Gut microbiome analysis for CRC detection: from raw sequences to machine learning models

Pankaj Chejara

2024-11-28

Table of contents

Introduction

This book is a result of the author's reflection of his learning of the field of bioinformatics. This book is created to bundle up all relevant learning materials and tutorials for gut microbiome analysis with a particular focus on early CRC detection.

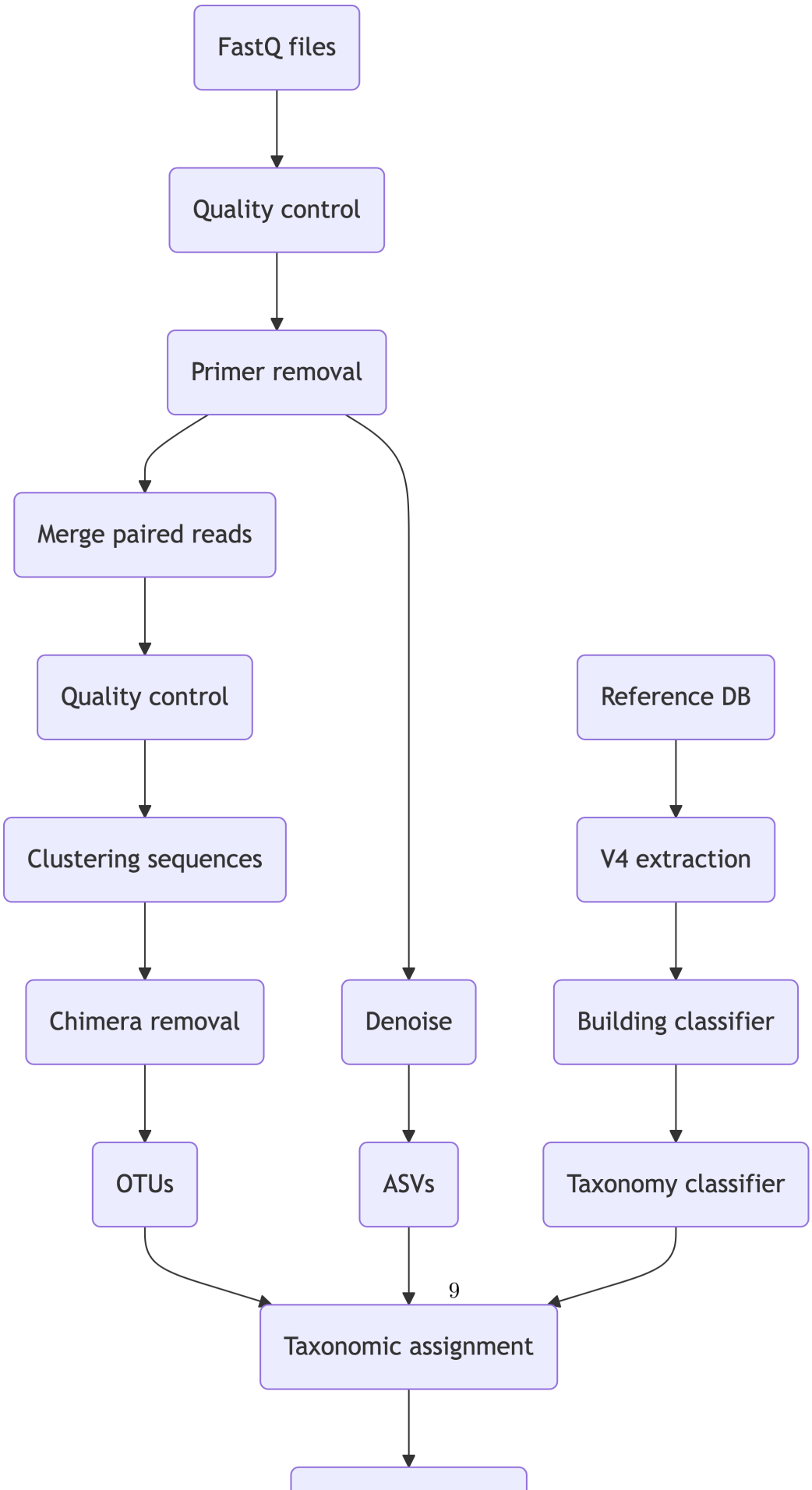
Part I

16S rRNA Sequence Processing

In this part of the book, we will dive into processing steps for 16S rRNA raw reads to obtain taxonomic information about microbiome represented in the data. There are two different ways to achieve that: Operational taxonomic units and Amplicon sequence variants.

You can check this resource to understand the differences between the two.

The diagram below shows all steps involved in processing 16S rRNA sequence data.


FastQ files
Quality control
Primer removal
Merge paired reads
Quality control
Clustering sequences
Chimera removal
OTUs
Denoise
ASVs
Reference DB
V4 extraction
Building classifier
Taxonomy classifier
Taxonomic assignment

Part II

References

1. https://astrobiomike.github.io/misc/amplicon_and_metagen

1 Reference database

We will first setup our reference database in QIIME2 (version 2024.10). For that we are going to use **SILVA database** (Pruesse et al. 2007; Quast et al. 2012). There are other databases as well, such as, Greengenes, RDP.

To simplify the process of setting up reference database, we will use an excellent plugin that is **Rescript** (Robeson et al. 2021). This plugin provides an already built pipeline to download and process different reference databases (read more here).

1.1 Installing RESCRIPt plugin

We will use the following command to install the plugin

```
pip install git+https://github.com/bokulich-lab/RESCRIPt.git
```

1.2 Preparing SILVA database as reference database

We will follow steps provided here to download and build SILVA database for use in QIIME.

1.2.1 Step-1: Download SILVA

```
qiime rescript get-silva-data \
    --p-version '138.1' \
    --p-target 'SSURef_NR99' \
    --o-silva-sequences silva-138.1-ssu-nr99-rna-seqs.qza \
    --o-silva-taxonomy silva-138.1-ssu-nr99-tax.qza
```

> ⚠ RESCRIPt error
>
> RESCRIPt plugin works well for some QIIME versions. In case if you encounter the problem of `cannot import name 'DNASequence' from 'q2_types.genome_data'` then re-

fer to this page for detailed instructions to resolve the issue. The same steps are also provided below.
We will **create an additional conda environment and install required packages** to allow rescript to function properly.

```
conda install -c conda-forge -c bioconda -c qiime2 \
-c https://packages.qiime2.org/qiime2/2023.9/shotgun/released/  \
-c defaults   xmltodict 'q2-types-genomics>2023.5' ncbi-datasets-pylib

pip install git+https://github.com/bokulich-lab/RESCRIPt.git

qiime dev refresh-cache

qiime rescript --help
```

1.2.2 Step-2: Converting rna sequences to dna sequences

The resultant sequences from the above step are of 'RNASequences' data type. To ensure a smooth downstream analysis, we will convert data type to 'DNASequences' using the following command.

```
qiime rescript reverse-transcribe \
    --i-rna-sequences silva-138.1-ssu-nr99-rna-seqs.qza \
    --o-dna-sequences silva-138.1-ssu-nr99-seqs.qza
```

The resultant qiime artifact `silva-138.1-ssu-nr99-seqs.qza` now can be used to train a classifier. Additional steps could also be integrated in the process before building the classifier. Such as **cutting low quality sequences**, **filtering based on length**, etc.

The link provides some of those examples and command to do that using `RESCRIPt` plugin.

1.3 Building a taxonomy classifer

Now we will move to train a taxonomy classifier which we are going to use later in our analysis to assign taxonomy labels to sequence data. Before doing that we will select V4 region from the SILVA database and use that extracted database for training our classifier.

This step of extracting V4 regions has been found to improve the performance.

1.3.1 Extract V4 region

This step extract the 16S region from the database using provided primers.

> Tip
>
> It is recommeded to use the same primers used in the 16S extraction process of dataset under study.

```
qiime feature-classifier extract-reads \
    --i-sequences silva-138.1-ssu-nr99-seqs.qza \
    --p-f-primer GTGYCAGCMGCCGCGGTAA \
    --p-r-primer GGACTACNVGGGTWTCTAAT \
    --p-n-jobs 2 \
    --p-read-orientation 'forward' \
    --o-reads silva-138.1-ssu-nr99-seqs-515f-806r.qza
```

1.3.2 Train the classifier

This step use a Naive-Bayes classifier and trains it on the extracted data from the previous step. The resultant classifier is stored as Qiime2 artifact which can be readily used for classification tasks using Qiime2.

```
qiime feature-classifier fit-classifier-naive-bayes \
  --i-reference-reads silva-138.1-ssu-nr99-seqs-515f-806r.qza \
  --i-reference-taxonomy silva-138.1-ssu-nr99-tax.qza \
  --o-classifier silva_classifier.qza
```

References

2 Quality control

The first step in the process is to learn about the quality of sequence reads. As Fastq files contains a quality score per base which indicates the level of accuracy for a particular base read. Our goal in the first step is to check is there something abnormal with the quality scores, e.g., do quality degrades after a particular read position.

Thankfully, we have tools which can automate this lookup process for us. The first tool is `Fastqc` which scans each sequence read file and then generate a report per file, providing essential information on many quality aspects of sequence data.

Now imagine going through each generated file to come up with some sort of understanding of the overall quality of datasets. It could be an exhausting task. Thanks again to the bioinformatics field, we have another tool `Multiqc` which summarizes multiple report files into a single report summary which makes it easier to determine the quality of sequence reads.

2.1 Generating quality report using fastqc

The fastqc command has the following syntax

```
fastqc  -o ./fastqc_results -t 10 ./raw_data/*.fastq.gz
```

Here, **-o** flag specifies output directory, and **-t** flag specifies number of threads to process the files.

2.2 Summarizing all quality reports using multiqc

The following command runs `multiqc` and generate a summary report of fastqc reports.

```
multiqc ./fastqc_results -o ./multiqc_results
```

2.3 Applying trimmomatic

Now, we will use insights gained from multiqc report to determine some parameters such as minimum length of sequences. These parameters then we will pass in the `trimmomatic` command.

```
trimmomatic -PE ADenoma12-2799_S12_L001_R1_001.fastq.gz ADenoma12-2799_S12_L001_R2_001.fastq
```

> **Tip**
>
> Similar command as above must be run for each paired (or single) end sequences file. To automate this, we can use a shell script.

```
#!/usr/bin/env bash
# author: pankaj chejara
# Script to iterate over paired read sequences to apply trimmomatic

samples=()

for filename in ./raw_data/*.fastq.gz; do
    base=$(basename "$filename" .fastq.gz)
    nf=$(echo $base | sed -e 's/.......$//');
    if ! [[ ${samples[@]} =~ $nf ]]
    then
      samples+=("$nf");
    fi
  done

for sample in "${samples[@]}"; do
  forward="./raw_data/${sample}_R1_001.fastq.gz"
  backward="./raw_data/${sample}_R2_001.fastq.gz"
  baseout="./trimm_outputs/${sample}.fastq.gz"
  trimmomatic PE -threads 20 $forward $backward -baseout $baseout CROP:200"
done;
```

References

3 Loading data in QIIME

Now, we will import our dataset into QIIME2 for further processing. There are different data formats supported by QIIME2 when it comes to importing. So, having a basic knowledge about those formats and their corresponding procedures to import them would be helpful.

More information can be accessed here

In our previous step, we applied filtering using trimmomatic tool. This filtering step generated new sequence files with names with suffixes like `1P_L001.fastq.gz`. We we will create a manifest file containing filepath of forward and backward sequence read files.

3.1 Create manifest file

To automate the generation of manifest file, we will use the following shell script

This script will generate a manifest file which looks like this

sample-id	absolute-filepath	direction
ADenoma12-2799	/Users/pankaj/Documents/Metrosert/Public dataset/zakular2014/trimm_outputs/ADenoma12-2799_S12_L001_1P.fastq.gz	forward
ADenoma12-2799	/Users/pankaj/Documents/Metrosert/Public dataset/zakular2014/trimm_outputs/ADenoma12-2799_S12_L001_2P.fastq.gz	backward
Ademona1-2065	/Users/pankaj/Documents/Metrosert/Public dataset/zakular2014/trimm_outputs/Ademona1-2065_S1_L001_1P.fastq.gz	forward

> **Tip**
>
> Do not forget to change the directories in the script as per your environment. The script scans `raw_data` directory to fetch all sample ids and then generate manifest file with filepaths (of files obtained after trimmomatic).

Listing 3.1 `create_manifest.sh`

```
# author: Pankaj chejara
# Script to create manifest file for qiime2

samples=()

for filename in ./raw_files/*.fastq.gz; do
    base=$(basename "$filename" .fastq.gz)
    nf=$(echo $base | sed -e 's/.......$//');
    if ! [[ ${samples[@]} =~ $nf ]]
    then
      samples+=("$nf");
    fi
  done

echo "sample-id    forward-absolute-filepath    reverse-absolute-filepath" > manifest.csv

for sample in "${samples[@]}"; do
  forward="$PWD/trimm_outputs/${sample}_1P.fastq.gz"
  backward="$PWD/trimm_outputs/${sample}_2P.fastq.gz"
  sampleid=$(echo $sample | sed -e 's/_.*//')
  echo "$sampleid    $forward    $backward" >> manifest.csv
  #echo "$sampleid,$backward,backward" >> manifest.csv
done;
```

3.2 Import sequences reads in QIIME

To import our dataset, we will use the generated manifest file and run the following command.

```
qiime tools import \
  --type 'SampleData[PairedEndSequencesWithQuality]' \
  --input-path manifest.csv \
  --input-format PairedEndFastqManifestPhred33V2 \
  --output-path paired-end-demux.qza
```

On successful execution of the above command, a new qiime artifact `paired-end-demux.qza` will be generated.

4 Removing primers

This step will remove primers from sequence read data. For that we will use QIIME with `cutadapt` plugin.

We will use the information from (Zackular et al. 2014) on forward and reverse primer used in sequencing. The following command remove primers from our dataset.

```
qiime cutadapt trim-paired \
--i-demultiplexed-sequences paired-end-demux.qza \
--p-front-f GTGYCAGCMGCCGCGGTAA \
--p-front-r GGACTACNVGGGTWTCTAAT \
--o-trimmed-sequences paired-end-demux-trimmed.qza"
```

References

5 ASVs

This step will extract Amplicon Sequence Variants (ASVs) from read sequences. This step involves the following two steps

1. **Denoising**: This step perform quality filtering and chimera removing. The results of this step is include a feature table and a frequency count table.
2. **Taxonomic classification**: This step assign taxonomic classification to the extracted features.

5.1 Denoising

This step will denoise our sequence dataset and also remove chimera sequences. We will use **dada2** (Callahan et al. 2016) for denoising.

This step performs the following three tasks

1. Quality filtering
2. Chimera checking
3. Paired-end joining

```
qiime dada2 denoise-paired \
  --i-demultiplexed-seqs demultiplexed-sequences.qza \
  --p-trunc-len-f 204 \
  --p-trim-left-r 1 \
  --p-trunc-len-r 205 \
  --o-representative-sequences asv-sequences-0.qza \
  --o-table feature-table-0.qza \
  --o-denoising-stats dada2-stats.qza
```

We will also generate a summary file to gain insights into the result of the command.

```
qiime metadata tabulate \
  --m-input-file dada2-stats.qza \
  --o-visualization dada2-stats-summ.qzv
```

5.2 Taxonomic classification

```
qiime feature-classifier classify-sklearn \
    --i-classifier /home/pankaj/reference_db/silva-138.1-ssu-nr99-515f-806r-classifier.qza \
    --i-reads asv-sequences-0.qza
    --o-classification rep-seqs-classification.qza
```

References

6 OTUs

This step generates Operational Taxonomic Units (OTUs). The following steps are executed to extract OTUs.

1. Merging paired-end read sequences
2. Filtering merged sequences using quality scores
3. Dereplicating merged sequences
4. Clustering sequences
5. Chimera removal
6. Taxonomic classification

6.1 Merging paired-end read sequences

As the first step, we will merge forward and backward read sequences into one. The resultant sequences will be used for next steps.

```
qiime vsearch merge-pairs \
--i-demultiplexed-seqs paired-end-demux-trimmed.qza \
--o-merged-sequences demux-joined.qza \
--o-unmerged-sequences demux-unjoined.qza
```

6.2 Filtering merged sequences using quality scores

Now we will use merged pairs and apply quality filtering.

```
qiime quality-filter q-score \
--i-demux demux-joined.qza \
--p-min-quality 20 \
--o-filtered-sequences demux-joined-filtered.qza
--o-filter-stats demux-joined-filter-stats.qza
```

6.3 Dereplicating merged sequences

This step identify duplicate sequences and remove them. This step reduces final sequences for clustering.

```
qiime vsearch dereplicate-sequences \
        --i-sequences demux-joined-filtered.qza \
        --o-dereplicated-table derep-joined-filtered_table.qza \
        --o-dereplicated-sequences derep-joined-filtered_seqs.qza
```

6.4 Clustering sequences

```
qiime vsearch cluster-features-de-novo \
        --i-table derep-joined-filtered_table.qza \
        --i-sequences derep-joined-filtered_seqs.qza \
        --p-perc-identity .97 \
        --o-clustered-table cluster-table.qza \
        --o-clustered-sequences cluster-seqs.qza  \
        --p-threads 8
```

6.5 Chimera removal

```
qiime vsearch uchime-ref \
    --i-table cluster-table.qza \
    --i-sequences  cluster-seqs.qza \
    --i-reference-sequences silva-138.1-ssu-nr99-seqs-515f-806r.qza \
    --p-threads 4 \
    --o-chimeras chimeras.qza \
    --o-nonchimeras non-chimera.qza \
    --o-stats chimera-stats.qza
```

```
qiime feature-table filter-features \
    --i-table cluster-table.qza \
    --m-metadata-file non-chimera.qza \
    --o-filtered-table table-nc.qza"
```

```
qiime feature-table filter-seqs \
    --i-data cluster-seqs.qza \
    --i-table table-nc.qza \
    --o-filtered-data seqs-nc.qza
```

6.6 Taxonomic classification

```
qiime feature-classifier classify-sklearn \
    --i-classifier silva_classifier.qza \
    --i-reads seqs-nc.qza \
    --o-classification seqs-nc-otus.qza
```

`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6ImNoYXB0ZXJzLzE2UyJ9 -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6ImNoYXB0ZXJzLzE2UyIsImJvb2tJdGVtVHlwZSI6ImNoY
```

7 Streamlining the analysis process

In the previous chapters, we have seen various steps in generating taxonomic assignment for raw reads data. Manual execution of all those steps, one by one, could be time-consuming and may be prone to errors. To address these challenges, there are several workflow management tools available (e.g., snakemake, nextflow). These tools automate the execution of the entire pipeline and also enable reproducibility of running the same workflow with different datasets.

Here, we will see how to use Snakemake, a workflow management tool, to streamline the entire process of taxonomy generation.

7.1 Snakemake configuration file

As the first step, we will create a configuration file where we will keep all needed information for our analysis. It includes project directory, names of output files created in the analysis, parameters values for tools, etc.

This configuration file has information in the format of `key:value`. The below file is a configuration file which we are going to use for our short analysis (i.e., primer removal -> asv generation -> taxonomy assignment).

```
## Basic config
project: trial
scratch: /Users/pankaj/Documents/Metrosert/public_dataset/zakular2014
raw_data: partial_raw_data
outputDIR: partial_raw_asv
manifest: manifest_partial.csv
metadata: metadata_partial.tsv

## 16S adapters from Zackular et al., 2014
primerF: GTGCCAGCMGCCGCGGTAA
primerR: GGACTACHVGGGTWTCTAAT
primer_err: 0.4
primer_overlap: 3

## Reference database
```

```
database: silva-138.1-ssu-nr99-seqs-515f-806r.qza
database_classifier: silva_classifier.qza
database_tax: silva-138.1-ssu-nr99-tax.qza

## DADA2 - ASV flags
truncation_err: 2
truncation_len-f: 150
truncation_len-r: 140
truncation_err: 2
quality_err: 2

## Diversity metrics
sampling_depth: 500
```

7.2 Pipeline

Now, we will put our analysis codes in one place to be used by Snakemake. To do that, we need to create a file without specifying any extension. For example, we create a file `Snakefile-asv`. Next, we will write our analysis codes in the file a specific format (which we will see later).

Basically, we will do the following

1. Specify configuration file
2. Extract all information provided in configuration file
3. Write rules for each analysis step

```
# Snakemake file - input quality controlled fastq reads to generate asv
# Pankaj chejara

# Base snakefile: https://github.com/shu251/tagseq-qiime2-snakemake/blob/master/Snakefile-asv

configfile: "config.yaml"


import io
import os
import pandas as pd
import pathlib

##############################################################
```

```
#                  SET CONFIG VARS
############################################################

PROJ = config["project"]
SCRATCH = config["scratch"]
INPUTDIR = config["raw_data"]
OUTPUTDIR = config['outputDIR']
MANIFEST = config["manifest"]
METADATA = config["metadata"]
SAMPLING_DEPTH= config['sampling_depth']

# Database information
DB = config["database"]
DB_classifier = config["database_classifier"]
DB_tax = config["database_tax"]


rule all:
  input:
    q2_import = SCRATCH + "/" + OUTPUTDIR +"/" +  PROJ + "-PE-demux.qza",
    q2_primerRM = SCRATCH + "/" + OUTPUTDIR +"/" +  PROJ + "-PE-demux-noprimer.qza",
    # Visualization
    raw = SCRATCH + "/" + OUTPUTDIR + "/viz/" + PROJ + "-PE-demux.qzv",
    primer = SCRATCH + "/" + OUTPUTDIR + "/viz/" + PROJ + "-PE-demux-noprimer.qzv",
    # Dada2 results
    table = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-asv-table.qza",
    rep = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-rep-seqs.qza",
    stats = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-stats-dada2.qza",
    stats_viz = SCRATCH + "/" + OUTPUTDIR + "/viz/" + PROJ + "-stats-dada2.qzv",
    # Taxonomic table
    sklearn = SCRATCH + "/" + OUTPUTDIR + "/asv/" +  PROJ + "-tax_sklearn.qza",
    table_biom = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "feature-table.biom",
    table_tsv = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-asv-table.tsv",
    table_tax = SCRATCH + "/" + OUTPUTDIR + "/asv/"  + "taxonomy.tsv",
    # Phylogenetic outputs
    aligned_seqs = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-aligned-rep-seqs
    aligned_masked = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-masked-aligned-
    unrooted_tree = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-unrooted-tree.qz
    rooted_tree = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-rooted-tree.qza",
    # Diversity metrics
    output_dir = directory(SCRATCH + "/qiime2/diversity")
```

```
############################################################
#                  LOAD DATA
############################################################
rule import_qiime:
  input:
    MANIFEST
  output:
    q2_import = SCRATCH + "/" + OUTPUTDIR +"/" + PROJ + "-PE-demux.qza"
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_q2.log"
  params:
    type="SampleData[PairedEndSequencesWithQuality]",
    input_format="PairedEndFastqManifestPhred33"
  shell:
    """qiime tools import \
    --type {params.type} \
    --input-path {input} \
    --output-path {output.q2_import} \
    --input-format {params.input_format}
    """

############################################################
#                  REMOVE PRIMERS
############################################################

rule rm_primers:
  input:
    q2_import = SCRATCH + "/" + OUTPUTDIR +"/" + PROJ + "-PE-demux.qza"
  output:
    q2_primerRM = SCRATCH + "/" + OUTPUTDIR +"/" + PROJ +  "-PE-demux-noprimer.qza"
  log:
     SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_primer_q2.log"

  shell:
    """qiime cutadapt trim-paired \
       --i-demultiplexed-sequences {input.q2_import} \
       --p-front-f {config[primerF]} \
       --p-front-r {config[primerR]} \
       --p-error-rate {config[primer_err]} \
       --p-overlap {config[primer_overlap]} \
       --o-trimmed-sequences {output.q2_primerRM}"""
```

```
############################################################
#                  QC STATS
############################################################

rule get_stats:
  input:
    q2_import = SCRATCH + "/" + OUTPUTDIR +"/" +  PROJ + "-PE-demux.qza",
    q2_primerRM = SCRATCH + "/" + OUTPUTDIR +"/" +  PROJ + "-PE-demux-noprimer.qza"
  output:
    raw = SCRATCH + "/" + OUTPUTDIR + "/viz/" + PROJ + "-PE-demux.qzv",
    primer = SCRATCH + "/" + OUTPUTDIR + "/viz/" + PROJ + "-PE-demux-noprimer.qzv"
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ +  "_getviz_q2.log"
  shell:
    """
     qiime demux summarize --i-data {input.q2_import} --o-visualization {output.raw}
     qiime demux summarize --i-data {input.q2_primerRM} --o-visualization {output.primer}
    """

############################################################
#                  DENOISE & ASVs
############################################################

rule dada2:
  input:
    q2_primerRM = SCRATCH + "/" + OUTPUTDIR +"/" + PROJ + "-PE-demux-noprimer.qza"
  output:
    table = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-asv-table.qza",
    rep = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-rep-seqs.qza",
    stats = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-stats-dada2.qza"
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_dada2_q2.log"
  shell:
    """qiime dada2 denoise-paired \
        --i-demultiplexed-seqs {input.q2_primerRM} \
        --p-trunc-q {config[truncation_err]} \
        --p-trunc-len-f {config[truncation_len-f]} \
        --p-trunc-len-r {config[truncation_len-r]} \
        --o-table {output.table} \
        --o-representative-sequences {output.rep} \
        --o-denoising-stats {output.stats}"""
```

```
rule dada2_stats:
  input:
    stats = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-stats-dada2.qza"
  output:
    stats_viz = SCRATCH + "/" + OUTPUTDIR + "/viz/" + PROJ + "-stats-dada2.qzv"
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_dada2-stats_q2.log"
  shell:
   """qiime metadata tabulate \
       --m-input-file {input.stats} \
       --o-visualization {output.stats_viz}"""


############################################################
#                 TAXONOMIC ASSIGNMENT
############################################################

rule assign_tax:
  input:
    rep = SCRATCH + "/" + OUTPUTDIR + "/asv/" +  PROJ + "-rep-seqs.qza",
    db_classified = DB_classifier
  output:
    sklearn = SCRATCH + "/" + OUTPUTDIR + "/asv/" +  PROJ + "-tax_sklearn.qza"
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ +  "_sklearn_q2.log"
  shell:
    """qiime feature-classifier classify-sklearn \
      --i-classifier {input.db_classified} \
      --i-reads {input.rep} \
      --o-classification {output.sklearn}"""


############################################################
#                 TAXONOMIC TABLE GENERATION
############################################################
rule gen_table:
  input:
    table = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-asv-table.qza"
  output:
    table_biom = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "feature-table.biom"
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_exportBIOM_q2.log"
```

```
  params:
    directory(SCRATCH + "/" + OUTPUTDIR + "/asv/")
  shell:
    "qiime tools export --input-path {input.table} --output-path {params}"

rule convert:
  input:
    table_biom = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "feature-table.biom"
  output:
    SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-asv-table.tsv"
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_exportTSV_q2.log"
  shell:
    "biom convert -i {input} -o {output} --to-tsv"

rule gen_tax:
  input:
    sklearn = SCRATCH + "/" + OUTPUTDIR + "/asv/" +  PROJ + "-tax_sklearn.qza"
  output:
     table_tax = SCRATCH + "/" + OUTPUTDIR + "/asv/"  + "taxonomy.tsv",
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_exportTAXTSV_q2.log"
  params:
    directory(SCRATCH + "/" + OUTPUTDIR + "/asv/")
  shell:
    "qiime tools export --input-path {input.sklearn} --output-path {params}"


############################################################
#                  PHYLOGENETIC TREE
############################################################
rule phy_tree:
  input:
     rep = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-rep-seqs.qza",
  output:
    aligned_seqs = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-aligned-rep-seqs
    aligned_masked = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-masked-aligned-
    unrooted_tree = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-unrooted-tree.qz
    rooted_tree = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-rooted-tree.qza",
  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_phylogeneticTREE_q2.log"
```

```
  shell:
    """qiime phylogeny align-to-tree-mafft-fasttree \
        --i-sequences {input.rep} \
        --o-alignment {output.aligned_seqs} \
        --o-masked-alignment {output.aligned_masked} \
        --o-tree {output.unrooted_tree} \
        --o-rooted-tree {output.rooted_tree}"""


############################################################
#                  DIVERSITY METRICS
############################################################
rule div_met:
  input:
     rooted_tree = SCRATCH + "/" + OUTPUTDIR + "/asv/" + "tree/" + PROJ + "-rooted-tree.qza"
     table = SCRATCH + "/" + OUTPUTDIR + "/asv/" + PROJ + "-asv-table.qza"
  output:
     output_dir = directory(SCRATCH + "/qiime2/diversity")

  log:
    SCRATCH + "/" + OUTPUTDIR + "/logs/" + PROJ + "_phylogeneticTREE_q2.log"

  shell:
    """qiime diversity core-metrics-phylogenetic \
        --i-phylogeny {input.rooted_tree} \
        --i-table {input.table} \
        --p-sampling-depth {SAMPLING_DEPTH} \
        --m-metadata-file {METADATA} \
        --output-dir {output.output_dir}"""
```

References

1. https://github.com/shu251/tagseq-qiime2-snakemake/blob/master/Snakefile-asv
2. https://forum.qiime2.org/t/qiime2-snakemake-workflow-tutorial-18s-16s-tag-sequencing/11334

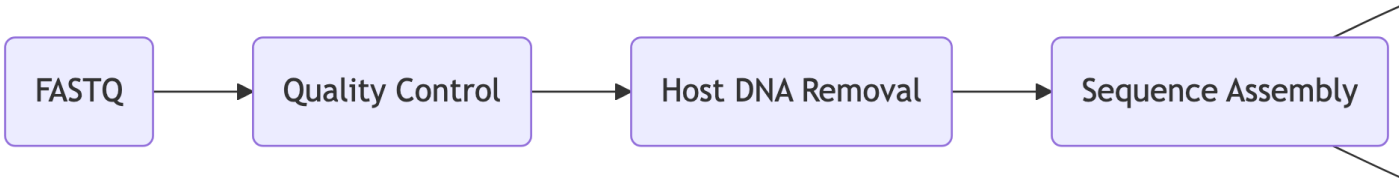
Part III

Metagenomics Sequence Processing

Metagenomic raw sequences consist of strings composed of four letters: A, C, T, and G, representing the nucleotide bases Adenine, Cytosine, Thymine, and Guanine, respectively. These bases form the building blocks of DNA. Machines that sequence DNA extract these bases from a DNA sample and store them in files, commonly in formats such as FASTQ or FASTA.

In this tutorial, we will focus on the widely adopted FASTQ format. A FASTQ file typically contains essential information about the sample, the nucleotide sequences, and a quality score for each identified base. The quality score indicates the probability of correctly identifying a specific base, providing a valuable measure of sequencing accuracy.

We will use FASTQ files to prepare data for further analysis. The figure below illustrates the processing pipeline and the tools involved in this workflow.



- **Quality Control**: This is the first step where we will assess the quality of raw sequences to identify low-quality reads and adapter contamination. Tools such as **FASTQC**, **MULTIQC**, and **Fastp** will help generate quality metrics like per-based quality scores and read length distribution. Trimming tools like **Trimmomatic** or **Trim-glore** then help clean the data.
- **Human DNA Removal**: This step removes host DNA from metagenomic raw sequences to ensure that the downstream analysis focuses fully to microbial sequences. Tools such as **bowtie2** and **samtools** will be used for this task. The reference human genome (e.g., GRCh38) will be downloaded and indexed before alignment.
- **Sequence Assembly**: In this step, we will assemble short reads into longer sequences. Tools like **SPAdes** (ideal for small database) and **MegaHit** (optimized for large datasets) will be employed for assembly.
- **Taxonomic Profiling**: This step assigns operational taxonomic units (OTUs) to assembeled and quality controled raw sequences to determine the taxonomic composition of the microbiome. To execute this task, we will explore the usage of tools such as **MetaPhlan2**, **BowTie2**, and **mOTUs**.

- **Functional Annotation**: While taxonomic profile identifies "who is there", functional annotation determines "what they are doing". It assigns functional roles and metabolic pathways present in provide metagenomic sequences. Tools like **HUMAnN** will be used for the task.

8 Quality control

We will start by looking into quality scores of our sequences to do a quality check. To achieve this task, we use FASTQC program with raw sequencing data. FASTQC provides an easy-to-understand html report, facilitating an efficient quality check of fastq files.

The command below run the `fastqc` program for each fastq (or compressed fastq.gz) file and produces a report for each file in the `output_directory`.

```
fastqc fastq_directory -o output_directory
```

In our case, we have multiple compressed fastq files with the file extension of `.fastq.gz`. We will use the expression `*.fastq.gz` to specify all those files for the processing. The following command run fastqc program for every file.

```
fastqc ./raw_data/*.fastq.gz -o ./fastqc_results
```

This execution results in the generation of a `html` and a `zip` file for each processed fastq file. The html file provides a report consisting of tables, figures about the quality check. The zip file contains all the figures, tables, and summaries. Figure 8.1 shows the distribution of quality scores for each position for a particular fastq file (`SRR6915103_1`).

Such report is generated for each `fastq` file and it can become cumbersome checking each of those files for quality check. To simplify this step and aggregate all such reports into a single one, we use MULTIQC program (Ewels et al. 2016).

The command below run multiqc program which generates a html report and a directory of images, tables, and summaries.

```
multiqc ./fastqc_results -o multiqc_results
```

The below image shows a snapshot of multiqc report.

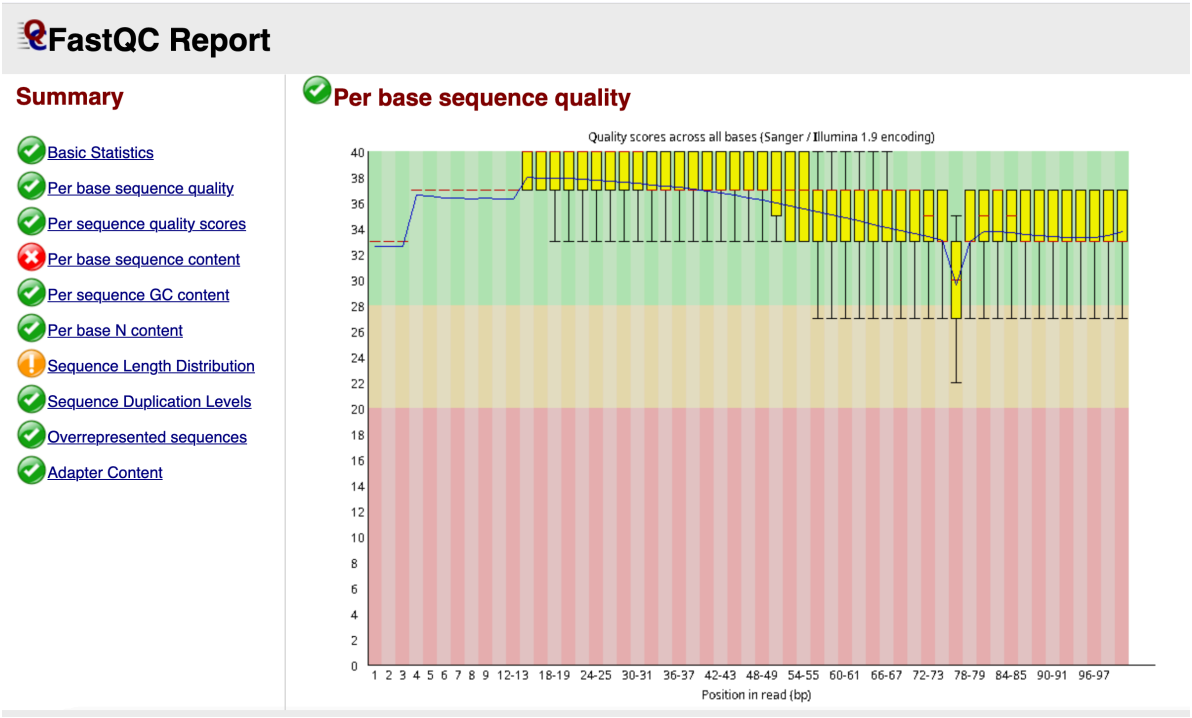


Figure 8.1: Qaulity score distribution

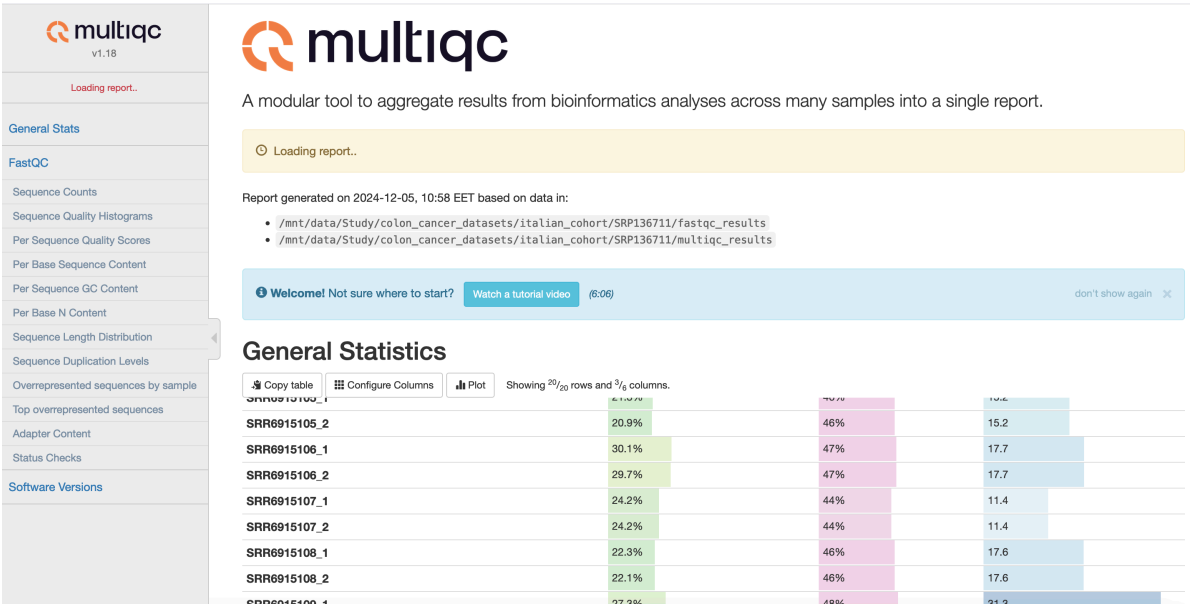


Figure 8.2: Multiqc report

8.1 Trimming sequences to enhance quality

This task involves removing sequences due to low quality or due to other measures (e.g., shorter than a particular length).

We will use here `trim-galore` (Krueger 2015) to perform the trimming task. The program runs for each paired-eng sequence. Therefore, we will write a script to iterate over all the fastq files in the directory.

The following script provides that functionality.

```
#!/bin/bash

# Create output directory if it doesn't exist
mkdir -p glore_output

# Loop over all R1 files ending with _1.fastq.gz
for r1 in raw_data/*_1.fastq.gz; do

  # Derive the corresponding R2 file by replacing _1 with _2
  r2="raw_data/$(basename "$r1" _1.fastq.gz)_2.fastq.gz"

  # Run Trim Galore for each pair and store output in glore_output
  trim_galore --nextera  --quality 20 --length 75 --paired "$r1" "$r2" --output_dir glore_ou
done
```

References

9 Host DNA Removal

In this step, we will remove human genomes from our raw sequences to ensure a high-quality analysis of gut microbiome. For this task, we will use `bowtie2` (Langmead and Salzberg 2012) and `samtools` (H. Li 2009) tools.

The removal of human genome is a multi-step process. These steps are given below

9.1 Download human genome index files for bowtie

The first step is to download bowtie2 indices for human genomes. There are multiple human genomes indices are available here. For this tutorial, we will use `hg19` human genome (Thomas et al. 2019).

The command below download the genome zipped file and extract human genome indices.

```
wget https://genome-idx.s3.amazonaws.com/bt/hg19.zip
unzip hg19.zip
```

9.2 Remove human genome

This step consisting of three tasks:

1. **Aligning raw sequences to human genomes**,
2. **Removing mapped sequences**
3. **Converting resultant files back to fastq format**.

The **first task** is executed by the following command

```
bowtie2 -x hg19/hg19  -1 reads_1.fastq.gz -2 reads_2.fastq.gz -S mapped.sam --very-sensitive
```

The **second task**, which remove mapped sequences, is performed by the following command.

```
# Convert sam file into bam file
samtools view -Sb mapped.sam > mapped.bam

# Extract unmapped sequences
samtools view -b -f 12 -F 256 mapped.bam > unmapped.bam
```

The **third task** is executed by the following command

```
# Convert bam file back to fastq format
samtools fastq unmapped.bam -1 unmapped_1.fastq -2 unmapped_2.fastq -0 unmapped_singletons.fa
```

References

10 Sequence Assembly

This step assembles short reads into a longer one for taxonomic and functional profiling. This operation has been found to imrove performance of taxonomic assignment (Tran and Phan 2020)

We will employ **MegaHit** tool (D. Li et al. 2015) for assembly. This tool is memory efficient, making it an ideal choice for larger datasets. The following command performs assembly task for a single paired-read sequence.

```
megahit -1 sample_R1.fastq -2 sample_R2.fastq -o megahit_output_dir
```

Here, `sample_R1.fastq` and `sample_R2.fastq` are forward and backward reads. The results are stored in `megahit_output_dir` specified using `-o` flag.

References

11 Taxonomic Profiling

In the field of bioinformatics, **taxonomy** is a classification scheme which typically classifies each living organism into a heirarchical order composed of different levels such as domain, kingdom, phylum, etc. (Figure 11.1). This classification facilitates an understanding of micro-organism community structure at different levels, from coarse (e.g., phylum) to refine (e.g., species).

Taxonomic profiling predics taxonomic operational units present in the metagenomic sequences at different levels of classification. To perform this task, we will employ **mOTUs** tool (Ruscheweyh et al. 2021).

```
motus profile --forward <sample_R1.fastq> --reverse <sample_R2.fastq> --output motus_output_
```

Here, `sample_R1.fastq` and `sample_R2.fastq` are forward and backward reads. The results are stored in `motus_output_dir` specified using `-output` flag. The database of marker genes specified by `<path_to_mOTUs_database>`.

References

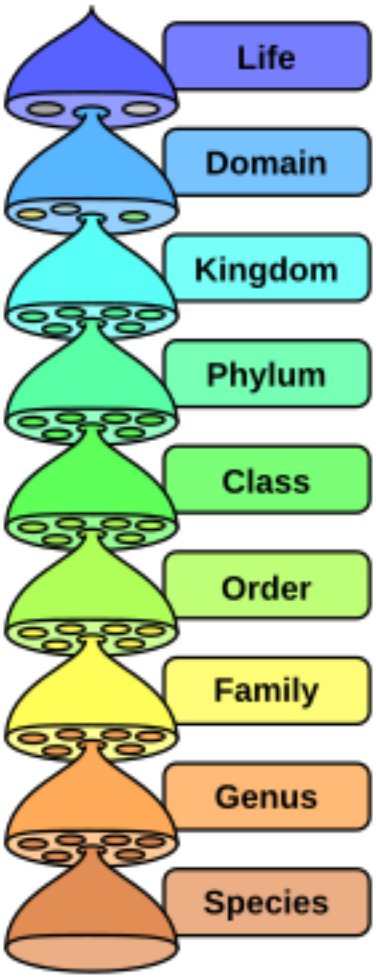


Figure 11.1: Biological classification (from WikiPedia)

12 Functional Profiling

This step determines metabolic pathways of present microbial commnuties in gut microbiome. todo:read about metabolic pathways of gut microbial communities.

We will employ **HUMAnN2** (Franzosa et al. 2018) for generating functional annotation of metagenomic data. This step utilises functional gene databases such as KEGG (Kyoto Encycolpedia of Genes and Genomes). The command below shows syntax for using HUMAnN2 for the task.

```
humann2 --input <input_file.fasta> --output <output_directory> --nucleotide-database /path/to
```

References

Part IV

./Analysis/Analysis_intro.qmd

13 Data loading

We will begin with a subset of dataset from Zeller et al. (2014). This dataset is from the fecal samples collected from **156 French patients**. In this section, we will load the dataset and explore its basic characteristics.

We will also extract the bacterial species column names. Those column names starting with `k__Bacteria` represents bacterial species.

```
import pandas as pd

# loading tab-seperated data file using pandas and transposing it
data = pd.read_csv('Nine_CRC_cohorts_taxon_profiles.tsv',sep='\t',header=None).T

# setting the first row as column names and then removing it
data = data.rename(columns=data.loc[0]).drop(0, axis=0)

# accessing Zeller et al., 2014 dataset
zeller_db = data.loc[data['dataset_name'] == 'ZellerG_2014',:]

# fetching microbacterial organism information-related columns
bacteria_colnames = [col for col in data.columns if 'k__Bacteria' in col]

# metadata colnames
metadata_colnames = ['dataset_name', 'sampleID', 'subjectID', 'body_site', 'study_condition'
                     'disease', 'age', 'age_category', 'gender', 'country','ajcc','alcohol',
                     'antibiotics_current_use','curator','disease_subtype','ever_smoke','fob
                     'hba1c','hdl','ldl','location','BMI']

print('Total features: ', zeller_db.shape[1])
```

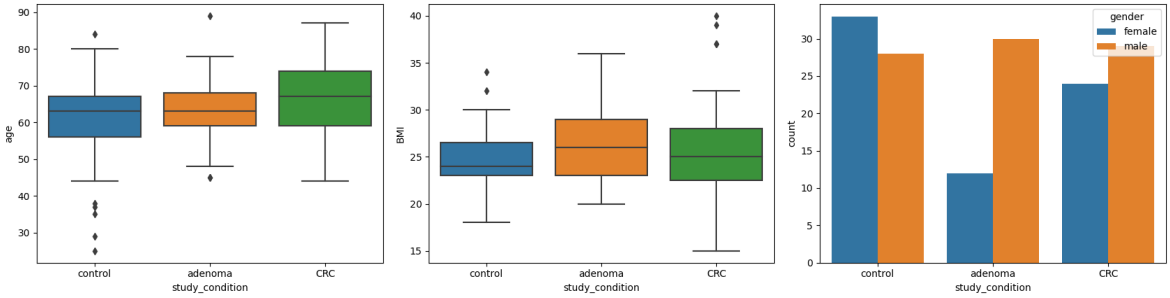
```
Total features:  829
```

For the rest of our analysis, we will focus on five metadata features along with OTUs. Those features are `age`, `gender`, `BMI`, `study_condition` and `ajcc`.

13.1 Age, BMI and gender distribution

Figure 13.1 shows distributions of age, BMI, and gender of patients across different study conditions: control, adenoma, and CRC.

- CRC patients are **slightly older than control cases.**
- There is **an increase in BMI for adenoma compared to control cases.**
- There are more **males with CRC than females.**



(a) Age distribution (b) BMI distribution (c) Gender distribution

Figure 13.1: Distribution across study conditions

14 Species filtering

We will filter out species whose **relative abundance does not exceed 0.001 in any sample**. This **criterion** is derived from the study by **Zeller et al., the same study** that provided the dataset (Zeller et al. 2014).

After applying this abundance filter, the dataset is reduced to **491 species**. We will now proceed with model development using these filtered species.

```
import numpy as np
import matplotlib.pyplot as plt

# dataset containing only bacterial microoganism's relative abundace
microbiome = zeller_db[bacteria_colnames]

# converting data types
for col in microbiome:
    microbiome.loc[:,col] = pd.to_numeric(microbiome[col], errors='coerce')

# fetching names of columns with abundance exceeding .001
columns_to_fetch = microbiome.columns[microbiome.max(axis=0) > 0.001]

# filtering dataset
microbiome_filtered = microbiome[columns_to_fetch]

plt.figure()
plt.bar([1,2],[len(microbiome.columns),len(columns_to_fetch)],alpha=.8)
plt.xticks([1,2],['before','after'])
plt.ylabel('Number of microbial species')
plt.title('Before and after species filtering')
plt.show()
```

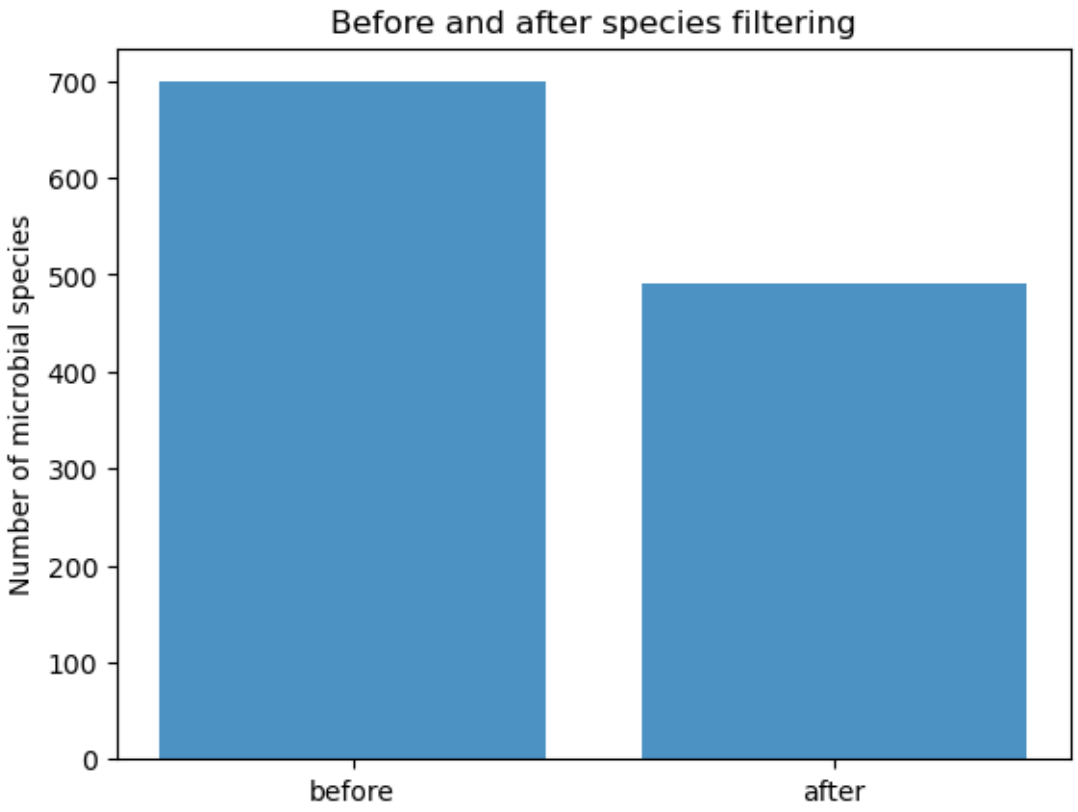


Figure 14.1: Before and after species filtering

14.1 References

15 Feature transformation

After filtering species on abundance criterion, we will now transform the data using a log-transformation. We will use the same transformational function that is used in Zeller et al. (2014). The transformational function is given in Equation 15.1.

$$log_{10}(x + x_0) \tag{15.1}$$

Here

x is a relative abundance value

x_0 is a small constant (1e-6)

We will apply this transformation on filtered species. The code below perform that step.

```
import numpy as np

# log transformation
microbiome_log = microbiome_filtered.applymap(lambda x: np.log10(x+.000001))
```

15.1 References

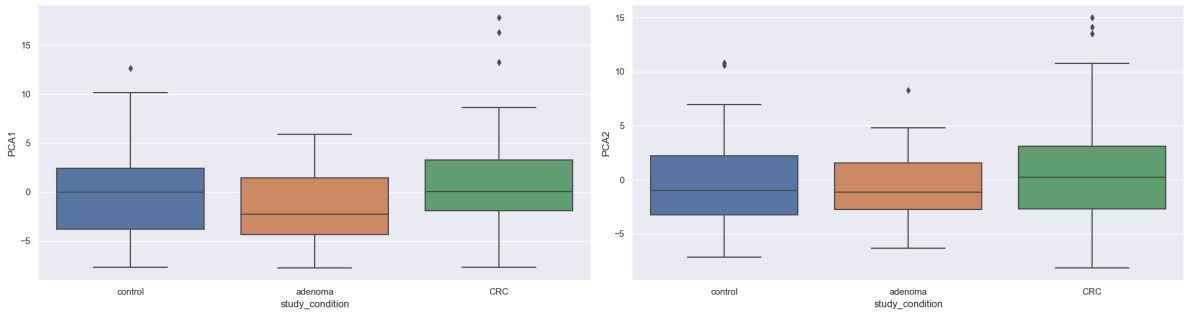
16 Principal Component Analysis

Principal Component Analysis (PCA) is a powerful dimensionality reduction technique that reduces the number of features (dimensions) while retaining the maximum variance in the data. By capturing the most important patterns in fewer components, PCA enhances data visualization and simplifies analysis.

We will now apply PCA to our filtered dataset and plot the resulting components to explore potential associations with study conditions. We will apply this transformation on filtered species. The code below perform that step.

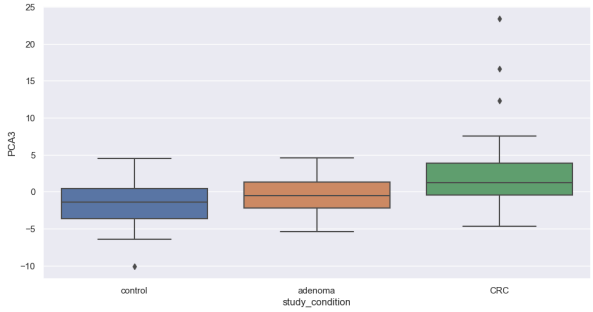
```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# log transformed data from the previous step
df = microbiome_log

# Step 1: Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Step 2: Apply PCA (reduce to 10 components for visualization)
pca = PCA(n_components=10)
pca_result = pca.fit_transform(scaled_data)

# Step 3: Create a DataFrame for PCA results
pca_df = pd.DataFrame(pca_result, columns=['PCA1', 'PCA2','PCA3', 'PCA4','PCA5',
                                           'PCA6','PCA7', 'PCA8', 'PCA9', 'PCA10'])

pca_df['study_condition'] = zeller_db['study_condition'].values
```

Figure 16.2 below shows the cumulative variance captured by 10 principal components, i.e. **26%**.

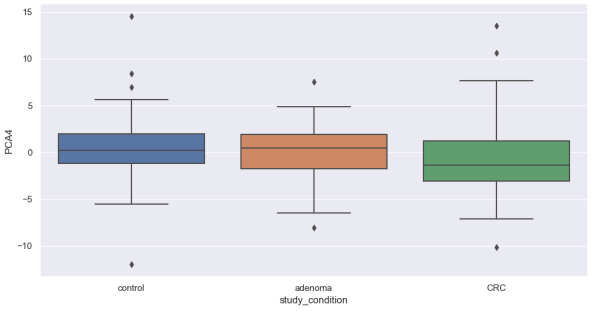


(a) PCA1

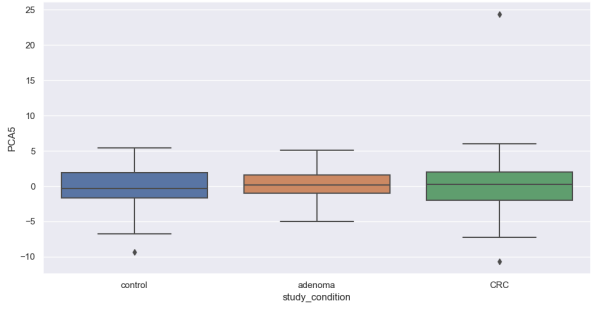
(b) PCA2



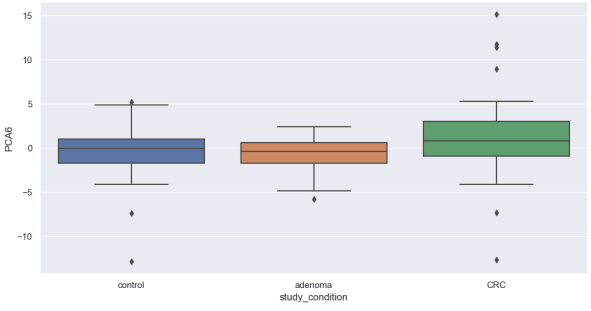
(c) PCA3



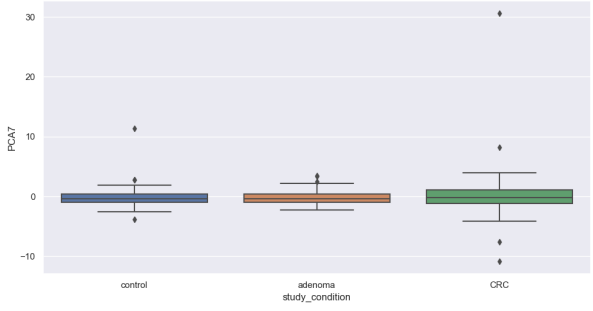
(d) PCA4



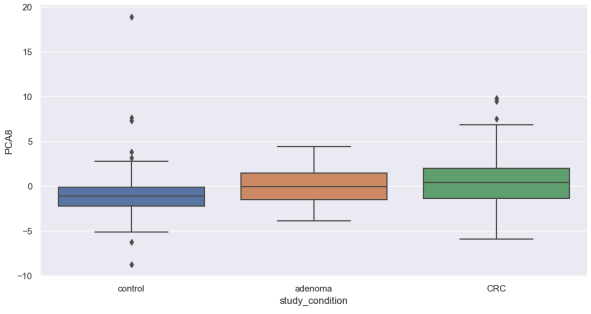
(e) PCA5



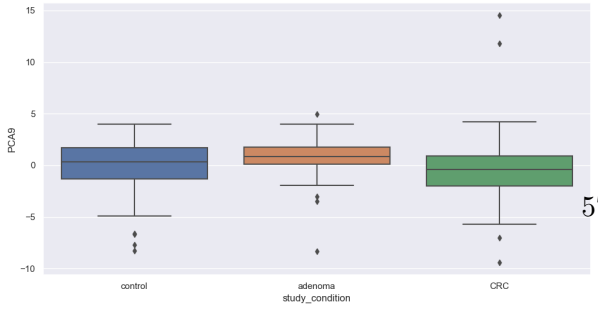
(f) PCA6



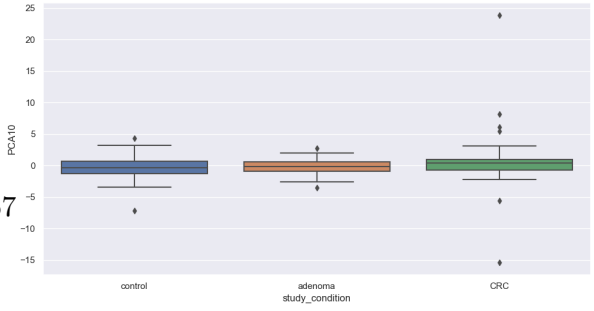
(g) PCA7



(h) PCA8



(i) PCA9



(j) PCA910

Figure 16.1: Principal components

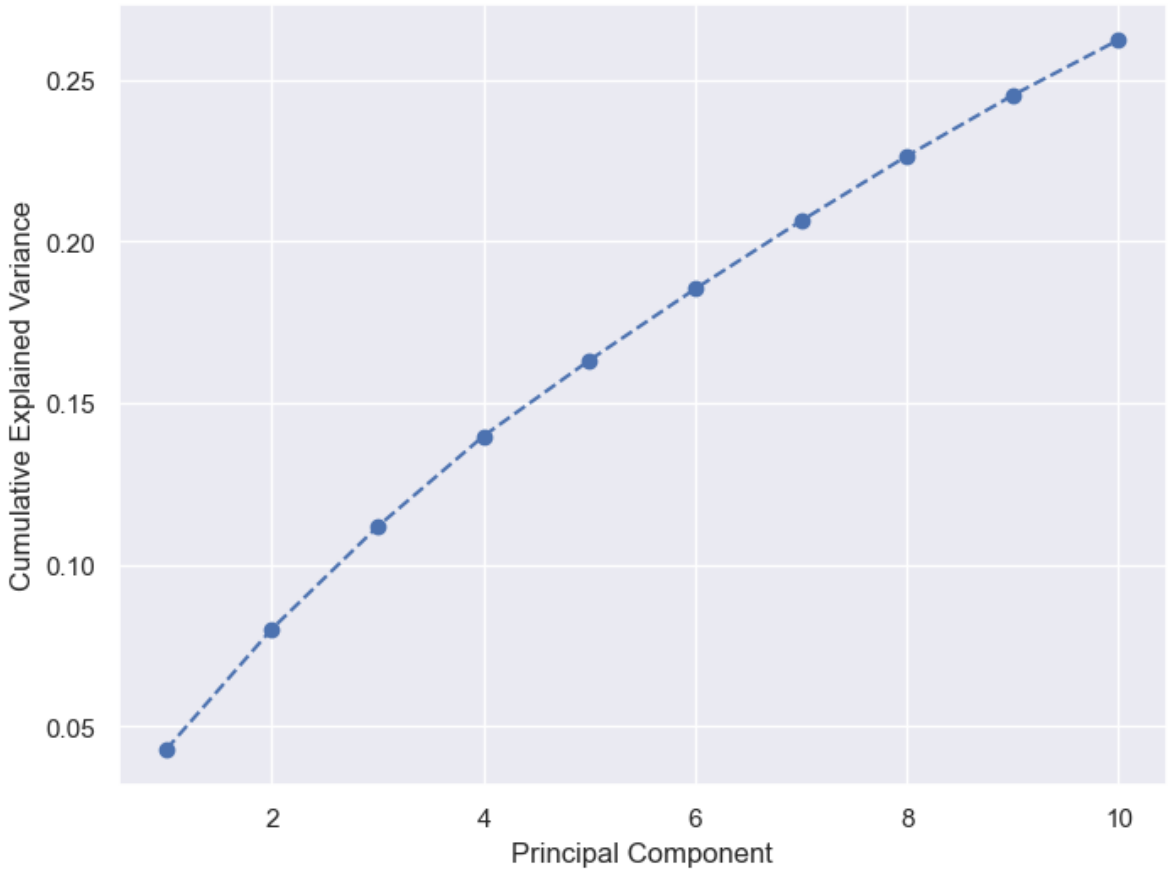


Figure 16.2: Cumulative Explained Variance by Principal Components

We will utilize PCA components in our modeling phase to build a CRC detection system.

17 Statistical analysis

Increasing research has found the association between microbial dysbiosis and CRC. Basing on that, we will check for differences in microbial composition of control and CRC cases. We will also investigate statistical significance of differences if found any.

17.1 Association between host's characteristics and CRC

We will now investigate for statistical differences among target classes in terms of patient demographics.

17.1.1 Age, BMI, and CRC

We start by investigating statistical significance of differences between age of control and CRC casese. For this test, we use **Mann-Whitney** test to investigate statistical significance of differences.

The below figures show that **there are statistical significant differences between `control` and `CRC` groups.

> **i** Age
>
> We found **statistical differences in age** of control and CRC group.

```
from skbio.diversity import alpha
from statannot import add_stat_annotation

# functions to process the data and prepare in a format supporting phyloseq analysis
def get_sample_table(df):
    df = df[metadata_colnames]
    df = df.set_index('subjectID')
    df.drop(['dataset_name','sampleID'],axis=1,inplace=True)
    return df
```

```
def get_otu_taxa_table(df):
    """
    This function returns otu table that contains relative abundance of species
    where columns are species and rows are cases.

    Args:
        df (dataframe): Dataframe of realtive abundance and metadata
    """
    df = df[bacteria_colnames + ['subjectID']]
    df.columns = ["OTU_{}".format(str(ind)) for ind, col in enumerate(bacteria_colnames)] +
    df.index = df['subjectID']
    taxa_table = get_taxa_table(bacteria_colnames)
    return df, taxa_table


def get_taxa_table(list_of_otus):
    """
    This function parse all present microbial species at different heirarchy levels, e.g., cl

    """
    otu = 0
    mapping = {}
    taxa_cols = ['kingdom','phylum','class','order','family','genus','species']
    df = pd.DataFrame(columns=taxa_cols)
    otu_mapping = {}
    otu_ids = []
    for ind, otu in enumerate(list_of_otus):
        tmp = {}
        for col in taxa_cols:
            tmp[col] = get_specific_label(otu, col)
        tmp_df = pd.DataFrame([tmp])
        df = pd.concat([df,tmp_df], ignore_index=[0])

        otu_id = "OTU_{}".format(str(ind))
        otu_mapping[otu] = otu_id
        otu_ids.append(otu)

    df['OTU'] = ['OTU_{}'.format(str(ind)) for ind in df.index]
    df = df.set_index('OTU')
    return df
```

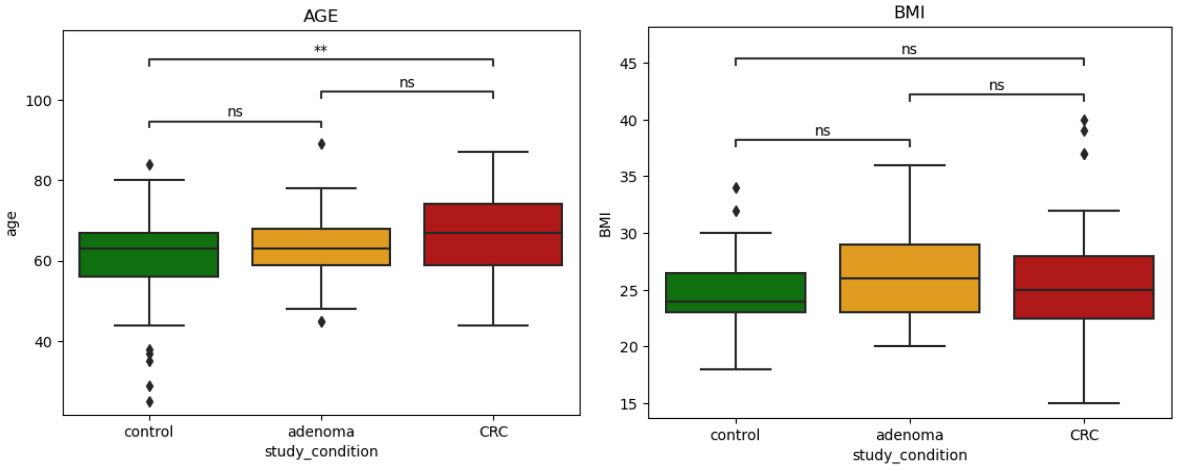
```
def get_specific_label(l, t):
    """
    This function parse the taxonomic assignment lable and fetch the specified information (e

    Args:
        l (str): string of taxonomy
        t (str): string specifying the requested information (e.g., kingdom, family, genus, e

    Returns:
        str: requested heirarichal info
    """
    taxa_order = {'kingdom':0,'phylum':1,'class':2,'order':3,'family':4,'genus':5,'species':6

    try:
        specific_label = l.split('|')[taxa_order[t]]

        return specific_label.strip().split('__')[1]
    except:
        return 'Unknown'


def get_otu_detail(taxa_table, otu_label, rank):
    return taxa_table[otu_label][rank]


def aggregate_by_taxonomy(otu_table, taxa_table, taxa_rank):
    """
    This function aggregates data based on specified
    taxa rank (e.g., kingdom, class, order, phylum, genus, species).
    """
    unique_values = (taxa_table[taxa_rank].unique())

    # mapping for otus to unique value of chosen taxa rank
    taxa_to_otu = {}

    # prepare the mapping
    for unique_value in unique_values:
        tdf = taxa_table.loc[taxa_table[taxa_rank] == unique_value, :]
        otus = tdf.index.to_list()
        taxa_to_otu[unique_value.strip()] = otus

    # create a dictionary for formulating expressions
```

```
    taxa_to_exp = {}

    for key in taxa_to_otu.keys():
        taxa_to_exp[key] = '{} = 0.000001'.format(key)
        for otu in taxa_to_otu[key]:
            taxa_to_exp[key] += ' + ' + otu
            otu_table[otu] = pd.to_numeric(otu_table[otu], errors='coerce')

    agg_df = otu_table

    for key, expr in taxa_to_exp.items():
        agg_df[key] = 0
        agg_df = agg_df.eval(expr, engine='python')

    agg_df = agg_df[list(unique_values)]

    return agg_df


def extend_with_alpha(df, metadata_features):
    """
    This function extends the dataframe with alpha diversity measures.

    Args:
        df: dataframe

        metadata_features: list of metadata feature names

    Returns:
        dataframe: extended dataframe with alpha diversity features
    """
    diversity_measures = pd.DataFrame()

    alpha_diversity_metrics = [
        "chao1",
        "shannon",
        "simpson",
        "simpson_e",
        "fisher_alpha",
        "berger_parker"
    ]
```

```
    shannon_diversity = df.apply(lambda x: alpha.shannon(x), axis=1)
    chao1_diversity   = df.apply(lambda x: alpha.chao1(x), axis=1)
    simpson_diversity   = df.apply(lambda x: alpha.simpson(x), axis=1)
    simpson_e_diversity   = df.apply(lambda x: alpha.simpson_e(x), axis=1)
    fisher_diversity   = df.apply(lambda x: alpha.fisher_alpha(x), axis=1)
    berger_parker_diversity   = df.apply(lambda x: alpha.berger_parker_d(x), axis=1)

    diversity_measures['shannon'] = shannon_diversity
    diversity_measures['chao1'] = chao1_diversity
    diversity_measures['simpson'] = simpson_diversity
    diversity_measures['simpson_e'] = simpson_e_diversity
    diversity_measures['fisher_alpha'] = fisher_diversity
    diversity_measures['berger_parker'] = berger_parker_diversity

    X_alpha = diversity_measures.reset_index().drop(['subjectID'], axis=1)
    X_extended = pd.concat([metadata_features,X_alpha],axis=1)

    return X_extended

# convert data tables into otu and taxa table
otu_table, taxa_table = get_otu_taxa_table(zeller_db)

# aggregating data at higher levels
phylum_agg = aggregate_by_taxonomy(otu_table, taxa_table, 'phylum')
genus_agg = aggregate_by_taxonomy(otu_table, taxa_table, 'genus')
order_agg = aggregate_by_taxonomy(otu_table, taxa_table, 'order')

color_palette = {'control':'green',
                 'adenoma':'orange',
                 'CRC':'#c80000'}

order = ['control', 'adenoma', 'CRC']
x = 'study_condition'

pairs = [
    ('control','adenoma'),
    ('control','CRC'),
    ('adenoma','CRC'),
]

metadata = zeller_db[metadata_colnames]
```

```
# changing data type of age and BMI
metadata['age'] = pd.to_numeric(metadata.age, errors='coerce')
metadata['BMI'] = pd.to_numeric(metadata.BMI, errors='coerce')
```

```
# plotting distribution

for ind, y in enumerate(['age','BMI']):
    plt.figure()
    ax = sns.boxplot(data=metadata, y=y,x=x, palette=color_palette, order= order)
    #annot = Annotator(ax, pairs=pairs, data=metadata, x=x, y=y, hue=x, hue_order=order, orde
    ax, test_results = add_stat_annotation(ax, box_pairs=pairs, data=metadata, x=x, y=y,
                                           hue_order=order, order=order,
                                           test='Mann-Whitney', text_format='star',comparison
                                           loc='inside', verbose=False)


    plt.title(f'{y.upper()}')

    plt.show()
```



(a) Age

(b) BMI

Figure 17.1: Age/BMI distribution

17.1.2 Gender and CRC

We now check the distribution of gender across different target groups (i.e., control, adenoma, CRC). We employ the Chi-squared test to investigate the statistical significance of differences

in gender distribution across different groups.

Figure 17.2 shows the frequency count of males/females across control, adenoma, and CRC groups. The differences were found to be statistically significant (p-value $< .05$).

> **Gender**
>
> We also found statistical differences in gender between control, adenoma and CRC group.

```
from scipy.stats import chi2_contingency

# Create a contingency table
contingency_table = pd.crosstab(metadata['gender'], metadata['study_condition'])

# Apply Fisher's Exact Test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# new figure
plt.figure()

# plot frequency plot
sns.countplot(data=metadata, x='study_condition',hue='gender')

# add p-value
plt.text(0.5, 32, f'$X^2 test$ p-value: {p_value:.4f}', fontsize=12, color='blue')
plt.show()
```

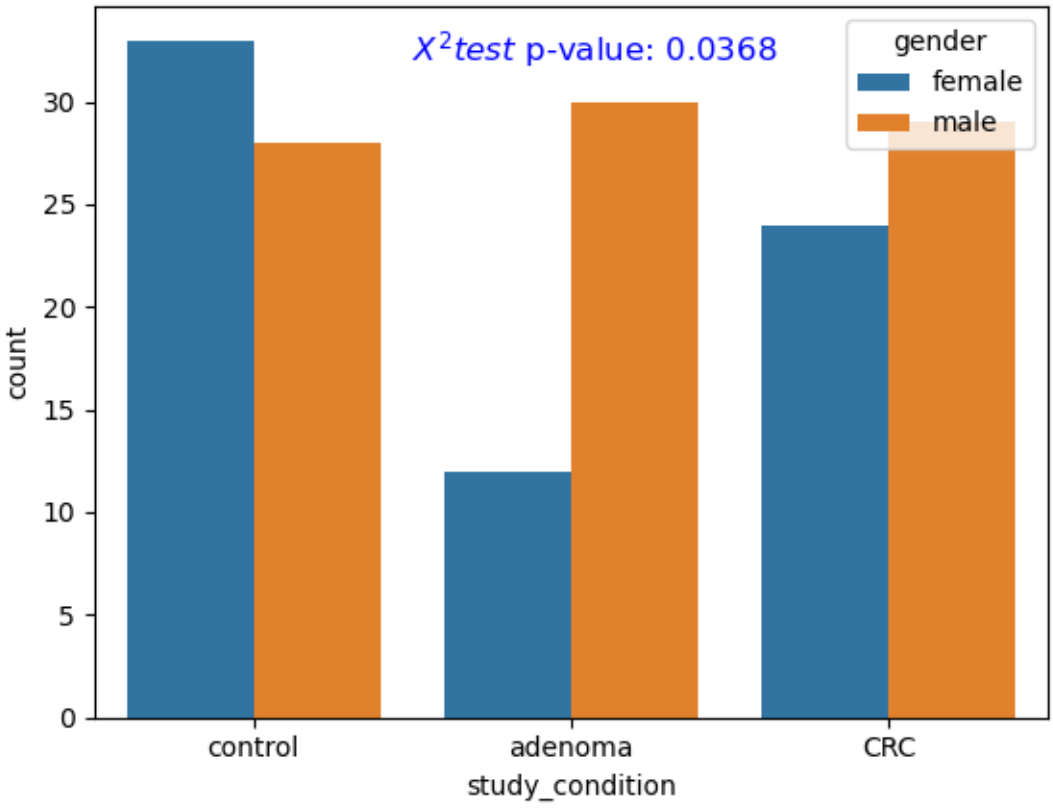


Figure 17.2: Gender distribution

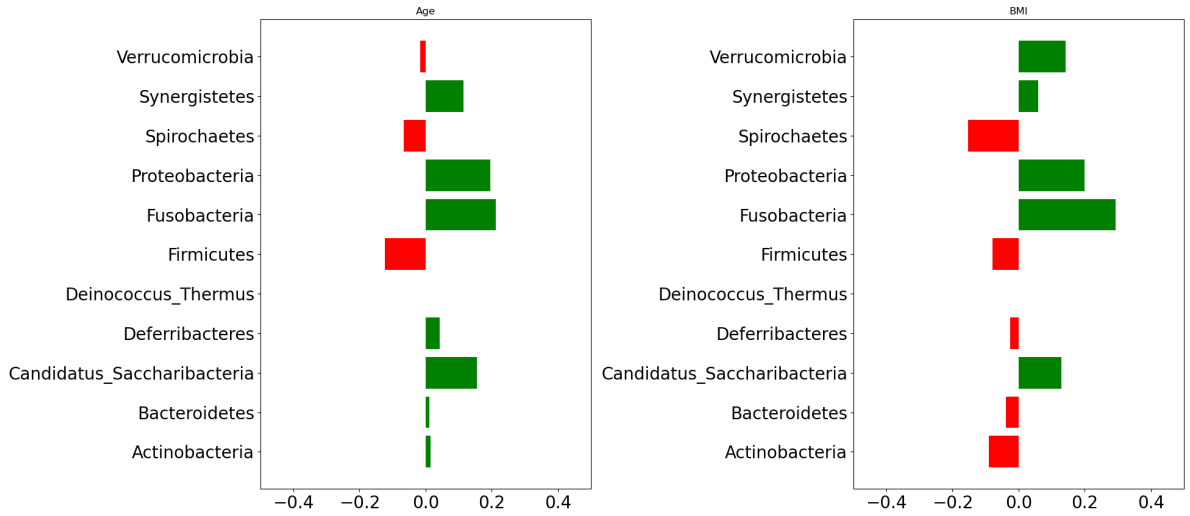
17.2 Association between host's characteristics and species abundance

We will now explore for association between host's characteristics (e.g., age, BMI) and species relative abundace. Figure 17.3 below shows correlation between host's characteristics and phylum level abundance data.

- **Firnicutes, Spirochaetes, and Verrucomicrobia** phylum levels are negatively correlated with age. That means as a person gets older these three phylum levels tend to get decreased.
- In case of BMI, **Spirochaetes, Firnicutes, Deferribacteres, Bacteroidetes, and Actinobacteria** are found negatively correlated. That implies an increase in BMI (which could be taken as an indication of obesity) is associated with decrease in those phylums.

```
# plotting distribution

def extend_abundance_metadata(df,meta):
    """
    This function extends abundance data with metadata information.

    Args:
        df (DataFrame): relative abundance data
        meta (DataFrame): host's characteristics

    Returns:
        DataFrame
    """
    return pd.concat([df,meta],axis=1)

# relative abundance aggregation at
#family_abundance = aggregate_by_taxonomy(otu_table, taxa_table, 'family')
genus_abundance = aggregate_by_taxonomy(otu_table, taxa_table, 'genus')
phylum_abundance = aggregate_by_taxonomy(otu_table, taxa_table, 'phylum')
metadata_ = metadata.set_index(metadata['subjectID'])

# plot age correlation plot
phylum_metadata = extend_abundance_metadata(phylum_abundance,metadata_[['age']])
phylum_corr = phylum_metadata.corr()
plt.figure(figsize=(7,10))
data_plot = phylum_corr['age'].drop('age')
bars = plt.barh(data_plot.index, data_plot, color=np.where(data_plot > 0, 'green', 'red'))
plt.xlim([-.5,.5])
plt.yticks(fontsize=20)
plt.xticks(fontsize=20)
plt.title('Age')
plt.show()

# plot bmi correlation plot
phylum_metadata = extend_abundance_metadata(phylum_abundance,metadata_[['BMI']])
phylum_corr = phylum_metadata.corr()
plt.figure(figsize=(7,10))
data_plot = phylum_corr['BMI'].drop('BMI')
bars = plt.barh(data_plot.index, data_plot, color=np.where(data_plot > 0, 'green', 'red'))
plt.xlim([-.5,.5])
plt.yticks(fontsize=20)
plt.xticks(fontsize=20)
```

```
plt.title('BMI')
plt.show()

phylum_metadata_ = phylum_metadata.copy()
phylum_metadata_['study_condition'] = metadata_['study_condition'].apply(
    lambda x: 'malign' if x == 'CRC' else 'benign')

df = phylum_metadata_.melt(id_vars='study_condition',value_vars=phylum_agg.columns)
```



(a) Correlation between age and phylum level abundance

(b) Correlation between BMI and phylum level abundance

Figure 17.3: Correlation with phylum abundace

Figure 17.4 (c) below shows **differences in microbial composition in terms of phylum level abundance among `benign` and `malign` tumors**. We can notice three phylums differ among benign and malign tumor groups. Those phylums are **Proteobacteria**, **Firnicutes**, and **Bacteroidetes**.

> We combined control with adenoma to create `benign` tumor class, and CRC class renamed as `malign` tumor.

```
plt.figure(figsize=(7,10))
sns.boxplot(data=df, y='variable',x='value', order=list(data_plot.index)[::-1], hue='study_c
plt.yticks(fontsize=20)
plt.xticks(fontsize=20)
plt.show()
```

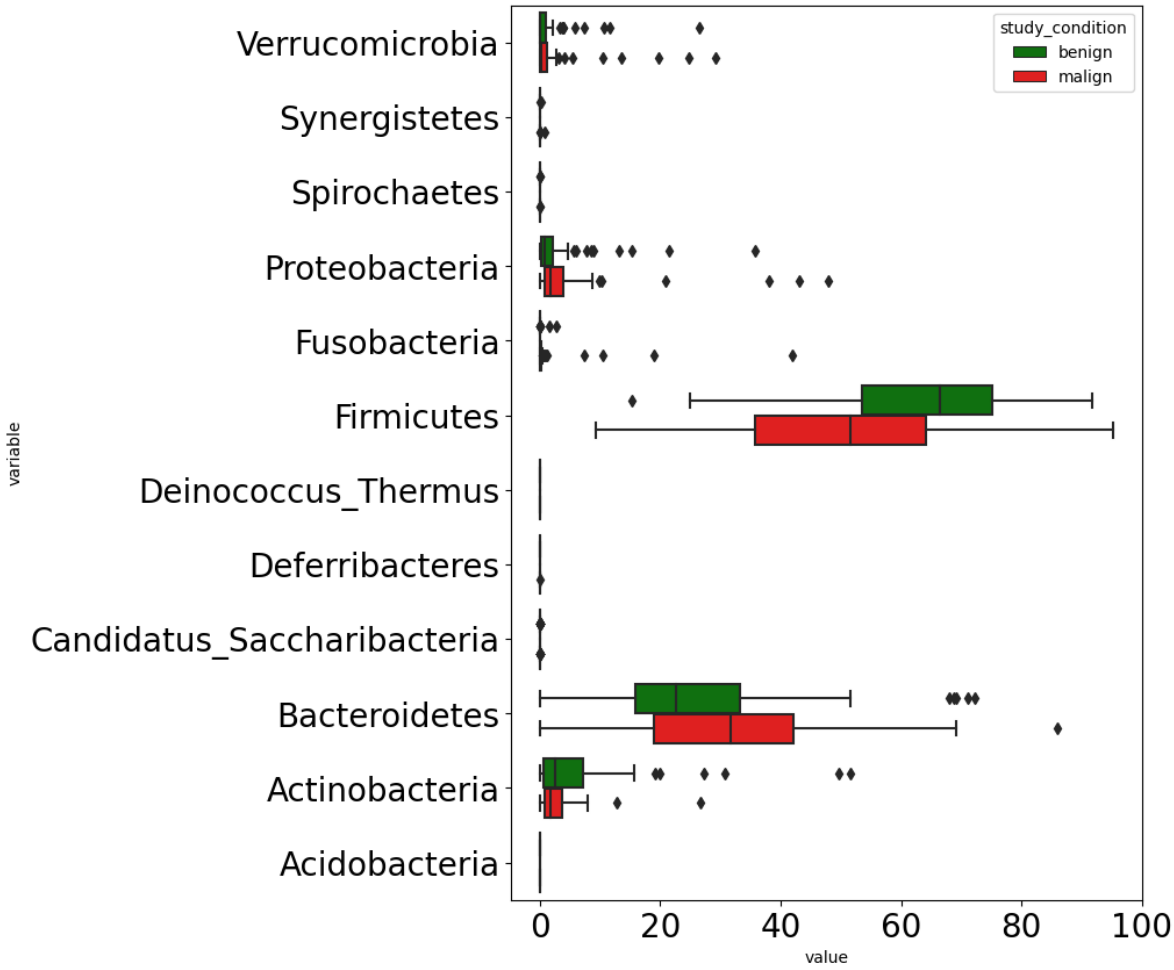


Figure 17.4: CRC and phylum abundace

17.3 Exploring Barteroidetes, Firmicutes, and Proteobacteria for theis association with CRc

We go further checking whether these differences are statistically significant or not. We employ `Mann-Whitney test` which is a non-parametric test for checking significance of differences in values from two independent groups.

Figure 17.5 below shows distributions of abundance at phylum levels across `benign` and `malign` cases for all three selected phylumns. The differences were found to be **statistically significant**.

```
color_palette = {'benign':'green','malign':'red'}

# selected phylums for statistical analysis
selected_phylums = ['Firmicutes','Proteobacteria','Bacteroidetes']

# extracting only selected phylum data
df_selected = df.loc[df['variable'].isin(selected_phylums),:]

# pairs for statistical test
pairs = [
    (('Firmicutes','benign'), ('Firmicutes','malign')),
    (('Proteobacteria','benign'), ('Proteobacteria','malign')),
    (('Bacteroidetes','benign'), ('Bacteroidetes','malign'))
]

# creating a new figure
plt.figure()

# plotting boxplot
ax = sns.boxplot(data=df_selected, x='variable', y='value',hue='study_condition', palette=col

# adding statistical annotation from Mann-Whitney test
ax, test_results = add_stat_annotation(ax, box_pairs=pairs, data=df_selected, x='variable', y
                                       hue='study_condition',hue_order=['benign','malign'],

                                       test='Mann-Whitney', text_format='star',comparisons_co
                                       loc='inside', verbose=False)

plt.xlabel('Phylums')

plt.show()
```

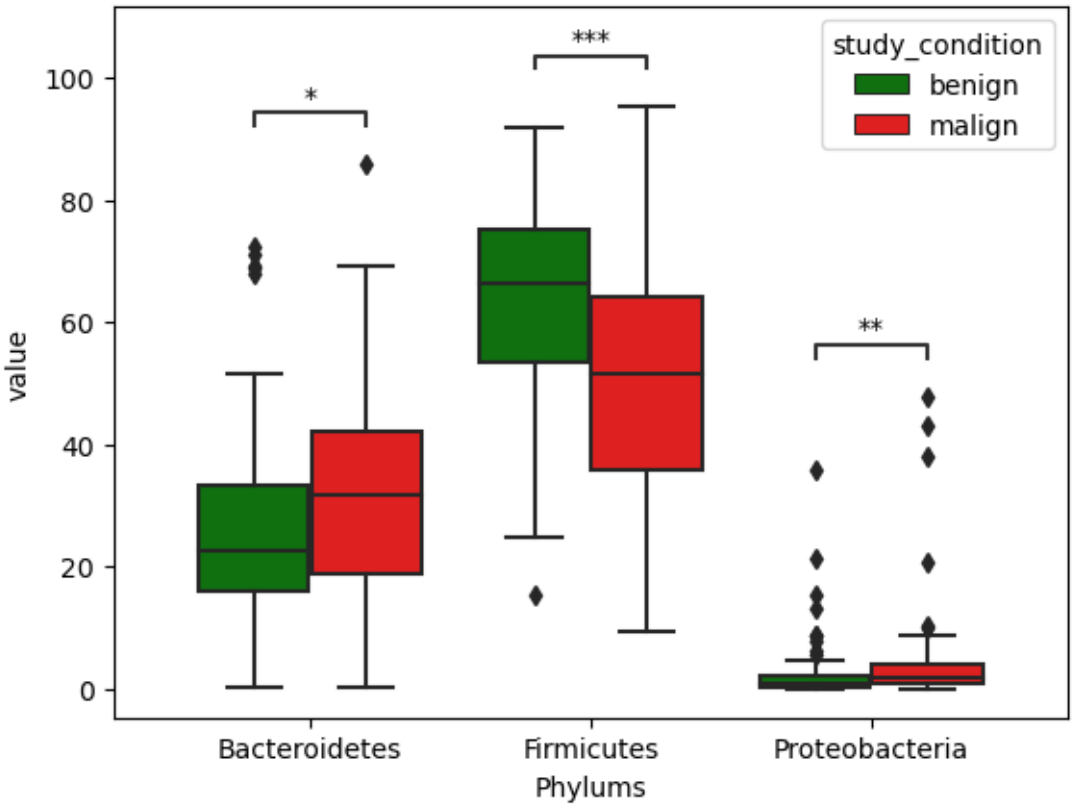


Figure 17.5: Phylum differences among benign and malign cases

Lets go further and check which genus and species under these phylums are statistically different in terms of relative abundance among `benign` and `malign` tumors.

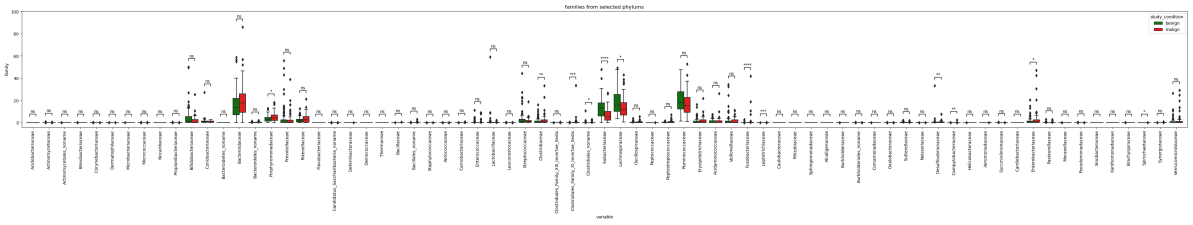
```
taxa_selected = taxa_table.loc[taxa_table['phylum'].isin(selected_phylums),:]

# fetching species related to selected phylums
selected_species_cols = list(taxa_table['species'].unique())

# fetching genus related to selected phylums
selected_genus_cols = list(taxa_table['genus'].unique())

# fetching family related to selected phylums
selected_family_cols = list(taxa_table['family'].unique())

# fetching order related to selected phylums
selected_order_cols = list(taxa_table['order'].unique())
```

```
# fetching order related to selected phylums
selected_class_cols = list(taxa_table['class'].unique())
```

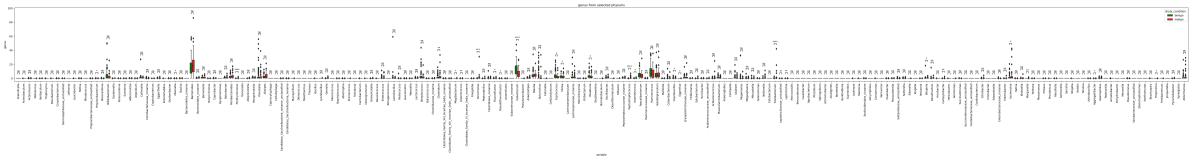
```
from statannotations.Annotator import Annotator

def plot_selected_taxa(selected_taxa, plot_at,figsize=(7,15),log_scale=False,title=""):
    """
    Args:
    ----
        selected_taxa(str): taxa which are selected for further exploration
        taxa_abun_df (dataframe): relative abundance data at taxa
        plot_at (str): taxa at which distribution will be plotted for benign and malign tumo


    """
    df_abundance = aggregate_by_taxonomy(otu_table, taxa_table, plot_at)
    metadata_ = metadata.set_index(metadata['subjectID'])

    taxa_abundance_selected = df_abundance[selected_taxa]

    taxa_abundance_selected['study_condition'] = metadata_['study_condition'].apply(
    lambda x: 'malign' if x == 'CRC' else 'benign')


    pairs = []

    for col in taxa_abundance_selected.columns:
        if col != 'study_condition':
            pairs.append(((col,'benign'),(col,'malign')))

    plt.figure(figsize=figsize)
    plot_df = taxa_abundance_selected.melt(id_vars='study_condition',value_vars=selected_taxa

    ax = sns.boxplot(data=plot_df,x='variable',y='value',hue='study_condition', palette=colo
    add_stat_annotation(ax, box_pairs=pairs, data=plot_df, x='variable', y='value',
                                        hue='study_condition',hue_order=['benign','malign'],

                                        test='Mann-Whitney', text_format='star',comparisons_c
                                        loc='inside', verbose=False)
    ax.set_ylabel(plot_at)
    plt.xticks(rotation='vertical')
```

```
    if log_scale:
        ax.set_yscale('log')
    plt.title(title)
    plt.show()
```

```
plot_selected_taxa(selected_family, plot_at='family', figsize=(50,5),title='families from sel
```



```
plot_selected_taxa(selected_genus, plot_at='genus', figsize=(80,5),title='genus from selected
```



18 Building a CRC classifier

Now, we will build our CRC classifier using gut microbial data from four different cohorts: `French`, `Italian`, `Austrian`, and `Germany`. Our dataset consists of relative abundance of gut microbial communities. The dataset also cotains classification of those communities at different taxonomic levels (e.g., phylum, genus, species).

We will follow the approach depicted in Figure 18.1 for building our CRC classifier. We will divide all cohorts using **70/30 split** into training and test set. All training sets will be utilized for building models and selecting the one with best performance. The selected model then applied on test set from different cohorts to assess generalizability of the model across cohorts.

18.1 Target class distribution across cohorts

Figure 18.2 below shows the ditribution of target class. For our modeling task, we have grouped `adenoma` and `control` into a single class of `benign` tumor. While, `CRC` class is treated as `malign` tumor.

- **French and Austrian cohorts are highly skewed** in terms of target class distribution, both having ~ 30% cases of CRC.
- While, **German and Italian cohorts have relatively balanced** cases of malign and benign tumors.

```
for country in ['france','austria','germany','italy']:
    plt.figure()

    # Get country specific dataset
    dataset = get_country_dataset('Nine_CRC_cohorts_taxon_profiles.tsv',country,country_datas

    # Plot class ditribution
    sns.countplot(data=dataset, x='target_class',alpha=.6,order=['benign','malign'],palette=
    plt.ylim([0,130])
    plt.show()
```

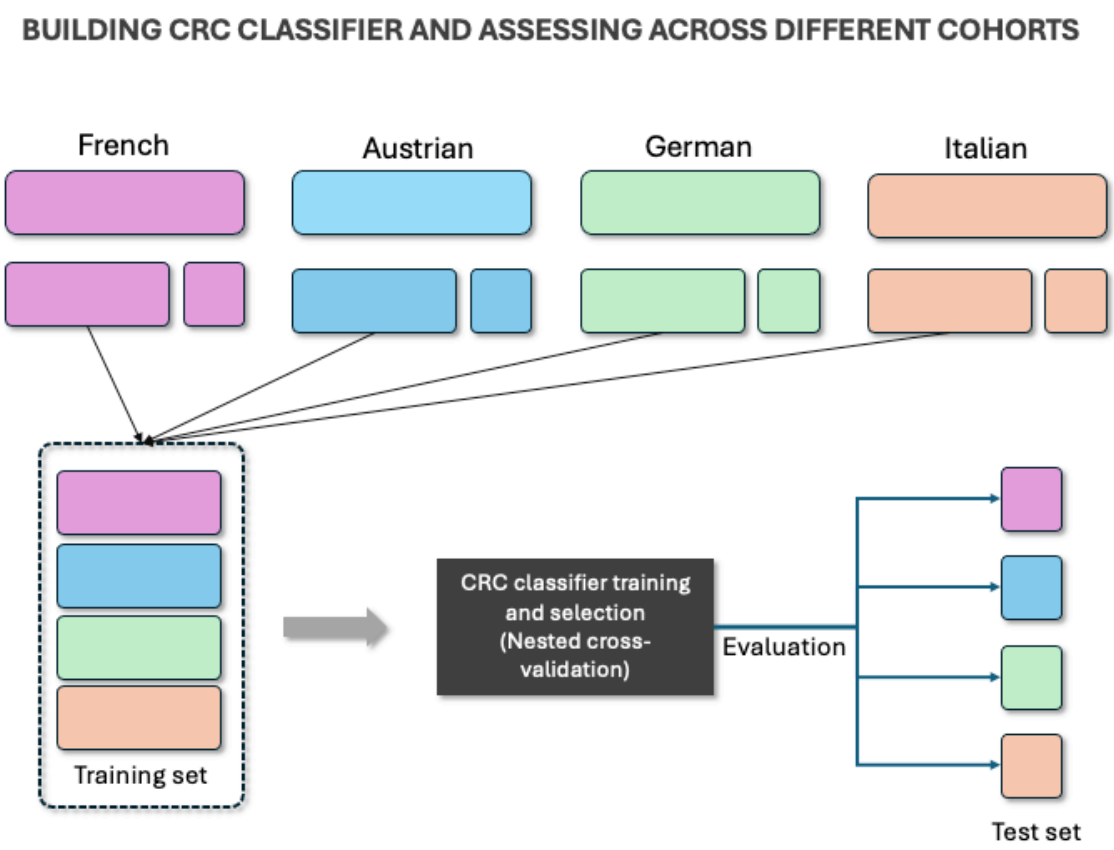
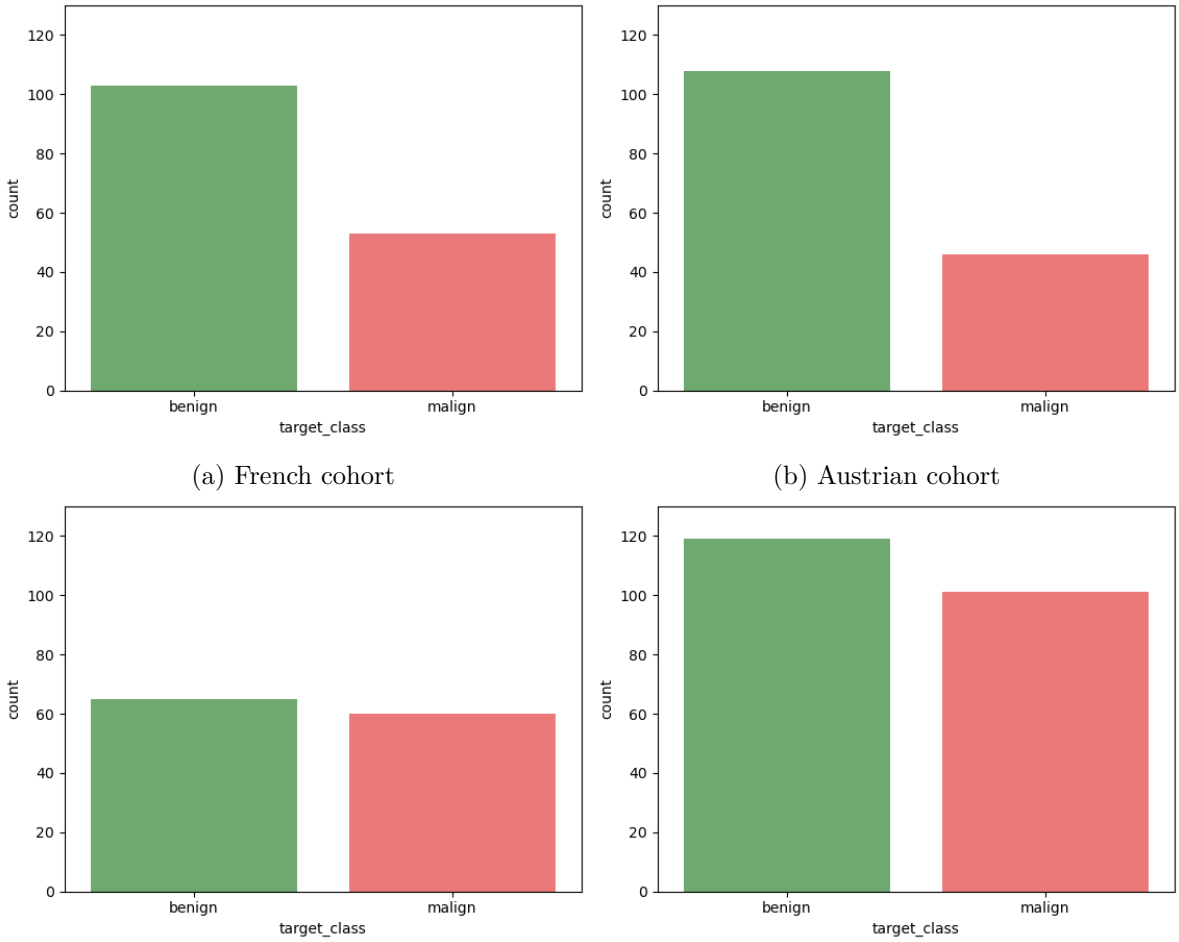


Figure 18.1: Methodology to build and assess CRC classifier



(a) French cohort

(b) Austrian cohort

(c) German cohort

(d) Italian cohort

Figure 18.2: Target class distribution in cohorts

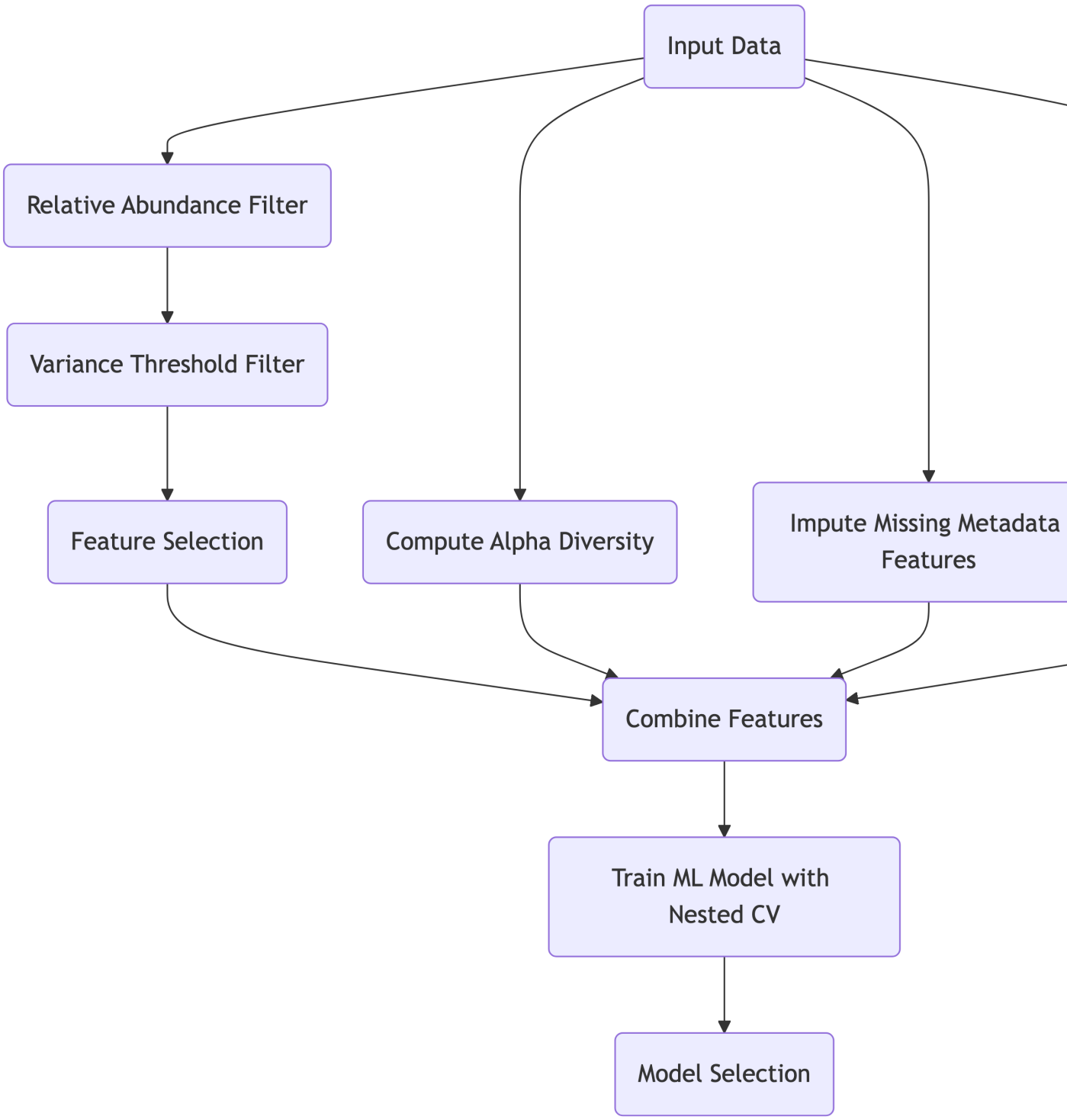
18.2 Train and test split of dataset

We will split each of aforementioned cohort using **70/30 split rule** resulting in a training set of 70% cases and a test set of 30% cases. We will use the test set only for our final evaluation of our models' performance.

```
from sklearn.model_selection import train_test_split

# List to store training and test seperately
train_X_data = []
test_X_data = []

for country in ['france','austria','germany','italy']:
    # Get country specific dataset
    dataset = get_country_dataset('Nine_CRC_cohorts_taxon_profiles.tsv',country,country_datas

    y = dataset['target_class']
    X = dataset.copy()

    # Split train and test
    train_x, test_x, train_y, test_y = train_test_split(X,y,test_size=0.20, random_state=42)

    # Store country
    train_x['groups'] = country
    test_x['groups'] = country

    # Storing train and test seperately
    train_X_data.append(train_x)
    test_X_data.append(test_x)

# Concatenating all country's dataset
train_X_df = pd.concat(train_X_data, axis=0)
test_X_df = pd.concat(test_X_data, axis=0)

# Preparing X and y
train_X = train_X_df.drop(['target_class'], axis=1)
train_y = train_X_df['target_class']

test_X_df.to_csv('test_X_df.csv',index=False)
```

18.3 Model building pipeline

The below diagram depicts our model building pipeline consisting of steps ranging from feature selection, dimensionality reduction to model selection. These steps are explained at length below.


Input Data
Relative Abundance Filter
Variance Threshold Filter
Feature Selection
Compute Alpha Diversity
Impute Missing Metadata Features
Combine Features
Train ML Model with Nested CV
Model Selection

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_selection import RFECV, SelectKBest, SelectFpr, f_classif, VarianceThres
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LassoCV
from sklearn.utils import resample

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, ro

from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion


# Creating custom pipeline processor
class ColumnSelector(BaseEstimator, TransformerMixin):
    """
    Custom selector based of feature names for pipeline
    """
    def __init__(self, variables):
        self.variables = variables

    def fit(self, X, y = None):
        return self

    def transform(self, X):
        X_dropped = X[self.variables]
        return X_dropped


class RelativeAbundanceFilter(BaseEstimator, TransformerMixin):
    """
    Custom selector based on relative abundance filtering
    """
    def __init__(self, threshold):
        self.threshold = threshold
        self.columns_select = None

    def fit(self, X, y=None):
```

```
        self.columns_to_select = X.columns[X.max(axis=0) > self.threshold]
        return self

    def transform(self, X):
        X_selected = X[self.columns_to_select]
        return X_selected


class VarianceThresholdFilter(BaseEstimator, TransformerMixin):
    """
    Custom selector based on relative abundance filtering
    """
    def __init__(self, threshold=0.0):
        self.threshold = threshold
        self.transformer = VarianceThreshold(self.threshold)

    def fit(self, X, y=None):
        self.transformer.fit(X,y)
        return self

    def transform(self, X):
        X_filtered = self.transformer.transform(X)
        return X_filtered


class FeatureSelector(BaseEstimator, TransformerMixin):
    """
    Select top k features based on specified strategy
    """
    def __init__(self, cv=10):
        self.estimator = LassoCV(cv=cv)
        self.feature_coef = pd.DataFrame()

    def fit(self, X, y=None):
        self.feature_coef.columns = X.columns

        for sample in resample(X,y):
            self.estimator.fit(X,y, random_state=42)
            coef_df = pd.DataFrame(self.estimator.coef_, columns=X.columns)
            self.feature_coef = pd.concat([self.feature_coef, coef_df], axis=0, ignore_index=

        return self
```

```
    def transform(self, X):
        X_selected = self.feature_selector.transform(X)
        df = pd.DataFrame(X_selected, columns=self.feature_names_out)

        return df


class AlphaFeatureExtender(BaseEstimator, TransformerMixin):
    """
    Custom transformer to extend relative abundace data with alpha diversity measure
    """
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Create an empty dataframe
        diversity_measures = pd.DataFrame()

        # alpha diversty measures
        alpha_diversity_metrics = [
            "chao1",
            "shannon",
            "simpson",
            "simpson_e",
            "fisher_alpha",
            "berger_parker"
        ]

        # Compute alpha diversity measures
        shannon_diversity = X.apply(lambda x: alpha.shannon(x), axis=1)
        chao1_diversity   = X.apply(lambda x: alpha.chao1(x), axis=1)
        simpson_diversity   = X.apply(lambda x: alpha.simpson(x), axis=1)
        simpson_e_diversity   = X.apply(lambda x: alpha.simpson_e(x), axis=1)
        fisher_diversity   = X.apply(lambda x: alpha.fisher_alpha(x), axis=1)
        berger_parker_diversity   = X.apply(lambda x: alpha.berger_parker_d(x), axis=1)

        # Add alpha measures to the dataframe
        diversity_measures['shannon'] = shannon_diversity
        diversity_measures['chao1'] = chao1_diversity
```

```
        diversity_measures['simpson'] = simpson_diversity
        diversity_measures['simpson_e'] = simpson_e_diversity
        diversity_measures['fisher_alpha'] = fisher_diversity
        diversity_measures['berger_parker'] = berger_parker_diversity

        return diversity_measures


class MetaDataImputer(BaseEstimator, TransformerMixin):
    """
    Imputer for specific metadata (i.e., age, genger, BMI)
    """
    def __init__(self):
        self.fill_values = dict()


    def fit(self, X, y=None):
        X['age'] = pd.to_numeric(X['age'], errors='coerce')
        X['BMI'] = pd.to_numeric(X['BMI'], errors='coerce')

        self.fill_values['age'] = X['age'].mean()
        self.fill_values['BMI'] = X['BMI'].mean()
        self.fill_values['gender'] = X['gender'].mode().values[0]
        return self

    def transform(self, X):
        if X['age'].dtype == 'O':
            X['age'] = pd.to_numeric(X['age'], errors='coerce')

        if X['BMI'].dtype == 'O':
            X['BMI'] = pd.to_numeric(X['BMI'], errors='coerce')

        for col in X.columns:
            X[col] = X[col].fillna(self.fill_values[col])
        X['gender'] = X['gender'].map({'female':0, 'male':1})
        return X


def prepare_dataset(X, return_mapping=False):
    """
    This function prepare dataframe for modeling task
    """
```

```
    otu, taxa = get_otu_table(X)
    metadata = get_metadata(X,['age','gender','BMI'])

    for col in otu.columns:
        otu[col] = otu[col].astype('float32')

    metadata['age'] = pd.to_numeric(metadata['age'], errors='coerce')
    metadata['BMI'] = pd.to_numeric(metadata['BMI'], errors='coerce')

    y= X['target_class'].map({'benign':0,'malign':1})
    y.reset_index(drop=True, inplace=True)
    otu.reset_index(drop=True, inplace=True)

    otu_to_species = taxa['species'].to_dict()

    X_ = pd.concat([otu, metadata], axis=1)
    if return_mapping:
        return X_, y, otu_to_species
    else:
        return X_, y
```

18.3.1 Applying filters to reduce feature space

We will start by reducing feature spaces through the following three steps.

1. Relative abundance filter
2. Variance threshold filter
3. Best features selection using Lasso

```
from sklearn.preprocessing import FunctionTransformer
from sklearn.decomposition import PCA

def log(x):
    return np.log(x+.000001)

# Prepare dataset for modeling
X, y, m = prepare_dataset(train_X_df, return_mapping=True)

# Create list of otus columns
otu_columns = [item for item in X.columns if item not in ['age','gender','BMI']]
```

```
# Pipeline for filtering
abundance_processing =  Pipeline([
                    ('abundance',ColumnSelector(otu_columns)),
                    ('relative',RelativeAbundanceFilter(threshold=.001)),
                    ('variance', VarianceThresholdFilter()),
                    ('log', FunctionTransformer(log)),
                    ('standard', MinMaxScaler())
                    ])

# Pipeline for computing alpha diversity
alpha_processing = Pipeline([
    ('abundance', ColumnSelector(otu_columns)),
    ('alpha_extended', AlphaFeatureExtender())
])

# Pipeline for imputing metadata
meta_processing = Pipeline([
    ('meta_columns', ColumnSelector(['age','BMI','gender'])),
    ('meta_imputer', MetaDataImputer())
])

# Pipeline for PCA
pca_processing = Pipeline([
    ('abundance',ColumnSelector(otu_columns)),
    ('pca',PCA()),
])

# Concatenating results
combined_features = FeatureUnion([
    ('abun_part', abundance_processing),
    ('alpha_part', alpha_processing),
    ('meta_part', meta_processing),
    ('pca_part',pca_processing),
])

# Create full pipeline
full_pipeline = Pipeline([
    ('combined_part', combined_features),
    ('select_part', ColumnSelector(final_features)),
    ('train', LogisticRegression(C=.1, penalty='l1',solver='liblinear')),
     ]
)
```

```
full_pipeline.fit(X,y)
```

```
Pipeline(steps=[('combined_part',
                 FeatureUnion(transformer_list=[('abun_part',
                                                 Pipeline(steps=[('abundance',
                                                                  ColumnSelector(variables=[
                                                                                 'OTU_16', 'OTU_28', 'OTU_108',
                                                                                 'OTU_111', 'OTU_144', 'OTU_153',
                                                                                 'OTU_157', 'OTU_158', 'OTU_161',
                                                                                 'OTU_178', 'OTU_218', 'OTU_348',
                                                                                 'OTU_353', 'OTU_367', 'OTU_372',
                                                                                 'OTU_375', 'OTU_378', 'OTU_381',
                                                                                 'OTU_388', 'OTU_446', 'OTU_464',
                                                                                 'OTU_465', 'OTU_466', 'OTU_468',
                                                                                 'OTU_470', 'OTU_503', 'OTU_535',
                                                                                 'OTU_544', 'OTU_546', 'OTU_547', ...])),
```

```
                ('train',
                 LogisticRegression(C=0.1, penalty='l1', solver='liblinear'))])
```

```
from imblearn.over_sampling import RandomOverSampler
from numpy.random import RandomState
from sklearn.feature_selection import SelectKBest

# Feature selection process
df_coef = pd.DataFrame()

feature_set = []

for i in range(100):
    sampler = RandomOverSampler(sampling_strategy=np.random.choice([1]), random_state=RandomS
    sample_X, sample_y = sampler.fit_resample(X_features, y)

    sel = SelectKBest(k=100)
    X_selected = sel.fit_transform(sample_X, sample_y)

    feature_set += list(X_selected.columns)

#import collections
#counter = collections.Counter(feature_set)
#final_features = [item[0] for item in counter.most_common(40)]

X_final = X_features[final_features]

lasso_param_grid = {
    "lasso__C": [0.001, .01, .1, 1, 10, 100],
}

dt_param_grid = {
    "dt__max_depth": [3,4,5,6],
    "dt__criterion":['gini','entropy'],
    "dt__min_samples_split":[2,4,6,8,10]
}

rf_param_grid = {
    "rf__n_estimators":[50,100,150,200],
    "rf__max_depth":[3,4,5,6],

}
```

```
lasso_pipe = Pipeline([
    ('scaler',StandardScaler()),
    ('lasso', LogisticRegression(penalty='l1',solver='liblinear', max_iter=10000))
])

dt_pipe = Pipeline([
    ('dt', DecisionTreeClassifier())
])

rf_pipe = Pipeline([
    ('scaler',StandardScaler()),
    ('rf', RandomForestClassifier())
])
```

```
def build_classifier(pipeline, param_grid, X,y, outer_cv=None, inner_cv=None):
    """
    This builds CRC classifier using nested cross-validation and print hyperparameters of the

    Args:
    -----
        model (sklearn model): machine learning model object

        param_grid (dict): parameters grid to search optimized parameters

        X (dataframe): pandas dataframe of dataset

        y (list): target class

        inner_cv_scoring (int): number of folds for inner cross-validation

        outer_cv_scoring (int): number of folds for outer cross-validation


    Returns:
    -----
        classifier (sklearn trained model)
    """

    # Outer validation: 10-fold cross-validation
    if outer_cv is None:
        outer_cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
    # Inner validation: 5-fold cross-validation for hyperparameter tuning
    if inner_cv is None:
        inner_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)


    aucs = []
    best_performance = 0
    best_estimator = None

    X = X.reset_index(drop=True)
    y = y.reset_index(drop=True)

    # Nested cross-validation
    for fold, (train_idx, test_idx) in enumerate(outer_cv.split(X, y), start=1):
        X_train, X_test = X.iloc[train_idx,:], X.iloc[test_idx,:]
        y_train, y_test = y[train_idx], y[test_idx]

        # Perform GridSearchCV with 10-fold cross-validation
        grid_search = GridSearchCV(
            pipeline, param_grid, scoring='roc_auc_ovr', cv=inner_cv, n_jobs=-1
        )

        # searching parameters first
        grid_search.fit(X_train, y_train)

        # Best model and hyperparameter
        best_model = grid_search.best_estimator_
        #print(f"Best Hyperparameter: {grid_search.best_params_}")

        # Fit the model and predict probabilities
        best_model.fit(X_train, y_train)
        y_prob = best_model.predict_proba(X_test)


        fpr, tpr, _ = roc_curve(y_test, y_prob[:,1])
        roc_auc = auc(fpr, tpr)

        if roc_auc > best_performance:
            best_estimator = best_model
            best_performance = roc_auc

        aucs.append(roc_auc)
```

```
        #print(f"  performance={roc_auc:.2f}")

    return best_estimator, best_performance, np.std(aucs)
```

```
clf, per, st = build_classifier(lasso_pipe, lasso_param_grid, X_final, y)

print(f' Best estimator (performance={per:.2f}):')
print(clf)
```

```
 Best estimator (performance=0.96):
Pipeline(steps=[('scaler', StandardScaler()),
                ('lasso',
                 LogisticRegression(C=0.1, max_iter=10000, penalty='l1',
                                    solver='liblinear'))])
```

```
clf, per, st = build_classifier(dt_pipe, dt_param_grid, X_final, y)

print(f' Best dt estimator (performance={per:.2f}±{st:.2f}):')
print(clf)
```

```
 Best dt estimator (performance=0.89±0.09):
Pipeline(steps=[('dt',
                 DecisionTreeClassifier(criterion='entropy', max_depth=4,
                                        min_samples_split=10))])
```

```
clf, per, st = build_classifier(rf_pipe, rf_param_grid, X_final, y)

print(f' Best rf estimator (performance={per:.2f}±{st:.2f}):')
print(clf)
```

```
 Best rf estimator (performance=0.96±0.07):
Pipeline(steps=[('scaler', StandardScaler()),
                ('rf', RandomForestClassifier(max_depth=6, n_estimators=200))])
```

18.4 Best models

We selected three different machine learning models based on their performance on entire dataset. The models were evluated in a nested cross-validation. The outer cross-validation

(10-fold) was used for model performance while the inner cross-validation (5-fold) was used for hyperparameter tuning.

1. LogisticRegression(C=.1) .96 AUC
2. DecisionTreeClassifier(criterion=entropy, max_depth=4, min_samples_split=10) AUC= 0.83±0.09
3. RandomForestClassifier(max_depth=6, n_estimators=200) AUC = 0.93±0.07

```
lg = LogisticRegression(C=.1, penalty='l1',solver='liblinear')
dt = DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_split=10)
rf = RandomForestClassifier(max_depth=6, n_estimators=200)
```

18.5 Model performance of selected model using LeaveOneGroupOut

We will now assess these models using leavel-one-group-out strategy. In this strategy, we will keep a cohort for testing while other cohorts datasets will be used for training. This will give us a better idea about our models' performance.

```
from sklearn.model_selection import LeaveOneGroupOut

def perform_logo(estimator, X, y, groups,title=''):
    """
    This function performs Leave-one-group-out evaluation of the model.
    """

    tprs = []
    aucs = []
    mean_fpr = np.linspace(0, 1, 100)
    aucs = []
    best_performance = 0
    best_estimator = None

    logo = LeaveOneGroupOut()

    # Nested cross-validation
    for fold, (train_idx, test_idx) in enumerate(logo.split(X, y,groups)):
        X_train, X_test = X.iloc[train_idx,:], X.iloc[test_idx,:]
        y_train, y_test = y[train_idx], y[test_idx]

        estimator.fit(X_train, y_train)
```

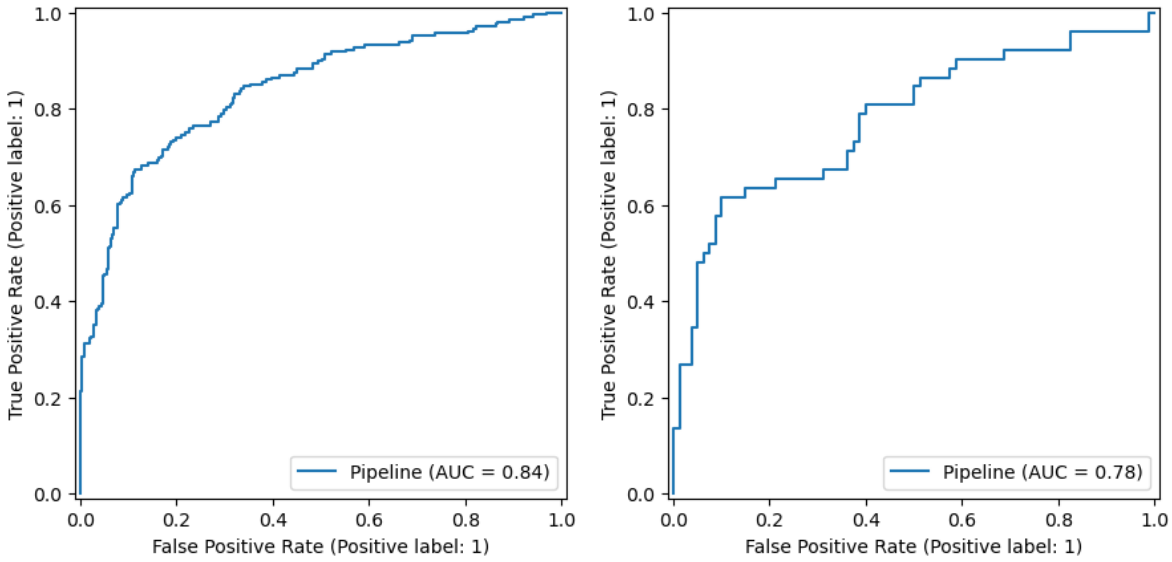
```
        y_prob = estimator.predict_proba(X_test)

        fpr, tpr, _ = roc_curve(y_test, y_prob[:,1])
        interp_tpr = np.interp(mean_fpr, fpr, tpr)
        interp_tpr[0] = 0.0
        tprs.append(interp_tpr)
        roc_auc = auc(fpr, tpr)

        aucs.append(roc_auc)

    # Average AUC across classes
    mean_auc = np.mean(aucs)

    # Plot the mean ROC curve
    # Calculate mean and standard deviation of TPRs
    mean_tpr = np.mean(tprs, axis=0)
    std_tpr = np.std(tprs, axis=0)

    # Plot the mean ROC curve
    plt.plot(mean_fpr, mean_tpr, color='blue', linestyle='--', lw=2, label=f'Mean ROC (AUC =

    # Fill the area between the mean TPR and ±1 standard deviation
    plt.fill_between(mean_fpr, mean_tpr - std_tpr, mean_tpr + std_tpr, color='blue', alpha=0

    # Plot the random chance line
    plt.plot([0, 1], [0, 1], linestyle='--', color='gray', lw=2, label='Chance')

    # Finalize the plot
    plt.title(title)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    #plt.grid(alpha=0.3)
    plt.show()
```

```
train_X, train_y = prepare_dataset(train_X_df)
test_X, test_y = prepare_dataset(test_X_df)

full_pipeline.fit(train_X, train_y)
```

```
Pipeline(steps=[('combined_part',
```

```
 FeatureUnion(transformer_list=[('abun_part',
                                 Pipeline(steps=[('abundance',
                                                  ColumnSelector(variables=[

 ColumnSelector(variables=['OTU_16', 'OTU_28', 'OTU_108',
                           'OTU_111', 'OTU_144', 'OTU_153',
                           'OTU_157', 'OTU_158', 'OTU_161',
                           'OTU_178', 'OTU_218', 'OTU_348',
                           'OTU_353', 'OTU_367', 'OTU_372',
                           'OTU_375', 'OTU_378', 'OTU_381',
                           'OTU_388', 'OTU_446', 'OTU_464',
                           'OTU_465', 'OTU_466', 'OTU_468',
                           'OTU_470', 'OTU_503', 'OTU_535',
                           'OTU_544', 'OTU_546', 'OTU_547', ...])),
('train',
 LogisticRegression(C=0.1, penalty='l1', solver='liblinear'))])
```

```
from sklearn import metrics

test_y_proba = full_pipeline.predict_proba(test_X)[:,1]

metrics.RocCurveDisplay.from_estimator(full_pipeline, train_X, train_y)
metrics.RocCurveDisplay.from_estimator(full_pipeline, test_X, test_y)
```



(a) Train (b) Test

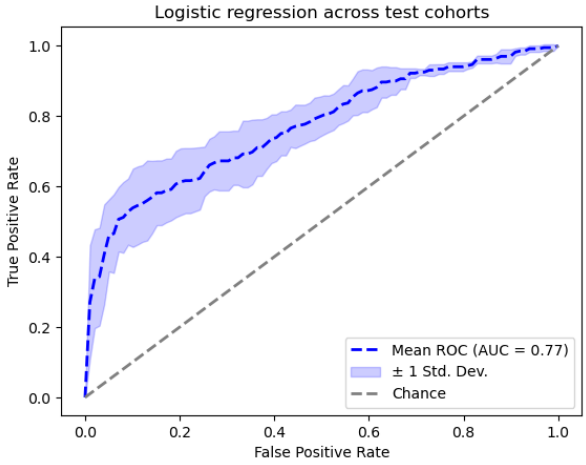
Figure 18.3: Lasso model performance

```
groups = train_X_df['groups'].map({'france':0,'austria':1,'germany':2,'italy':3})
groups = groups.reset_index(drop=True)
```

```
#train_X, train_y, transformer = prepare_dataset(train_X_df)

perform_logo(full_pipeline, X, y, groups,'Logistic regression across test cohorts')
#perform_logo(dt, train_X,train_y, groups,'Decision tree across cohorts (training)')
#perform_logo(rf, train_X,train_y, groups,'Random forest across cohorts (training)')
```

18.5.1 Model performance on test data from different cohorts

Now, we will evaluate our trained model which were trained using data from different cohorts. This evaluation is done on a seperate dataset which was kept aside earlier.



Model training performance across cohorts (LOGO)

Figure 18.4: Logistic regression

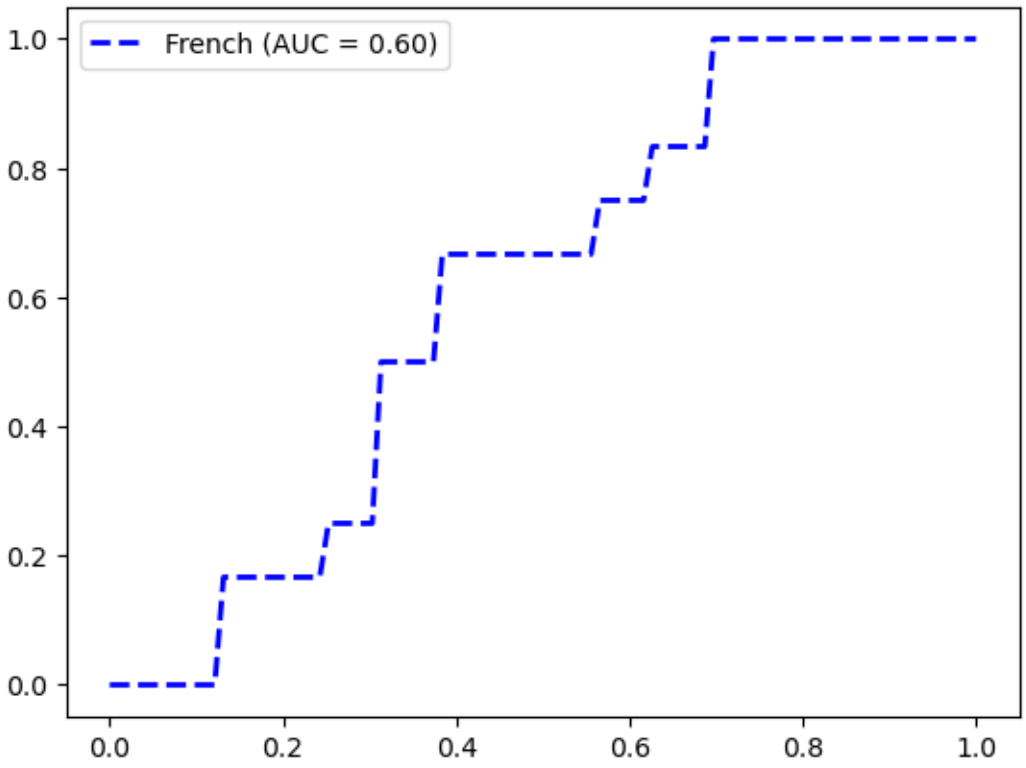
```
lg.fit(train_X, train_y)
dt.fit(train_X, train_y)
rf.fit(train_X, train_y)
```

```
RandomForestClassifier(max_depth=6, n_estimators=200)
```

```
train_pre_X, train_y = prepare_dataset(train_X_df)

combined_features.fit(train_pre_X, train_y)

train_X = combined_features.transform(train_pre_X)
```

```
from sklearn.metrics import RocCurveDisplay

estimator = lg

mean_fpr = np.linspace(0, 1, 100)
cn_test_set = test_X_df.loc[test_X_df['country']=='ITA',:]
cn_test_pre_X, cn_test_y = prepare_dataset(cn_test_set)
cn_test_X = combined_features.transform(cn_test_pre_X)
y_prob = estimator.predict_proba(cn_test_X)
fpr, tpr, _ = roc_curve(cn_test_y, y_prob[:,1])
roc_auc = auc(fpr, tpr)
interp_tpr = np.interp(mean_fpr, fpr, tpr)
```

```
interp_tpr[0] = 0.0

# Plot the mean ROC curve
plt.plot(mean_fpr, interp_tpr, color='blue', linestyle='--', lw=2, label=f'French (AUC = {roc
plt.legend()
```



19 Generalizability evaluation

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

Callahan, Benjamin J, Paul J McMurdie, Michael J Rosen, Andrew W Han, Amy Jo A Johnson, and Susan P Holmes. 2016. "DADA2: High-Resolution Sample Inference from Illumina Amplicon Data." *Nature Methods* 13 (7): 581–83.

Ewels, Philip, Måns Magnusson, Sverker Lundin, and Max Käller. 2016. "MultiQC: Summarize Analysis Results for Multiple Tools and Samples in a Single Report." *Bioinformatics* 32 (19): 3047–48.

Franzosa, Eric A, Lauren J McIver, Gholamali Rahnavard, Luke R Thompson, Melanie Schirmer, George Weingart, Karen Schwarzberg Lipson, et al. 2018. "Species-Level Functional Profiling of Metagenomes and Metatranscriptomes." *Nature Methods* 15 (11): 962–68.

Krueger, Felix. 2015. "Trim Galore!: A Wrapper Around Cutadapt and FastQC to Consistently Apply Adapter and Quality Trimming to FastQ Files, with Extra Functionality for RRBS Data." *Babraham Institute*.

Langmead, Ben, and Steven L Salzberg. 2012. "Fast Gapped-Read Alignment with Bowtie 2." *Nature Methods* 9 (4): 357–59.

Li, Dinghua, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. 2015. "MEGAHIT: An Ultra-Fast Single-Node Solution for Large and Complex Metagenomics Assembly via Succinct de Bruijn Graph." *Bioinformatics* 31 (10): 1674–76.

Li, Heng. 2009. "The Sequence Alignment/Map (SAM) Format and SAMtools 1000 Genome Project Data Processing Subgroup." *Bioinformatics* 25: 1.

Pruesse, Elmar, Christian Quast, Katrin Knittel, Bernhard M. Fuchs, Wolfgang Ludwig, Jörg Peplies, and Frank Oliver Glöckner. 2007. "SILVA: A Comprehensive Online Resource for Quality Checked and Aligned Ribosomal RNA Sequence Data Compatible with ARB." *Nucleic Acids Research* 35 (21): 7188–96. https://doi.org/10.1093/nar/gkm864.

Quast, Christian, Elmar Pruesse, Pelin Yilmaz, Jan Gerken, Timmy Schweer, Pablo Yarza, Jörg Peplies, and Frank Oliver Glöckner. 2012. "The SILVA Ribosomal RNA Gene Database Project: Improved Data Processing and Web-Based Tools." *Nucleic Acids Research* 41 (D1): D590–96. https://doi.org/10.1093/nar/gks1219.

Robeson, Michael S, Devon R O'Rourke, Benjamin D Kaehler, Michal Ziemski, Matthew R Dillon, Jeffrey T Foster, and Nicholas A Bokulich. 2021. "RESCRIPt: Reproducible Sequence Taxonomy Reference Database Management." *PLoS Computational Biology* 17 (11): e1009581.

Ruscheweyh, Hans-Joachim, Alessio Milanese, Lucas Paoli, Anna Sintsova, Daniel R Mende, Georg Zeller, and Shinichi Sunagawa. 2021. "mOTUs: Profiling Taxonomic Composition, Transcriptional Activity and Strain Populations of Microbial Communities." *Current Protocols* 1 (8): e218.

Thomas, Andrew Maltez, Paolo Manghi, Francesco Asnicar, Edoardo Pasolli, Federica Armanini, Moreno Zolfo, Francesco Beghini, et al. 2019. "Metagenomic Analysis of Colorectal Cancer Datasets Identifies Cross-Cohort Microbial Diagnostic Signatures and a Link with Choline Degradation." *Nature Medicine* 25 (4): 667–78.

Tran, Quang, and Vinhthuy Phan. 2020. "Assembling Reads Improves Taxonomic Classification of Species." *Genes* 11 (8): 946.

Zackular, Joseph P, Mary AM Rogers, Mack T Ruffin IV, and Patrick D Schloss. 2014. "The Human Gut Microbiome as a Screening Tool for Colorectal Cancer." *Cancer Prevention Research* 7 (11): 1112–21.

Zeller, Georg, Julien Tap, Anita Y Voigt, Shinichi Sunagawa, Jens Roat Kultima, Paul I Costea, Aurélien Amiot, et al. 2014. "Potential of Fecal Microbiota for Early-Stage Detection of Colorectal Cancer." *Molecular Systems Biology* 10 (11): 766.