

Introduction to Ancient Metagenomics

2023-07-05

Table of contents

Introduction	1
Authors	3
Acknowledgements	19
Financial Support	19
Institutional Support	19
Infrastructural Support	20
Before you Start	21
Creating a conda environment	22
Additional Software	23
I Theory	25
Lectures	27
1 Introduction to NGS Sequencing	31
1.1 Lecture	31
1.2 Readings	31
1.3 Questions to think about	32
2 Introduction to Ancient DNA	33
2.1 Lecture	33
2.2 Questions to think about	33
3 Introduction to Metagenomics	35
3.1 Introduction	35
3.2 Lecture	35
3.3 Questions to think about	35
4 Introduction to Microbial Genomics	37
4.1 Lecture	37
4.2 Questions to think about	37

5 Introduction to Evolutionary Biology	39
II Useful Skills	41
Introduction to the Command Line (Bare Bones Bash)	43
Introduction to R	43
Introduction to Python	43
Introduction to Git and GitHub	44
6 Introduction to the Command Line	45
7 Lecture	47
7.1 Session 1	47
7.2 Session 2	47
8 Introduction to R and the Tidyverse	49
8.1 Lecture	49
8.2 The working environment	50
8.3 Loading data into tibbles	50
8.4 Plotting data in tibbles	52
8.5 Conditional queries on tibbles	55
8.6 Transforming and manipulating tibbles	58
8.7 Combining tibbles with join operations	62
9 Introduction to Python and Pandas	69
9.1 Lecture	69
9.2 Introduction to data manipulation in Python with Pandas and visualization with plotnine	70
9.3 Overview:	71
9.4 o - Foreword, working in a jupyter environment	71
9.5 1 - Loading required libraries	73
9.6 2 - Foreword on Pandas	73
9.7 3 - Reading data with Pandas	74
9.8 5 - Computing basic statistics	119
9.9 6 - Filtering	123
9.10 7 - GroupBy operations, and computing statistics on grouped values .	142
9.11 8 - Reshaping data, from wide to long and back	144
9.12 9 - Joining two different tables	149
9.13 10 - Visualizing some of the results with Plotnine	158
9.14 11 - Bonus, dealing with ill-formatted columns	161
10 Introduction to Git(Hub)	169
10.1 Introduction	169
10.2 Lecture	169
10.3 SSH setup	169
10.4 The only 6 commands you really need to know	170

10.5 Working collaboratively	173
10.6 Pull requests	174
10.7 Questions to think about	174
III Ancient Metagenomics	175
Taxonomic Profiling	177
Functional Profiling	177
<i>De novo</i> Assembly	177
11 Taxonomic Profiling, OTU Tables and Visualisation	179
11.1 Lecture	179
11.2 Download and Subsample	180
11.3 Hands on introduction to ancient microbiome analysis	182
12 Functional Profiling	255
12.1 Lecture	255
12.2 Preparation	255
12.3 HUMAnN ₃ Pathways	256
12.4 humann ₃ tables	257
12.5 Sample Clustering with PCA	263
13 Introduction to <i>de novo</i> Genome Assembly	271
13.1 Introduction	271
IV Ancient Genomics	273
Genome Mapping	275
Phylogenomics	275
14 Genome Mapping	277
14.1 Lecture	277
14.2 Mapping to a Reference Genome	277
15 Introduction to Phylogenomics	289
15.1 Lecture	289
15.2 Preparation	289
15.3 Basic concepts in phylogenomics	289
15.4 The start: DNA sequence alignment	292
15.5 Distance-based phylogenetic methods	296
15.6 Character-based phylogenetic methods: maximum parsimony and probabilistic approaches	304
V Ancient Metagenomic Resources	319
Accessing Ancient Metagenome Data	321
Ancient Metagenomic Pipelines	321

16 Introduction to AncientMetagenomeDir	323
16.1 Lecture	323
16.2 Introduction	323
16.3 Finding Ancient Metagenomic Data	324
16.4 AncientMetagenomeDir	324
16.5 Further Improving Metadata Reporting in Ancient Metagenomics	326
16.6 Running AMDirT	326
16.7 Inspecting AMDirT Output	331
16.8 Git Practise	333
16.9 Summary	334
17 Ancient Metagenomic Pipelines	335
17.1 Lecture	335
17.2 Introduction	335
17.3 What is nf-core/eager?	336
17.4 Steps in the pipeline	336
17.5 How to build an nf-core/eager command: A practical introduction	337
17.6 Top Tips for nf-core/eager success	338
17.7 Questions to think about	339
Summary	341
VI Appendices	343
18 Resources	345
18.1 Introduction to NGS Sequencing	345
References	347
19 Tools	349
19.1 Introduction to R and the Tidyverse	349
19.2 Introduction to Python and Pandas	349
19.3 Introduction to Git(Hub)	349
19.4 Functional Profiling	349
19.5 <i>De novo</i> assembly	350
19.6 Genome Mapping	350
19.7 Phylogenomics	350
19.8 Ancient Metagenomic Pipelines	350

Introduction

Ancient metagenomics applies cutting-edge metagenomic methods to the degraded DNA content of archaeological and palaeontological specimens. The rapidly growing field is currently uncovering a wealth of novel information for both human and natural history, from identifying the causes of devastating pandemics such as the Black Death, to revealing how past ecosystems changed in response to long-term climatic and anthropogenic change, to reconstructing the microbiomes of extinct human relatives. However, as the field grows, the techniques, methods, and workflows used to analyse such data are rapidly changing and improving.

In this book we will go through the main steps of ancient metagenomic bioinformatic workflows, familiarising students with the command line, demonstrating how to process next-generation-sequencing (NGS) data, and showing how to perform de novo metagenomic assembly. Focusing on host-associated ancient metagenomics, the book consists of a combination of theory and hands-on exercises, allowing readers to become familiar with the types of questions and data researchers work with.

By the end of the textbook, readers will have an understanding of how to effectively carry out the major bioinformatic components of an ancient metagenomic project in an open and transparent manner.

Note

If you export the PDF or ePub versions of this book, some sections maybe excluded (such as videos, and embedded slide decks). Always refer to this website in doubt.

All material was originally developed for the [SPAAM Summer School: Introduction to Ancient Metagenomics](#)

Authors

The creation of this text book was developed through a series of ...

2022

**✉ James
Fellows Yates**

is an archaeology-trained biomolecular archaeologist and convert to palaeogenomics, and is recently pivoting to bioinformatics. He specialises in ancient metagenomics analysis, generating tools and high-throughput approaches and high-quality pipelines for validating and analysing ancient (oral) microbiomes and palaeogenomic data.

2022



✉ **Christina Warinner** is Group Leader of Microbiome Sciences at the Max Planck Institute for Evolutionary Anthropology in Leipzig, Germany, and Associate Professor of Anthropology at Harvard University. She serves on the Leadership Team of the Max Planck-Harvard Research Center for the Archaeoscience of the Ancient Mediterranean (MHAAM), and is a Professor in the Faculty of Biological Sciences at Friedrich Schiller University in Jena, Germany. Her research focuses on the use of metagenomics and paleoproteomics to better understand past human diet, health, and the evolution of the human

2022

**✉ Aida
Andrade**

Valtueña is a geneticist interested in pathogen evolution, with particular interest in prehistoric pathogens. She has been exploring new methods to analyse ancient pathogen data to understand their past function and ecology to inform models of pathogen emergence.

2022



 **Alexander**

Herbig is a bioinformatician and group leader for Computational Pathogenomics at the Max Planck Institute for Evolutionary Anthropology. His main interest is in studying the evolution of human pathogens and in methods development for pathogen detection and bacterial genomics.

2022

**✉ Alex**

Hübner is a computational biologist, who originally studied biotechnology, before switching to evolutionary biology during his PhD. For his postdoc in the Warinner lab, he focuses on investigating whether new methods in the field of modern metagenomics can be directly applied to ancient DNA data. Here, he is particularly interested in the *de novo* assembly of ancient metagenomic sequencing data and the subsequent analysis of its results.

2022



 **Alina Hiss**

is a PhD student in the Computational Pathogenomics group at the Max Planck Institute for Evolutionary Anthropology. She is interested in the evolution of human pathogens and working on material from the Carpathian basin to gain insights about the presence and spread of pathogens in the region during the Early Medieval period.

2022

**✉ Arthur****Kocher**

initially trained as a veterinarian.

He then pursued a PhD in the field of disease ecology, during which he studied the impact of biodiversity changes on the transmission of zoonotic diseases using molecular tools such as DNA metabarcoding.

During his Post-Docs, he extended his research focus to evolutionary aspects of pathogens, which he currently investigates using ancient genomic data and Bayesian phylogenetics.

2022



✉ Clemens Schmid is a computational archaeologist pursuing a PhD in the group of Stephan Schiffels at the department of Archaeogenetics at the Max Planck Institute for Evolutionary Anthropology. He is trained both in archaeology and computer science and currently develops computational methods for the spatiotemporal co-analysis of archaeological and ancient genomic data. He worked in research projects on the European Neolithic, Copper and Bronze age and maintains research software in R, C++ and Haskell.

2022

**✉ Irina**

Velsko is a postdoc in the Microbiome group of the department of Archaeogenetics at the Max Planck Institute for Evolutionary Anthropology. She did her PhD work on oral microbiology and immunology of the living, and now works on oral microbiomes of the living and the dead. Her work focuses on the evolution and ecology of dental plaque biofilms, both modern and ancient, and the complex interplay between microbiomes and their hosts.



2022

 **Maxime**

Borry is a doctoral researcher in bioinformatics at the Max Planck Institute for Evolutionary Anthropology in Germany. After an undergraduate in life sciences and a master in Ecology, followed by a master in bioinformatics, he is now working on the completion of his PhD, focused on developing new tools and data analysis of ancient metagenomic samples.

2022

**✉ Megan**

Michel is a PhD student jointly affiliated with the Archaeogenetics Department at the Max Planck Institute for Evolutionary Anthropology and the Human Evolutionary Biology Department at Harvard University. Her research focuses on using computational genomic analyses to understand how pathogens have co-evolved with their hosts over the course of human history.

2022



✉ **Nikolay Oskolkov** is a bioinformatician at Lund University and the bioinformatics platform of SciLifeLab, Sweden. He defended his PhD in theoretical physics in 2007, and switched to life sciences in 2012. His research interests include mathematical statistics and machine learning applied to genetics and genomics, single cell and ancient metagenomics data analysis.

2022



✉ Sebastian Duchene is an Australian Research Council Fellow at the Doherty Institute for Infection and Immunity at the University of Melbourne, Australia. Prior to joining the University of Melbourne he obtained his PhD and conducted postdoctoral work at the University of Sydney. His research is in molecular evolution and epidemiology of infectious pathogens, notably viruses and bacteria, and developing Bayesian phylodynamic methods.

2022



✉ **Thisreas Lamnidis** is a human population geneticist interested in European population history after the Bronze Age. To gain the required resolution to differentiate between Iron Age European populations, he is developing analytical methods based on the sharing of rare variation between individuals. He has also contributed to pipelines that streamline the processing and analysis of genetic data in a reproducible manner, while also facilitating dissemination of information among interdisciplinary colleagues.

Acknowledgements

We would like to thank the following supporters of the original summer schools and eventual textbook.

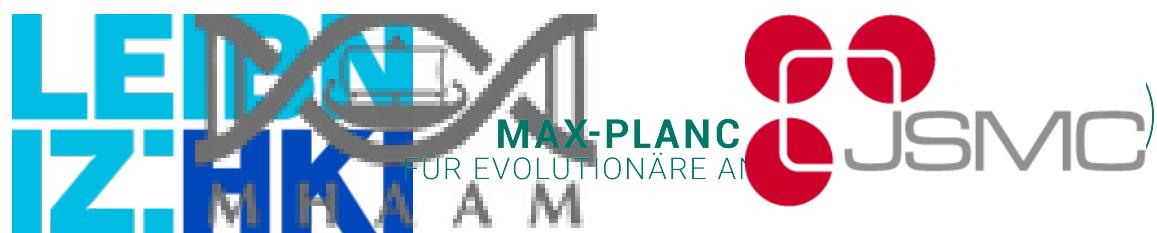
Financial Support



WERNER SIEMENS-STIFTUNG

The content of this textbook was developed from the SPAAM Summer School: Introduction to Ancient Metagenomics summer school series, sponsored by the Werner Siemens-Stiftung (Grant: Paleobiotechnology, awarded to Pierre Stallforth, Hans-Knöll Institute, and Christina Warinner, Max Planck Institute for Evolutionary Anthropology)

Institutional Support



Infrastructural Support



GERMAN NETWORK FOR BIOINFORMATICS INFRASTRUCTURE

The practical sessions of the summers schools work was supported by the BMBF-funded de.NBI Cloud within the German Network for Bioinformatics Infrastructure (de.NBI) (o31A532B, o31A533A, o31A533B, o31A534A, o31A535A, o31A537A, o31A537B, o31A537C, o31A537D, o31A538A). z

Before you Start

The summer school course that this textbook is derived from was designed to be as practical as possible. This means that most of the chapters are designed to act as a walkthrough to guide you through the steps on how to generate and analyse data for each of the major steps of an ancient metagenomics project.

The summer school utilised cloud computing to provide a consistent computing platform for all participants, however all tools and data demonstrated are open-source and publicly available.

Warning

Bioinformatics often involve large computing resource requirements! While we aim to make example data and processing as efficient as possible, we cannot guarantee that they will all be able to work on standard laptops or desktop computing - most likely due to memory/RAM requirements. As a guide, the cloud nodes used during the summer school had 16 cores and 32 GB of RAM.

To follow the practical chapters of this text book, you will require:

- A unix based operating system (e.g., Linux, MacOS, or possibly Windows with Linux Subsystem - however the latter has not been tested)
- A corresponding Unix terminal
- An internet connection
- A web browser
- A `conda` installation with `bioconda` configured.
 - Conda is a very popular package manager for installing software in bioinformatics. `bioconda` is the main source of bioinformatics software for conda.
 - To speed up installation, we would also highly recommend setting up the `libmamba-solver`

For each chapter we will provide a link to a tar archive that will contain the raw data and a `conda.yml` file that specifies the software environment for that chapter.

Before loading the environment for the exercises, the environment will need to be

installed using the `yml` with the instructions below, and then activated. A list of the software in each chapter's environment can be found in the [Appendix](#).

Creating a conda environment

Once conda is installed and bioconda configured, at the beginning of each chapter, to create the conda environment from the `yml` file, you will need to run the following:

1. Download the Zenodo tar archive file either the download button and or 'Copy link address' the URL and run

```
curl -O <url>
## or
wget <url>
```

2. Extract the tar archive with

```
tar -xvf <tar_file>.tar.gz
```

and change into the directory with

```
cd <directory>/
```

3. Within the resulting directory run the following conda command to install the software into it's dedicated environment

```
conda env create -f <env_file>.yml
```

::: {.callout-note} Note: you only have to run the environment creation once. :::

4. Follow the instructions as prompted. Once created, you can see a list of environments with

```
conda env list
```

5. To load the relevant environment, you can run

```
conda activate <name_of_environment>.yml
```

6. Once finished with the chapter, you can deactivate the environment with

```
conda deactivate
```

To reuse the environment, just run step 4 and 5 as necessary.

? Tip

To delete a conda software environment, just get the path listed on `conda env list` and delete the folder with `rm -rf <path>`.

Additional Software .

For some chapters you may need the following software/and or data manually installed, which are not available on bioconda:

- *De novo* assembly
 - MetaWrap

```
conda create -n metawrap-env python=2.7
conda activate metawrap-env
conda install biopython bwa maxbin2 metabat2 samtools=1.9
cd ~/bin/
git clone https://github.com/bxlab/metaWRAP.git
echo "export PATH=$PATH:~/bin/metaWRAP/bin" >> ~/.bashrc
```

- Functional Profiling
 - HUMAnN3 UniRef database (where the functional providing conda environment is already activated - see the Functional Profiling chapter for more details)

```
humann3_databases --download uniref uniref90_ec_filtered_diamond /vol/volume/5c-functional
```

- Phylogenomics
 - [Tempest](#) (v1.5.3)
 - It is also recommended to assign the following bash variable so you can access the tool without the full path

```
export tempest='bash /home/ubuntu/bin/TempEst_v1.5.3/bin/tempest'
```

- [MEGAX](#) (v11.0.11)

Part I

Theory

In the first section of this book we will introduce the basic concepts of a range of topics related to ancient DNA, from how Next Generation Sequencing (NGS) sequencing works, to the fundamental biochemistry of ancient DNA, to the phylogenomic analysis of reconstructed genomes.

The content of this section of the book were originally delivered as lectures, and each chapter will have a recording of the lectures and the accompanying slides.

Lectures

Introduction to NGS Sequencing

In this chapter, we will introduce how we are able to convert DNA molecules to human readable sequences of A, C, T, and Gs, which we can subsequently can computationally analyse.

The field of Ancient DNA was revolutionised by the development of ‘Next Generation Sequencing’ (NGS), which relies on sequencing of millions of *short* fragments of DNA in parallel. The global leading DNA sequencing company is Illumina, and the technology used by Illumina is also most popular by palaeogeneticists. Therefore we will go through the various technologies behind Illumina next-generation sequencing machines.

We will also look at some important differences in the way different models of Illumina sequences work, and how this can influence ancient DNA research. Finally we will cover the structure of ‘FASTQ’ files, the most popular file format for representing the DNA sequence output of NGS sequencing machines.

Introduction to Ancient DNA

This chapter introduces you to ancient DNA and the enormous technological changes that have taken place since the field’s origins in 1984. Starting with the quagga and proceeding to microbes, we discuss where ancient microbial DNA can be found in the archaeological record and examine how ancient DNA is defined by its condition, not by a fixed age.

We next cover genome basics and take an in-depth look at the way DNA degrades over time. We detail the fundamentals of DNA damage, including the specific chemical processes that lead to DNA fragmentation and C->T miscoding lesions. We then demystify the DNA damage “smiley plot” and explain the how the plot’s symmetry or asymmetry is related to the specific enzymes used to repair DNA during library construction. We discuss how DNA damage is and is not clock-like, how to interpret and troubleshoot DNA damage plots, and how DNA damage patterns can be used to authenticate ancient samples, specific taxa, and even sequences. We cover laboratory strategies for removing or reducing damage for greater accuracy for genotype calling, and we discuss the pros and cons of single-stranded library protocols. We then take a closer look at proofreading and non-proofreading polymerases and note key steps

in NGS library preparation during which enzyme selection is critical in ancient DNA studies.

Finally, we examine the big picture of why DNA damage matters in ancient microbial studies, and its effects on taxonomic identification of sequences, accurate genome mapping, and metagenomic assembly.

Introduction to Metagenomics

This chapter introduces you to the basics of metagenomics, with an emphasis on tools and approaches that are used to study ancient metagenomes. We begin by covering the basic terminology used in metagenomics and microbiome research and discuss how the field has changed over time. We examine the species concept for microbes and challenges that arise in classifying microbial species with respect to taxonomy and phylogeny. We then proceed to taxonomic profiling and discuss the pros and cons of different taxonomic profilers.

Afterwards, we explain how to estimate preservation in ancient metagenomic samples and how to clean up your datasets and remove contaminants. Finally, we discuss strategies for exploring and comparing the ecological diversity in your samples, including different strategies for data normalization, distance calculation, and ordination.

Introduction to Microbial Genomics

The field of microbial genomics aims at the reconstruction and comparative analyses of genomes for gaining insights into the genetic foundation and evolution of various functional aspects such as virulence mechanisms in pathogens.

Including data from ancient samples into this comparative assessment allows for studying these evolutionary changes through time. This, for example, provides insights into the emergence of human pathogens and their development in conjunction with human cultural transitions.

In this chapter we will look examples for how to utilise data from ancient genomes in comparative studies of human pathogens and today's practical sessions will highlight methodologies for the reconstruction of microbial genomes.

Introduction to Evolutionary Biology

Pathogen genome data are an invaluable source of information about the evolution and spread of these organisms. This chapter will focus on molecular phylogenetic methods and the insight that they can reveal from improving our understanding of ancient evolution to the epidemiological dynamics of current outbreaks.

The first section will introduce phylogenetic trees and a set of core terms and concepts for their interpretation. Next, it will focus on some of the most popular approaches to inferring phylogenetic trees; those based on genetic distance, maximum likelihood, and Bayesian inference. These methods carry important considerations

regarding the process that generated the data, computational capability, and data quality, all of which will be discussed here. Finally, we will direct our attention to examples of analyses of ancient and modern pathogens (e.g. *Yersinia pestis*, Hepatitis B virus, SARS-CoV-2) and critically assess appropriate choice of models and methods.

Chapter 1

Introduction to NGS Sequencing

1.1 Lecture

PDF version of the slide lectures can be downloaded from [here](#).

1.2 Readings

1.2.1 Reviews

(Schuster 2008)

(Shendure and Ji 2008)

(Slatko, Gardner, and Ausubel 2018)

(Dijk et al. 2014)

1.2.2 Sequencing Library Construction

(Kircher, Sawyer, and Meyer 2012)

(Meyer and Kircher 2010)

1.2.3 Errors and Considerations

(Ma et al. 2019)

(Sinha et al. 2017)

(Valk et al. 2019)

1.3 Questions to think about

- Why is Illumina sequencing technologies useful for aDNA?
- What problems can the 2-colour chemistry technology of NextSeq and NovaSeqs cause in downstream analysis?
- Why is ‘Index-Hopping’ a problem?
- What is good software to evaluate the quality of your sequencing runs?

Chapter 2

Introduction to Ancient DNA

2.1 Lecture

PDF version of these slides can be downloaded from [here](#).

2.2 Questions to think about

- What is ancient DNA?
- Where do we find ancient DNA from microbes?
- How does DNA degrade?
- How do I interpret a DNA damage plot?
- How is DNA damage used to authenticate ancient genomes and samples?
- What methods are available for managing DNA damage?
- How does DNA damage matter for my analyses?

Chapter 3

Introduction to Metagenomics

3.1 Introduction

3.2 Lecture

PDF version of these slides can be downloaded from [here](#).

3.3 Questions to think about

- What is a metagenome? a microbiota? a microbiome?
- What is ancient metagenomics?
- What challenges do DNA degradation and sample decay pose for ancient metagenomics
- How do you find out “who’s there” in your samples?
- How do alignment based and k-mer based taxonomic profilers differ? What are the advantages and disadvantages of each?
- Why does database selection matter?
- How do you estimate the preservation and integrity of your ancient metagenome?
- What are tools you can use to identify poorly preserved samples and remove contaminant taxa?
- What aspects of diversity are important in investigating microbial communities?
- Which distance metrics are commonly used to compare the beta-diversity of microbial communities and why? What are some advantages and disadvantages to these different approaches?

Chapter 4

Introduction to Microbial Genomics

4.1 Lecture

PDF version of these slides can be downloaded from [here](#).

4.2 Questions to think about

Chapter 5

Introduction to Evolutionary Biology

5.0.1 Lecture

PDF version of these slides can be downloaded from [here](#).

Part II

Useful Skills

In this section, we will cover some useful computational skills that will likely be important for executing the analysis phase of any ancient DNA projects. With a focus on open- and reproducible science, we will cover introducing the command line, common programming languages to help automate your analyses, and also how to use Git(Hub) for sharing code.

Introduction to the Command Line (Bare Bones Bash)

Computational work in metagenomics often involves connecting to remote servers to run analyses via the use of command line tools. Bash is a programming language that is used as the main command line interface of most UNIX systems, and hence most remote servers a user will encounter. By learning bash, users can work more efficiently and reproducibly on these remote servers.

In this chapter we will introduce the basic concepts of bash and the command line. Students will learn how to move around the filesystem and interact with files, how to chain multiple commands together using “pipes”, and how to use loops and regular expressions to simplify the running of repetitive tasks.

Finally, readers will learn how to create a bash script of their own, that can run a set of commands in sequence. This session requires no prior knowledge of bash or the command line and is meant to serve as an entry-level introduction to basic programming concepts that can be applicable in other programming languages too.

Introduction to R

R is an interpreted programming language with a particular focus on data manipulation and analysis. It is very well established for scientific computing and supported by an active community developing and maintaining a huge ecosystem of software packages for both general and highly derived applications.

In this chapter we will explore how to use R for a simple, standard data science workflow. We will import, clean, and visualise context and summary data for and from our ancient metagenomics analysis workflow. On the way we will learn about the RStudio integrated development environment, dip into the basic logic and syntax of R and finally write some first useful code within the tidyverse framework for tidy, readable and reproducible data analysis.

This chapter will be targeted at beginners without much previous experience with R or programming and will kickstart your journey to master this powerful tool.

Introduction to Python

While R has traditionally been the language of choice for statistical programming for many years, Python has taken away some of the hegemony thanks to its numerous available libraries for machine and deep learning. With its ever increasing collection

of libraries for statistics and bioinformatics, Python has now become one of the most used language in the bioinformatics community.

In this tutorial, mirroring to the R session, we will learn how to use the Python libraries Pandas for importing, cleaning, and manipulating data tables, and producing simple plots with the Python sister library of ggplot2, plotnine.

We will also get ourselves familiar with the Jupyter notebook environment, often used by many high performance computing clusters as an interactive scripting interface.

This chapter is meant for participants with a basic experience in R/tidyverse, but assumes no prior knowledge of Python/Jupyter.

Introduction to Git and GitHub

As the size and complexity of metagenomic analyses continues to expand, effectively organizing and tracking changes to scripts, code, and even data, continues to be a critical part of ancient metagenomic analyses. Furthermore, this complexity is leading to ever more collaborative projects, with input from multiple researchers.

In this chapter, we will introduce ‘Git’, an extremely popular version control system used in bioinformatics and software development to store, track changes, and collaborate on scripts and code. We will also introduce, GitHub, a cloud-based service for Git repositories for sharing data and code, and where many bioinformatic tools are stored. We will learn how to access and navigate course materials stored on GitHub through the web interface as well as the command line, and we will create our own repositories to store and share the output of upcoming sessions.

Chapter 6

Introduction to the Command Line



Tip

For this chapter's exercises, you will only need a unix terminal, and do not need any special software installed

Chapter 7

Lecture

7.1 Session 1

For a full screen version on the presentation and press f on your keyboard.

[Intro to Bash](#)

PDF version of these slides can be downloaded from [here](#).

The teaching material for the FULL BareBonesBash course can be found on the [Bare-BonesBash website](#)

7.2 Session 2

For a full screen version click on the presentation and press f on your keyboard.

[Intro to Bash](#)

PDF version of these slides can be downloaded from [here](#).

The teaching material for the FULL BareBonesBash course can be found on the [Bare-BonesBash website](#)

Chapter 8

Introduction to R and the Tidyverse

Note

This session is typically ran held in parallel to the Introduction to Python and Pandas. Participants of the summer schools chose which to attend based on their prior experience. We recommend the introduction to R session if you have no experience with neither R nor Python.

Tip

For this chapter's exercises, if not already performed, you will need to create the **conda** environment from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate r-python
```

8.1 Lecture

PDF version of these slides can be downloaded from [here](#).

8.2 The working environment

8.2.1 R, RStudio and the tidyverse

- R is a fully featured programming language, but it excels as an environment for (statistical) data analysis (<https://www.r-project.org>)
- RStudio is an integrated development environment (IDE) for R (and other languages): (<https://www.rstudio.com/products/rstudio>)
- The tidyverse is a collection of R packages with well-designed and consistent interfaces for the main steps of data analysis: loading, transforming and plotting data (<https://www.tidyverse.org>)
 - This introduction works with tidyverse ~v1.3.0
 - We will learn about `readr`, `tibble`, `ggplot2`, `dplyr`, `magrittr` and `tidyR`
 - `forcats` will be briefly mentioned
 - `purrr` and `stringr` are left out

8.3 Loading data into tibbles

8.3.1 Reading data with `readr`

- With R we usually operate on data in our computer's memory
- The tidyverse provides the package `readr` to read data from text files into the memory
- `readr` can read from our file system or the internet
- It provides functions to read data in almost any (text) format:

```
readr::read_csv()    # .csv files
readr::read_tsv()    # .tsv files
readr::read_delim()  # tabular files with an arbitrary separator
readr::read_fwf()    # fixed width files
readr::read_lines()  # read linewise to parse yourself
```

- `readr` automatically detects column types – but you can also define them manually

8.3.2 How does the interface of `read_csv` work

- We can learn more about a function with `?.`. To open a help file: `?readr::read_csv`
- `readr::read_csv` has many options to specify how to read a text file

```
read_csv(
  file,                      # The path to the file we want to read
  col_names = TRUE,           # Are there column names?
  col_types = NULL,           # Which types do the columns have? NULL -> auto
  locale = default_locale(),  # How is information encoded in this file?
  na = c("", "NA"),           # Which values mean "no data"
  trim_ws = TRUE,             # Should superfluous white-spaces be removed?
  skip = 0,                   # Skip X lines at the beginning of the file
  n_max = Inf,                # Only read X lines
  skip_empty_rows = TRUE,     # Should empty lines be ignored?
  comment = "",               # Should comment lines be ignored?
  name_repair = "unique",    # How should "broken" column names be fixed
  ...
)
```

8.3.3 What does `readr` produce? The tibble

```
sample_table_path <- "/vol/volume/3b-1-introduction-to-r-and-the-tidyverse/ancientmetagenome-hostas
sample_table_url <-
"https://raw.githubusercontent.com/SPAAM-community/AncientMetagenomeDir/b187df6ebd23df2eb42935fd5020
ancientmetagenome-hostassociated/samples/ancientmetagenome-hostassociated_samples.tsv"

samples <- readr::read_tsv(sample_table_url)
```

- The tibble is a “data frame”, a tabular data structure with rows and columns
- Unlike a simple array, each column can have another data type

```
print(samples, n = 3)
```

8.3.4 How to look at a tibble

```
samples          # Typing the name of an object will print it to the console
str(samples)     # A structural overview of an object
summary(samples) # A human-readable summary of an object
View(samples)    # RStudio's interactive data browser
```

- R provides a very flexible indexing operation for `data.frames` and `tibbles`

<code>samples[1,1]</code>	# Access the first row and column
<code>samples[1,]</code>	# Access the first row
<code>samples[,1]</code>	# Access the first column

```
samples[c(1,2,3),c(2,3,4)]          # Access values from rows and columns
samples[,-c(1,2)]                   # Remove the first two columns
samples[,c("site_name", "material")] # Columns can be selected by name
```

- tibbles are mutable data structures, so their content can be overwritten

```
samples[1,1] <- "Cheesecake2015"      # replace the first value in the first column
```

8.4 Plotting data in tibbles

8.4.1 ggplot2 and the “grammar of graphics”

- ggplot2 offers an unusual, but powerful and logical interface
- The following example describes a stacked bar chart

```
library(ggplot2) # Loading a library to use its functions without ::

ggplot(           # Every plot starts with a call to the ggplot() function
  data = samples # This function can also take the input tibble
) +              # The plot consists of functions linked with +
  geom_bar(       # "geoms" define the plot layers we want to draw
    mapping = aes( # The aes() function maps variables to visual properties
      x = publication_year, # publication_year -> x-axis
      fill = community_type # community_type -> fill color
    )
  )
```

- `geom_*`: data + geometry (bars) + statistical transformation (sum)

8.4.2 ggplot2 and the “grammar of graphics”

- This is the plot described above: number of samples per community type through time

```
ggplot(samples) +
  geom_bar(aes(x = publication_year, fill = community_type))
```

8.4.3 ggplot2 features many geoms

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and a + expand_limits()
Ensure limits include values across all plots.
```

TWO VARIABLES both continuous

```
e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1,
nudge_y = 1) -> x, y, label, alpha, angle, color,
family, fontface, hjust, lineheight, size, vjust
```

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight
```

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

```
b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(intercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:155, radius = 1))
```

continuous function

```
i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size
```

continuous bivariate distribution

```
h + geom_hex()
x, y, alpha, color, fill, size
```

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
```

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size
```

continuous function

```
i + geom_line()
x, y, alpha, color, group, linetype, size
```

discrete

```
d <- ggplot(mpg, aes(frt))
d + geom_bar()
```

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state, map = map)
+ expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size, weight)
```

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

```
b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(intercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:155, radius = 1))
```

continuous function

```
i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size
```

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))

h + geom_hex()
x, y, alpha, color, fill, size
```

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
```

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size
```

continuous function

```
i + geom_line()
x, y, alpha, color, group, linetype, size
```

discrete

```
d <- ggplot(mpg, aes(frt))
d + geom_bar()
```

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state, map = map)
+ expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size, weight)
```

THREE VARIABLES

```
sealsSz <- width(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, color, group, linetype, size, weight
```

continuous function

```
i + geom_raster(aes(fill = z))
x, y, alpha, color, fill, group, linetype, size, weight
```

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha,
color, fill, group, linetype, shape, size, weight
```

continuous function

```
i + geom_line()
x, y, alpha, color, group, linetype, size
```

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))

h + geom_hex()
x, y, alpha, color, fill, size
```

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state, map = map)
+ expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size, weight)
```

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

```
b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(intercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:155, radius = 1))
```

continuous function

```
i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size
```

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))

h + geom_hex()
x, y, alpha, color, fill, size
```

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
```

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size
```

continuous function

```
i + geom_line()
x, y, alpha, color, group, linetype, size
```

discrete

```
d <- ggplot(mpg, aes(frt))
d + geom_bar()
```

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state, map = map)
+ expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size, weight)
```

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

```
b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(intercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:155, radius = 1))
```

continuous function

```
i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size
```

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))

h + geom_hex()
x, y, alpha, color, fill, size
```

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
```

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size
```

continuous function

```
i + geom_line()
x, y, alpha, color, group, linetype, size
```

discrete

```
d <- ggplot(mpg, aes(frt))
d + geom_bar()
```

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state, map = map)
+ expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size, weight)
```

- RStudio shares helpful cheatsheets for the tidyverse and beyond: <https://www.rstudio.com/resources/cheatsheets>

8.4.4 scales control the behaviour of visual elements

- Another plot: Boxplots of sample age through time

```
ggplot(samples) +
  geom_boxplot(aes(x = as.factor(publication_year), y = sample_age))
```

- This is not well readable, because extreme outliers dictate the scale

8.4.5 scales control the behaviour of visual elements

- We can change the **scale** of different visual elements - e.g. the y-axis

```
ggplot(samples) +
  geom_boxplot(aes(x = as.factor(publication_year), y = sample_age)) +
```

```
scale_y_log10()
```

- The log-scale improves readability

8.4.6 scales control the behaviour of visual elements

- (Fill) color is a visual element of the plot and its scaling can be adjusted

```
ggplot(samples) +
  geom_boxplot(aes(x = as.factor(publication_year), y = sample_age,
                    fill = as.factor(publication_year))) +
  scale_y_log10() + scale_fill_viridis_d(option = "C")
```

8.4.7 Defining plot matrices via facets

- Splitting up the plot by categories into **facets** is another way to visualize more variables at once

```
ggplot(samples) +
  geom_count(aes(x = as.factor(publication_year), y = material)) +
  facet_wrap(~archive)
```

- Unfortunately the x-axis became unreadable

8.4.8 Setting purely aesthetic settings with theme

- Aesthetic changes like this can be applied as part of the theme

```
ggplot(samples) +
  geom_count(aes(x = as.factor(publication_year), y = material)) +
  facet_wrap(~archive) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```

8.4.9 Exercise 1

1. Look at the `mtcars` dataset and read up on the meaning of its variables
2. Visualize the relationship between *Gross horsepower* and *1/4 mile time*
3. Integrate the *Number of cylinders* into your plot

8.4.10 Possible solutions 1

1. Look at the mtcars dataset and read up on the meaning of its variables

```
?mtcars
```

2. Visualize the relationship between *Gross horsepower* and *1/4 mile time*

```
ggplot(mtcars) + geom_point(aes(x = hp, y = qsec))
```

3. Integrate the *Number of cylinders* into your plot

```
ggplot(mtcars) + geom_point(aes(x = hp, y = qsec, color = as.factor(cyl)))
```

8.5 Conditional queries on tibbles

8.5.1 Selecting columns and filtering rows with `select` and `filter`

- The dplyr package includes powerful functions to subset data in tibbles based on conditions
- `dplyr::select` allows to select columns

```
dplyr::select(samples, project_name, sample_age) # reduce to two columns
dplyr::select(samples, -project_name, -sample_age) # remove two columns
```

- `dplyr::filter` allows for conditional filtering of rows

```
dplyr::filter(samples, publication_year == 2014) # samples published in 2014
dplyr::filter(samples, publication_year == 2014 |
             publication_year == 2018) # samples from 2015 OR 2018
dplyr::filter(samples, publication_year %in% c(2014, 2018)) # match operator: %in%
dplyr::filter(samples, sample_host == "Homo sapiens" &
              community_type == "oral") # oral samples from modern humans
```

8.5.2 Chaining functions together with the pipe `%>%`

- The pipe `%>%` in the magrittr package is a clever infix operator to chain data and operations

```
library(magrittr)
samples %>% dplyr::filter(publication_year == 2014)
```

- It forwards the LHS as the first argument of the function appearing on the RHS
- That allows for sequences of functions (“tidyverse style”)

```
samples %>%
dplyr::select(sample_host, community_type) %>%
dplyr::filter(sample_host == "Homo sapiens" & community_type == "oral") %>%
nrow() # count the rows
```

- magrittr also offers some more operators, among which the extraction %\$% is particularly useful

```
samples %>%
dplyr::filter(material == "tooth") %$%
sample_age %>% # extract the sample_age column as a vector
max()           # get the maximum of said vector
```

8.5.3 Summary statistics in base R

- Summarising and counting data is indispensable and R offers all operations you would expect in its base package

```
nrow(samples)          # number of rows in a tibble
length(samples$site_name) # length/size of a vector
unique(samples$material) # unique elements of a vector
min(samples$sample_age) # minimum
max(samples$sample_age) # maximum
mean(samples$sample_age) # mean
median(samples$sample_age) # median
var(samples$sample_age) # variance
sd(samples$sample_age) # standard deviation
quantile(samples$sample_age, probs = 0.75) # sample quantiles for the given probs
```

- many of these functions can ignore missing values with an option na.rm = TRUE

8.5.4 Group-wise summaries with group_by and summarise

- These summary statistics are particular useful when applied to conditional subsets of a dataset
- dplyr allows such summary operations with a combination of group_by and summarise

```

samples %>%
dplyr::group_by(material) %>% # group the tibble by the material column
dplyr::summarise(
  min_age = min(sample_age), # a new column: min age for each group
  median_age = median(sample_age), # a new column: median age for each group
  max_age = max(sample_age) # a new column: max age for each group
)

```

- grouping can be applied across multiple columns

```

samples %>%
dplyr::group_by(material, sample_host) %>% # group by material and host
dplyr::summarise(
  n = dplyr::n(), # a new column: number of samples for each group
  .groups = "drop" # drop the grouping after this summary operation
)

```

8.5.5 Sorting and slicing tibbles with arrange and slice

- dplyr allows to arrange tibbles by one or multiple columns

```

samples %>% dplyr::arrange(publication_year)           # sort by publication year
samples %>% dplyr::arrange(publication_year,
                           sample_age)                  # ... and sample age
samples %>% dplyr::arrange(dplyr::desc(sample_age)) # sort descending on sample age

```

- Sorting also works within groups and can be paired with slice to extract extreme values per group

```

samples %>%
dplyr::group_by(publication_year) %>% # group by publication year
dplyr::arrange(dplyr::desc(sample_age)) %>% # sort by age within (!) groups
dplyr::slice_head(n = 2) %>%                 # keep the first two samples per group
dplyr::ungroup()                            # remove the still lingering grouping

```

- Slicing is also the relevant operation to take random samples from the observations in a tibble

```

samples %>% dplyr::slice_sample(n = 20)

```

8.5.6 Exercise 2

1. Determine the number of cars with four *forward gears* (gear) in the mtcars dataset

2. Determine the mean *1/4 mile time* (qsec) per *Number of cylinders* (cyl) group
3. Identify the least efficient cars for both *transmission types* (am)

8.5.7 Possible solutions 2

1. Determine the number of cars with four *forward gears* (gear) in the mtcars dataset

```
mtcars %>% dplyr::filter(gear == 4) %>% nrow()
```

2. Determine the mean *1/4 mile time* (qsec) per *Number of cylinders* (cyl) group

```
mtcars %>% dplyr::group_by(cyl) %>% dplyr::summarise(qsec_mean = mean(qsec))
```

3. Identify the least efficient cars for both *transmission types* (am)

```
#mtcars3 <- tibble::rownames_to_column(mtcars, var = "car") %>% tibble::as_tibble()
mtcars %>% dplyr::group_by(am) %>% dplyr::arrange(mpg) %>% dplyr::slice_head()
```

8.6 Transforming and manipulating tibbles

8.6.1 Renaming and reordering columns and values with `rename`, `relocate` and `recode`

- Columns in tibbles can be renamed with `dplyr::rename` and reordered with `dplyr::relocate`

```
samples %>% dplyr::rename(country = geo_loc_name) # rename a column
samples %>% dplyr::relocate(site_name, .before = project_name) # reorder columns
```

- Values in columns can also be changed with `dplyr::recode`

```
samples$sample_host %>% dplyr::recode(`Homo sapiens` = "modern human")
```

- R supports explicitly ordinal data with factors, which can be reordered as well
- factors can be handled more easily with the `forcats` package

```
ggplot(samples) + geom_bar(aes(x = community_type)) # bars are alphabetically ordered
sa2 <- samples
sa2$cto <- forcats::fct_reorder(sa2$community_type, sa2$community_type, length)
# fct_reorder: reorder the input factor by a summary statistic on an other vector
ggplot(sa2) + geom_bar(aes(x = community_type)) # bars are ordered by size
```

8.6.2 Adding columns to tibbles with `mutate` and `transmute`

- A common application of data manipulation is adding derived columns. `dplyr` offers that with `mutate`

```
samples %>%
dplyr::mutate(
  archive_summary = paste0(archive, ":", archive_accession) # combines two other
) %$% archive_summary # columns
```

- `dplyr::transmute` removes all columns but the newly created ones

```
samples %>%
dplyr::transmute(
  sample_name = tolower(sample_name), # overwrite this column
  publication_doi # select this column
)
```

- `tibble::add_column` behaves as `dplyr::mutate`, but gives more control over column position

```
samples %>% tibble::add_column(., id = 1:nrow(.), .before = "project_name")
```

8.6.3 Conditional operations with `ifelse` and `case_when`

- `ifelse` allows to implement conditional `mutate` operations, that consider information from other columns, but that gets cumbersome easily

```
samples %>% dplyr::mutate(hemi = ifelse(latitude >= 0, "North", "South")) %$% hemi
samples %>% dplyr::mutate(
  hemi = ifelse(is.na(latitude), "unknown", ifelse(latitude >= 0, "North", "South"))
) %$% hemi
```

- `dplyr::case_when` is a much more readable solution for this application

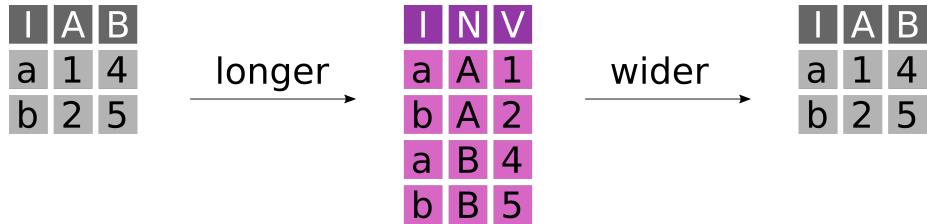
```

samples %>% dplyr::mutate(
  hemi = dplyr::case_when(
    latitude >= 0 ~ "North",
    latitude < 0 ~ "South",
    TRUE           ~ "unknown" # TRUE catches all remaining cases
  )
) %$% hemi

```

8.6.4 Long and wide data formats

- For different applications or to simplify certain analysis or plotting operations data often has to be transformed from a **wide** to a **long** format or vice versa



- A table in **wide** format has N key columns and N value columns
- A table in **long** format has N key columns, one descriptor column and one value column

8.6.5 A wide dataset

```

carsales <- tibble::tribble(
  ~brand, ~`2014`, ~`2015`, ~`2016`, ~`2017`,
  "BMW",   20,      25,      30,      45,
  "VW",    67,      40,     120,      55
)
carsales

```

- Wide format becomes a problem, when the columns are semantically identical. This dataset is in wide format and we can not easily plot it
- We generally prefer data in long format, although it is more verbose with more duplication. “Long” format data is more “tidy”

8.6.6 Making a wide dataset long with pivot_longer

```

carsales_long <- carsales %>% tidyverse::pivot_longer(
  cols = tidyselect:::num_range("", range = 2014:2017), # set of columns to transform
  names_to = "year",           # the name of the descriptor column we want
  names_transform = as.integer, # a transformation function to apply to the names
  values_to = "sales"         # the name of the value column we want
)

carsales_long

```

8.6.7 Making a long dataset wide with pivot_wider

```

carsales_wide <- carsales_long %>% tidyverse::pivot_wider(
  id_cols = "brand", # the set of id columns that should not be changed
  names_from = year, # the descriptor column with the names of the new columns
  values_from = sales # the value column from which the values should be extracted
)

carsales_wide

```

- Applications of wide datasets are adjacency matrices to represent graphs, covariance matrices or other pairwise statistics
- When data gets big, then wide formats can be significantly more efficient (e.g. for spatial data)

8.6.8 Exercise 3

1. Move the column gear to the first position of the mtcars dataset
2. Make a new dataset mtcars2 with the column mpg and an additional column am_v, which encodes the *transmission type* (am) as either "manual" or "automatic"
3. Count the number of cars per *transmission type* (am_v) and *number of gears* (gear). Then transform the result to a wide format, with one column per *transmission type*.

8.6.9 Possible solutions 3

- Move the column gear to the first position of the mtcars dataset

```
mtcars %>% dplyr::relocate(gear, .before = mpg)
```

- Make a new dataset mtcars2 with the column gear and an additional column am_v, which encodes the *transmission type* (am) as either "manual" or "automatic"

```
mtcars2 <- mtcars %>% dplyr::mutate(
  gear, am_v = dplyr::case_when(am == 0 ~ "automatic", am == 1 ~ "manual")
)
```

- Count the number of cars in mtcars2 per *transmission type* (am_v) and *number of gears* (gear). Then transform the result to a wide format, with one column per *transmission type*.

```
mtcars2 %>% dplyr::group_by(am_v, gear) %>% dplyr::tally() %>%
  tidyr::pivot_wider(names_from = am_v, values_from = n)
```

8.7 Combining tibbles with join operations

8.7.1 Types of joins

Joins combine two datasets x and y based on key columns

- Mutating joins add columns from one dataset to the other
 - Left join: Take observations from x and add fitting information from y
 - Right join: Take observations from y and add fitting information from x
 - Inner join: Join the overlapping observations from x and y
 - Full join: Join all observations from x and y, even if information is missing
- Filtering joins remove observations from x based on their presence in y
 - Semi join: Keep every observation in x that is in y
 - Anti join: Keep every observation in x that is not in y

8.7.2 A second dataset

```
library_table_path <- "/vol/volume/3b-1-introduction-to-r-and-the-tidyverse/ancientmetag
library_table_url <-
"https://raw.githubusercontent.com/SPAAM-community/AncientMetagenomeDir/b187df6ebd23df
ancientmetagenome-hostassociated/libraries/ancientmetagenome-hostassociated_libraries.ts
libraries <- readr::read_tsv(library_table_url)
```

```
print(libraries, n = 3)
```

8.7.3 Meaningful subsets

```
samsub <- samples %>% dplyr::select(project_name, sample_name, sample_age)
libsub <- libraries %>% dplyr::select(project_name, sample_name, library_name, read_count)
```

```
print(samsub, n = 3)
print(libsub, n = 3)
```

8.7.4 Left join

Take observations from x and add fitting information from y

A	B	C		A	B	D		A	B	C	D
a	t	1		a	t	3		a	t	1	3
b	u	2		b	u	2		b	u	2	2
c	v	3		d	w	1		c	v	3	-

```
left <- dplyr::left_join(
  x = samsub,                                # 1060 observations
  y = libsub,                                 # 1657 observations
  by = c("project_name", "sample_name") # the key columns by which to join
)

print(left, n = 1)
```

- Left joins are the most common join operation: Add information from another dataset

8.7.5 Right join

Take observations from y and add fitting information from x

A B C		A B D		A B C D
a t 1	+	a t 3	=	a t 1 3
b u 2		b u 2		b u 2 2
c v 3		d w 1		d w - 1

```
right <- dplyr::right_join(
  x = samsub,                               # 1060 observations
  y = libsub,                                # 1657 observations
  by = c("project_name", "sample_name")
)

print(right, n = 1)
```

- Right joins are almost identical to left joins – only x and y have reversed roles

8.7.6 Inner join

Join the overlapping observations from x and y

A B C		A B D		A B C D
a t 1	+	a t 3	=	a t 1 3
b u 2		b u 2		b u 2 2
c v 3		d w 1		

```
inner <- dplyr::inner_join(
  x = samsub,                               # 1060 observations
  y = libsub,                                # 1657 observations
  by = c("project_name", "sample_name")
)

print(inner, n = 1)
```

- Inner joins are a fast and easy way to check, to which degree two dataset overlap

8.7.7 Full join

Join all observations from x and y, even if information is missing

A	B	C		A	B	D		A	B	C	D
a	t	1		a	t	3		a	t	1	3
b	u	2		b	u	2		b	u	2	2
c	v	3		d	w	1		c	v	3	-

+

A	B	C		A	B	D		A	B	C	D
a	t	1		a	t	3		a	t	1	3
b	u	2		b	u	2		b	u	2	2
c	v	3		d	w	1		c	v	3	-

=

A	B	C	D	A	B	C	D
a	t	1	3	a	t	1	3
b	u	2	2	b	u	2	2
c	v	3	-	c	v	3	-

d w - 1

```
full <- dplyr::full_join(
  x = samsub,                               # 1060 observations
  y = libsub,                                # 1657 observations
  by = c("project_name", "sample_name")
)

print(full, n = 1)
```

- Full joins allow to preserve every bit of information

8.7.8 Semi join

Keep every observation in x that is in y

A	B	C		A	B	D		A	B	C	
a	t	1		a	t	3		a	t	1	
b	u	2		b	u	2		b	u	2	
c	v	3		d	w	1		c	v	3	

+

A	B	C		A	B	D		A	B	C	
a	t	1		a	t	3		a	t	1	
b	u	2		b	u	2		b	u	2	
c	v	3		d	w	1		c	v	3	

=

A	B	C		A	B	C	
a	t	1		a	t	1	
b	u	2		b	u	2	

```
semi <- dplyr::semi_join(
  x = samsub,                               # 1060 observations
  y = libsub,                                # 1657 observations
  by = c("project_name", "sample_name")
)

print(semi, n = 1)
```

- Semi joins are underused operations to filter datasets

8.7.9 Anti join

Keep every observation in x that is not in y

A	B	C		A	B	D		A	B	C
a	t	1	+	a	t	3	=	c	v	3
b	u	2		b	u	2				
c	v	3		d	w	1				

```
anti <- dplyr::anti_join(
  x = samsub,                      # 1060 observations
  y = libsub,                      # 1657 observations
  by = c("project_name", "sample_name")
)

print(anti, n = 1)
```

- Anti joins allow to quickly specify incomplete datasets and missing information

8.7.10 Exercise 4

Consider the following additional dataset:

```
gear_opinions <- tibble::tibble(gear = c(3, 5), opinion = c("boring", "wow"))
```

1. Add my opinions about gears to the mtcars dataset
2. Remove all cars from the dataset for which I don't have an opinion

8.7.11 Possible Solutions 4

1. Add my opinions about gears to the mtcars dataset

```
dplyr::left_join(mtcars, gear_opinions, by = "gear")
```

2. Remove all cars from the dataset for which I don't have an opinion

```
dplyr::anti_join(mtcars, gear_opinions, by = "gear")
```


Chapter 9

Introduction to Python and Pandas

Note

This session is typically ran held in parallel to the Introduction to R and Tidyverse. Participants of the summer schools chose which to attend based on their prior experience. We recommend the introduction to R session if you have no experience with either R nor Python.

Tip

For this chapter's exercises, if not already performed, you will need to create the **conda environment** from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate r-python
```

9.1 Lecture

PDF version of these slides can be downloaded from [here](#).

This session is run using a Jupyter notebook. This can be found [here](#). However, it will already be installed on compute nodes during the summer school.

⚠ Warning

We highly recommend viewing this walkthrough via the Jupyter notebook above! The output of commands on the website for this walkthrough are displayed in their own code blocks - be wary of what you copy-paste!

```
from IPython.core.display import SVG
```

9.2 Introduction to data manipulation in Python with Pandas and visualization with plotnine

Maxime Borry
SPAAM Summer School 2022

```
SVG(filename='img/whoami.svg')
```

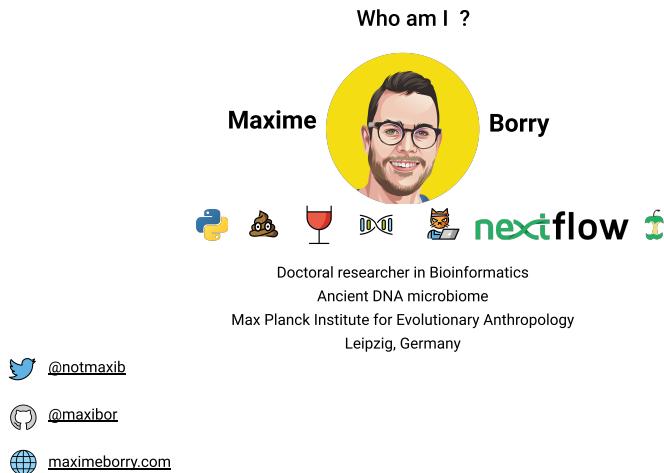


Figure 9.1: svg

Over the last few years, Python has gained an immense amount of popularity thanks to its numerous libraries in the field of machine learning, statistical data analysis, and bioinformatics. While a few years ago, it was often necessary to go back to R for performing routine data manipulation and analysis tasks, nowadays Python has a vast ecosystem of libraries for doing just that.

Today, we will do a quick introduction of the most popular libraries for data analysis:

- [pandas](#), for reading and manipulation tabular data

- [plotnine](#), the Python clone of ggplot2

9.3 Overview:

- o - Foreword, working in a jupyter environment
- 1 - Loading required libraries
- 2 - Foreword on Pandas
- 3 - Reading data with Pandas
- 4 - Dealing with missing data
- 5 - Computing basic statistics
- 6 - Filtering
- 8 - GroupBy operations
- 9 - Joining different tables
- 10 - Visualization with Plotnine

9.4 o - Foreword, working in a jupyter environment

9.4.1 This is a markdown cell

With some features of the markdown syntax, such as:

- **bold** **bold**
- *italic* *italic*
- inline code

`inline code`

- [links](#) [links](<https://www.google.com/>)
- Images



- Latex code $y = ax + b$
 $y = ax + b$

```
print("This is a code cell in Python")
```

This is a code cell in Python

```
! echo "This is code cell in bash"
```

This is code cell in bash

```
%%bash
```

```
echo "This a multiline code cell"
echo "in bash"
```

This a multiline code cell
in bash

9.5 1 - Loading required libraries

```
import pandas as pd
import numpy as np
from plotnine import *

pd.__version__
'1.4.3'

np.__version__
'1.23.1'

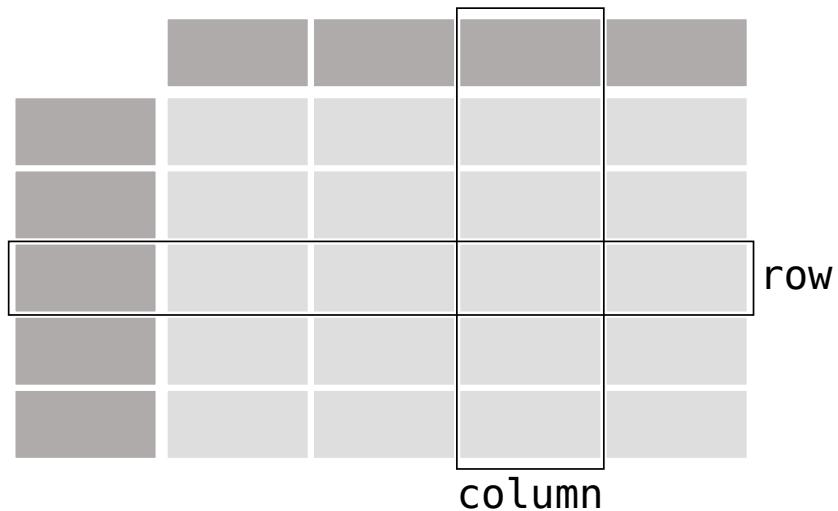
! conda list | grep plotnine

plotnine          0.9.0           pyhd8ed1ab_0    conda-forge
```

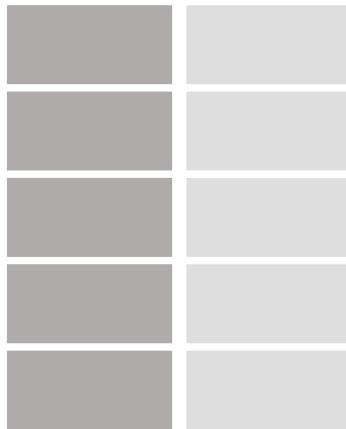
9.6 2 - Foreword on Pandas

9.6.1 Pandas terminology

DataFrame



Series



The pandas getting started tutorial: pandas.pydata.org/docs/getting_started

9.7 3 - Reading data with Pandas

```
sample_table_url = "https://raw.githubusercontent.com/SPAM-community/AncientMetagenomeD...
ancientmetagenome-hostassociated/samples/ancientmetagenome-hostassociated_samples.tsv"
library_table_url = "https://raw.githubusercontent.com/SPAM-community/AncientMetagenomeD...
ancientmetagenome-hostassociated/libraries/ancientmetagenome-hostassociated_libraries.ts...
```

Getting help in Python

```
help(pd.read_csv)
```

Help on function `read_csv` in module `pandas.io.parsers.readers`:

```
read_csv(filepath_or_buffer: 'FilePath | ReadCsvBuffer[bytes] | ReadCsvBuffer[str]' ,
sep=<no_default>, delimiter=None, header='infer', names=<no_default>, index_col=None,
usecols=None, squeeze=None, prefix=<no_default>, mangle_dupe_cols=True,
dtype: 'DtypeArg | None' = None, engine: 'CSVEngine | None' = None, converters=None,
true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0,
nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False,
skip_blank_lines=True, parse_dates=None, infer_datetime_format=False, keep_date_col=False,
date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None,
compression: 'CompressionOptions' = 'infer', thousands=None, decimal: 'str' = '.',
```

```
lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None,
comment=None, encoding=None, encoding_errors: 'str | None' = 'strict', dialect=None,
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None, storage_options: 'StorageOptions' = None)
Read a comma-separated values (csv) file into DataFrame.
```

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for
`IO Tools <https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html>`_.

Parameters

`filepath_or_buffer` : str, path object or file-like object

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is expected. A local file could be: file:///localhost/path/to/table.csv.

If you want to pass in a path object, pandas accepts any ``os.PathLike``.

By file-like object, we refer to objects with a ``read()`` method, such as a file handle (e.g. via builtin ``open`` function) or ``StringIO``.

`sep` : str, default ','

Delimiter to use. If `sep` is None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator by Python's builtin sniffer tool, ``csv.Sniffer``. In addition, separators longer than 1 character and different from ``'\s+'`` will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: ``'\r\t``.

`delimiter` : str, default ``None``

Alias for `sep`.

`header` : int, list of int, None, default 'infer'

Row number(s) to use as the column names, and the start of the data. Default behavior is to infer the column names: if no names are passed the behavior is identical to ``header=0`` and column names are inferred from the first line of the file, if column names are passed explicitly then the behavior is identical to ``header=None``. Explicitly pass ``header=0`` to be able to replace existing names. The header can be a list of integers that specify row locations for a multi-index on the columns e.g. [0,1,3]. Intervening rows that are not specified will be skipped (e.g. 2 in this example is skipped). Note that this parameter ignores commented lines and empty lines if ``skip_blank_lines=True``, so ``header=0`` denotes the first line of

```

    data rather than the first line of the file.
names : array-like, optional
    List of column names to use. If the file contains a header row,
    then you should explicitly pass ``header=0`` to override the column names.
    Duplicates in this list are not allowed.
index_col : int, str, sequence of int / str, or False, optional, default ``None``
    Column(s) to use as the row labels of the ``DataFrame``, either given as
    string name or column index. If a sequence of int / str is given, a
    MultiIndex is used.

Note: ``index_col=False`` can be used to force pandas to *not* use the first
column as the index, e.g. when you have a malformed file with delimiters at
the end of each line.
usecols : list-like or callable, optional
    Return a subset of the columns. If list-like, all elements must either
    be positional (i.e. integer indices into the document columns) or strings
    that correspond to column names provided either by the user in `names` or
    inferred from the document header row(s). If ``names`` are given, the document
    header row(s) are not taken into account. For example, a valid list-like
    `usecols` parameter would be ``[0, 1, 2]`` or ``['foo', 'bar', 'baz']``.
    Element order is ignored, so ``usecols=[0, 1]`` is the same as ``[1, 0]``.
    To instantiate a DataFrame from ``data`` with element order preserved use
    ``pd.read_csv(data, usecols=['foo', 'bar'])[['foo', 'bar']]`` for columns
    in ``['foo', 'bar']`` order or
    ``pd.read_csv(data, usecols=['foo', 'bar'])[['bar', 'foo']]``

If callable, the callable function will be evaluated against the column
names, returning names where the callable function evaluates to True. An
example of a valid callable argument would be ``lambda x: x.upper() in
['AAA', 'BBB', 'DDD']``. Using this parameter results in much faster
parsing time and lower memory usage.
squeeze : bool, default False
    If the parsed data only contains one column then return a Series.

    .. deprecated:: 1.4.0
    Append ``.squeeze("columns")`` to the call to ``read_csv`` to squeeze
    the data.
prefix : str, optional
    Prefix to add to column numbers when no header, e.g. 'X' for X0, X1, ...
    .. deprecated:: 1.4.0
    Use a list comprehension on the DataFrame's columns after calling ``read_csv``.
mangle_dupe_cols : bool, default True
    Duplicate columns will be specified as 'X', 'X.1', ... 'X.N', rather than
    'X'...'X'. Passing in False will cause data to be overwritten if there

```

```

        are duplicate names in the columns.

dtype : Type name or dict of column -> type, optional
    Data type for data or columns. E.g. {'a': np.float64, 'b': np.int32,
        'c': 'Int64'}
    Use `str` or `object` together with suitable `na_values` settings
        to preserve and not interpret dtype.
    If converters are specified, they will be applied INSTEAD
        of dtype conversion.

engine : {'c', 'python', 'pyarrow'}, optional
    Parser engine to use. The C and pyarrow engines are faster, while the python engine
    is currently more feature-complete. Multithreading is currently only supported by
        the pyarrow engine.

... versionadded:: 1.4.0

The "pyarrow" engine was added as an *experimental* engine, and some features
    are unsupported, or may not work correctly, with this engine.

converters : dict, optional
    Dict of functions for converting values in certain columns. Keys can either
        be integers or column labels.

true_values : list, optional
    Values to consider as True.

false_values : list, optional
    Values to consider as False.

skipinitialspace : bool, default False
    Skip spaces after delimiter.

skiprows : list-like, int or callable, optional
    Line numbers to skip (0-indexed) or number of lines to skip (int)
        at the start of the file.

If callable, the callable function will be evaluated against the row
    indices, returning True if the row should be skipped and False otherwise.
    An example of a valid callable argument would be ``lambda x: x in [0, 2]``.

skipfooter : int, default 0
    Number of lines at bottom of file to skip (Unsupported with engine='c').

nrows : int, optional
    Number of rows of file to read. Useful for reading pieces of large files.

na_values : scalar, str, list-like, or dict, optional
    Additional strings to recognize as NA/NaN. If dict passed, specific
    per-column NA values. By default the following values are interpreted as
    NaN: '', '#N/A', '#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN', '-nan',
        '1.#IND', '1.#QNAN', '<NA>', 'N/A', 'NA', 'NULL', 'NaN', 'n/a',
        'nan', 'null'.

keep_default_na : bool, default True
    Whether or not to include the default NaN values when parsing the data.

Depending on whether `na_values` is passed in, the behavior is as follows:

```

- * If `keep_default_na` is True, and `na_values` are specified, `na_values` is appended to the default NaN values used for parsing.
- * If `keep_default_na` is True, and `na_values` are not specified, only the default NaN values are used for parsing.
- * If `keep_default_na` is False, and `na_values` are specified, only the NaN values specified `na_values` are used for parsing.
- * If `keep_default_na` is False, and `na_values` are not specified, no strings will be parsed as NaN.

Note that if `na_filter` is passed in as False, the `keep_default_na` and `na_values` parameters will be ignored.

`na_filter : bool, default True`

Detect missing value markers (empty strings and the value of na_values). In data without any NAs, passing `na_filter=False` can improve the performance of reading a large file.

`verbose : bool, default False`

Indicate number of NA values placed in non-numeric columns.

`skip_blank_lines : bool, default True`

If True, skip over blank lines rather than interpreting as NaN values.

`parse_dates : bool or list of int or names or list of lists or dict, default False`
The behavior is as follows:

- * boolean. If True -> try parsing the index.
- * list of int or names. e.g. If [1, 2, 3] -> try parsing columns 1, 2, 3 each as a separate date column.
- * list of lists. e.g. If [[1, 3]] -> combine columns 1 and 3 and parse as a single date column.
- * dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call result 'foo'

If a column or index cannot be represented as an array of datetimes, say because of an unparsable value or a mixture of timezones, the column or index will be returned unaltered as an object data type. For non-standard datetime parsing, use ``pd.to_datetime`` after ``pd.read_csv``. To parse an index or column with a mixture of timezones, specify ``date_parser`` to be a partially-applied :func:`pandas.to_datetime` with ``utc=True``. See :ref:`io.csv.mixed_timezones` for more.

Note: A fast-path exists for iso8601-formatted dates.

`infer_datetime_format : bool, default False`

If True and `parse_dates` is enabled, pandas will attempt to infer the format of the datetime strings in the columns, and if it can be inferred, switch to a faster method of parsing them. In some cases this can increase the parsing speed by 5-10x.

```

keep_date_col : bool, default False
    If True and `parse_dates` specifies combining multiple columns then
        keep the original columns.
date_parser : function, optional
    Function to use for converting a sequence of string columns to an array of
    datetime instances. The default uses ``dateutil.parser.parser`` to do the
    conversion. Pandas will try to call `date_parser` in three different ways,
    advancing to the next if an exception occurs: 1) Pass one or more arrays
    (as defined by `parse_dates`) as arguments; 2) concatenate (row-wise) the
    string values from the columns defined by `parse_dates` into a single array
    and pass that; and 3) call `date_parser` once for each row using one or
    more strings (corresponding to the columns defined by `parse_dates`) as
    arguments.
dayfirst : bool, default False
    DD/MM format dates, international and European format.
cache_dates : bool, default True
    If True, use a cache of unique, converted dates to apply the datetime
    conversion. May produce significant speed-up when parsing duplicate
    date strings, especially ones with timezone offsets.

    .. versionadded:: 0.25.0
iterator : bool, default False
    Return TextFileReader object for iteration or getting chunks with
    ``get_chunk()``.

    .. versionchanged:: 1.2

        ``TextFileReader`` is a context manager.
chunksize : int, optional
    Return TextFileReader object for iteration.
    See the `IO Tools docs
<https://pandas.pydata.org/pandas-docs/stable/io.html#io-chunking>`_
    for more information on ``iterator`` and ``chunksize``.

    .. versionchanged:: 1.2

        ``TextFileReader`` is a context manager.
compression : str or dict, default 'infer'
    For on-the-fly decompression of on-disk data. If 'infer' and '%s' is
    path-like, then detect compression from the following extensions: '.gz',
    '.bz2', '.zip', '.xz', or '.zst' (otherwise no compression). If using
    'zip', the ZIP file must contain only one data file to be read in. Set to
    ``None`` for no decompression. Can also be a dict with key ``'method'`` set
    to one of {``'zip'``, ``'gzip'``, ``'bz2'``, ``'zstd'``} and other
    key-value pairs are forwarded to `` zipfile.ZipFile``, `` gzip.GzipFile``,
    `` bz2.BZ2File``, or `` zstandard.ZstdDecompressor``, respectively. As an

```

example, the following could be passed for Zstandard decompression using a custom compression dictionary:

```
``compression={'method': 'zstd', 'dict_data': my_compression_dict}``.
```

.. versionchanged:: 1.4.0 Zstandard support.

`thousands` : str, optional

Thousands separator.

`decimal` : str, default `.'

Character to recognize as decimal point (e.g. use `,' for European data).

`lineterminator` : str (length 1), optional

Character to break file into lines. Only valid with C parser.

`quotechar` : str (length 1), optional

The character used to denote the start and end of a quoted item. Quoted items can include the delimiter and it will be ignored.

`quoting` : int or csv.QUOTE_* instance, default 0

Control field quoting behavior per ``csv.QUOTE_*`` constants. Use one of QUOTE_MINIMAL (0), QUOTE_ALL (1), QUOTE_NONNUMERIC (2) or QUOTE_NONE (3).

`doublequote` : bool, default ``True``

When `quotechar` is specified and `quoting` is not ``QUOTE_NONE``, indicate whether or not to interpret two consecutive `quotechar` elements INSIDE a field as a single ```quotechar``` element.

`escapechar` : str (length 1), optional

One-character string used to escape other characters.

`comment` : str, optional

Indicates remainder of line should not be parsed. If found at the beginning of a line, the line will be ignored altogether. This parameter must be a single character. Like empty lines (as long as ```skip_blank_lines=True```), fully commented lines are ignored by the parameter `header` but not by

``skiprows``. For example, if ```comment='#'```, parsing

``#empty\na,b,c\n1,2,3`` with ```header=0``` will result in 'a,b,c' being treated as the header.

`encoding` : str, optional

Encoding to use for UTF when reading/writing (ex. 'utf-8'). `List of Python standard encodings

<<https://docs.python.org/3/library/codecs.html#standard-encodings>>`_.

.. versionchanged:: 1.2

When ```encoding``` is ``None`` , ```errors="replace"``` is passed to ```open()```. Otherwise, ```errors="strict"``` is passed to ```open()```. This behavior was previously only the case for ```engine="python"```.

.. versionchanged:: 1.3.0

```encoding_errors``` is a new argument. ```encoding``` has no longer an

influence on how encoding errors are handled.

```
encoding_errors : str, optional, default "strict"
 How encoding errors are treated. `List of possible values
 <>`_ .
 .. versionadded:: 1.3.0

dialect : str or csv.Dialect, optional
 If provided, this parameter will override values \(default or not\) for the
 following parameters: `delimiter`, `doublequote`, `escapechar`,
 `skipinitialspace`, `quotechar`, and `quoting`. If it is necessary to
 override values, a ParserWarning will be issued. See csv.Dialect
 documentation for more details.
error_bad_lines : bool, optional, default ``None``
 Lines with too many fields \(e.g. a csv line with too many commas\) will by
 default cause an exception to be raised, and no DataFrame will be returned.
 If False, then these "bad lines" will be dropped from the DataFrame that is
 returned.

 .. deprecated:: 1.3.0
 The ``on_bad_lines`` parameter should be used instead to specify behavior upon
 encountering a bad line instead.
warn_bad_lines : bool, optional, default ``None``
 If error_bad_lines is False, and warn_bad_lines is True, a warning for each
 "bad line" will be output.

 .. deprecated:: 1.3.0
 The ``on_bad_lines`` parameter should be used instead to specify behavior upon
 encountering a bad line instead.
on_bad_lines : {'error', 'warn', 'skip'} or callable, default 'error'
 Specifies what to do upon encountering a bad line \(a line with too many fields\).
 Allowed values are :
 - 'error', raise an Exception when a bad line is encountered.
 - 'warn', raise a warning when a bad line is encountered and skip that line.
 - 'skip', skip bad lines without raising or warning when they are encountered.

 .. versionadded:: 1.3.0

 - callable, function with signature
 ``bad_line: list\[str\] -> list\[str\] | None`` that will process a single
 bad line. ``bad_line`` is a list of strings split by the ``sep``.
 If the function returns ``None``, the bad line will be ignored.
 If the function returns a new list of strings with more elements than
 expected, a ``ParserWarning`` will be emitted while dropping extra elements.
```

```

Only supported when ``engine="python"``

.. versionadded:: 1.4.0

delim_whitespace : bool, default False
 Specifies whether or not whitespace (e.g. ``' '`` or ``' '``) will be
 used as the sep. Equivalent to setting ``sep='\\s+'``. If this option
 is set to True, nothing should be passed in for the ``delimiter``
 parameter.

low_memory : bool, default True
 Internally process the file in chunks, resulting in lower memory use
 while parsing, but possibly mixed type inference. To ensure no mixed
 types either set False, or specify the type with the `dtype` parameter.
 Note that the entire file is read into a single DataFrame regardless,
 use the `chunksize` or `iterator` parameter to return the data in chunks.
 (Only valid with C parser).

memory_map : bool, default False
 If a filepath is provided for `filepath_or_buffer`, map the file object
 directly onto memory and access the data directly from there. Using this
 option can improve performance because there is no longer any I/O overhead.

float_precision : str, optional
 Specifies which converter the C engine should use for floating-point
 values. The options are ``None`` or 'high' for the ordinary converter,
 'legacy' for the original lower precision pandas converter, and
 'round_trip' for the round-trip converter.

.. versionchanged:: 1.2

storage_options : dict, optional
 Extra options that make sense for a particular storage connection, e.g.
 host, port, username, password, etc. For HTTP(S) URLs the key-value pairs
 are forwarded to ``urllib`` as header options. For other URLs (e.g.
 starting with "s3://", and "gcs://") the key-value pairs are forwarded to
 ``fsspec``. Please see ``fsspec`` and ``urllib`` for more details.

.. versionadded:: 1.2

>Returns

DataFrame or TextParser
 A comma-separated values (csv) file is returned as two-dimensional
 data structure with labeled axes.

See Also

DataFrame.to_csv : Write DataFrame to a comma-separated values (csv) file.

```

```
read_csv : Read a comma-separated values (csv) file into DataFrame.
read_fwf : Read a table of fixed-width formatted lines into DataFrame.
```

#### Examples

-----

```
>>> pd.read_csv('data.csv') # doctest: +SKIP
```

```
sample_df = pd.read_csv(sample_table_url, sep="\t")
library_df = pd.read_csv(library_table_url, sep="\t")
```

```
sample_df.project_name.nunique()
```

45

```
library_df.project_name.nunique()
```

43

### 9.7.1 Listing the columns of the sample dataframe

```
sample_df.columns
```

```
Index(['project_name', 'publication_year', 'publication_doi', 'site_name',
 'latitude', 'longitude', 'geo_loc_name', 'sample_name', 'sample_host',
 'sample_age', 'sample_age_doi', 'community_type', 'material', 'archive',
 'archive_project', 'archive_acquisition'],
 dtype='object')
```

### 9.7.2 Looking at the data type of the sample dataframe

```
sample_df.dtypes
```

project_name	object
publication_year	int64
publication_doi	object
site_name	object
latitude	float64
longitude	float64
geo_loc_name	object
sample_name	object
sample_host	object
sample_age	int64
sample_age_doi	object
community_type	object

```
material object
archive object
archive_project object
archive_acquisition object
dtype: object
```

- int64 is for integers
- floating64 is for floating point precision numbers, also known as double in some other programming languages
- object is a general type in pandas for everything that is not a number, interval, categorical, or date

### 9.7.3 Let's inspect our data

What is the size of our dataframe ?

```
sample_df.shape
```

```
(1060, 16)
```

This dataframe has **1060** rows, and **16** columns

Let's look at the first 5 rows

```
sample_df.head()
```

```
project_name
publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
```

```
archive_project
archive_accession
o
Warinner2014
2014
10.1038/ng.2906
Dalheim
51.565
8.840
Germany
B61
Homo sapiens
900
10.1038/ng.2906
oral
dental calculus
SRA
PRJNA216965
SRS473742,SRS473743,SRS473744,SRS473745
1
Warinner2014
2014
10.1038/ng.2906
Dalheim
51.565
8.840
Germany
G12
Homo sapiens
900
10.1038/ng.2906
```

oral  
dental calculus  
SRA  
PRJNA216965  
SRS473747,SRS473746,SRS473748,SRS473749,SRS473750  
2  
Weyrich2017  
2017  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
Gola Forest  
7.657  
-10.841  
Sierra Leone  
Chimp  
Pan troglodytes  
100  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
oral  
dental calculus  
SRA  
PRJNA685265  
SRS7890499  
3  
Weyrich2017  
2017  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
El Sidrón Cave  
43.386  
-5.328  
Spain  
ElSidron1

Homo sapiens neanderthalensis

49000

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890498

4

Weyrich2017

2017

10.1038/nature21674

El Sidrón Cave

43.386

-5.329

Spain

ElSidron2

Homo sapiens neanderthalensis

49000

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890496

**Unlike R, Python is 0 based language, meaning the first element is of index 0, not like R where it is 1.**

Let's look at the last 5 rows

```
sample_df.tail()
```

```
project_name
```

publication\_year  
publication\_doi  
site\_name  
latitude  
longitude  
geo\_loc\_name  
sample\_name  
sample\_host  
sample\_age  
sample\_age\_doi  
community\_type  
material  
archive  
archive\_project  
archive\_acquisition  
1055  
Kazarina2021b  
2021  
[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)  
St. Gertrude's Church, Riga  
56.958  
24.121  
Latvia  
T2  
Homo sapiens  
400  
[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)  
oral  
tooth  
ENA  
PRJEB47251

ERS7283094,ERS7283095

1056

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Gertrude's Church, Riga

56.958

24.121

Latvia

T<sub>3</sub>

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283096,ERS7283097

1057

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Gertrude's Church, Riga

56.958

24.121

Latvia

T<sub>9</sub>

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283098, ERS7283099

1058

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

Dom Square, Riga

56.949

24.104

Latvia

TZA3

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283100, ERS7283101

1059

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Peter's Church, Riga

56.947

24.109

Latvia

TZA4

Homo sapiens

```
500
10.1016/j.jasrep.2021.103213
oral
tooth
ENA
PRJEB47251
ERS7283102,ERS7283103
Let's randomly inspect 5 rows
```

```
sample_df.sample(n=5)

project_name
publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
archive_project
archive_accession
413
Neukamm2020
2020
10.1186/s12915-020-00839-8
Abusir el-Meleq
```

29.240  
31.100  
Egypt  
Abusir1576  
Homo sapiens  
2200  
10.1038/ncomms15694  
skeletal tissue  
bone  
ENA  
PRJEB33848  
ERS3635981  
754  
Rampelli2021  
2021  
10.1038/s42003-021-01689-y  
El Salt  
38.687  
-0.508  
Spain  
V3  
Homo sapiens neanderthalensis  
44700  
10.1038/s42003-021-01689-y  
gut  
sediment  
ENA  
PRJEB41665  
ERS5428042  
436  
Neukamm2020

2020

10.1186/s12915-020-00839-8

Abusir el-Meleq

29.240

31.100

Egypt

Abusir1606

Homo sapiens

2600

10.1186/s12915-020-00839-8

skeletal tissue

bone

ENA

PRJEB33848

ERS3635928

474

Neukamm2020

2020

10.1186/s12915-020-00839-8

Abusir el-Meleq

29.240

31.100

Egypt

Abusir1654

Homo sapiens

2300

10.1038/ncomms15694

oral

tooth

ENA

PRJEB33848

ERS3635960  
 573  
 Philips2017  
 2017  
 10.1186/s12864-020-06810-9  
 Kowalewko  
 52.699  
 17.605  
 Poland  
 PCAoo4o  
 Homo sapiens  
 1900  
 10.1186/s12864-020-06810-9  
 oral  
 tooth  
 SRA  
 PRJNA354503  
 SRS1815407

### Accessing the data by index/columns

The are different way of selecting of subset of a dataframe

Selecting by the row index

```
selecting the 10th row, and all columns
sample_df.iloc[9, :]
```

project_name	Weyrich2017
publication_year	2017
publication_doi	10.1038/nature21674
site_name	Stuttgart-Mühlhausen I
latitude	48.839
longitude	9.227
geo_loc_name	Germany
sample_name	EuroLBK1
sample_host	Homo sapiens
sample_age	7400

```
sample_age_doi 10.1038/nature21674
community_type oral
material dental calculus
archive SRA
archive_project PRJNA685265
archive_accession SRS7890488
Name: 9, dtype: object

selecting the 10th to 12th row, and all columns
sample_df.iloc[9:12, :]

project_name
publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
archive_project
archive_accession

9
Weyrich2017
2017
10.1038/nature21674
Stuttgart-Mühlhausen I
48.839
9.227
```

Germany

EuroLBK1

Homo sapiens

7400

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890488

10

Weyrich2017

2017

10.1038/nature21674

Stuttgart-Mühlhausen I

48.839

9.227

Germany

EuroLBK2

Homo sapiens

7400

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890485

11

Weyrich2017

2017

10.1038/nature21674

```
Stuttgart-Mühlhausen I
```

```
48.839
```

```
9.227
```

```
Germany
```

```
EuroLBK3
```

```
Homo sapiens
```

```
7400
```

```
10.1038/nature21674
```

```
oral
```

```
dental calculus
```

```
SRA
```

```
PRJNA685265
```

```
SRS7890490
```

```
selecting the 10th to 12th row, and the first to the 4th column
sample_df.iloc[9:12, 0:4]
```

```
project_name
```

```
publication_year
```

```
publication_doi
```

```
site_name
```

```
9
```

```
Weyrich2017
```

```
2017
```

```
10.1038/nature21674
```

```
Stuttgart-Mühlhausen I
```

```
10
```

```
Weyrich2017
```

```
2017
```

```
10.1038/nature21674
```

```
Stuttgart-Mühlhausen I
```

```
11
```

```
Weyrich2017
```

```
2017
```

```
10.1038/nature21674
```

```
Stuttgart-Mühlhausen I
```

```
selecting the column site_name
sample_df['site_name']
```

```
0 Dalheim
1 Dalheim
2 Gola Forest
3 El Sidrón Cave
4 El Sidrón Cave
...
1055 St. Gertrude's Church, Riga
1056 St. Gertrude's Church, Riga
1057 St. Gertrude's Church, Riga
1058 Dom Square, Riga
1059 St. Peter's Church, Riga
Name: site_name, Length: 1060, dtype: object
```

```
Also valid, but less preferred
sample_df.site_name
```

```
0 Dalheim
1 Dalheim
2 Gola Forest
3 El Sidrón Cave
4 El Sidrón Cave
...
1055 St. Gertrude's Church, Riga
1056 St. Gertrude's Church, Riga
1057 St. Gertrude's Church, Riga
1058 Dom Square, Riga
1059 St. Peter's Church, Riga
Name: site_name, Length: 1060, dtype: object
```

```
Removing a row
sample_df.drop(0)
```

```
project_name
```

```
publication_year
```

```
publication_doi
```

```
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
archive_project
archive_accession
1
Warinner2014
2014
10.1038/ng.2906
Dalheim
51.565
8.840
Germany
G12
Homo sapiens
900
10.1038/ng.2906
oral
dental calculus
SRA
PRJNA216965
SRS473747,SRS473746,SRS473748,SRS473749,SRS473750
2
```

Weyrich2017

2017

10.1038/nature21674

Gola Forest

7.657

-10.841

Sierra Leone

Chimp

Pan troglodytes

100

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890499

3

Weyrich2017

2017

10.1038/nature21674

El Sidrón Cave

43.386

-5.328

Spain

ElSidron1

Homo sapiens neanderthalensis

49000

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265  
SRS7890498  
4  
Weyrich2017  
2017  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
El Sidrón Cave  
43.386  
-5.329  
Spain  
ElSidron2  
Homo sapiens neanderthalensis  
49000  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
oral  
dental calculus  
SRA  
PRJNA685265  
SRS7890496  
5  
Weyrich2017  
2017  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
Spy Cave  
50.480  
4.674  
Belgium  
Spy1  
Homo sapiens neanderthalensis  
35800  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)

oral

dental calculus

SRA

PRJNA685265

SRS7890491

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

1055

Kazarina2021b

2021

[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)

St. Gertrude's Church, Riga

56.958

24.121

Latvia

T2

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283094, ERS7283095

1056

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Gertrude's Church, Riga

56.958

24.121

Latvia

T<sub>3</sub>

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283096, ERS7283097

1057

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Gertrude's Church, Riga

56.958

24.121

Latvia

T<sub>9</sub>

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283098, ERS7283099

1058

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

Dom Square, Riga

56.949

24.104

Latvia

TZA<sub>3</sub>

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283100, ERS7283101

1059

Kazarina2021b

2021

```
10.1016/j.jasrep.2021.103213
```

```
St. Peter's Church, Riga
```

```
56.947
```

```
24.109
```

```
Latvia
```

```
TZA4
```

```
Homo sapiens
```

```
500
```

```
10.1016/j.jasrep.2021.103213
```

```
oral
```

```
tooth
```

```
ENA
```

```
PRJEB47251
```

```
ERS7283102,ERS7283103
```

```
1059 rows × 16 columns
```

```
Removing a column
sample_df.drop('project_name', axis=1)

publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
```

```
archive_project
archive_acquisition
o
2014
10.1038/ng.2906
Dalheim
51-565
8.840
Germany
B61
Homo sapiens
900
10.1038/ng.2906
oral
dental calculus
SRA
PRJNA216965
SRS473742,SRS473743,SRS473744,SRS473745
1
2014
10.1038/ng.2906
Dalheim
51-565
8.840
Germany
G12
Homo sapiens
900
10.1038/ng.2906
oral
dental calculus
```

SRA  
PRJNA216965  
SRS473747,SRS473746,SRS473748,SRS473749,SRS473750  
2  
2017  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
Gola Forest  
7.657  
-10.841  
Sierra Leone  
Chimp  
Pan troglodytes  
100  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
oral  
dental calculus  
SRA  
PRJNA685265  
SRS7890499  
3  
2017  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
El Sidrón Cave  
43.386  
-5.328  
Spain  
ElSidron1  
Homo sapiens neanderthalensis  
49000  
[10.1038/nature21674](https://doi.org/10.1038/nature21674)  
oral

dental calculus

SRA

PRJNA685265

SRS7890498

4

2017

10.1038/nature21674

El Sidrón Cave

43.386

-5.329

Spain

ElSidron2

Homo sapiens neanderthalensis

49000

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890496

...

...

...

...

...

...

...

...

...

...

...

...  
...  
...  
...  
...  
**1055**  
**2021**  
[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)  
St. Gertrude's Church, Riga  
56.958  
**24.121**  
Latvia  
T2  
Homo sapiens  
**400**  
[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)  
oral  
tooth  
ENA  
PRJEB47251  
ERS7283094, ERS7283095  
**1056**  
**2021**  
[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)  
St. Gertrude's Church, Riga  
56.958  
**24.121**  
Latvia  
T3  
Homo sapiens  
**400**

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283096, ERS7283097

1057

2021

10.1016/j.jasrep.2021.103213

St. Gertrude's Church, Riga

56.958

24.121

Latvia

T9

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283098, ERS7283099

1058

2021

10.1016/j.jasrep.2021.103213

Dom Square, Riga

56.949

24.104

Latvia

TZA3

Homo sapiens

```
400
10.1016/j.jasrep.2021.103213
oral
tooth
ENA
PRJEB47251
ERS7283100,ERS7283101
1059
2021
10.1016/j.jasrep.2021.103213
St. Peter's Church, Riga
56.947
24.109
Latvia
TZA4
Homo sapiens
500
10.1016/j.jasrep.2021.103213
oral
tooth
ENA
PRJEB47251
ERS7283102,ERS7283103
1060 rows × 15 columns
```

#### 4 - Dealing with missing data

Checking is some entries if the table have missing data (NA or NaN)

```
sample_df.isna()
project_name
publication_year
```





False

False

False

3

False

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

1055

False

False

False

False

False

False

False

```
False
False
False
False
False
False
False
False
False
1056
False
1057
False
False
False
False
```

False

```
making the sum by row - axis=1
sample_df.isna().sum(axis=1)
```

```
0 0
1 0
2 0
3 0
4 0
 ..
1055 0
1056 0
1057 0
1058 0
1059 0
Length: 1060, dtype: int64
```

Sorting by decreasing order to check which rows have missing values

```
sample df.isna().sum(axis=1).sort_values(ascending=False)
```

```

800 2
962 2
992 2
801 2
802 2
...
362 0
363 0
364 0
365 0
1059 0
Length: 1060, dtype: int64

```

```

sample_df.iloc[800, :]

project_name FellowsYates2021
publication_year 2021
publication_doi 10.1073/pnas.2021655118
site_name Not specified
latitude NaN
longitude NaN
geo_loc_name Democratic Republic of the Congo
sample_name GDC002.A
sample_host Gorilla gorilla gorilla
sample_age 200
sample_age_doi 10.1073/pnas.2021655118
community_type oral
material dental calculus
archive ENA
archive_project PRJEB34569
archive_accession ERS3774403
Name: 800, dtype: object

```

What to do now ? The ideal scenario would be to correct or impute the data. However, sometimes, the only thing we can do is remove the row with missing data, with the `.dropna()` function.  
Here, we're just going to ignore them, and deal with it individually if necessary

## 9.8 5 - Computing basic statistics

TLDR: use the `describe()` function, the equivalent of `summarize` in R

```

sample_df.describe()

publication_year

```

```
latitude
longitude
sample_age
count
1060.000000
1021.000000
1021.000000
1060.000000
mean
2019.377358
40.600493
3.749624
3588.443396
std
1.633877
18.469421
43.790316
9862.416855
min
2014.000000
-34.030000
-121.800000
100.000000
25%
2018.000000
29.240000
-1.257000
200.000000
50%
2020.000000
45.450000
```

```
14.381000
1000.000000
75%
2021.000000
52.699000
23.892000
2200.000000
max
2021.000000
79.000000
159.346000
102000.000000
```

Let's look at various individual summary statistics. We can run them on the whole dataframe (for int or float columns), or on a subset of columns

```
sample_df.mean()

/var/folders/1c/l1qb09f15jddsh65f6xv1n_r0000gp/T/ipykernel_69168/2260452167.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
```

```
publication_year 2019.377358
latitude 40.600493
longitude 3.749624
sample_age 3588.443396
dtype: float64
```

```
sample_df['publication_year'].describe()

count 1060.000000
mean 2019.377358
std 1.633877
min 2014.000000
25% 2018.000000
```

```
50% 2020.000000
75% 2021.000000
max 2021.000000
Name: publication_year, dtype: float64

The average publication year
sample_df['publication_year'].mean()

2019.377358490566

The median publication year
sample_df['publication_year'].median()

2020.0

The minimum, or oldest publication year
sample_df['publication_year'].min()

2014

The maximum, or most recent publication year
sample_df['publication_year'].max()

2021

The number of sites
sample_df['site_name'].nunique()

246

The number of samples from the different hosts
sample_df['sample_host'].value_counts()

Homo sapiens 741
Ursus arctos 85
Ambrosia artemisiifolia 46
Arabidopsis thaliana 34
Homo sapiens neanderthalensis 32
Pan troglodytes schweinfurthii 26
Gorilla beringei beringei 15
Canis lupus 12
Gorilla gorilla gorilla 8
Mammuthus primigenius 8
Pan troglodytes verus 7
Rangifer tarandus 6
```

```

Gorilla beringei graueri 6
Pan troglodytes ellioti 6
Papio hamadryas 5
Alouatta palliata 5
Conepatus chinga 4
Gerbilliscus boehmi 4
Strigocuscus celebensis 4
Papio anubis 2
Gorilla beringei 2
Papio sp. 1
Pan troglodytes 1
Name: sample_host, dtype: int64

```

```

The quantile of the publication years
sample_df['publication_year'].quantile(np.arange(0,1,0.1))

```

```

0.0 2014.0
0.1 2017.0
0.2 2018.0
0.3 2018.0
0.4 2020.0
0.5 2020.0
0.6 2020.0
0.7 2021.0
0.8 2021.0
0.9 2021.0
Name: publication_year, dtype: float64

```

```

We can also visualize it with built-in plot functions of pandas
sample_df['publication_year'].plot.hist()

```

```
<AxesSubplot:ylabel='Frequency'>
```

## 9.9 6 - Filtering

There are different ways of filtering data with Pandas:

- The **classic** method with bracket indexing/subsetting
- The `query()` method

The classic method

```

Getting all the publications before 2015
sample_df[sample_df['publication_year'] < 2015]

```

project\_name

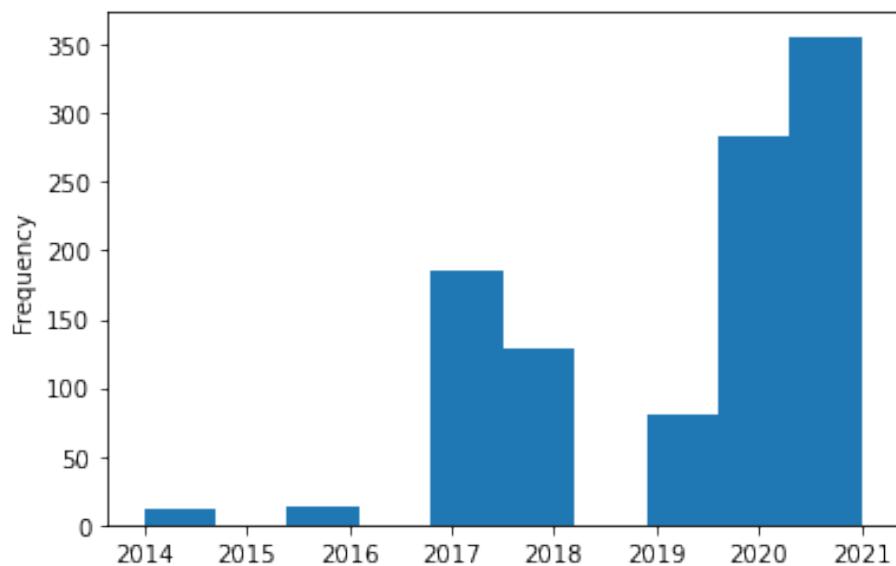


Figure 9.2: png

publication\_year  
publication\_doi  
site\_name  
latitude  
longitude  
geo\_loc\_name  
sample\_name  
sample\_host  
sample\_age  
sample\_age\_doi  
community\_type  
material  
archive  
archive\_project  
archive\_accession  
o

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

B61

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473742,SRS473743,SRS473744,SRS473745

1

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

G12

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473747,SRS473746,SRS473748,SRS473749,SRS473750

272

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP4

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428959

273

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP10

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428961

274

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP18

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428962

275

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP37

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428963

276

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP9

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428960

277

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46  
Mexico  
TP48  
*Homo sapiens*  
400  
10.7183/1045-6635.23.4.467  
skeletal tissue  
bone  
SRA  
PRJNA205039  
SRS428964  
278  
Campana2014  
2014  
10.1186/1756-0500-7-111  
Teposcolula Yucundaa  
17.490  
-97.46  
Mexico  
TPo2,TP10,TP15,TP26  
*Homo sapiens*  
400  
10.7183/1045-6635.23.4.467  
skeletal tissue  
bone  
SRA  
PRJNA205039  
SRS428958  
279  
Campana2014  
2014

10.1186/1756-0500-7-111  
Teposcolula Yucundaa  
17.490  
-97.46  
Mexico  
TP32,TP42,TP45,TP48  
Homo sapiens  
400  
10.7183/1045-6635.23.4.467  
skeletal tissue  
bone  
SRA  
PRJNA205039  
SRS428972  
500  
Appelt2014  
2014  
10.1128/AEM.03242-13  
Place d'Armes, Namur  
50.460  
4.86  
Belgium  
4.453  
Homo sapiens  
600  
10.1128/AEM.03242-13  
gut  
palaeofaeces  
SRA  
PRJNA230469  
SRS510175

```
Getting all the publications before 2015, only in the Northern hemisphere
sample_df[(sample_df['publication_year'] < 2015) & (sample_df['longitude'] > 0)]
```

project\_name  
publication\_year  
publication\_doi  
site\_name  
latitude  
longitude  
geo\_loc\_name  
sample\_name  
sample\_host  
sample\_age  
sample\_age\_doi  
community\_type  
material  
archive  
archive\_project  
archive\_accession  
o  
Warinner2014  
2014  
10.1038/ng.2906  
Dalheim  
51.565  
8.84  
Germany  
B61  
Homo sapiens  
900  
10.1038/ng.2906

oral  
dental calculus  
SRA  
PRJNA216965  
SRS473742,SRS473743,SRS473744,SRS473745  
1  
Warinner2014  
2014  
10.1038/ng.2906  
Dalheim  
51.565  
8.84  
Germany  
G12  
Homo sapiens  
900  
10.1038/ng.2906  
oral  
dental calculus  
SRA  
PRJNA216965  
SRS473747,SRS473746,SRS473748,SRS473749,SRS473750  
500  
Appelt2014  
2014  
10.1128/AEM.03242-13  
Place d'Armes, Namur  
50.460  
4.86  
Belgium  
4.453

Homo sapiens  
600  
10.1128/AEM.03242-13  
gut  
palaeofaeces  
SRA  
PRJNA230469  
SRS510175

This syntax can rapidly become quite cumbersome, which is why we can also use the query() method

```
Getting all the publications before 2015
sample_df.query("publication_year < 2015")

project_name
publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
archive_project
archive_accession
o
Warinner2014
```

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

B61

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473742,SRS473743,SRS473744,SRS473745

1

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

G12

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473747,SRS473746,SRS473748,SRS473749,SRS473750

272

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP4

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428959

273

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP10

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone  
SRA  
PRJNA205039  
SRS428961  
274  
Campana2014  
2014  
10.1186/1756-0500-7-111  
Teposcolula Yucundaa  
17.490  
-97.46  
Mexico  
TP18  
Homo sapiens  
400  
10.7183/1045-6635.23.4.467  
skeletal tissue  
bone  
SRA  
PRJNA205039  
SRS428962  
275  
Campana2014  
2014  
10.1186/1756-0500-7-111  
Teposcolula Yucundaa  
17.490  
-97.46  
Mexico  
TP37  
Homo sapiens

400  
10.7183/1045-6635.23.4.467  
skeletal tissue  
bone  
SRA  
PRJNA205039  
SRS428963  
276  
Campana2014  
2014  
10.1186/1756-0500-7-111  
Teposcolula Yucundaa  
17.490  
-97.46  
Mexico  
TP9  
Homo sapiens  
400  
10.7183/1045-6635.23.4.467  
skeletal tissue  
bone  
SRA  
PRJNA205039  
SRS428960  
277  
Campana2014  
2014  
10.1186/1756-0500-7-111  
Teposcolula Yucundaa  
17.490  
-97.46

Mexico

TP48

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428964

278

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TPo2,TP10,TP15,TP26

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428958

279

Campana2014

2014

10.1186/1756-0500-7-111

Teposcolula Yucundaa

17.490

-97.46

Mexico

TP32,TP42,TP45,TP48

Homo sapiens

400

10.7183/1045-6635.23.4.467

skeletal tissue

bone

SRA

PRJNA205039

SRS428972

500

Appelt2014

2014

10.1128/AEM.03242-13

Place d'Armes, Namur

50.460

4.86

Belgium

4.453

Homo sapiens

600

10.1128/AEM.03242-13

gut

palaeofaeces

SRA

PRJNA230469

SRS510175

```
Getting all the publications before 2015, only the Northern hemisphere
sample_df.query("publication_year < 2015 and longitude > 0 ")

project_name
publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
archive_project
archive_acquisition
o
Warinner2014
2014
10.1038/ng.2906
Dalheim
51565
8.84
Germany
B61
Homo sapiens
900
10.1038/ng.2906
```

oral

dental calculus

SRA

PRJNA216965

SRS473742,SRS473743,SRS473744,SRS473745

1

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

G12

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473747,SRS473746,SRS473748,SRS473749,SRS473750

500

Appelt2014

2014

10.1128/AEM.03242-13

Place d'Armes, Namur

50.460

4.86

Belgium

4.453

Homo sapiens  
600  
10.1128/AEM.03242-13  
gut  
palaeofaeces  
SRA  
PRJNA230469  
SRS510175

## 9.10 7 - GroupBy operations, and computing statistics on grouped values

The “`groupBy`” operation, as the name suggests, allows us to group values by a grouping key, and perform a groupwise operation.

For example, we can group by the `sample_host` and get the age of the **youngest** sample in each group

```
sample_df.groupby("sample_host")['sample_age'].min()
```

sample_host	sample_age
Alouatta palliata	200
Ambrosia artemisiifolia	100
Arabidopsis thaliana	100
Canis lupus	400
Conepatus chinga	100
Gerbilliscus boehmi	100
Gorilla beringei	100
Gorilla beringei beringei	200
Gorilla beringei graueri	200
Gorilla gorilla gorilla	200
Homo sapiens	100
Homo sapiens neanderthalensis	35800
Mammuthus primigenius	41800
Pan troglodytes	100
Pan troglodytes elliotti	200
Pan troglodytes schweinfurthii	100
Pan troglodytes verus	200
Papio anubis	100
Papio hamadryas	100
Papio sp.	100
Rangifer tarandus	100

```
Strigocuscus celebensis 100
Ursus arctos 100
Name: sample_age, dtype: int64
```

Here `min()` is a so-called aggregation function

Notice that `.value_counts()` is actually a special case of `.groupby()`

```
sample_df.groupby("sample_host")["sample_host"].count()
```

sample_host	
Alouatta palliata	5
Ambrosia artemisiifolia	46
Arabidopsis thaliana	34
Canis lupus	12
Conepatus chinga	4
Gerbilliscus boehmi	4
Gorilla beringei	2
Gorilla beringei beringei	15
Gorilla beringei graueri	6
Gorilla gorilla gorilla	8
Homo sapiens	741
Homo sapiens neanderthalensis	32
Mammuthus primigenius	8
Pan troglodytes	1
Pan troglodytes elliotti	6
Pan troglodytes schweinfurthii	26
Pan troglodytes verus	7
Papio anubis	2
Papio hamadryas	5
Papio sp.	1
Rangifer tarandus	6
Strigocuscus celebensis	4
Ursus arctos	85
Name: sample_host, dtype: int64	

## 9.11 8 - Reshaping data, from wide to long and back

### Wide data format

A	B	C
1.1	4.2	5.6
1.0	4.5	5.8

### Tidy data format

Condition	Value
A	1.1
	1.0
B	4.2
	4.5
C	5.6
	5.8

#### 9.11.1 From wide to long/tidy

The tidy format, or long format idea is that one column = one kind of data. Unfortunately for this tutorial, the AncientMetagenomeDir tables are already in the tidy format (good), so we'll see an example of the wide format just below

```
wide_df = pd.DataFrame(
 [
 [150, 155, 157, 160],
 [149, 153, 154, 155]
])
```

```
, index = ['John', 'Jack']
, columns = [1991, 1992, 1993, 1994]
).rename_axis('individual').reset_index()
wide_df
```

individual

1991

1992

1993

1994

o

John

150

155

157

160

1

Jack

149

153

154

155

In this hypothetic dataframe, we have the years as column, the individual as index, and their height as value.

We'll reformat to the tidy/long format using the `.melt()` function

```
tidy_df = wide_df.melt(id_vars='individual', var_name='birthyear', value_name='height')
```

individual

birthyear

height

o

John

1991

150

1

Jack

1991

149

2

John

1992

155

3

Jack

1992

153

4

John

1993

157

5

Jack

1993

154

6

John

1994

160

7

Jack

1994

155

Bonus: How to deal with a dataframe with the kind of data indicated in the column name, typically like so

```
wide_df = pd.DataFrame(
 [
 [150, 155, 157, 160],
 [149, 153, 154, 155]
]
, index = ['John', 'Jack'])
, columns = ["year-1991", "year-1992", "year-1993", "year-1994"]
).rename_axis('individual').reset_index()
wide_df

individual
year-1991
year-1992
year-1993
year-1994
o
John
150
155
157
160
1
Jack
149
153
154
155
pd.wide_to_long(wide_df, ['year'], i='individual', j='birthyear', sep="-").rename(columns={'year':
height
individual
birthyear
```

John

1991

150

Jack

1991

149

John

1992

155

Jack

1992

153

John

1993

157

Jack

1993

154

John

1994

160

Jack

1994

155

### 9.11.2 From long/tidy to wide format using the `.pivot()` function.

```
tidy_df.pivot(index='individual', columns='birthyear', values='height')
```

```
/Users/maxime/mambaforge/envs/intro-data/lib/python3.10/site-packages/pandas/core/algorith
```

```
birthyear
```

```
1991
```

```
1992
```

```
1993
```

```
1994
```

```
individual
```

```
Jack
```

```
149
```

```
153
```

```
154
```

```
155
```

```
John
```

```
150
```

```
155
```

```
157
```

```
160
```

## 9.12 9 - Joining two different tables

In AncientMetagenomeDir, the information about each sample is located in sample table, and about the library in the library table.

To match these two together, we need to join the tables together.

To do so, we need a column in common between the two tables, the so-called **joining key** (this key can be the index)



For the samples and libraries dataframes, the joining key is the column `sample_name`

```
sample_df.merge(library_df, on='sample_name').columns
Index(['project_name_x', 'publication_year_x', 'publication_doi', 'site_name',
 'latitude', 'longitude', 'geo_loc_name', 'sample_name', 'sample_host',
 'sample_age', 'sample_age_doi', 'community_type', 'material',
 'archive_x', 'archive_project_x', 'archive_accession', 'project_name_y'],
```

```
'publication_year_y', 'data_publication_doi', 'archive_y',
'archive_project_y', 'archive_sample_accession', 'library_name',
'strand_type', 'library_polymerase', 'library_treatment',
'library_concentration', 'instrument_model', 'library_layout',
'library_strategy', 'read_count', 'archive_data_accession',
'download_links', 'download_md5s', 'download_sizes'],
dtype='object')
```

We have some duplicate columns that we can get rid of:

```
merged_df = sample_df.merge(library_df.drop(['project_name', 'publication_year', 'archiv
merged_df

project_name
publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
...
library_treatment
library_concentration
instrument_model
library_layout
library_strategy
read_count
archive_data_accession
download_links
download_md5s
download_sizes
o
```

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

B61

Homo sapiens

900

...

none

NaN

Illumina HiSeq 2000

SINGLE

WGS

13228381

SRR957738

[ftp://sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957738/...](ftp://sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957738/)

9c4oc43b5d455e760ae8db924347fob2

953396663

1

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

B61

Homo sapiens

900  
...  
none  
NaN  
Illumina HiSeq 2000  
SINGLE  
WGS  
13260566  
SRR957739  
[ftp.sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957739/...](ftp.sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957739/)  
dec1507f742de109529638bfooe0732f  
1026825795  
2  
Warinner2014  
2014  
<10.1038/ng.2906>  
Dalheim  
51-565  
8.84  
Germany  
B61  
Homo sapiens  
900  
...  
none  
NaN  
Illumina HiSeq 2000  
SINGLE  
WGS  
8869866  
SRR957740

ftp.sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957740/...

bc49c59f489b4009206f8abcb737d55d

661500786

3

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

B61

Homo sapiens

900

...

none

NaN

Illumina HiSeq 2000

SINGLE

WGS

11275013

SRR957741

ftp.sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957741/...

eo2e3549ddd3ba6dc278a7f573c07321

877360302

4

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565



...  
...  
...  
...  
...  
...  
...  
...  
1802  
Maixner2021  
2021  
[10.1016/j.cub.2021.09.031](https://doi.org/10.1016/j.cub.2021.09.031)  
Edlersbergwerk - oben, Hallstatt  
47.56°  
13.63  
Austria  
2612  
Homo sapiens  
150  
...  
none  
NaN  
Illumina MiSeq  
PAIRED  
WGS  
1858404  
ERR5766179  
[ftp.sra.ebi.ac.uk/vol1/fastq/ERR576/009/ERR576...](ftp://sra.ebi.ac.uk/vol1/fastq/ERR576/009/ERR576...)  
542787c645boaebe15c66cc926d3f69;obc58d56be3c3...  
86783041;98100690  
1803  
Maixner2021

2021

10.1016/j.cub.2021.09.031

Edlersbergwerk - oben, Hallstatt

47.56°

13.63

Austria

2612

Homo sapiens

150

...

none

NaN

Illumina MiSeq

PAIRED

WGS

1603064

ERR5766180

ftp.sra.ebi.ac.uk/vol1/fastq/ERR576/000/ERR576...

022bb28da460e66590e974b4135bdd2e;f88acec67b648...

74375931;77621627

1804

Maixner2021

2021

10.1016/j.cub.2021.09.031

Edlersbergwerk - oben, Hallstatt

47.56°

13.63

Austria

2612

Homo sapiens

150

...

none

NaN

Illumina MiSeq

PAIRED

WGS

1075088

ERR5766181

<ftp.sra.ebi.ac.uk/vol1/fastq/ERR576/001/ERR576...>

57fc575d32db14f1d5c1ed7f6a106e91;4f57b9d978b53...

51852071;56288763

1805

Maixner2021

2021

<10.1101/j.cub.2021.09.031>

Edlersbergwerk - oben, Hallstatt

47.560

13.63

Austria

2612

Homo sapiens

150

...

none

NaN

Illumina HiSeq 2500

PAIRED

WGS

138836358

ERR5766182

<ftp.sra.ebi.ac.uk/vol1/fastq/ERR576/002/ERR576...>

```
64e63df8da7542957d1d9eb08e764d38;3fc6cba02c74d...
4332353625;4420486328
1806
Maixner2021
2021
10.1016/j.cub.2021.09.031
Edlersbergwerk - oben, Hallstatt
47.560
13.63
Austria
2612
Homo sapiens
150
...
none
NaN
HiSeq X Ten
PAIRED
WGS
84192332
ERR5766183
ftp.sra.ebi.ac.uk/vol1/fastq/ERR576/003/ERR576...
43ac661c4e211ed6ee2940dcab8b13cb;88de66a85df92...
3128863954;3460789287
1807 rows × 31 columns
```

## 9.13 10 - Visualizing some of the results with Plot-nine

Plotnine is the Python clone of ggplot2, the syntax is identical, which makes it great if you're working with data in tidy/long format, and are already familiar with the ggplot2 syntax

```
ggplot(merged_df, aes(x='publication_year')) + geom_histogram() + theme_classic()

/Users/maxime/mambaforge/envs/intro-data/lib/python3.10/
site-packages/plotnine/stats/stat_bin.py:95:
PlotnineWarning: 'stat_bin()' using 'bins = 15'. Pick better value with 'binwidth'.
```

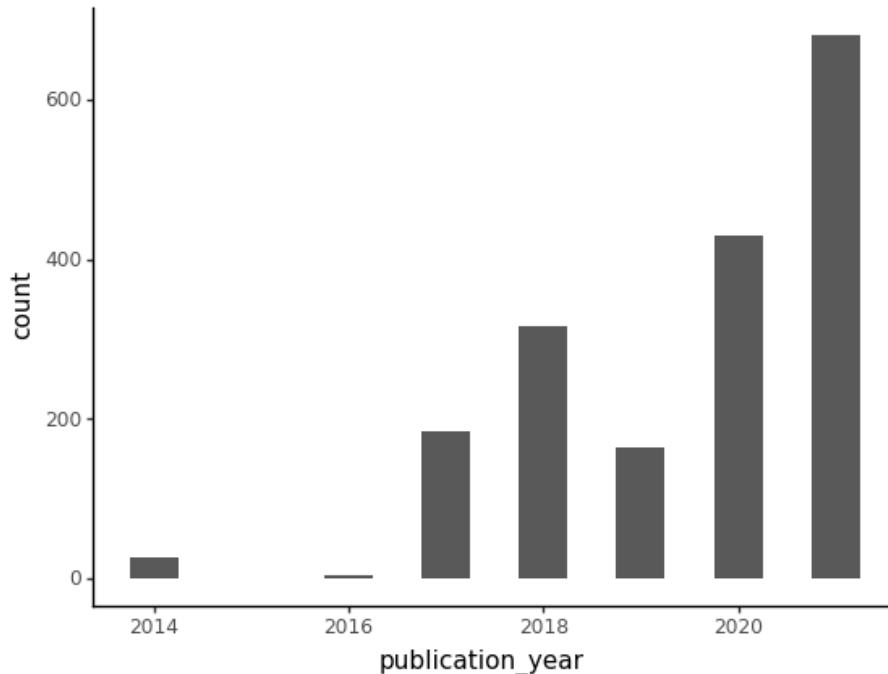


Figure 9.3: png

&lt;ggplot: (366051178)&gt;

We can start to ask some questions, for example, is the sequencing depth increasing with the years ?

```
merged_df['publication_year'] = merged_df['publication_year'].astype('category')

ggplot(merged_df, aes(x='publication_year', y='np.log10(read_count)', fill='publication_year')) +
 geom_jitter(alpha=0.1) + geom_boxplot(alpha=0.8) + theme_classic()

<ggplot: (366112582)>
```

We could ask the same question, but first grouping the samples by publication year

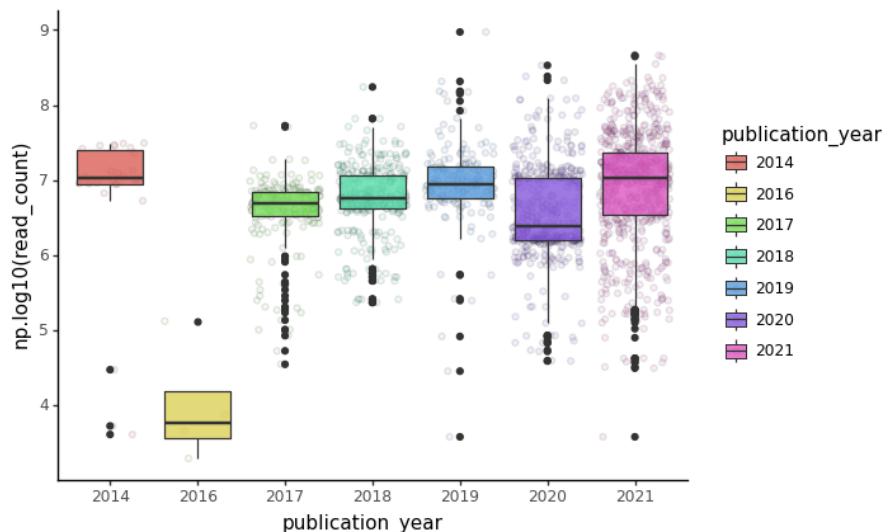


Figure 9.4: png

```
avg_read_count_by_year = merged_df.groupby('publication_year')['read_count'].mean().to_f
```

```
publication_year
```

```
read_count
```

```
0
```

```
2014
```

```
1.437173e+07
```

```
1
```

```
2016
```

```
3.653450e+04
```

```
2
```

```
2017
```

```
5.712598e+06
```

```
3
```

```
2018
```

```
9.273287e+06
```

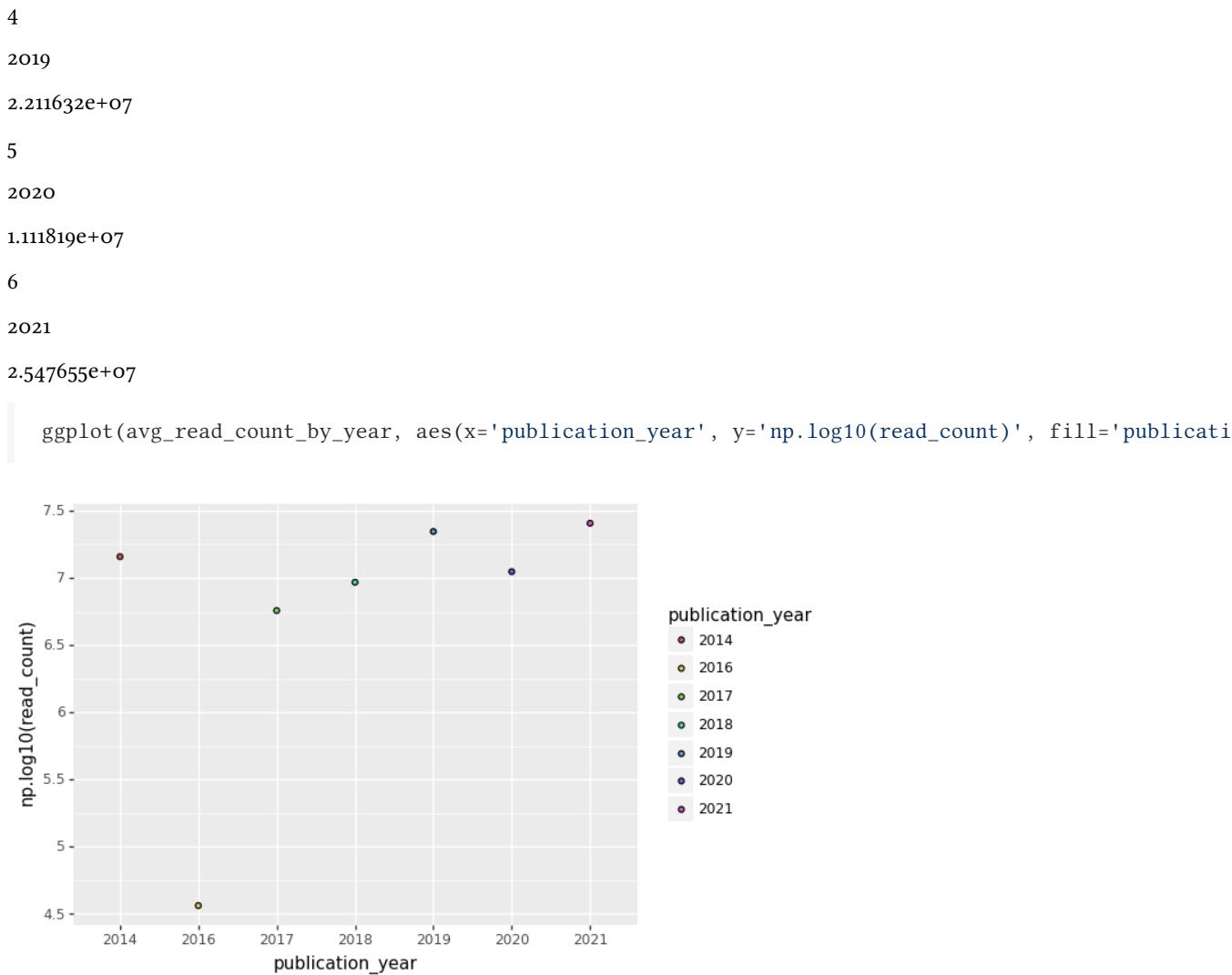


Figure 9.5: png

&lt;ggplot: (366206706)&gt;

**Your turn ! Make a plot to investigate the relation between the type of library treatment throughout the publication years**

## 9.14 11 - Bonus, dealing with ill-formatted columns

Sometimes, columns can contain entries which could be split in multiple columns, typically values separated by a comma. In AncientMetagenomeDir, this is the case

with the archive accession column.

Here is how we would solve it with pandas

```
sample_df.assign(archive_accession = sample_df.archive_accession.str.split(",")).explode()

project_name
publication_year
publication_doi
site_name
latitude
longitude
geo_loc_name
sample_name
sample_host
sample_age
sample_age_doi
community_type
material
archive
archive_project
archive_accession
o
Warinner2014
2014
10.1038/ng.2906
Dalheim
51-565
8.840
Germany
B61
Homo sapiens
900
```

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473742

o

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.840

Germany

B61

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473743

o

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.840

Germany

B61

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473744

o

Warinner2014

2014

10.1038/ng.2906

Dalheim

51-565

8.840

Germany

B61

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473745

1

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.840

Germany

G12

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473747

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

1057

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Gertrude's Church, Riga

56.958

24.121

Latvia

T9

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283099

1058

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

Dom Square, Riga

56.949

24.104

Latvia

TZA3

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283100

1058

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

Dom Square, Riga

56.949

24.104

Latvia

TZA3

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283101

1059

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Peter's Church, Riga

56.947

24.109

Latvia

TZA4

Homo sapiens

500

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283102

1059

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Peter's Church, Riga

56.947

24.109

Latvia

TZA4

Homo sapiens

500

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283103

1262 rows × 16 columns

# Chapter 10

## Introduction to Git(Hub)

### 10.1 Introduction

In this walkthrough, we will introduce the version control system **Git** as well as **Github**, a remote hosting service for version controlled repositories. Git and Github are increasingly popular tools for tracking data, collaborating on research projects, and sharing data and code, and learning to use them will help in many aspects of your own research. For more information on the benefits of using version control systems, see the slides.



#### Tip

For this chapter's exercises, if not already performed, you will need to create the `conda` environment from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate git-eager
```

### 10.2 Lecture

PDF version of these slides can be downloaded from [here](#).

### 10.3 SSH setup

To begin, you will set up an SSH key to facilitate easier authentication when transferring data between local and remote repositories. In other words, follow this section of the tutorial so that you never have to type in your github password again! Begin by activating the conda environment for this section (see **Preparation** above).

```
conda activate git-eager
```

Next, generate your own ssh key, replacing the email below with your own address.

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

I recommend saving the file to the default location and skipping passphrase setup. To do this, simply press enter without typing anything.

You should now (hopefully!) have generated an ssh key. To check that it worked, run the following commands to list the files containing your public and private keys and check that the ssh program is running.

```
cd ~/.ssh/
ls id*
eval "$(ssh-agent -s)"
```

Now you need to give ssh your key to record:

```
ssh-add ~/.ssh/id_ed15519
```

Next, open your webbrowser and navigate to your github account. Go to settings -> SSH & GPG Keys -> New SSH Key. Give your key a title and paste the public key that you just generated on your local machine.

```
cat ~/.ssh/id_ed15519
```

Finally, press Add SSH key. To check that it worked, run the following command on your local machine. You should see a message telling you that you've successfully authenticated.

```
ssh -T git@github.com
```

For more information about setting up the SSH key, including instructions for different operating systems, check out github's documentation: <https://docs.github.com/es/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>.

## 10.4 The only 6 commands you really need to know

Now that you have set up your own SSH key, we can begin working on some version controlled data! Navigate to your github homepage and create a new repository. You can choose any name for your new repo (including the default). Add a README file, then select Create Repository.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner \* Repository name \*

 meganemichel /

Great repository names are short and memorable. Need inspiration? How about [super-duper-dollop](#)?

Description (optional)

 **Public**  
Anyone on the internet can see this repository. You choose who can commit.

 **Private**  
You choose who can see and commit to this repository.

Initialize this repository with:

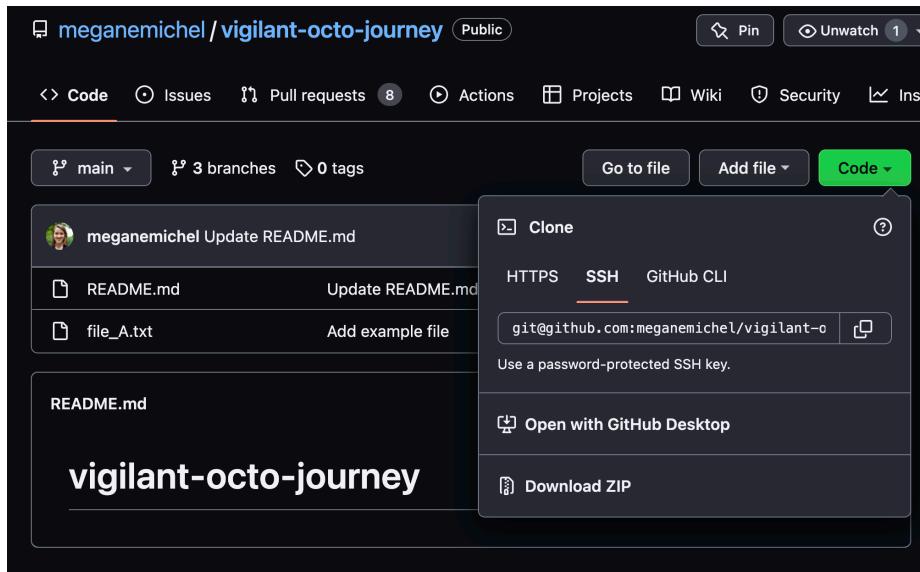
Skip this step if you're importing an existing repository.

Add a README file

### Note

For the remainder of the session, replace the name of my repository (`vigilant-octo-journey`) with your own repo name.

Change into the directory where you would like to work, and let's get started! First, we will learn to **clone** a remote repository onto your local machine. Navigate to your new repo, select the *Code* dropdown menu, select SSH, and copy the address as shown below.



Back at your command line, clone the repo as follows:

```
git clone git@github.com:meganemichel/vigilant-octo-journey.git
```

Next, let's **add** a new or modified file to our 'staging area' on our local machine.

```
cd vigilant-octo-journey
echo "test_file" > file_A.txt
echo "Just an example repo" >> README.md
git add file_A.txt
```

Now we can check what files have been locally changed, staged, etc. with **status**.

```
git status
```

You should see that `file_A.txt` is staged to be committed, but `README.md` is NOT. Try adding `README.md` and check the status again.

Now we need to package or save the changes into a **commit** with a message describing the changes we've made. Each commit comes with a unique hash ID and will be stored forever in git history.

```
git commit -m "Add example file"
```

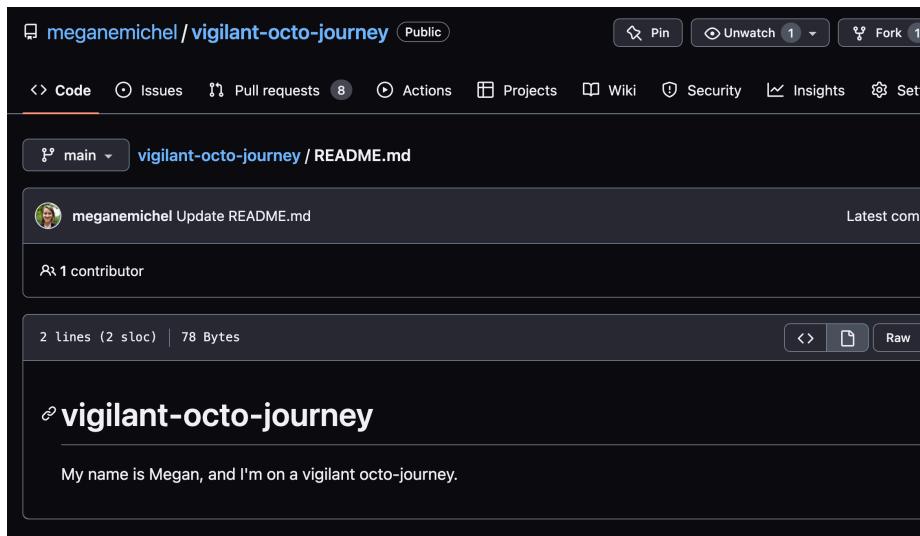
Finally, let's **push** our local commit back to our remote repository.

```
git push
```

What if we want to download new commits from our remote to our local repository?

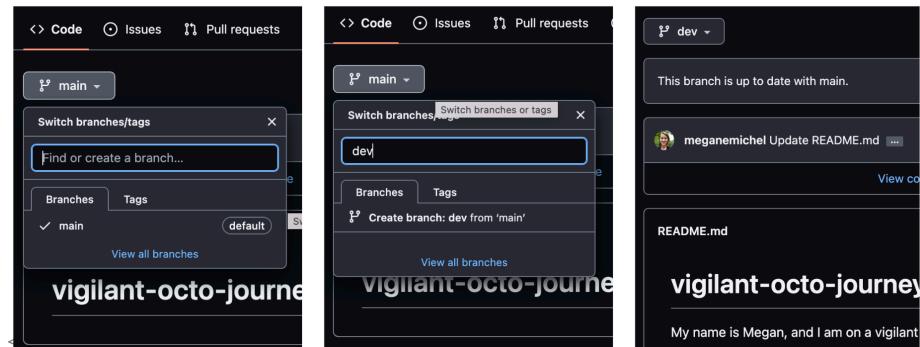
```
git pull
```

You should see that your repository is already up-to-date, since we have not made new changes to the remote repo. Let's try making a change to the remote repository's `README` file (as below). Then, back on the command line, pull the repository again.



## 10.5 Working collaboratively

Github facilitates simultaneous work by small teams through branching, which generates a copy of the main repository within the repository. This can be edited without breaking the ‘master’ version. First, back on github, make a new branch of your repository.



From the command line, you can create a new branch as follows:

```
git switch -c new_branch
```

To switch back to the main branch, use

```
git switch main
```

Note that you **must commit changes** for them to be saved to the desired branch!

## **10.6 Pull requests**

A **Pull request** (aka PR) is used to propose changes to a branch from another branch. Others can comment and make suggestions before your changes are merged into the main branch. For more information on creating a pull request, see github's documentation: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request>.

## **10.7 Questions to think about**

1. Why is using a version control software for tracking data and code important?
2. How can using Git(Hub) help me to collaborate on group projects?

## **Part III**

# **Ancient Metagenomics**



The techniques in this section of the book can be used in a variety of stages of any ancient metagenomics projects, for screening for pathogens (what species should I target for downstream genomic mapping?), for differential abundance analysis (does the community make of this sample change between different cultural periods?), but also for reference-free assembly of genomes (can I recover the genome architecture of a variety of species in my sample?). It focuses on the concept of ‘many samples to many genomes’ using high-throughput techniques and algorithms and trying to analyse data at whole ‘community’ levels.

## Taxonomic Profiling

TBC

## Functional Profiling

The value of microbial taxonomy lies in the implied biochemical properties of a given taxon. Historically taxonomy was determined by growth characteristics and cell properties, and more recently through genomic and genetic similarity.

The genomic content of microbial taxa, specifically the presence or absence of genes, determine how those taxa interact with their environment, including all the biochemical processes they participate in, both internally and externally. Strains within any microbial species may have different genetic content and therefore may behave strikingly differently in the same environment, which cannot be determined through taxonomic profiling. Functionally profiling a microbial community, or determining all of the genes present independent of the species they are derived from, reveals the biochemical reactions and metabolic products the community may perform and produce, respectively.

This approach may provide insights to community activity and environmental interactions that are hidden when using taxonomic approaches alone. In this chapter we will perform functional profiling of metagenomic communities to assess their genetic content and inferred metabolic pathways.

## *De novo* Assembly

*De novo* assembly of ancient metagenomic samples enables the recovery of the genetic information of organisms without requiring any prior knowledge about their genomes. Therefore, this approach is very well suited to study the biological diversity of species that have not been studied well or are simply not known yet.

In this chapter, we will show you how to prepare your sequencing data and subsequently *de novo* assemble them. Furthermore, we will then learn how we can actually evaluate what organisms we might have assembled and whether we obtained enough

data to reconstruct a whole metagenome-assembled genome. We will particularly focus on the quality assessment of these reconstructed genomes and how we can ensure that we obtained high-quality genomes.

# Chapter 11

## Taxonomic Profiling, OTU Tables and Visualisation



### Tip

For this chapter's exercises, if not already performed, you will need to create the **conda environment** from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate r-python
```

### 11.1 Lecture

PDF version of these slides can be downloaded from [here](#).

This session is run using a Jupyter notebook. This can be found [here](#). However, it will already be installed on compute nodes during the summer school.



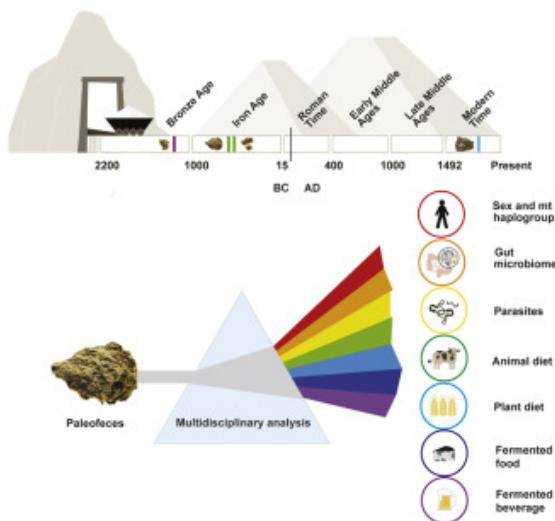
### Warning

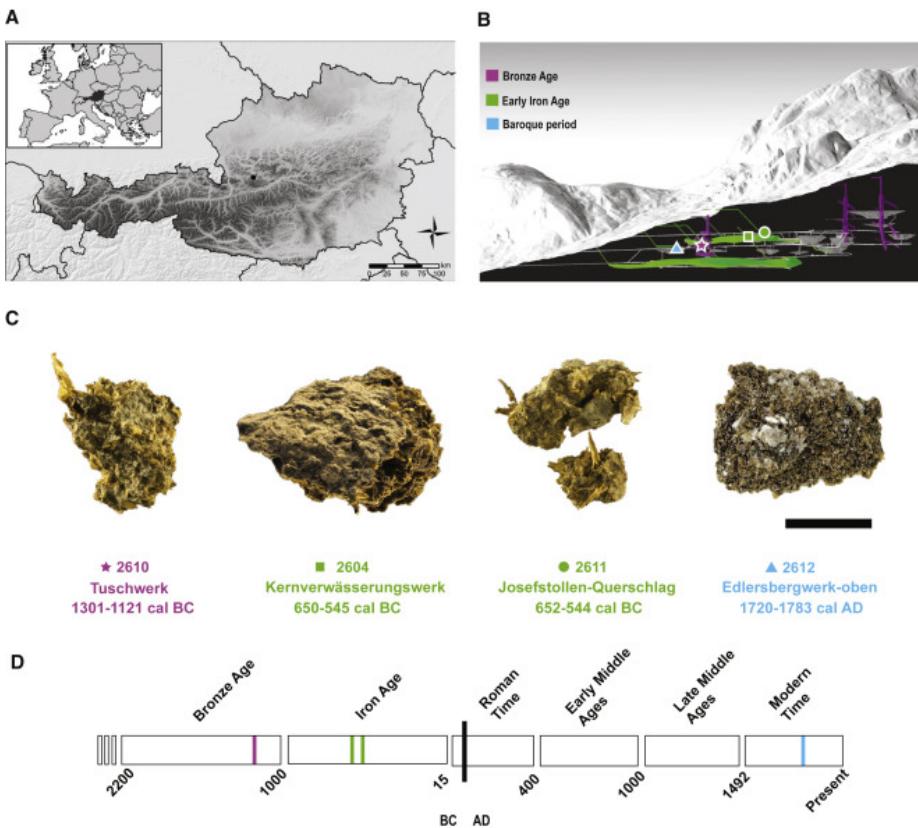
We highly recommend viewing this walkthrough via the Jupyter notebook above! The output of commands on the website for this walkthrough are displayed in their own code blocks - be wary of what you copy-paste!

## 11.2 Download and Subsample

```
import subprocess
import glob
from pathlib import Path
```

For this tutorial, we will be using the ERR5766177 library from the sample 2612 published by [Maixner et al. 2021](#)





### 11.2.1 Subsampling the sequencing files to make the analysis quicker for this tutorial

```

def subsample(filename, outdir, depth=1000000):
 basename = Path(filename).stem
 cmd = f"seqtk sample -s42 {filename} {depth} > {outdir}/{basename}_subsample_{depth}.fastq"
 print(cmd)
 subprocess.check_output(cmd, shell=True)

for f in glob.glob("../data/raw/*"):
 outdir = "../data/subsampled"
 subsample(f, outdir)

seqtk sample -s42 ../data/raw/ERR5766177_PE.mapped.hostremoved.fwd.fq.gz 1000000 >
 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.fwd.fq_subsample_1000000.fastq
seqtk sample -s42 ../data/raw/ERR5766177_PE.mapped.hostremoved.rev.fq.gz 1000000 >
 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.rev.fq_subsample_1000000.fastq

```

```
! gzip -f ./data/subsampled/*.fastq
```

## 11.3 Hands on introduction to ancient microbiome analysis

Author: Maxime Borry

Date: 12/08/2021

In this tutorial, we're going to go through the steps necessary to:

- generate a taxonomic profile table with [metaphlan v3.0](#)
- have a look at metaphlan results with [Pavian](#) and generate a [Krona plot](#)
- Compare the diversity of our samples vs the diversity of modern human gut samples
- Compare the composition of our samples vs modern gut samples, and see where they fit in a lower dimensional space

### 11.3.1 o. Quick intro to Jupyter Notebooks

This a markdown cell

```
print("This is a python cell")
```

This is a python cell

```
! echo "This is how to run a single line bash command"
```

This is how to run a single line bash command

```
%%bash
echo "This how to run a multi"
echo "lines bash command"
```

This how to run a multi  
lines bash command

### 11.3.2 1. Data pre-processing

Before starting to analyze our data, we will need to pre-process them to remove reads mapping to the host genome, here, *Homo sapiens*

To do so, I've used [nf-core/eager](#)

I've already pre-processed the data, and the resulting cleaned files are available in the [data/eager\\_cleaned](#), but the basic eager command to do so is:

```
nextflow run nf-core/eager -profile <docker/singularity/podman/conda/institute> --input '*_R{1,2}.fastq'
--fasta 'human_genome.fasta' --hostremoval_input_fastq
```

### 11.3.3 2. Adapter sequence trimming and low-quality bases trimming

Sequencing adapters are small DNA sequences added prior to DNA sequencing to allow the DNA fragments to attach to the sequencing flow cells. Because these adapters could interfere with downstream analyses, we need to remove them before proceeding any further. Furthermore, because the quality of the sequencing is not always optimal, we need to remove bases of lower sequencing quality to might lead to spurious results in downstream analyses.

To perform both of these tasks, we'll use the program [fastp](#).

```
! fastp -h

option needs value: --html
usage: fastp [options] ...
options:
 -i, --in1 read1 input file name (string [=])
 -o, --out1 read1 output file name (string [=])
 -I, --in2 read2 input file name (string [=])
 -O, --out2 read2 output file name (string [=])
 --unpaired1 for PE input, if read1 passed QC but read2 not, it will be written to unpaired1
 Default is to discard it. (string [=])
 --unpaired2 for PE input, if read2 passed QC but read1 not, it will be written to unpaired2
 If --unpaired2 is same as --unpaired1 (default mode), both unpaired reads will be
 written to this same file. (string [=])
 --overlapped_out for each read pair, output the overlapped region if it has no any mismatched
 base. (string [=])
 --failed_out specify the file to store reads that cannot pass the filters. (string [=])
 -m, --merge for paired-end input, merge each pair of reads into a single read if they are
 overlapped. The merged reads will be written
 --merged_out to the file given by --merged_out, the unmerged reads will be written to the
 files specified by --out1 and --out2. The merging mode is disabled by default.
 in the merging mode, specify the file name to store merged output, or specify
 --stdout to stream the merged output (string [=])
 --include_unmerged in the merging mode, write the unmerged or unpaired reads to the file specified
 by --merge. Disabled by default.
 -6, --phred64 indicate the input is using phred64 scoring (it'll be converted to phred33),
 so the output will still be phred33
 -z, --compression compression level for gzip output (1 ~ 9). 1 is fastest, 9 is smallest, def
 --stdin input from STDIN. If the STDIN is interleaved paired-end FASTQ, please also add
 --stdout stream passing-filters reads to STDOUT. This option will result in interleaving
 FASTQ output for paired-end output. Disabled by default.
```

--interleaved\_in indicate that <in1> is an interleaved FASTQ which contains both forward and reverse reads. Disabled by default.

--reads\_to\_process specify how many reads/pairs to be processed. Default 0 means all. If >0, don't overwrite existing files. Overwriting is allowed by default.

--fix\_mgi\_id the MGI FASTQ ID format is not compatible with many BAM operations. This option forces the conversion of the MGI ID to standard FASTQ IDs.

-V, --verbose output verbose log information (i.e. when every 1M reads are processed).

-A, --disable\_adapter\_trimming adapter trimming is enabled by default. If this option is specified, it will disable adapter trimming.

-a, --adapter\_sequence the adapter for read1. For SE data, if not specified, the adapter sequence is determined from the first read. For PE data, this is used if R1/R2 are found not overlapped. (string)

--adapter\_sequence\_r2 the adapter for read2 (PE data only). This is used if R1/R2 are found not overlapped. If not specified, it will be the same as <adapter\_sequence> (string)

--adapter\_fasta specify a FASTA file to trim both read1 and read2 (if PE) by all bases.

--detect\_adapter\_for\_pe by default, the auto-detection for adapter is for SE data input. This option to enable it for PE data.

-f, --trim\_front1 trimming how many bases in front for read1, default is 0 (int [=0])

-t, --trim\_tail1 trimming how many bases in tail for read1, default is 0 (int [=0])

-b, --max\_len1 if read1 is longer than max\_len1, then trim read1 at its tail to long as max\_len1. Default 0 means no limitation (int [=0])

-F, --trim\_front2 trimming how many bases in front for read2. If it's not specified, it will be the same as <trim\_front1>.

-T, --trim\_tail2 trimming how many bases in tail for read2. If it's not specified, it will be the same as <trim\_tail1>.

-B, --max\_len2 if read2 is longer than max\_len2, then trim read2 at its tail to long as max\_len2. Default 0 means no limitation. If it's not specified, it will follow <max\_len1>.

-D, --dedup enable deduplication to drop the duplicated reads/pairs.

--dup\_calc\_accuracy accuracy level to calculate duplication (1~6), higher level uses more memory. Default 1 for no-dedup mode, and 3 for dedup mode. (int [=0])

--dont\_eval\_duplication don't evaluate duplication rate to save time and use less memory.

-g, --trim\_poly\_g force polyG tail trimming, by default trimming is automatically done. The minimum length to detect polyG in the read tail. 10 by default.

--poly\_g\_min\_len disable polyG tail trimming, by default trimming is automatic.

-G, --disable\_trim\_poly\_g enable polyX trimming in 3' ends.

-x, --trim\_poly\_x the minimum length to detect polyX in the read tail. 10 by default.

--poly\_x\_min\_len move a sliding window from front (5') to tail, drop the bases in its mean quality < threshold, stop otherwise.

-5, --cut\_front move a sliding window from front (5') to tail, drop the bases in its mean quality < threshold, stop otherwise.

-3, --cut\_tail move a sliding window from tail (3') to front, drop the bases in its mean quality < threshold, stop otherwise.

-r, --cut\_right move a sliding window from front to tail, if meet one window with < threshold, drop the bases in the window and the right part, and the window size option shared by cut\_front, cut\_tail or cut\_size.

-W, --cut\_window\_size the mean quality requirement option shared by cut\_front, cut\_tail or cut\_right. Range: 1~36 default: 20 (Q20) (int [=20])

--cut\_front\_window\_size the window size option of cut\_front, default to cut\_window\_size.

--cut\_front\_mean\_quality the mean quality requirement option for cut\_front, default to cut\_mean\_quality.

--cut\_tail\_window\_size the window size option of cut\_tail, default to cut\_window\_size.

--cut\_tail\_mean\_quality the mean quality requirement option for cut\_tail, default to cut\_mean\_quality.

--cut\_right\_window\_size the window size option of cut\_right, default to cut\_window\_size.

--cut\_right\_mean\_quality the mean quality requirement option for cut\_right, default to cut\_mean\_quality.

```

-Q, --disable_quality_filtering quality filtering is enabled by default. If this option is specified,
-q, --qualified_quality_phred the quality value that a base is qualified. Default 15 means phred qua
-u, --unqualified_percent_limit how many percents of bases are allowed to be unqualified (0~100). Defa
-n, --n_base_limit if one read's number of N base is >n_base_limit, then this read/pair is di
-e, --average_qual if one read's average quality score <avg_qual, then this read/pair is disc
 Default 0 means no requirement (int [=0])
-L, --disable_length_filtering length filtering is enabled by default. If this option is specified, l
-1, --length_required reads shorter than length_required will be discarded, default is 15. (in
--length_limit reads longer than length_limit will be discarded, default 0 means no limita
-y, --low_complexity_filter enable low complexity filter. The complexity is defined as the percenta
 that is different from its next base (base[i] != base[i+1]).
-Y, --complexity_threshold the threshold for low complexity filter (0~100). Default is 30, which m
--filter_by_index1 specify a file contains a list of barcodes of index1 to be filtered out, o
--filter_by_index2 specify a file contains a list of barcodes of index2 to be filtered out, o
--filter_by_index_threshold the allowed difference of index barcode for index filtering, default 0
-c, --correction enable base correction in overlapped regions (only for PE data), default i
--overlap_len_require the minimum length to detect overlapped region of PE reads. This will aff
 adapter trimming and correction. 30 by default. (int [=30])
--overlap_diff_limit the maximum number of mismatched bases to detect overlapped region of PE
 This will affect overlap analysis based PE merge, adapter trimming and correctio
--overlap_diff_percent_limit the maximum percentage of mismatched bases to detect overlapped region
 This will affect overlap analysis based PE merge, adapter trimming and correctio
-U, --umi enable unique molecular identifier (UMI) preprocessing
--umi_loc specify the location of UMI, can be (index1/index2/read1/read2/per_index/pe
--umi_len if the UMI is in read1/read2, its length should be provided (int [=0])
--umi_prefix if specified, an underline will be used to connect prefix and UMI (i.e.
 prefix=UMI, UMI=AATTCT, final=UMI_AATTCT). No prefix by default (string [=])
--umi_skip if the UMI is in read1/read2, fastp can skip several bases following UMI, def
-p, --overrepresentation_analysis enable overrepresented sequence analysis.
-P, --overrepresentation_sampling one in (--overrepresentation_sampling) reads will be computed for ov
 analysis (1~10000), smaller is slower, default is 20. (int [=20])
-j, --json the json format report file name (string [=fastp.json])
-h, --html the html format report file name (string [=fastp.html])
-R, --report_title should be quoted with ' or ", default is "fastp report" (string [=fastp re
-w, --thread worker thread number, default is 3 (int [=3])
-s, --split split output by limiting total split file number with this option (2~999), a
 will be added to output name (0001.out.fq, 0002.out.fq...), disabled by default
-S, --split_by_lines split output by limiting lines of each file with this option(>=1000), a
 added to output name (0001.out.fq, 0002.out.fq...), disabled by default (long
-d, --split_prefix_digits the digits for the sequential number padding (1~10), default is 4, so th
 0001.xxx, 0 to disable padding (int [=4])
--cut_by_quality5 DEPRECATED, use --cut_front instead.
--cut_by_quality3 DEPRECATED, use --cut_tail instead.
--cut_by_quality_aggressive DEPRECATED, use --cut_right instead.
--discard_unmerged DEPRECATED, no effect now, see the introduction for merging.
-?, --help print this message

```

```

%%bash
fastp \
--in1 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.fwd fq_subsample_1000000.f
--in2 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.fwd fq_subsample_1000000.f
--merge \
--merged_out ../results/fastp/ERR5766177.merged.fastq.gz \
--include_unmerged \
--dedup \
--json ../results/fastp/ERR5766177.fastp.json \
--html ../results/fastp/ERR5766177.fastp.html \

```

Read1 before filtering:  
total reads: 1000000  
total bases: 101000000  
Q20 bases: 99440729(98.4562%)  
Q30 bases: 94683150(93.7457%)

Read2 before filtering:  
total reads: 1000000  
total bases: 101000000  
Q20 bases: 99440729(98.4562%)  
Q30 bases: 94683150(93.7457%)

Merged and filtered:  
total reads: 1994070  
total bases: 201397311  
Q20 bases: 198330392(98.4772%)  
Q30 bases: 188843169(93.7665%)

Filtering result:  
reads passed filter: 1999252  
reads failed due to low quality: 728  
reads failed due to too many N: 20  
reads failed due to too short: 0  
reads with adapter trimmed: 282  
bases trimmed due to adapters: 18654  
reads corrected by overlap analysis: 0  
bases corrected by overlap analysis: 0

Duplication rate: 0.2479%

Insert size peak (evaluated by paired-end reads): 31

Read pairs merged: 228  
% of original read pairs: 0.0228%

```
% in reads after filtering: 0.0114339%
```

```
JSON report: ../results/fastp/ERR5766177.fastp.json
HTML report: ../results/fastp/ERR5766177.fastp.html
```

```
fastp --in1 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.fwd.fq_subsample_1000000.fastq.gz \
--in2 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.fwd.fq_subsample_1000000.fastq.gz --merge \
--merged_out ../results/fastp/ERR5766177.merged.fastq.gz --include_unmerged --dedup \
--json ../results/fastp/ERR5766177.fastp.json --html ../results/fastp/ERR5766177.fastp.html
fastp v0.23.2, time used: 11 seconds
```

### 3. Taxonomic profiling with Metaphlan

```
! metaphlan --help
```

```
usage: metaphlan --input_type {fastq,fasta,bowtie2out,sam} [--force]
 [--bowtie2db METAPHLAN_BOWTIE2_DB] [-x INDEX]
 [--bt2_ps BowTie2_presets] [--bowtie2_exe BOWTIE2_EXE]
 [--bowtie2_build BOWTIE2_BUILD] [--bowtie2out FILE_NAME]
 [--min_mapq_val MIN_MAPQ_VAL] [--no_map] [--tmp_dir]
 [--tax_lev TAXONOMIC_LEVEL] [--min_cu_len]
 [--min_alignment_len] [--add_viruses] [--ignore_eukaryotes]
 [--ignore_bacteria] [--ignore_archaea] [--stat_q]
 [--perc_nonzero] [--ignore_markers IGNORE_MARKERS]
 [--avoid_disqm] [--stat] [-t ANALYSIS_TYPE]
 [--nreads NUMBER_OF_READS] [--pres_th PRESENCE_THRESHOLD]
 [--clade] [--min_ab] [-o output_file] [--sample_id_key name]
 [--use_groupRepresentative] [--sample_id value]
 [-s sam_output_file] [--legacy-output] [--CAMI_format_output]
 [--unknown_estimation] [--biom biom_output] [--mdelim mdelim]
 [--nproc N] [--install] [--force_download]
 [--read_min_len READ_MIN_LEN] [-v] [-h]
 [INPUT_FILE] [OUTPUT_FILE]
```

#### DESCRIPTION

MetaPhlAn version 3.1.0 (25 Jul 2022):

METAGenomic PHylogenetic ANalysis for metagenomic taxonomic profiling.

AUTHORS: Francesco Beghini (francesco.beghini@unitn.it), Nicola Segata (nicola.segata@unitn.it), Duy Tin Pham (duytin.pham@unitn.it), Francesco Asnicar (f.asnicar@unitn.it), Aitor Blanco Miguez (aitor.blancomiguez@unitn.it)

#### COMMON COMMANDS

We assume here that MetaPhlAn is installed using the several options available (pip, conda, PyPi)

Also BowTie2 should be in the system path with execution and read permissions, and Perl should be in the system path.

---

===== MetaPhlAn clade-abundance estimation =====

The basic usage of MetaPhlAn consists in the identification of the clades (from phyla to species) present in the metagenome obtained from a microbiome sample and their relative abundance. This correspond to the default analysis type (-t rel\_ab).

- \* Profiling a metagenome from raw reads:  
`$ metaphlan metagenome.fastq --input_type fastq -o profiled_metagenome.txt`
  - \* You can take advantage of multiple CPUs and save the intermediate BowTie2 output for re-running MetaPhlAn extremely quickly:  
`$ metaphlan metagenome.fastq --bowtie2out metagenome.bowtie2.bz2 --nproc 5 --input_type fastq`
  - \* If you already mapped your metagenome against the marker DB (using a previous MetaPhlAn run), you can obtain the results in few seconds by using the previously saved --bowtie2out file and specifying the input (--input\_type bowtie2out):  
`$ metaphlan metagenome.bowtie2.bz2 --nproc 5 --input_type bowtie2out -o profiled_metagenome.txt`
  - \* bowtie2out files generated with MetaPhlAn versions below 3 are not compatible. Starting from MetaPhlAn 3.0, the BowTie2 output now includes the size of the profiled metagenome. If you want to re-run MetaPhlAn using these files you should provide the metagenome size via --nreads:  
`$ metaphlan metagenome.bowtie2.bz2 --nproc 5 --input_type bowtie2out --nreads 520000 -o profiled_metagenome.txt`
  - \* You can also provide an externally BowTie2-mapped SAM if you specify this format with --input\_type. Two steps: first apply BowTie2 and then feed MetaPhlAn with the obtained sam:  
`$ bowtie2 --sam-no-hd --sam-no-sq --no-unal --very-sensitive -S metagenome.sam -x \${mpa_dir}/metaphlan_databases/mpa_v30_CHOCOPhlan_201901 -U metagenome.fastq`  
`$ metaphlan metagenome.sam --input_type sam -o profiled_metagenome.txt`
  - \* We can also natively handle paired-end metagenomes, and, more generally, metagenomes stored in multiple files (but you need to specify the --bowtie2out parameter):  
`$ metaphlan metagenome_1.fastq,metagenome_2.fastq --bowtie2out metagenome.bowtie2.bz2 --nproc 5`
- 

---

===== Marker level analysis =====

MetaPhlAn introduces the capability of characterizing organisms at the strain level using non aggregated marker information. Such capability comes with several slightly different flavours and are a way to perform strain tracking and comparison across multiple samples. Usually, MetaPhlAn is first ran with the default -t to profile the species present in the community, and then a strain-level profiling can be performed to zoom-in into specific species of interest. This operation can be performed quickly as it exploits the --bowtie2out intermediate.

file saved during the execution of the default analysis type.

- \* The following command will output the abundance of each marker with a RPK (reads per kilo-base) higher 0.0. (we are assuming that metagenome\_outfmt.bz2 has been generated before as shown above).
 

```
$ metaphlan -t marker_ab_table metagenome_outfmt.bz2 --input_type bowtie2out -o marker_abundance_table.txt
```

 The obtained RPK can be optionally normalized by the total number of reads in the metagenome to guarantee fair comparisons of abundances across samples. The number of reads in the metagenome needs to be passed with the '--nreads' argument
- \* The list of markers present in the sample can be obtained with '-t marker\_pres\_table'.
 

```
$ metaphlan -t marker_pres_table metagenome_outfmt.bz2 --input_type bowtie2out -o marker_abundance_table.txt
```

 The '--pres\_th' argument (default 1.0) set the minimum RPK value to consider a marker present
- \* The list '-t clade\_profiles' analysis type reports the same information of '-t marker\_ab\_table' but the markers are reported on a clade-by-clade basis.
 

```
$ metaphlan -t clade_profiles metagenome_outfmt.bz2 --input_type bowtie2out -o marker_abundance_table.txt
```
- \* Finally, to obtain all markers present for a specific clade and all its subclades, the '-t clade\_specific\_strain\_tracker' should be used. For example, the following command is reporting the presence/absence of the markers for the *B. fragilis* species and its strains the optional argument '--min\_ab' specifies the minimum clade abundance for reporting the markers
 

```
$ metaphlan -t clade_specific_strain_tracker --clade s__Bacteroides_fragilis metagenome_outfmt.bz2 --input_type bowtie2out -o marker_abundance_table.txt
```

---

positional arguments:

INPUT_FILE	the input file can be: * a fastq file containing metagenomic reads OR * a BowTie2 produced SAM file. OR * an intermediary mapping file of the metagenome generated by a previous MetaPhlAn run If the input file is missing, the script assumes that the input is provided using the standard input, or named pipes.
IMPORTANT: the type of input needs to be specified with --input_type	
OUTPUT_FILE	the tab-separated output file of the predicted taxon relative abundances [stdout if not present]

Required arguments:

--input_type {fastq,fasta,bowtie2out,sam}	set whether the input is the FASTA file of metagenomic reads or the SAM file of the mapping of the reads against the MetaPhlAn db.
-------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------

## Mapping arguments:

```
--force Force profiling of the input file by removing the bowtie2out file
--bowtie2db METAPHLAN_BOWTIE2_DB
 Folder containing the MetaPhlAn database. You can specify the location by exposing
 DEFAULT_DB_FOLDER variable in the shell.
 [default /Users/maxime/mambaforge/envs/summer_school_microbiome/lib/python3.6/site-packages/metaphlan/bowtie2_db]
-x INDEX, --index INDEX
 Specify the id of the database version to use. If "latest", MetaPhlAn will get the latest
 If an index name is provided, MetaPhlAn will try to use it, if available, and skip
 If the database files are not found on the local MetaPhlAn installation they
 will be automatically downloaded [default latest]
--bt2_ps BowTie2 presets
 Presets options for BowTie2 (applied only when a FASTA file is provided)
 The choices enabled in MetaPhlAn are:
 * sensitive
 * very-sensitive
 * sensitive-local
 * very-sensitive-local
 [default very-sensitive]
--bowtie2_exe BOWTIE2_EXE
 Full path and name of the BowTie2 executable. This option allowsMetaPhlAn to run
 executable even when it is not in the system PATH or the system PATH is unreachable
--bowtie2_build BOWTIE2_BUILD
 Full path to the bowtie2-build command to use, deafult assumes that 'bowtie2-build'
--bowtie2out FILE_NAME
 The file for saving the output of BowTie2
--min_mapq_val MIN_MAPQ_VAL
 Minimum mapping quality value (MAPQ) [default 5]
--no_map
 Avoid storing the --bowtie2out map file
--tmp_dir
 The folder used to store temporary files [default is the OS dependent tmp directory]
```

## Post-mapping arguments:

```
--tax_lev TAXONOMIC_LEVEL
 The taxonomic level for the relative abundance output:
 'a' : all taxonomic levels
 'k' : kingdoms
 'p' : phyla only
 'c' : classes only
 'o' : orders only
 'f' : families only
 'g' : genera only
 's' : species only
 [default 'a']
--min_cu_len minimum total nucleotide length for the markers in a clade for
 estimating the abundance without considering sub-clade abundances
 [default 2000]
```

```
--min_alignment_len The sam records for aligned reads with the longest subalignment
length smaller than this threshold will be discarded.
 [default None]
--add_viruses Allow the profiling of viral organisms
--ignore_eukaryotes Do not profile eukaryotic organisms
--ignore_bacteria Do not profile bacterial organisms
--ignore_archaea Do not profile archeal organisms
--stat_q Quantile value for the robust average
 [default 0.2]
--perc_nonzero Percentage of markers with a non zero relative abundance for misidentify a species
 [default 0.33]
--ignore_markers IGNORE_MARKERS
 File containing a list of markers to ignore.
--avoid_disqm Deactivate the procedure of disambiguating the quasi-markers based on the
marker abundance pattern found in the sample. It is generally recommended
to keep the disambiguation procedure in order to minimize false positives
--stat Statistical approach for converting marker abundances into clade abundances
'avg_g' : clade global (i.e. normalizing all markers together) average
'avg_l' : average of length-normalized marker counts
'tavg_g' : truncated clade global average at --stat_q quantile
'tavg_l' : truncated average of length-normalized marker counts (at --stat_q)
'wavg_g' : winsorized clade global average (at --stat_q)
'wavg_l' : winsorized average of length-normalized marker counts (at --stat_q)
'med' : median of length-normalized marker counts
 [default tavg_g]
```

Additional analysis types and arguments:

```
-t ANALYSIS_TYPE Type of analysis to perform:
* rel_ab: profiling a metagenomes in terms of relative abundances
* rel_ab_w_read_stats: profiling a metagenomes in terms of relative abundances and estima-
 the number of reads coming from each clade.
* reads_map: mapping from reads to clades (only reads hitting a marker)
* clade_profiles: normalized marker counts for clades with at least a non-null marker
* marker_ab_table: normalized marker counts (only when > 0.0 and normalized by metagenome
* marker_counts: non-normalized marker counts [use with extreme caution]
* marker_pres_table: list of markers present in the sample (threshold at 1.0 if not differ-
* clade_specific_strain_tracker: list of markers present for a specific clade, specified
 [default 'rel_ab']
--nreads NUMBER_OF_READS
The total number of reads in the original metagenome. It is used only when
-t marker_table is specified for normalizing the length-normalized counts
with the metagenome size as well. No normalization applied if --nreads is not
specified
--pres_th PRESENCE_THRESHOLD
Threshold for calling a marker present by the -t marker_pres_table option
--clade The clade for clade_specific_strain_tracker analysis
```

```
--min_ab The minimum percentage abundance for the clade in the clade_specific_strain

Output arguments:
-o output_file, --output_file output_file
 The output file (if not specified as positional argument)
--sample_id_key name Specify the sample ID key for this analysis. Defaults to 'SampleID'.
--use_group_representative
 Use a species as representative for species groups.
--sample_id value Specify the sample ID for this analysis. Defaults to 'Metaphlan_Analysis'
-s sam_output_file, --samout sam_output_file
 The sam output file
--legacy-output Old MetaPhlAn2 two columns output
--CAMI_format_output Report the profiling using the CAMI output format
--unknown_estimation Scale relative abundances to the number of reads mapping to known clades
--biom biom_output, --biom_output_file biom_output
 If requesting biom file output: The name of the output file in biom format
--mdelim mdelim, --metadata_delimiter_char mdelim
 Delimiter for bug metadata: - defaults to pipe. e.g. the pipe in k_Bacteria|p_
Other arguments:
--nproc N The number of CPUs to use for parallelizing the mapping [default 4]
--install Only checks if the MetaPhlAn DB is installed and installs it if not. All other
--force_download Force the re-download of the latest MetaPhlAn database.
--read_min_len READ_MIN_LEN
 Specify the minimum length of the reads to be considered when parsing the input
 'read_fastx.py' script, default value is 70
-v, --version Prints the current MetaPhlAn version and exit
-h, --help show this help message and exit

metaphlan ../results/fastp/ERR5766177.merged.fastq.gz \
 --input_type fastq \
 --bowtie2out ../results/metaphlan/ERR5766177.bt2.out \
 --nproc 4 \
 > ../results/metaphlan/ERR5766177.metaphlan_profile.txt
```

The main results files that we're interested in is located at [./results/metaphlan/ERR5766177.metaphlan\\_profile.txt](#)

It's a tab separated file, with taxons in rows, with their relative abundance in the sample

```
! head ../results/metaphlan/ERR5766177.metaphlan_profile.txt

#mpa_v30_CHOCOPhlan_201901
#/home/maxime_borry/.conda/envs/maxime/envs/summer_school_microbiome/bin/metaphlan ../resul
--input_type fastq --bowtie2out ../results/metaphlan/ERR5766177.bt2.out --nproc 8
```

```
#SampleID Metaphlan_Analysis
#clade_name NCBI_tax_id relative_abundance additional_species
k_Bacteria 2 82.23198
k_Archaea 2157 17.76802
k_Bacteria|p_Firmicutes 2|1239 33.47957
k_Bacteria|p_Bacteroidetes 2|976 28.4209
k_Bacteria|p_Actinobacteria 2|201174 20.33151
k_Archaea|p_Euryarchaeota 2157|28890 17.76802
```

### 11.3.4 4. Visualizing the taxonomic profile

#### 11.3.4.1 4.1 Visualizing metaphlan taxonomic profile with Pavian

Pavian is an interactive app to explore results of different taxonomic classifiers

There are different ways to run it:

- If you have docker installed (and are somehow familiar with it)

```
docker pull 'florianbw/pavian'
docker run --rm -p 5000:80 florianbw/pavian
```

Then open your browser and visit [localhost:5000](http://localhost:5000)

- If you are familiar with R

```
if (!require(remotes)) { install.packages("remotes") }
remotes::install_github("fbreitwieser/pavian")

pavian::runApp(port=5000)
```

Then open your browser and visit [localhost:5000](http://localhost:5000)

- Otherwise, just visit [fbreitwieser.shinyapps.io/pavian](http://fbreitwieser.shinyapps.io/pavian).

#### 11.3.4.2 4.2 Visualizing metaphlan taxonomic profile with Krona

```
%%bash
python ./scripts/metaphlan2krona.py -p ./results/metaphlan/ERR5766177.metaphlan_profile.txt -k ...
ktImportText -o ./results/krona/ERR5766177_krona.html ./results/krona/ERR5766177_krona.out

Writing ./results/krona/ERR5766177_krona.html...
```

### 11.3.5 5. Getting modern reference data

In order to compare our sample with modern reference samples, I used the curatedMetagenomicsData package, which provides both curated metadata, and pre-

computed metaphlan taxonomic profiles for published modern human samples. The full R code to get these data is available in [curatedMetagenomics/get\\_sources.Rmd](#)

I pre-selected 200 gut microbiome samples from non-westernized (100) and westernized (100) from healthy, non-antibiotic users donors.

```
library(curatedMetagenomicData)
library(tidyverse)

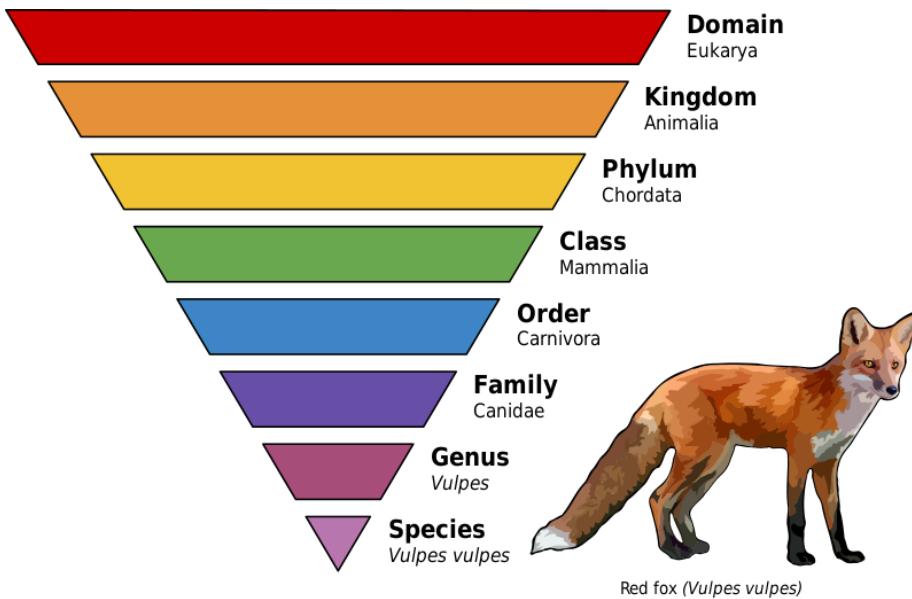
sampleMetadata %>%
 filter(body_site=='stool' & antibiotics_current_use == 'no' & disease == 'healthy') %>%
 group_by(non_westernized) %>%
 sample_n(100) %>%
 ungroup() -> selected_samples

selected_samples %>%
 returnSamples("relative_abundance") -> rel_ab

data_ranks = splitByRanks(rel_ab)

for (r in names(data_ranks)){
 print(r)
 assay_rank = as.data.frame(assay(data_ranks[[r]]))
 print(paste0("../data/curated_metagenomics/modern_sources_",tolower(r),".csv"))
 write.csv(assay_rank, paste0("../data/curated_metagenomics/modern_sources_",tolower(r),".csv"))
```

- The resulting metaphlan taxonomic profiles (split by taxonomic ranks) are available at [../data/curated\\_metagenomics](#)
- The associated metadata is available at [../data/metadata/curated\\_metagenomics\\_modern\\_sources.csv](#)



### 11.3.6 6. Bringing together ancient and modern data

This is the moment where we will use the [Pandas Python](#) library to perform some data manipulation.

We will also use the [Taxopy](#) library to work with taxonomic information.

```
! pip install taxopy
```

Requirement already satisfied: taxopy in /Users/maxime/mambaforge/envs/summer\_school\_microbiome/lib/python3.7/site-packages

```
import pandas as pd
import taxopy
import pickle
import gzip

with gzip.open("../data/taxopy/taxdb.p.gz", 'rb') as tdb:
 taxo_db = pickle.load(tdb)

! head ../results/metaphlan/ERR5766177.metaphlan_profile.txt

#mpa_v30_CHOCOPh1An_201901
#/home/maxime_borry/.conda/envs/maxime/envs/summer_school_microbiome/bin/metaphlan ../results/fastp/ER
--input_type fastq --bowtie2out ../results/metaphlan/ERR5766177.bt2.out --nproc 8
#SampleID Metaphlan_Analysis
#clade_name NCBI_tax_id relative_abundance additional_species
```

```

k__Bacteria 2 82.23198
k__Archaea 2157 17.76802
k__Bacteria|p__Firmicutes 2|1239 33.47957
k__Bacteria|p__Bacteroidetes 2|976 28.4209
k__Bacteria|p__Actinobacteria 2|201174 20.33151
k__Archaea|p__Euryarchaeota 2157|28890 17.76802

ancient_data = pd.read_csv("../results/metaphlan/ERR5766177.metaphlan_profile.txt",
 comment="#",
 delimiter="\t",
 names=['clade_name','NCBI_tax_id','relative_abundance','add...')

ancient_data.head()

clade_name
NCBI_tax_id
relative_abundance
additional_species
o
k__Bacteria
2
82.23198
NaN
1
k__Archaea
2157
17.76802
NaN
2
k__Bacteria|p__Firmicutes
2|1239
33.47957
NaN
3
k__Bacteria|p__Bacteroidetes

```

2|976

28.42090

NaN

4

k\_\_Bacteria|p\_\_Actinobacteria

2|201174

20.33151

NaN

```
ancient_data.sample(10)
```

clade\_name

NCBI\_tax\_id

relative\_abundance

additional\_species

1

k\_\_Archaea

2157

17.76802

NaN

46

k\_\_Bacteria|p\_\_Bacteroidetes|c\_Bacteroidia|o...

2|976|200643|171549|171552|838|165179

25.75544

k\_\_Bacteria|p\_\_Bacteroidetes|c\_Bacteroidia|o...

55

k\_\_Bacteria|p\_\_Firmicutes|c\_\_Clostridia|o\_\_Clo...

2|1239|186801|186802|186803|189330|88431

0.91178

NaN

18

k\_\_Archaea|p\_\_Euryarchaeota|c\_Halobacteria|o...

2157|28890|183963|2235  
0.71177  
NaN  
36  
k\_Bacteria|p\_Actinobacteria|c\_Actinobacteri...  
2|201174|1760|85004|31953|1678  
9.39377  
NaN  
65  
k\_Bacteria|p\_Actinobacteria|c\_Actinobacteri...  
2|201174|1760|85004|31953|1678|216816  
0.05447  
k\_Bacteria|p\_Actinobacteria|c\_Actinobacteri...  
37  
k\_Bacteria|p\_Firmicutes|c\_Clostridia|o\_Clo...  
2|1239|186801|186802|186803  
2.16125  
NaN  
38  
k\_Bacteria|p\_Firmicutes|c\_Clostridia|o\_Clo...  
2|1239|186801|186802|541000|216851  
1.24537  
NaN  
26  
k\_Bacteria|p\_Actinobacteria|c\_Actinobacteri...  
2|201174|1760|85004|31953  
9.39377  
NaN  
48  
k\_Bacteria|p\_Firmicutes|c\_Clostridia|o\_Clo...  
2|1239|186801|186802|541000|1263|40518

```
14.96816
```

```
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clo...
```

Because for this analysis, we're only going to look at the relative abundance, we'll only use this column, an the **TAXID** information

```
ancient_data = (
 ancient_data
 .rename(columns={'NCBI_tax_id': 'TAXID'})
 .drop(['clade_name', 'additional_species'], axis=1)
)
```

Always investigate your data at first !

```
ancient_data.relative_abundance.sum()
```

```
700.00007
```

**Pause and think:** A relative abundance of 700%, really ?

Let's proceed further and try to understand what's happening.

```
ancient_data.head()
```

```
TAXID
```

```
relative_abundance
```

```
0
```

```
2
```

```
82.23198
```

```
1
```

```
2157
```

```
17.76802
```

```
2
```

```
2|1239
```

```
33.47957
```

```
3
```

```
2|976
```

```
28.42090
```

```
4
```

```
2|201174
```

```
20.33151
```

To make sense of the TAXID, we will use taxopy to get all the taxonomic related informations such as:

- name of the taxon
- rank of the taxon
- lineage of the taxon

```
This function is here to help us get the taxon information
from the metaphlan taxonomic ID lineage, of the following form
2|976|200643|171549|171552|838|165179

def to_taxopy(taxid_entry, taxo_db):
 """Returns a taxopy taxon object
 Args:
 taxid_entry(str): metaphlan TAXID taxonomic lineage
 taxo_db(taxopy database)
 Returns:
 (bool): Returns a taxopy taxon object
 """
 taxid = taxid_entry.split("|")[-1] # get the last element
 try:
 if len(taxid) > 0:
 return taxopy.Taxon(int(taxid), taxo_db) # if it's not empty, get the taxon
 else:
 return taxopy.Taxon(12908, taxo_db) # otherwise, return the taxon associated
 except taxopy.exceptions.TaxidError as e:
 return taxopy.Taxon(12908, taxo_db)

ancient_data['taxopy'] = ancient_data['TAXID'].apply(to_taxopy, taxo_db=taxo_db)

ancient_data.head()

TAXID
relative_abundance
taxopy
o
2
82.23198
s__Bacteria
```

```
1
2157
17.76802
s__Archaea
2
2|1239
33.47957
s__Bacteria;c__Terrabacteria group;p__Firmicutes
3
2|976
28.42090
s__Bacteria;c__FCB group;p__Bacteroidetes
4
2|201174
20.33151
s__Bacteria;c__Terrabacteria group;p__Actinoba...
ancient_data = ancient_data.assign(
 rank = ancient_data.taxopy.apply(lambda x: x.rank),
 name = ancient_data.taxopy.apply(lambda x: x.name),
 lineage = ancient_data.taxopy.apply(lambda x: x.name_lineage),
)
ancient_data
TAXID
relative_abundance
taxopy
rank
name
lineage
o
2
```

82.23198

s\_Bacteria

superkingdom

Bacteria

[Bacteria, cellular organisms, root]

1

2157

17.76802

s\_Archaea

superkingdom

Archaea

[Archaea, cellular organisms, root]

2

2|1239

33.47957

s\_Bacteria;c\_Terrabacteria group;p\_Firmicutes

phylum

Firmicutes

[Firmicutes, Terrabacteria group, Bacteria, ce...

3

2|976

28.42090

s\_Bacteria;c\_FCB group;p\_Bacteroidetes

phylum

Bacteroidetes

[Bacteroidetes, Bacteroidetes/Chlorobi group, ...

4

2|201174

20.33151

s\_Bacteria;c\_Terrabacteria group;p\_Actinoba...

phylum

Actinobacteria

[Actinobacteria, Terrabacteria group, Bacteria...

...

...

...

...

...

...

...

62

2|1239|186801|186802|186803|572511|33039

0.24910

s\_\_Bacteria;c\_\_Terrabacteria group;p\_\_Firmicut...

species

[Ruminococcus] torques

[[Ruminococcus] torques, Mediterraneibacter, L...

63

2|201174|84998|84999|84107|1472762|1232426

0.17084

s\_\_Bacteria;c\_\_Terrabacteria group;p\_\_Actinoba...

species

[Collinsella] massiliensis

[[Collinsella] massiliensis, Enorma, Coriobact...

64

2|1239|186801|186802|186803|189330|39486

0.07690

s\_\_Bacteria;c\_\_Terrabacteria group;p\_\_Firmicut...

species

Dorea formicigenerans

[Dorea formicigenerans, Dorea, Lachnospiraceae...

65

```

2|201174|1760|85004|31953|1678|216816
0.05447
s__Bacteria;c__Terrabacteria group;p__Actinoba...
species
Bifidobacterium longum
[Bifidobacterium longum, Bifidobacterium, Bifi...
66
2|1239|186801|186802|541000|1263|1262959
0.01440
s__Bacteria;c__Terrabacteria group;p__Firmicut...
species
Ruminococcus sp. CAG:488
[Ruminococcus sp. CAG:488, environmental sampl...
67 rows × 6 columns

Because our modern data are split by ranks, we'll first split our ancient sample by rank

```

Which of the entries are at the species rank level ?

```

ancient_species = ancient_data.query("rank == 'species'")

ancient_species.head()

TAXID
relative_abundance
taxopy
rank
name
lineage
46
2|976|200643|171549|171552|838|165179
25.75544
s__Bacteria;c__FCB group;p__Bacteroidetes;c__B...

```

species

*Prevotella copri*

[*Prevotella copri*, *Prevotella*, *Prevotellaceae*,...]

47

2157|28890|183925|2158|2159|2172|2173

17.05626

s\_\_Archaea;p\_\_Euryarchaeota;c\_\_Methanomada gro...

species

*Methanobrevibacter smithii*

[*Methanobrevibacter smithii*, *Methanobrevibacte...*]

48

2|1239|186801|186802|541000|1263|40518

14.96816

s\_\_Bacteria;c\_\_Terrabacteria group;p\_\_Firmicut...

species

*Ruminococcus bromii*

[*Ruminococcus bromii*, *Ruminococcus*, *Oscillospi...*]

49

2|1239|186801|186802|186803|841|301302

13.57908

s\_\_Bacteria;c\_\_Terrabacteria group;p\_\_Firmicut...

species

*Roseburia faecis*

[*Roseburia faecis*, *Roseburia*, *Lachnospiraceae*,...]

50

2|201174|84998|84999|84107|102106|74426

9.49165

s\_\_Bacteria;c\_\_Terrabacteria group;p\_\_Actinoba...

species

*Collinsella aerofaciens*

[*Collinsella aerofaciens*, *Collinsella*, *Corioba...*]

Let's do a bit of renaming to prepare for what's coming next

```
ancient_species = ancient_species[['relative_abundance', 'name']].set_index('name').renam

ancient_species.head()

ERR5766177
name
Prevotella copri
25.75544
Methanobrevibacter smithii
17.05626
Ruminococcus bromii
14.96816
Roseburia faecis
13.57908
Collinsella aerofaciens
9.49165

ancient_phylums = ancient_data.query("rank == 'phylum'")

ancient_phylums = ancient_phylums[['relative_abundance', 'name']].set_index('name').renam

ancient_phylums

ERR5766177
name
Firmicutes
33.47957
Bacteroidetes
28.42090
Actinobacteria
20.33151
Euryarchaeota
```

17.76802

Now, let's go back to the 700% relative abundance issue...

```
ancient_data.groupby('rank')['relative_abundance'].sum()
```

rank	
class	99.72648
family	83.49854
genus	97.56524
no rank	19.48331
order	99.72648
phylum	100.00000
species	100.00002
superkingdom	100.00000

Name: relative\_abundance, dtype: float64

Seems better, right ?

**Pause and think: why don't we get exactly 100% ?**

Now let's load our modern reference samples

```
modern_phylums = pd.read_csv("../data/curated_metagenomics/modern_sources_phylum.csv", index_col=0)
modern_phylums.head()
```

deo28ad4-7ae6-11e9-a106-68b59976a384

PNP\_Main\_283

PNP\_Validation\_55

G8o275

PNP\_Main\_363

SAMEA7045572

SAMEA7045355

HD-13

EGARoooo1420773\_9002000001423910

SID5428-4

...

A46\_o2\_1FE

TZ\_87532

A94\_o1\_1FE

KHG\_7  
LDK\_4  
KHG\_9  
A48\_o1\_1FE  
KHG\_1  
TZ\_81781  
Ao9\_o1\_1FE  
Bacteroidetes  
0.00000  
17.44332  
82.86400  
69.99087  
31.93081  
51.76204  
53.32801  
74.59667  
8.81074  
26.39694  
...  
1.97760  
1.49601  
67.21410  
4.29848  
68.16890  
38.59709  
14.81828  
10.13908  
57.14031  
11.61544  
Firmicutes  
95.24231

60.47031

16.53946

22.81977

65.23075

41.96928

45.77661

23.51065

54.35341

62.23094

...

76.68499

78.13269

29.72394

33.51772

19.11149

46.87139

72.68136

35.43789

40.57101

24.72113

Proteobacteria

4.49959

0.77098

0.05697

4.07757

0.27316

3.33972

0.02001

1.72865

0.00000

1.81016

...  
16.57250  
0.76159  
2.35058  
9.83772  
5.32392  
0.19699  
3.64655  
17.64151  
0.30580  
56.20177  
Actinobacteria  
0.25809  
10.27631  
0.45187  
1.11902  
2.31075  
2.92715  
0.77667  
0.16403  
36.55138  
1.19951  
...  
3.01814  
19.20468  
0.69913  
46.99479  
7.39093  
14.26365  
5.47750  
36.77145

```
1.16426
```

```
7.40894
```

```
Verrucomicrobia
```

```
0.00000
```

```
0.00784
```

```
0.00000
```

```
1.99276
```

```
0.25451
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.09940
```

```
3.29795
```

```
...
```

```
0.05011
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
0.00000
```

```
5 rows × 200 columns
```

```
modern_species = pd.read_csv("../data/curated_metagenomics/modern_sources_species.csv", index_col=0)

modern_species.head()

deo28ad4-7ae6-11e9-a106-68b59976a384
PNP_Main_283
```

PNP\_Validation\_55  
G80275  
PNP\_Main\_363  
SAMEA7045572  
SAMEA7045355  
HD-13  
EGARoooo1420773\_9002000001423910  
SID5428-4  
...  
A46\_o2\_iFE  
TZ\_87532  
A94\_o1\_iFE  
KHG\_7  
LDK\_4  
KHG\_9  
A48\_o1\_iFE  
KHG\_1  
TZ\_81781  
A09\_o1\_iFE  
Bacteroides vulgatus  
o.o  
0.60446  
1.59911  
4.39085  
0.04494  
4.66505  
2.99431  
29.30325  
1.48560  
0.98818  
...

0.20717

0.0

0.00309

0.48891

0.00000

0.02230

0.00000

0.15112

0.0

0.00836

Bacteroides stercoris

0.0

0.00546

0.00000

0.00000

2.50789

0.00000

20.57498

8.28443

1.23261

0.00000

...

0.00000

0.0

0.00000

0.00693

0.00000

0.02603

0.00000

0.19318

0.0

0.00000  
Acidaminococcus intestini  
0.0  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.32822  
0.00000  
...  
0.00000  
0.0  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
Eubacterium sp CAG 38  
0.0  
0.06712  
0.81149  
0.05247  
0.26027  
0.00000  
0.00000

2.62415  
0.46585  
0.23372  
...  
0.78140  
0.0  
0.00000  
0.00499  
0.00000  
0.02446  
0.00000  
0.00000  
0.0  
0.00000  
Parabacteroides distasonis  
0.0  
1.34931  
2.00672  
5.85067  
0.59019  
7.00027  
1.28075  
0.61758  
0.07383  
2.80355  
...  
0.11423  
0.0  
0.01181  
0.01386  
0.03111

```
0.07463
0.15597
0.07541
0.0
0.01932
5 rows × 200 columns
```

Now, let's merge our ancient sample with the modern data in one single table

```
all_species = ancient_species.merge(modern_species, left_index=True, right_index=True, how='inner')
all_phylums = ancient_phylums.merge(modern_phylums, left_index=True, right_index=True, how='inner')
```

Finally, let's load the metadata

```
metadata = pd.read_csv("../data/metadata/curated_metagenomics_modern_sources.csv")

metadata.head()

study_name
sample_id
subject_id
body_site
antibiotics_current_use
study_condition
disease
age
infant_age
age_category
...
hla_drb11
birth_order
age_twins_started_to_live_apart
zigosity
brinkman_index
alcohol_numeric
```

breastfeeding\_duration  
formula\_first\_day  
ALT  
eGFR  
o  
ShaoY\_2019  
deo28ad4-7ae6-11e9-a106-68b59976a384  
Co1528\_ba  
stool  
no  
control  
healthy  
o.o  
4.o  
newborn  
...  
NaN  
1  
ZeeviD\_2015  
PNP\_Main\_283  
PNP\_Main\_283  
stool

no  
control  
healthy  
NaN  
NaN  
adult  
...  
NaN  
2  
ZeeviD\_2015  
PNP\_Validation\_55  
PNP\_Validation\_55  
stool  
no  
control  
healthy  
NaN  
NaN  
adult  
...  
NaN  
NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

3

VatanenT\_2016

G8o275

To14806

stool

no

control

healthy

1.0

NaN

child

...

NaN

4

```
ZeeviD_2015
PNP_Main_363
PNP_Main_363
stool
no
control
healthy
NaN
NaN
adult
...
NaN
5 rows × 130 columns
```

### 11.3.7 7. Comparing ancient and modern samples

#### 11.3.7.1 7.1 Taxonomic composition

One common plot in microbiome papers is a stacked barplot, often at the phylum or family level.

First, we'll do some renaming, to make the value of the metadata variables a bit easier to understand

```
group_info = (
 metadata['non_westernized']
```

```
.map({'no': 'westernized', 'yes': 'non_westernized'}) # for the non_westernized in the modern samples
.to_frame(name='group').set_index(metadata['sample_id']) # rename the column to group
.reset_index()
.append({'sample_id': 'ERR5766177', 'group': 'ancient'}, ignore_index=True) # add the ancient sample
)
group_info

/var/folders/1c/l1qb09f15jddsh65f6xv1n_r0000gp/T/ipykernel_40830/27419655.py:2:
FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version.
Use pandas.concat instead.
 metadata['non_westernized']

sample_id
group
o
deo28ad4-7ae6-11e9-a106-68b59976a384
westernized
1
PNP_Main_283
westernized
2
PNP_Validation_55
westernized
3
G8o275
westernized
4
PNP_Main_363
westernized
...
...
...
196
A48_o1_1FE
non_westernized
```

```

197
KHG_1
non_westernized

198
TZ_81781
non_westernized

199
Ao9_o1_1FE
non_westernized

200
ERR5766177
ancient

201 rows × 2 columns

```

We need transform our data in [tidy](#) format to plot with [plotnine](#), a python clone of [ggplot](#).

We then add the group information (Westernized, non westernized, or ancient sample), and compute the mean abundance for each phylum.

First we transpose the dataframe to have the samples as index, and the phylums as columns

We then add the metadata information

```

(
 all_phylums
 .transpose()
 .merge(group_info, left_index=True, right_on='sample_id')
)

```

```

Actinobacteria
Apicomplexa
Ascomycota
Bacteroidetes
Basidiomycota
Candidatus Melainabacteria
Chlamydiae

```

Chloroflexi

Cyanobacteria

Deferribacteres

...

Fusobacteria

Lentisphaerae

Planctomycetes

Proteobacteria

Spirochaetes

Synergistetes

Tenericutes

Verrucomicrobia

sample\_id

group

200

20.33151

0.0

0.0

28.42090

0.0

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

0.00000

0.00000

o.o  
o.o  
o.oooooo  
ERR5766177  
ancient  
o  
o.25809  
o.o  
o.o  
o.oooooo  
o.o  
o.o  
o.o  
o.o  
o.o  
o.o  
o.o  
o.o  
...  
o.o  
o.oooooo  
o.o  
4.49959  
o.oooooo  
o.o  
o.o  
o.oooooo  
deo28ad4-7  
westernized  
1  
10.27631  
o.o  
o.o

17.44332

0.0

0.0

0.0

0.0

0.0

0.0

...

0.0

0.01486

0.0

0.77098

0.00000

0.0

0.0

0.00784

PNP\_Main\_283

westernized

2

0.45187

0.0

0.0

82.86400

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

0.05697

0.00000

0.0

0.0

0.00000

PNP\_Validation\_55

westernized

3

1.11902

0.0

0.0

69.99087

0.0

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

4.07757

0.00000

0.0

0.0

1.99276

G80275

westernized

195

14.26365

0.0

0.0

38.59709

0.0

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

0.19699

0.00000

0.0

0.0

0.00000

KHG\_9

non\_westernized

196

5.47750

0.0

0.0

14.81828

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

3.64655

0.09964

0.0

o.o

o.ooooo

A48\_o1\_1FE

non\_westernized

197

36.77145

o.o

o.o

10.13908

o.o

o.o

o.o

o.o

o.o

o.o

...

o.o

o.ooooo

o.o

17.64151

o.ooooo

o.o

o.o

o.ooooo

KHG\_1

non\_westernized

198

1.16426

o.o

o.o

57.14031

0.0

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

0.30580

0.70467

0.0

0.0

0.00000

TZ\_81781

non\_westernized

199

7.40894

0.0

0.0

11.61544

0.0

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

```

o.o
56.20177
o.oooooo
o.o
o.o
o.oooooo
Ao9_o1_1FE
non_westernized
201 rows × 24 columns

```

Now, we need it in the tidy format

```

tidy_phylums = (
 all_phylums
 .transpose()
 .merge(group_info, left_index=True, right_on='sample_id')
 .melt(id_vars=['sample_id', 'group'], value_name='relative_abundance', var_name='Phylum', ignore_index=True)
)

```

Finally, we only want to keep the mean relative abundance for each phylum

```
tidy_phylums = tidy_phylums.groupby(['group', 'Phylum']).mean().reset_index()
```

```

tidy_phylums.groupby('group')['relative_abundance'].sum()

group
ancient 100.000000
non_westernized 99.710255
westernized 99.905089
Name: relative_abundance, dtype: float64

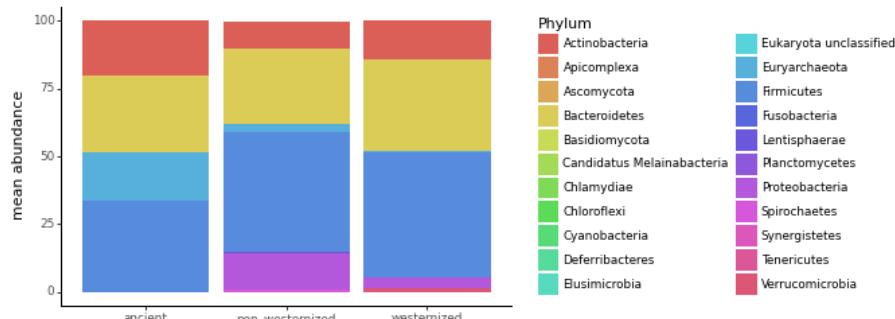
```

```

from plotnine import *

ggplot(tidy_phylums, aes(x='group', y='relative_abundance', fill='Phylum')) \
+ geom_bar(position='stack', stat='identity') \
+ ylab('mean abundance') \
+ xlab("") \
+ theme_classic()

```



```
<ggplot: (406187548)>
```

### 11.3.7.2 7.2 Ecological diversity

#### 7.2.1 Alpha diversity

Alpha diversity is the measure of diversity within each sample. It is used to estimate how many species are present in a sample, and how diverse they are.

We'll use the python library [scikit-bio](#) to compute it, and the [plotnine](#) library (a python port of [ggplot2](#) to visualize the results).

```
import skbio
```

Let's compute the [species richness](#), the [Shannon](#), and [Simpson index of diversity](#) index

```
shannon = skbio.diversity.alpha_diversity(metric='shannon', counts=all_species.transpose())
simpson = skbio.diversity.alpha_diversity(metric='simpson', counts=all_species.transpose())
richness = (all_species != 0).astype(int).sum(axis=0)
alpha_diversity = (shannon.to_frame(name='shannon')
 .merge(simpson.to_frame(name='simpson'), left_index=True, right_index=True)
 .merge(richness.to_frame(name='richness'), left_index=True, right_index=True))
alpha_diversity
```

```
shannon
```

```
simpson
```

```
richness
```

```
ERR5766177
```

```
3.032945
```

```
0.844769
```

```
21
```

```
deo28ad4-7ae6-11e9-a106-68b59976a384
```

0.798112

0.251280

11

PNP\_Main\_283

5.092878

0.954159

118

PNP\_Validation\_55

3.670162

0.812438

72

G8o275

3.831358

0.876712

66

...

...

...

...

KHG\_9

3.884285

0.861683

87

A48\_o1\_1FE

4.377755

0.930024

53

KHG\_1

3.733834

0.875335

108

```
TZ_81781
```

```
2.881856
```

```
0.719491
```

```
44
```

```
Ao9_o1_1FE
```

```
2.982322
```

```
0.719962
```

```
75
```

```
201 rows × 3 columns
```

Let's load the group information from the metadata

```
alpha_diversity = (
 alpha_diversity
 .merge(metadata[['sample_id', 'non_westernized']], left_index=True, right_on='sample_id')
 .set_index('sample_id')
 .rename(columns={'non_westernized': 'group'})
)
alpha_diversity['group'] = alpha_diversity['group'].replace({'yes': 'non_westernized', 'no': 'westernized'})
```

```
alpha_diversity
```

```
shannon
```

```
simpson
```

```
richness
```

```
group
```

```
sample_id
```

```
ERR5766177
```

```
3.032945
```

```
0.844769
```

```
21
```

```
ERR5766177
```

```
deo28ad4-7ae6-11e9-a106-68b59976a384
```

```
0.798112
```

```
0.251280
```

11

westernized

PNP\_Main\_283

5.092878

0.954159

118

westernized

PNP\_Validation\_55

3.670162

0.812438

72

westernized

G8o275

3.831358

0.876712

66

westernized

...

...

...

...

...

KHG\_9

3.884285

0.861683

87

non\_westernized

A48\_o1\_1FE

4.377755

0.930024

53

```
non_westernized
```

```
KHG_1
```

```
3.733834
```

```
0.875335
```

```
108
```

```
non_westernized
```

```
TZ_81781
```

```
2.881856
```

```
0.719491
```

```
44
```

```
non_westernized
```

```
Ao9_o1_1FE
```

```
2.982322
```

```
0.719962
```

```
75
```

```
non_westernized
```

```
201 rows × 4 columns
```

```
alpha_diversity = alpha_diversity.melt(id_vars='group', value_name='alpha diversity', va
```

```
alpha_diversity
```

```
group
```

```
diversity_index
```

```
alpha diversity
```

```
sample_id
```

```
ERR5766177
```

```
ERR5766177
```

```
shannon
```

```
3.032945
```

```
deo28ad4-7ae6-11e9-a106-68b59976a384
```

```
westernized
```

shannon

0.798112

PNP\_Main\_283

westernized

shannon

5.092878

PNP\_Validation\_55

westernized

shannon

3.670162

G8o275

westernized

shannon

3.831358

...

...

...

...

KHG\_9

non\_westernized

richness

87.000000

A48\_o1\_1FE

non\_westernized

richness

53.000000

KHG\_1

non\_westernized

richness

108.000000

TZ\_81781

```
non_westernized
```

```
richness
```

```
44.000000
```

```
A09_o1_1FE
```

```
non_westernized
```

```
richness
```

```
75.000000
```

```
603 rows × 3 columns
```

```
g = ggplot(alpha_diversity, aes(x='group', y='alpha diversity', color='group'))
g += geom_violin()
g += geom_jitter()
g += theme_classic()
g += facet_wrap(~diversity_index, scales = 'free')
g += theme(axis_text_x=element_text(rotation=45, hjust=1))
g += scale_color_manual({'ERR5766177':'#DB5F57','westernized': '#5F57DB','non_westernized': '#55A85A'})
g += theme(subplots_adjust={'wspace': 0.15})
g
```

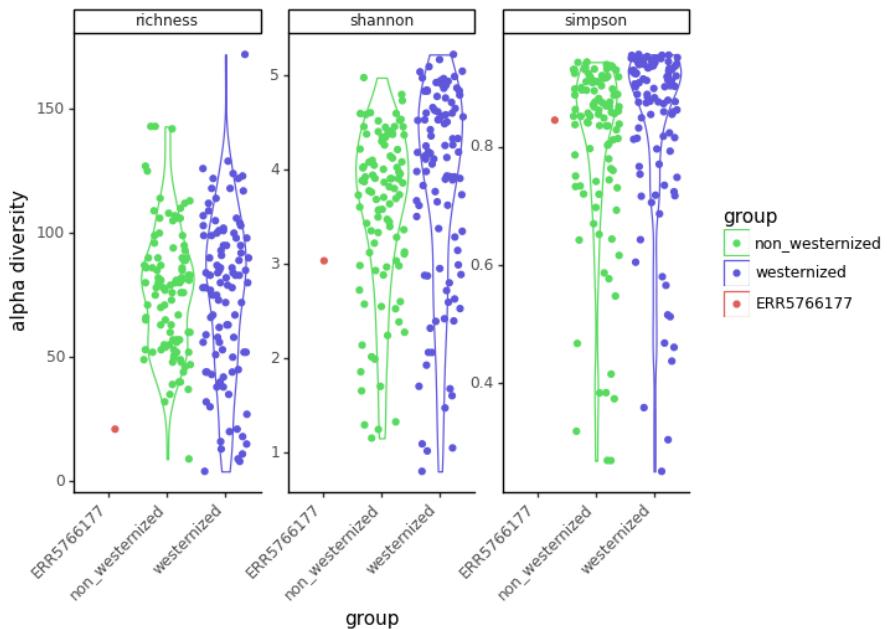


Figure 11.1: png

```
<ggplot: (407407577)>
```

**Pause and think: Why do we observe a smaller species richness and diversity in our sample ?**

### 11.3.7.3 7.2.2 Beta diversity

The Beta diversity is the measure of diversity between a pair of samples. It is used to compare the diversity between samples and see how they relate.

We will compute the beta diversity using the `bray-curtis` dissimilarity

```
beta_diversity = skbio.diversity.beta_diversity(metric='braycurtis', counts=all_species.transpose())
```

We get a distance matrix

```
print(beta_diversity)
```

```
201x201 distance matrix
IDs:
'ERR5766177', 'de028ad4-7ae6-11e9-a106-68b59976a384', 'PNP_Main_283', ...
Data:
[[0. 1. 0.81508134 ... 0.85716612 0.69790092 0.8303726]
 [1. 0. 0.99988327 ... 0.99853413 0.994116 0.99877258]
 [0.81508134 0.99988327 0. ... 0.82311942 0.87202543 0.91363156]
 ...
 [0.85716612 0.99853413 0.82311942 ... 0. 0.84253376 0.76616679]
 [0.69790092 0.994116 0.87202543 ... 0.84253376 0. 0.82409272]
 [0.8303726 0.99877258 0.91363156 ... 0.76616679 0.82409272 0.]]
```

To visualize this distance matrix in a lower dimensional space, we'll use a `PCoA`, which is a method very similar to a PCA, but taking a distance matrix as input.

```
pcoa = skbio.stats.ordination.pcoa(beta_diversity)
```

```
/Users/maxime/mambaforge/envs/summer_school_microbiome/lib/python3.9/site-packages/skbio/stats/ordination.py:100: UserWarning: The result contains negative eigenvalues. Please compare their magnitude with the magnitude of some of the positive ones. If the negative ones are smaller, it's probably safe to ignore them, but if they are large in magnitude, then you may want to consider using a different method.
 See the Notes section for more details. The smallest eigenvalue is -0.25334842745723996 and the largest is 1.0.
```

```
pcoa.samples
```

PC1

PC2

PC3

PC4  
PC5  
PC6  
PC7  
PC8  
PC9  
PC10  
...  
PC192  
PC193  
PC194  
PC195  
PC196  
PC197  
PC198  
PC199  
PC200  
PC201  
ERR5766177  
0.216901  
-0.039778  
0.107412  
0.273272  
0.020540  
0.114876  
-0.256332  
-0.151069  
0.097451  
0.060211  
...  
0.0

O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
deo28ad4-7ae6-11e9-a106-68b59976a384  
-0.099355  
0.145224  
-0.191676  
0.127626  
0.119754  
-0.132209  
-0.097382  
0.036728  
0.081294  
-0.056686  
...  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O  
O.O

PNP\_Main\_283

-0.214108

-0.147466

0.116027

0.090059

0.076644

0.111536

0.092115

0.026477

-0.006460

-0.018592

...

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

PNP\_Validation\_55

0.244827

-0.173996

-0.311197

-0.012836

0.031759

0.117548

0.148715

-0.135641

0.034730

-0.009395

...

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

G8o275

-0.261358

-0.077147

-0.254374

-0.065932

0.088538

0.165970

-0.005260

-0.028739

-0.002016

0.015719

...

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

KHG\_9

0.296057

-0.150300

0.013941

0.032649

-0.147692

0.019663

-0.063120

-0.034453

-0.073514

0.070085

...

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

A48\_o1\_1FE

0.110621

0.030971

0.154231

-0.185961

-0.008512

-0.103420

0.028169

-0.044530

0.041902

0.068597

...

0.0

0.0



0.405716

-0.139297

-0.075026

-0.079716

-0.053264

-0.119271

0.068261

-0.018821

0.198152

-0.012792

...

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

Ao9\_01\_lFE

0.089101

0.471135

0.069629

-0.125644

-0.036793

0.115151

0.060507

-0.0000912

-0.027239

-0.138436

•

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

201 rows × 201 columns

Let's look at the variance explained by the first axes by using a scree plot

```
var_explained = pcoa.proportion_explained[:9].to_frame(name='variance explained').reset_index()

ggplot(var_explained, aes(x='PC', y='variance explained', group=1)) \
+ geom_point() \
+ geom_line() \
+ theme_classic()

gplot: (407531271)>
```

In this scree plot, we're looking for the "elbow", where there is a drop in the slope. Here, it seems that most of the variance is captured by the 3 first principal components

```
pcoa_embed = pcoa.samples[['PC1', 'PC2', 'PC3']].rename_axis('sample').reset_index()

pcoa_embed = (
 pcoa_embed
 .merge(metadata[['sample_id', 'non_westernized']], left_on='sample', right_on='sample')
 .drop('sample_id', axis=1)
 .rename(columns={'non_westernized': 'group'})
)
pcoa_embed['group'] = pcoa_embed['group'].replace({'yes': 'non_westernized', 'no': 'westernized'})
```

Let's first look at these components with 2D plots

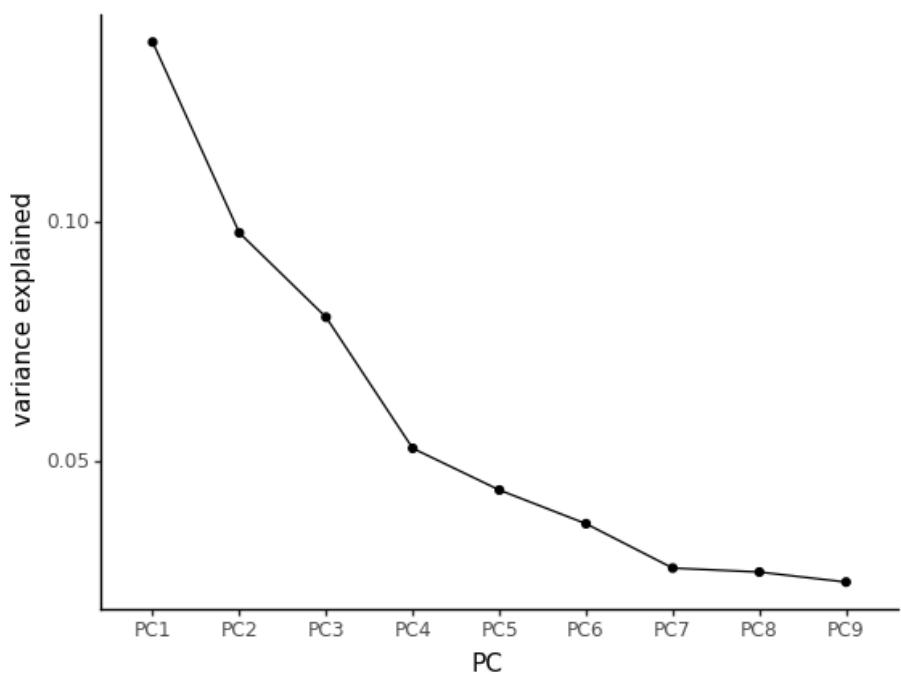


Figure 11.2: png



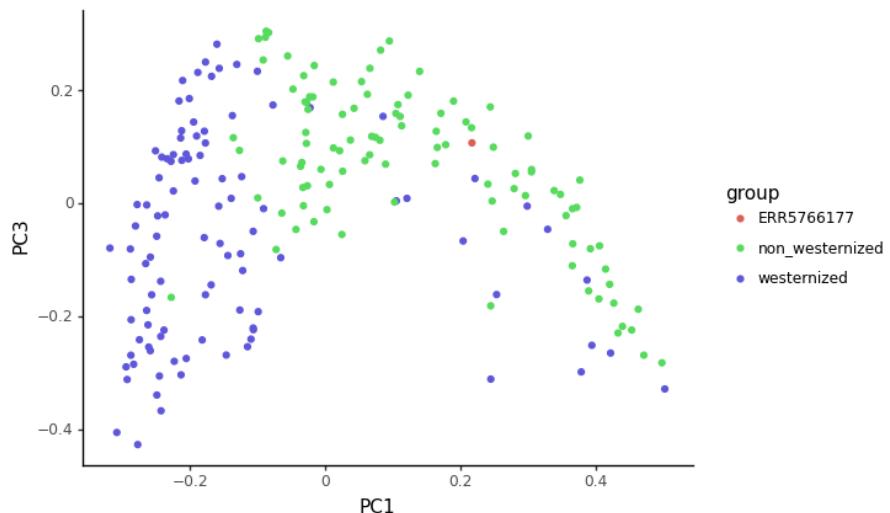


Figure 11.4: png

**! Important**

3D PLOT HERE NOT DISPLAYED DUE TO RENDERING LIMITATIONS - PLEASE SEE JUPYTER NOTEBOOK

**Pause and think: How do you think this embedding represents how our sample relates to modern reference samples ?**

We can also visualize this distance matrix using a clustered heatmap, where pairs of sample with a small beta diversity are clustered together

```
import seaborn as sns
import scipy.spatial as sp, scipy.cluster.hierarchy as hc

pcoa_embed['colour'] = pcoa_embed['group'].map({'ERR5766177': '#DB5F57', 'westernized': '#5F57DB', 'non_westernized': '#5F57DB'})

linkage = hc.linkage(sp.distance.squareform(beta_diversity.to_data_frame()), method='average')

sns.clustermap(
 beta_diversity.to_data_frame(),
 row_linkage=linkage,
 col_linkage=linkage,
 row_colors = pcoa_embed['colour'].to_list()
```

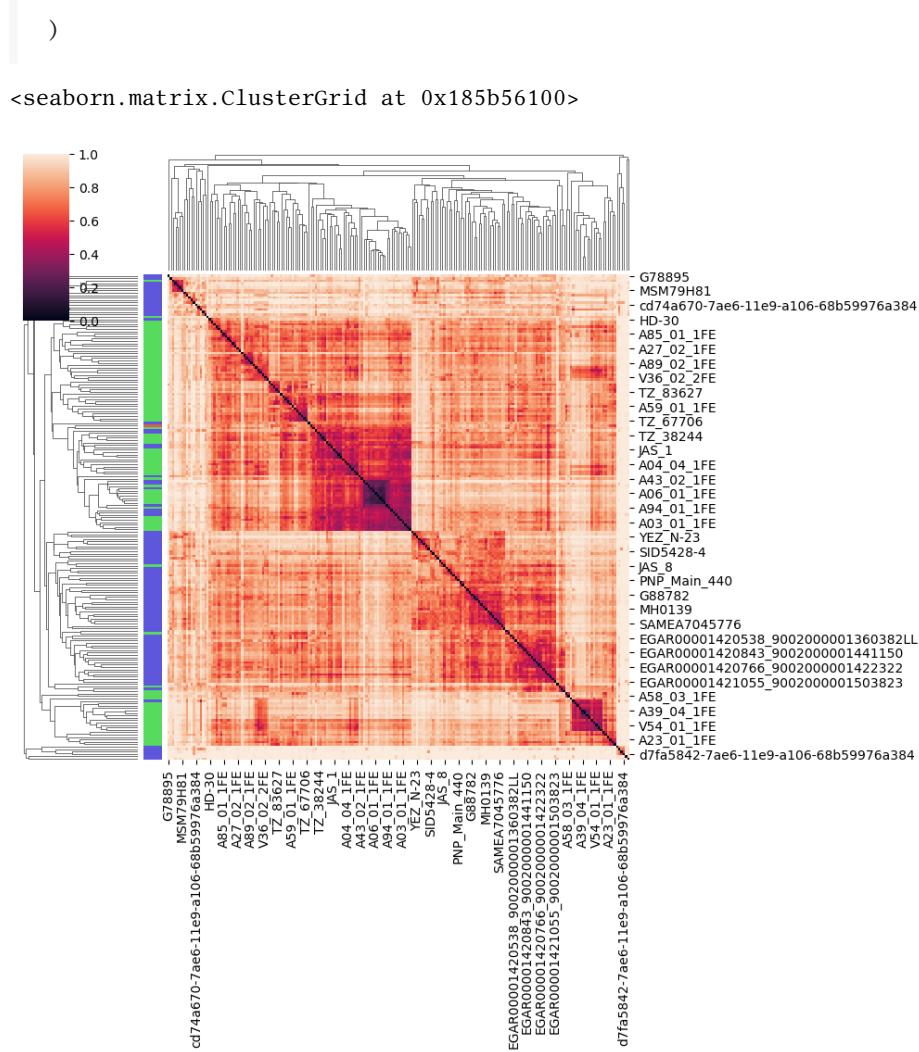


Figure 11.5: png

### 11.3.8 8. Additional steps

#### 11.3.8.1 8.1 Source tracking

Sourcetracker is a program that can estimate the proportion of different sources (reference biomes) contained in a sample (a sink). However, because of the statistical framework that it uses (MCMC with [Gibbs sampling](#)), we recommend to limit the number of source samples to greatly reduce runtime

First, you will need to transform our relative abundance table to counts for Source-

## Tracker

```
all_species_counts = all_species.multiply(1000000).astype(int)

min_count = all_species_counts.sum(axis=0).min()
min_count

95327810
```

Exporting the count table to tsv

```
all_species_counts.to_csv("../results/sourcetracker2/all_species_counts.tsv", sep="\t", index_label=0)
```

Converting to `biom` format

```
!biom convert -i ../results/sourcetracker2/all_species_counts.tsv \
-o ../results/sourcetracker2/all_species_counts.biom \
--table-type="Taxon table" --to-hdf5
```

Converting the metadata to Sourtracker format

```
st2_metadata = metadata[['sample_id', 'non_westernized']].rename(columns={'non_westernized':'Env', 'Env': 'Env'})[['Env', 'Env']]
st2_metadata['SourceSink'] = ['source'] * st2_metadata.shape[0]
```

We subset it to select only 10 samples from each source

```
st2_metadata = st2_metadata.groupby('Env').sample(10).reset_index()

st2_metadata = st2_metadata.append({'#SampleID': 'ERR5766177', 'Env': '-', 'SourceSink': 'sink'}, ignore_index=True)[['#SampleID', 'SourceSink', 'Env']].set_index('#SampleID')

/var/folders/1c/11qb09f15jddsh65f6xv1n_r0000gp/T/ipykernel_40830/2882312005.py:1: FutureWarning:
```

The `frame.append` method is deprecated and will be removed from pandas in a future version. Use `pandas.concat`

```
st2_metadata.to_csv("../results/sourcetracker2/labels_st2.tsv", sep="\t", index_label='#SampleID')

sourcetracker2 gibbs \
-i ../results/sourcetracker2/all_species_counts.biom \
-m ../results/sourcetracker2/labels_st2.tsv \
-o ../results/sourcetracker2/st2 \
--source_rarefaction_depth 95327810 \
```

```
--sink_rarefaction_depth 95327810 \
--jobs 10
```

**Because SourceTracker is relying on MCMC sampling, it can very slow to run (which is why we won't run it here)**

Among alternative faster solutions for source tracking are (among others):

- FEAST ([article](#), [code](#)),
- Sourcepredict ([article](#), [code](#))

#### 11.3.8.2 8.2 The next steps:

- Damage Analysis ([mapDamage](#), [DamageProfiler](#), [PyDamage](#))
- Assembly ([megahit](#), [metaSPAdes](#)), binning ([metabat2](#), [maxbin2](#), [dastool](#)), and bin validation ([checkm](#), [gunc](#))
- Functional analysis ([Prokka](#), [Humann](#))
- Differential abundance ([Maaslin2](#), [Lefse](#), [Songbird](#), GLM, Mixed effect models).  
Nice review by [Wallen 2021](#)
- genotyping
- Phylogenies
- ...

## Chapter 12

# Functional Profiling



### Tip

For this chapter's exercises, if not already performed, you will need to create the **conda environment** from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate phylogenomics-functional
```

### 12.1 Lecture

PDF version of these slides can be downloaded from [here](#).

### 12.2 Preparation

The data and conda environment `.yaml` file for this practical session can be downloaded from here: <https://doi.org/10.5281/zenodo.6983188>. See instructions on page.

Change into the session directory

```
cd /<path>/<to>/5c-functional-genomics/
```

Load the conda environment.

```
conda activate phylogenomics-functional
```

Open R Studio from within the conda environment, and we can load the required libraries for this walkthrough.

```
library(mixOmics) ## For PCA generation

Utility packages (pretty stuff)
library(knitr)
library(data.table)
library(tidyverse)
library(gplots)
library(ggrepel)
library(viridis)
library(patchwork)
```

## 12.3 HUMAnN<sub>3</sub> Pathways

First, we need to run HUMMAn3 to align reads against gene databases and convert to gene family names counts.

### ⚠ Warning

We will not run HUMAnN<sub>3</sub> here as it requires very large databases and takes a long time to run, so we will give you the commands you normally would run but we provide with you pre-made results files before you (see below).

```
DO NOT RUN!

run humann3
humann3 --input file.fastq --output output --threads <threads>

join all output tables (can do for both gene and pathways)
humann_join_tables -i output/ -o gene_families_joined.tsv --file_name unmapped_gene_fami

normalize the output (here by tss - total sum scaling, can do for both gene and pathway)
humann_renorm_table --input gene_families_joined.tsv --output gene_families_joined_cpm.t

regroup the table to combine gene families (standardise gene family IDs across taxa)
humann_regroup_table --input gene_families_joined_cpm.tsv --output gene_families_joined_cpm

give the gene families names
humann_rename_table --input gene_families_joined_cpm_ur90rxn.tsv --output gene_families_joined_cpm_ur90rxn
```

## 12.4 humann3 tables

First lets load a pre-made pathway abundance file

```
load the species and genus tables generated with humann3
humann3_path_full <- fread("./pathabundance_joined_cpm.tsv")
humann3_path_full <- as_tibble(humann3_path_full)

clean the file names
humann3_path_full <- rename(humann3_path_full, Pathway = `# Pathway`)
colnames(humann3_path_full) <- gsub(".unmapped_Abundance","", colnames(humann3_path_full))
colnames(humann3_path_full) <- gsub(".SG1","", colnames(humann3_path_full))

remove unmapped and ungrouped reads
humann3_path <- humann3_path_full %>% filter(!str_detect(Pathway, "UNMAPPED|UNINTEGRATED"))
```

Then lets load associated sample metadata to help make it easier for comparative analysis and make actual informative inferences.

The data being used in this session, is from [Velsko et al. 2022 \(PNAS Nexus\)](#), where we tried to find associations between dental pathologies and taxonomic and genome content. We had a large skeletal collection from a single site in the Netherlands, with a lot of osteological metadata. The study aimed to see if there were any links between the oral microbiome and groups of dental pathologies.

```
load the metadata file
full_metadata <- fread("full_combined_metadata.tsv")

Example of metadata
tibble(full_metadata %>%
 filter(Site_code == "MID") %>%
 select(Site, Time_period, Library_ID, Sequencing_instrument, Pipenotch, Max_Perio_Score, `%teet`)
```

First step: we can pre-define various functions for generate PCAs we will use downstream - you don't have to worry about these too much they are just custom functions to quickly plot PCAs from a mixOmics PCA output object with ggplot, but we leave the code here for if you're curious.

```
plot PCA with colored dots and the title including the # of species or genera
plot_pca <- function(df, pc1, pc2, color_group, shape_group, ncomps) {
 metadata_group_colors <- get(paste(color_group, "_colors", sep = ""))
 metadata_group_shapes <- get(paste(shape_group, "_shapes", sep = ""))

 pca.list <- mixOmics::pca(df, ncomp = ncomps, logratio = 'CLR')
```

```

Pull out loadings
exp_var <- paste0(round(pca.list$explained_variance * 100, 2), "%")
df_X <- pca.list$variates$X %>%
 as.data.frame() %>%
 rownames_to_column("Library_ID") %>%
 inner_join(full_metadata, by = "Library_ID")

color_group = df_X[[color_group]]
shape_group = df_X[[shape_group]]

Selecting which PCs to plot
if (pc1 == 'PC1') {
 pc1 <- df_X$PC1
 exp_var_pc1 <- exp_var[1]
 xaxis <- c("PC1")
} else if (pc1 == 'PC2') {
 pc1 <- df_X$PC2
 exp_var_pc1 <- exp_var[2]
 xaxis <- c("PC2")
} else if (pc1 == 'PC3') {
 pc1 <- df_X$PC3
 exp_var_pc1 <- exp_var[3]
 xaxis <- c("PC3")
}

if (pc2 == 'PC1') {
 pc2 <- df_X$PC1
 exp_var_pc2 <- exp_var[1]
 yaxis <- c("PC1")
} else if (pc2 == 'PC2') {
 pc2 <- df_X$PC2
 exp_var_pc2 <- exp_var[2]
 yaxis <- c("PC2")
} else if (pc2 == 'PC3') {
 pc2 <- df_X$PC3
 exp_var_pc2 <- exp_var[3]
 yaxis <- c("PC3")
}

Generate figure
pca_plot <- ggplot(df_X, aes(pc1, pc2)) +
 geom_point(aes(fill = color_group, shape = shape_group), size = 4.5, stroke = 0.3) +
 scale_fill_manual(values = metadata_group_colors) +
 scale_shape_manual(values = metadata_group_shapes) +

```

```

stat_ellipse() +
xlab(paste(xaxis, " - ", exp_var_pc1)) +
ylab(paste(yaxis, " - ", exp_var_pc2)) +
theme_minimal(base_size = 16) +
theme(text = element_text(size=16)) +
theme(legend.title = element_blank(),
 legend.key.size = unit(2,"mm"),
 legend.text = element_text(size = 6)) +
theme(legend.position = "top")

return(pca_plot)
}

for continuous data
plot_pca_cont <- function(df, pc1, pc2, color_group, shape_group, ncomps, title_text) {

pca.list <- mixOmics::pca(df, ncomp = ncomps, logratio = 'CLR')

exp_var <- paste0(round(pca.list$explained_variance * 100, 2), "%")
df_X <- pca.list$variates$X %>%
 as.data.frame() %>%
 rownames_to_column("Library_ID") %>%
 inner_join(full_metadata, by = "Library_ID")

color_group = df_X[[color_group]]
shape_group = df_X[[shape_group]]

if (pc1 == 'PC1') {
 pc1 <- df_X$PC1
 exp_var_pc1 <- exp_var[1]
 xaxis <- c("PC1")
} else if (pc1 == 'PC2') {
 pc1 <- df_X$PC2
 exp_var_pc1 <- exp_var[2]
 xaxis <- c("PC2")
} else if (pc1 == 'PC3') {
 pc1 <- df_X$PC3
 exp_var_pc1 <- exp_var[3]
 xaxis <- c("PC3")
}

if (pc2 == 'PC1') {
 pc2 <- df_X$PC1
 exp_var_pc2 <- exp_var[1]
}

```

```

 yaxis <- c("PC1")
 } else if (pc2 == 'PC2') {
 pc2 <- df_X$PC2
 exp_var_pc2 <- exp_var[2]
 yaxis <- c("PC2")
 } else if (pc2 == 'PC3') {
 pc2 <- df_X$PC3
 exp_var_pc2 <- exp_var[3]
 yaxis <- c("PC3")
 }

pca_plot <- ggplot(df_X, aes(pc1, pc2, fill = color_group, shape = shape_group)) +
 geom_point(size = 5, color = "black") +
 scale_fill_viridis_c(option = "C") +
 scale_shape_manual(values = c(24, 21)) +
 # stat_ellipse() +
 xlab(paste(xaxis, " - ", exp_var_pc1)) +
 ylab(paste(yaxis, " - ", exp_var_pc2)) +
 theme_minimal(base_size = 16) +
 theme(text = element_text(size=16)) +
 theme(legend.title = element_blank(),
 legend.key.size = unit(2,"mm"),
 legend.text = element_text(size = 6)) +
 theme(legend.position = "top") +
 ggtitle(title_text) + theme(plot.title = element_text(size = 10))

return(pca_plot)
}

plot_pca_bi <- function(df, pc1, pc2, metadata_group, columntitle) {
 metadata_group_colors <- get(paste(metadata_group, "_colors", sep = ""))
 metadata_group_shapes <- get(paste(metadata_group, "_shapes", sep = ""))

 arrow_pc <- enquo(columntitle)

 exp_var <- paste0(round(df$explained_variance * 100, 2), "%") # explained variance f

 # select only the PCs from the PCA and add metadata
 df_X <- df$variates$X %>%
 as.data.frame() %>%
 rownames_to_column("Library_ID") %>%
 inner_join(full_metadata, by = "Library_ID")

 metadata_group = df_X[[metadata_group]]
}

```

```

corr_lam <- df$sdev[c("PC1", "PC2", "PC3")] * sqrt(nrow(df_X))

df_X <- df_X %>%
 mutate(PC1 = PC1 / corr_lam[1],
 PC2 = PC2 / corr_lam[2],
 PC3 = PC3 / corr_lam[3])

select the correct PC column and explained variance for PC1
if (pc1 == 'PC1') {
 Pc1 <- df_X$PC1
 exp_var_pc1 <- exp_var[1]
 xaxis <- c("PC1")
} else if (pc1 == 'PC2') {
 Pc1 <- df_X$PC2
 exp_var_pc1 <- exp_var[2]
 xaxis <- c("PC2")
} else if (pc1 == 'PC3') {
 Pc1 <- df_X$PC3
 exp_var_pc1 <- exp_var[3]
 xaxis <- c("PC3")
}

select the correct PC column and explained variance for PC2
if (pc2 == 'PC1') {
 Pc2 <- df_X$PC1
 exp_var_pc2 <- exp_var[1]
 yaxis <- c("PC1")
} else if (pc2 == 'PC2') {
 Pc2 <- df_X$PC2
 exp_var_pc2 <- exp_var[2]
 yaxis <- c("PC2")
} else if (pc2 == 'PC3') {
 Pc2 <- df_X$PC3
 exp_var_pc2 <- exp_var[3]
 yaxis <- c("PC3")
}

Identify the 10 pathways that have highest positive and negative loadings in the selected PC
pws_10 <- df$loadings$X %>%
 as.data.frame(.) %>%
 rownames_to_column(var = "Pathway") %>%
 separate(Pathway, into = "Pathway", sep = ":", extra = "drop") %>%
 top_n(10, !arrow_pc)

```

```

neg_10 <- df$loadings$X %>%
 as.data.frame(.) %>%
 rownames_to_column(var = "Pathway") %>%
 separate(Pathway, into = "Pathway", sep = ":", extra = "drop") %>%
 top_n(-10, !arrow_pc)

pca_plot_bi <- ggplot(df_X, aes(x = Pc1, y = Pc2)) +
 geom_point(size = 3.5, aes(shape = metadata_group, fill = metadata_group)) +
 geom_segment(data = pws_10,
 aes(xend = get(paste(pc1)), yend = get(paste(pc2))),
 x = 0, y = 0, colour = "black",
 size = 0.5,
 arrow = arrow(length = unit(0.03, "npc"))) +
 geom_label_repel(data = pws_10,
 aes(x = get(paste(pc1)), y = get(paste(pc2)), label = Pathway),
 size = 2.5, colour = "grey20", label.padding = 0.2, force = 5, max.overlaps = 12)
 geom_segment(data = neg_10,
 aes(xend = get(paste(pc1)), yend = get(paste(pc2))),
 x = 0, y = 0, colour = "grey50",
 size = 0.5,
 arrow = arrow(length = unit(0.03, "npc"))) +
 geom_label_repel(data = neg_10,
 aes(x = get(paste(pc1)), y = get(paste(pc2)), label = Pathway),
 size = 2.5, colour = "grey20", label.padding = 0.2, max.overlaps = 12)
 labs(x = paste(xaxis, " - ", exp_var_pc1),
 y = paste(yaxis, " - ", exp_var_pc2)) +
 scale_fill_manual(values = metadata_group_colors) +
 scale_shape_manual(values = metadata_group_shapes) +
 theme_minimal() + theme(text = element_text(size = 16)) +
 theme(text = element_text(size=16)) +
 theme(legend.position = "top")

return(pca_plot_bi)
}

```

As we are dealing with aDNA, and we often have bad samples, its sometimes interesting to see differences between well/badly preserved samples at all stages of analysis.

Therefore we may generate results for all samples. However for actual analysis where we want to interpret biological differences, should exclude outliers (in this case highly contaminated samples - as identified by the decontam package - see [Velsko et al. 2022 – PNAS Nexus](#) for more details).

We can make a list the outliers from the previous authentication analyses.

```

outliers_mpa3 <- c("EXB059.A2101", "EXB059.A2501", "EXB015.A3301", "EXB034.A2701",
 "EXB059.A2201", "EXB059.A2301", "EXB059.A2401", "LIB058.A0103", "LIB058.A0106", "LIB0
poor_samples_mpa3 <- c("CS28", "CS38", "CSN", "ELR003.A0101", "ELR010.A0101",
 "KT09calc", "MID024.A0101", "MID063.A0101", "MID092.A0101")
outliersF <- str_c(outliers_mpa3, collapse = "|")

```

## 12.5 Sample Clustering with PCA

### 12.5.1 Pathway abundance analyses

Once we've removed outlier samples, our first simple question is - what is the functional relationships of the groups?

Can we already see distinctive patterns between the different groups in our dataset?

To do this lets clean up the data a bit (cleaning names, removing samples with no metadata etc.), normalise (via a 'centered-log-ratio' transform ), and run a PCA.

Once we've done this we should always check our PCA's Scree plot first.

```

humann3_path_11 <- humann3_path %>%
 filter(!str_detect(Pathway, "\\|")) %>%
 # no full_metadata, remove these
 select(-c("MID025.A0101", "MID033.A0101", "MID052.A0101", "MID056.A0101",
 "MID065.A0101", "MID068.A0101", "MID076.A0101", "MID078.A0101")) %>%
 # remove poorly preserved samples
 select(-c("MID024.A0101", "MID063.A0101", "MID092.A0101")) %>%
 select(-matches("EXB|LIB")) %>%
 # inner_join(., humann3_path.decontam_noblocks_presence_more_30, by = "Pathway") %>%
 gather("Library_ID", "Counts", 2:ncol(.)) %>%
 mutate(Counts = Counts + 1) %>%
 spread(Pathway, Counts) %>%
 column_to_rownames("Library_ID")

prepare to run a PCA
check the number of components to retain by tuning the PCA
mixOmics::tune.pca(humann3_path_11, logratio = 'CLR')

humann3_all_otu.pca <- mixOmics::pca(humann3_path_11, ncomp = 3, logratio = 'CLR')
humann3_all_pca_values <- humann3_all_otu.pca$variates$X %>%
 as.data.frame() %>%
 rownames_to_column("Library_ID") %>%

```

```
inner_join(., full_metadata, by = "Library_ID")
```

We can see the first couple of PCs in the scree plot account for a good chunk of the variation of our dataset, so lets visualise the PCA itself.

We visualise the PCA with one of our custom functions defined above, and colour by the Pipe notch metadata column.

```
pipenotch colors/shapes
Pipenotch_colors = c("#311068", "#C83E73")
Pipenotch_shapes = c(16, 17)

by minimum number of pipenotches
pipenotch <- plot_pca_cont(humann3_path_11, "PC1", "PC2", "Min_no_Pipe_notches", "Pipenotch"
pipenotch
```

We can see there is a slight separation between the groups, but how do we find out which pathways are maybe driving this pattern?

For this we can generate a PCA bi-plot which show what loadings are driving the spread of the samples.

```
Pipenotch_colors = c("#311068", "#C83E73")
Pipenotch_shapes = c(24, 21)

biplot <- plot_pca_bi(humann3_all_otu.pca, "PC1", "PC2", "Pipenotch", PC1)
biplot
```

From the biplot we can see which pathways are differentiating along PC1.

We can pull these IDs out to find out what pathways there are from the biplot object itself.

```
make a table of the pathways to save, to use again later in another R notebook
humann3_pathway_biplot_list <- biplot$plot_env$pws_10 %>% arrange(desc(PC1)) %>% select(
humann3_pathway_biplot_list <- humann3_pathway_biplot_list %>%
 bind_rows(biplot$plot_env$neg_10 %>% arrange(desc(PC1)) %>% select(Pathway, PC1, PC2)%>%
```

### 12.5.2 Species contributions to pathways

However, this ID numbers aren't very informative to us. At this point we have to do a bit of literature review/database scraping to pull the human-readable names/descriptions of the IDs - which we have already done for you.

We can load these back into our environment

```
PC biplot loading top 10s
humann3_pathway_biplot_list <- fread("./humann3_pathway_biplot_list.tsv")
humann3_pathway_biplot_list <- humann3_pathway_biplot_list %>%
 rename(Pathway = pathway) %>%
 mutate(Path = sapply(Pathway, function(f) {
 unlist(str_split(f, ":"))[1]
 })) %>%
 select(Pathway, Path, everything()) %>%
remove 3 of the 4 ubiquinol pathways w/identical loadings
 filter(!str_detect(Pathway, "5856|5857|6708"))

tibble(humann3_pathway_biplot_list)
```

We now have the pathway ID, and a pathway description for each of the loadings of the PCA.

## PC1

While we have the pathways, we don't *who* contributed these.

For this, we can join our pathway table back onto the original output from HUMANn3 we loaded at the beginning, which includes the taxa information.

```
list the 10 orthologs with strongest loading in PC1 + values
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%
 filter(Direction == "PC1+") %>%
 pull(Path) %>%
 str_c(., collapse = "|") # need this format for filtering in the next step

select only those 10 pathways from the list, and split the column with names into 3 (Pathway, Genus, Species)
humann3_path_pc1pws <- humann3_path %>%
 filter(str_detect(Pathway, humann3_path_biplot_pc)) %>%
 filter(str_detect(Pathway, "\\\\|")) %>%
 gather("SampleID", "CPM", 2:ncol(.)) %>%
 mutate(Pathway = str_replace_all(Pathway, "\\\\.s___.", "|s___.")) %>%
 separate(., Pathway, into = c("Pathway", "Genus", "Species"), sep = "\\\\" |") %>%
 mutate(Species = replace_na(Species, "unclassified"),
 Genus = str_replace_all(Genus, "g___.", ""),
 Species = str_replace_all(Species, "s___.", "")) %>%
 inner_join(., humann3_pathway_biplot_list %>%
 select(Pathway, Path) %>%
 distinct(.), by = "Pathway") %>%
 inner_join(., humann3_pathway_biplot_list %>%
 filter(Direction == "PC1+")) %>%
```

```

 select(Path), by = "Path") %>%
select(-Pathway) %>%
 select(Path, everything()) %>%
 arrange(Path)

tibble(humann3_path_pc1pws)

```

We can now see who contributed which pathway, and also the abundance information (CPM)!

Given many taxa may contribute to the same pathway, we may want to see which taxa are more ‘dominantly’ contributing to this.

For this we can calculate of all copies of a given pathway what fraction comes from which taxa (you can imagine this like ‘depth’ coverage in genomic analysis), based on the percentage of the total copies per million for that pathway.

```

calculate the % for each pathway contributed by each genus
humann3_path_pc1pws_stats <- humann3_path_pc1pws %>%
 group_by(Path, Genus) %>%
 summarise(Sum = sum(CPM)) %>%
 mutate(Percent = Sum/sum(Sum)*100) %>%
 ungroup(.)

create the list of 10 orthologs again, but don't collapse the list as above
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%
 filter(Direction == "PC1+") %>%
 arrange(Path) %>%
 pull(Path)

calculate the total % of all genera that contribute < X% to each ortholog
humann3_path_pc1pws_stats_extra <- lapply(humann3_path_biplot_pc, function(eclass) {
 high_percent <- humann3_path_pc1pws_stats %>%
 filter(Path == eclass) %>%
 filter(Percent < 5) %>%
 summarise(Remaining = sum(Percent)) %>%
 mutate(Path = eclass,
 Genus = "Other")
}) %>%
 bind_rows(.)

add this additional % to the main table
humann3_path_pcbi_bar_df <- humann3_path_pc1pws_stats_extra %>%
 rename(Percent = Remaining) %>%
 bind_rows(., humann3_path_pc1pws_stats %>%

```

```

 select(-Sum)) %>%
select(Path, Genus, Percent) %>%
mutate(Direction = "PC1+") %>%
distinct()

```

And we can visualize the contributors to the top 10 pathways driving the main variation along PC1 (with the assumption these maybe the most biological significant, and to reduce the numbers of pathways we have to research).

For the loadings falling in the positive direction of the PC1:

```

plot the values in a bar chart
paths_sp_pc1 <- humann3_path_pcbi_bar_df %>%
 # filter(Direction == "PC1+", Genus != "Other") %>% # removing Other plots all species/unassigned
 filter(Percent >= 5 | (Percent <= 5 & Genus == "Other")) %>% # filter out the genera with % < 5,
 # filter(Percent >= 5) %>% # filter out the genera with % < 5, but keep Other < 5
 mutate(
 Genus = fct_relevel(Genus, "Other", "unclassified", "Aggregatibacter", "Capnocytophaga", "Cardiobac
 "Eikenella", "Haemophilus", "Kingella", "Lautropia", "Neisseria", "Ottowia", "Str
 Path = fct_relevel(Path, humann3_pathway_biplot_list %>%
 filter(Direction == "PC1+")) %>%
 pull(Path))) %>%
 ggplot(., aes(x=Path, y=Percent, fill = Genus)) +
 geom_bar(stat = "identity") +
 theme_minimal() +
 scale_fill_manual(values = c("#0D0887FF", "#969696", "#5D01A6FF", "#7E03A8FF",
 "#9C179EFF", "#B52F8cff", "#CC4678FF", "#DE5F65FF",
 "#ED7953FF", "#F89441FF", "#FDB32FFF", "#FBD424FF", "#F0F921FF")) +
 theme(text = element_text(size=18),
 axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)) +
 ylab("Percent") +
 ggtitle("Metacyc pathways - PC1 positive") +
 theme(title = element_text(size=10))

viridis_pal(option = "B")(13)
paths_sp_pc1

```

### Warning

PWY-5345 has no species assignment to that pathway.

And the negative loadings:

```

list the 10 orthologs with strongest loading in PC1 + values
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%
 filter(Direction == "PC1-") %>%
 arrange(Path) %>%
 pull(Path) %>%
 str_c(., collapse = "|") # need this format for filtering in the next step

select only those 10 orthologs from the list, and split the column with names into 3 (
humann3_path_pc1neg <- humann3_path %>%
 filter(str_detect(Pathway, humann3_path_biplot_pc)) %>%
 filter(str_detect(Pathway, "\\\\|")) %>%
 gather("SampleID", "CPM", 2:ncol(.)) %>%
 mutate(Pathway = str_replace_all(Pathway, "\\\\.s_\\", "|s_\\")) %>%
 separate(., Pathway, into = c("Pathway", "Genus", "Species"), sep = "\\\\|") %>%
 mutate(Species = replace_na(Species, "unclassified"),
 Genus = str_replace_all(Genus, "g_\\", ""),
 Species = str_replace_all(Species, "s_\\", "")) %>%
 inner_join(., humann3_pathway_biplot_list %>%
 select(Pathway, Path) %>%
 distinct(., by = "Pathway")) %>%
 inner_join(., humann3_pathway_biplot_list %>%
 filter(Direction == "PC1-") %>%
 select(Path), by = "Path") %>%
 select(-Pathway) %>%
 select(Path, everything()) %>%
 arrange(Path)

calculate the % for each ortholog contributed by each genus
humann3_path_pc1neg_stats <- humann3_path_pc1neg %>%
 group_by(Path, Genus) %>%
 summarize(Sum = sum(CPM)) %>%
 mutate(Percent = Sum/sum(Sum)*100) %>%
 ungroup(.)

create the list of 10 orthologs again, but don't collapse the list as above
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%
 filter(Direction == "PC1-") %>%
 arrange(Path) %>%
 pull(Path)

calculate the total % of all genera that contribute < X% to each ortholog
humann3_path_pc1neg_stats_extra <- lapply(humann3_path_biplot_pc, function(eclass) {
 high_percent <- humann3_path_pc1neg_stats %>%
 filter(Path == eclass) %>%

```

```

filter(Percent < 5) %>%
summarise(Remaining = sum(Percent)) %>%
mutate(Path = eclass,
Genus = "Other")
}) %>%
bind_rows(.)

add this additional % to the main table
humann3_path_pcbi_bar_df <- humann3_path_pcbi_bar_df %>%
bind_rows(humann3_path_pc1neg_stats_extra %>%
rename(Percent = Remaining) %>%
bind_rows(., humann3_path_pc1neg_stats %>%
select(-Sum)) %>%
select(Path, Genus, Percent) %>%
mutate(Direction = "PC1-")) %>%
distinct()

plot the values in a bar chart
paths_sp_pc2 <- humann3_path_pcbi_bar_df %>%
filter(Direction == "PC1-") %>% # removing Other plots all species/unassigned - no need to filter
filter(Percent >= 5 | (Percent <= 5 & Genus == "Other")) %>% # filter out the genera with % < 5,
mutate(Genus = fct_relevel(Genus, "Other", "unclassified", "Desulfobulbus", "Desulfomicrobium", "Meth
Path = fct_relevel(Path, humann3_pathway_biplot_list %>%
filter(Direction == "PC1-") %>%
pull(Path))) %>%
ggplot(., aes(x=Path, y=Percent, fill = Genus)) +
geom_bar(stat = "identity") +
theme_minimal() +
scale_fill_manual(values = c("#0D0887FF", "#969696", "#B52F8CFF", "#ED7953FF", "#FCFFA4FF")) +
theme(text = element_text(size=18),
axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)) +
facet_wrap(~pathrtholog, nrow=2) +
ylab("Percent") +
ggtitle("Metacyc pathways - PC1 negative") +
theme(title = element_text(size=12))

paths_sp_pc2

```

### 12.5.3 Final Visualisation

Finally, we can stick the biplot and the taxon contribution plots together!

```

h3biplots <- biplot + paths_sp_pc2 + paths_sp_pc1 +
plot_layout(widths = c(2, 1, 1))

```

```
ggsave("./h3_paths_biplots.pdf", plot = h3biplots,
 device = "pdf", scale = 1, width = 20, height = 9.25, units = c("in"), dpi = 300

system(paste0('firefox "h3_paths_biplots.pdf')))
```

This allows us to evaluate all the information together.

From this point onwards, we would have to do manual research/literature reviews into each of the pathways, see if they make ‘sense’ to the sample type and associated groups of samples, and evaluate whether they are interesting or not..

# Chapter 13

## Introduction to *de novo* Genome Assembly



Tip

For this chapter's exercises, if not already performed, you will need to create the **conda environment** from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate microbial-genomics
```

### 13.0.1 Lecture

PDF version of these slides can be downloaded from [here](#).

### 13.1 Introduction



## **Part IV**

# **Ancient Genomics**



A natural extension to any ancient \_meta\_genomics project is to further investigate the specific genomes of the plethora of species and strains you may have detected. In this section of the book, we will look at the specific techniques used to reconstruct ancient genomes using standard genomics reference-based methods, but as always in the context of the short and damaged DNA fragments that are typical of ancient DNA.

## Genome Mapping

An important step in the reconstruction of full genomic sequences is mapping. Even relatively short genomes usually cannot be sequenced as a single consecutive piece. Instead, millions of short sequence reads are generated from genomic fragments. These reads can be several hundred nucleotides in length but are considerably shorter for ancient DNA (aDNA).

For many applications involving comparative genomics these ‘reads’ have to be aligned to one or multiple already-reconstructed reference genomes in order to identify differences between the sequenced genome and any given contextual dataset. Aligning millions of short reads to much longer genome sequences in a time-efficient and accurate manner is a bioinformatics challenge for which numerous algorithms and tools have been developed. Each of these programs comes with a variety of parameters that can significantly alter the results and default settings are often not optimal when working with aDNA. Furthermore, read mapping procedures are often part of complex computational genomics pipelines and are therefore not directly applied by many users.

In this chapter we will take a look at specific challenges during read mapping when dealing with aDNA. We will get an overview of common input and output formats and manually apply a read mapper to aDNA data studying the direct effects of variation in mapping parameters. We will conclude the session with an outlook on genotyping, which is an important follow-up analysis step, that in turn is very relevant for downstream analyses such as phylogenetics.

## Phylogenomics

Phylogenetic trees are central tools for studying the evolution of microorganisms, as they provide essential information about their relationships and timing of divergence between microbial strains.

In this chapter, we will introduce basic phylogenetic concepts and definitions, and provide guidance on how to interpret phylogenetic trees. We will then learn how to reconstruct phylogenetic trees from DNA sequences using various methods ranging from distance-based methods to probabilistic approaches, including maximum likelihood and Bayesian phylogenetics. In particular, we will learn how to use ancient genomic data to reconstruct time-calibrated trees with BEAST2.



## Chapter 14

# Genome Mapping



### Tip

For this chapter's exercises, if not already performed, you will need to create the `conda` environment from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate microbial-genomics
```

### 14.1 Lecture

PDF version of these slides can be downloaded from [here](#).

### 14.2 Mapping to a Reference Genome

One way of reconstructing genomic information from DNA sequencing reads is mapping/aligning them to a reference genome. This allows for identification of differences between the genome from your sample and the reference genome. This information can be used for example for comparative analyses such as in phylogenetics. For a detailed explanation of the read alignment problem and an overview of concepts for solving it, please see <https://doi.org/10.1146/annurev-genom-090413-025358>.

In this session we will map two samples to the *Yersinia pestis* (plague) genome using different parameter sets. We will do this “manually” in the sense that we will use all necessary commands one by one in the terminal. These commands usually run in the back when you apply DNA sequencing data processing pipelines.

### 14.2.1 Preparation

The data and conda environment .yaml file for this practical session can be downloaded from here: <https://doi.org/10.5281/zenodo.6983174>. See instructions on page.

We will open a terminal and then navigate to the working directory of this session:

```
cd /<path>/<to>/4b-genome-mapping/
```

Then, we need to activate the conda environment of this session. By this all the necessary tools can be accessed in the current terminal session:

```
conda activate microbial-genomics
```

We will be using the Burrows-Wheeler Aligner (Li et al. 2009 – <http://bio-bwa.sourceforge.net>). There are different algorithms implemented for different types of data (e.g. different read lengths). Here, we use BWA backtrack (*bwa aln*), which is well suitable for Illumina sequences up to 100bp. Other algorithms are *bwa mem* and *bwa sw* for longer reads.

### 14.2.2 Reference Genome

For mapping we need a reference genome in FASTA format. Ideally we use a genome from the same species that our data relates to or, if not available, a closely related species. The selection of the correct reference genome is highly relevant. E.g. if the chosen genome differs too much from the organism the data relates to, it might not be possible to map most of the reads. Reference genomes can be retrieved from comprehensive databases such as [NCBI](#).

In your directory, you can find 2 samples and your reference. As a first step we will index our reference genome (make sure you are inside your directory).

The first index we will generate is for *bwa*.

```
bwa index YpestisC092.fa
```

The second index will be used by the genome browser we will apply to our results later on:

```
samtools faidx YpestisC092.fa
```

We need to build a third index that is necessary for the genotyping step, which comes later after mapping:

```
picard CreateSequenceDictionary R=YpestisC092.fa
```

### 14.2.3 Mapping Parameters

We will be using *bwa aln*, but we need to specify parameters. For now we will concentrate on the “seed length” and the “maximum edit distance”. We will use the default setting for all other parameters during this session. The choice of the right parameters depend on many factors such as the type of data and the specific use case. One aspect is the mapping sensitivity, i.e. how different a read can be from the chosen reference and still be mapped. In this context we generally differentiate between *strict* and *lenient* mapping parameters.

As many other mapping algorithms *bwa* uses a so-called “seed-and-extend” approach. I.e. it initially maps the first  $N$  nucleotides of each read to the genome with relatively few mismatches and thereby determines candidate positions for the more time-intensive full alignment.

A short seed length will generate more such candidate positions and therefore mapping will take longer, but it will also be more sensitive, i.e. there can be more differences between the read and the genome. Long seeds are less sensitive but the mapping procedure is faster.

In this session we will use the following two parameter sets:

#### Lenient

Allow for more mismatches → -n 0.01

Short seed length → -l 16

#### Strict

Allow for less mismatches → -n 0.1

Long seed length → -l 32

We will be working with pre-processed files (`sample1.fastq.gz`, `sample2.fastq.gz`), i.e. any quality filtering and removal of sequencing adapters is already done.

We will map each file once with lenient and once with strict parameters. For this, we will make 4 separate directories, to avoid mixing up files:

```
mkdir sample1_lenient sample2_lenient sample1_strict sample2_strict
```

### 14.2.4 Mapping Sample1

Let's begin with a lenient mapping of sample1.

Go into the corresponding folder:

```
cd sample1_lenient
```

Perform the *bwa* alignment, here for sample1, and specify lenient mapping parameters:

```
bwa aln -n 0.01 -l 16 ./YpestisC092.fa ../sample1.fastq.gz > reads_file.sai
```

Proceed with writing the mapping in *sam* format ([https://en.wikipedia.org/wiki/SAM\\_\(file\\_format\)](https://en.wikipedia.org/wiki/SAM_(file_format))):

```
bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ./YpestisC092.fa reads_file
```

Note that we have specified the sequencing platform (Illumina) by creating a so-called “Read Group” (-r). This information is used later during the genotyping step.

Convert SAM file to binary format (BAM file):

```
samtools view -b -S reads_mapped.sam > reads_mapped.bam
```

For processing of *sam* and *bam* files we use *SAMtools* (Li et al. 2009 – <http://samtools.sourceforge.net/>).

-b specifies to output in BAM format. (-S specifies input is SAM, can be omitted in recent versions.)

Now we sort the *bam* file → Sort alignments by leftmost coordinates:

```
samtools sort reads_mapped.bam > reads_mapped_sorted.bam
```

The sorted bam file needs to be indexed → more efficient for further processing:

```
samtools index reads_mapped_sorted.bam
```

Deduplication → Removal of reads from duplicated fragments:

```
samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam
```

```
samtools index reads_mapped_sorted_dedup.bam
```

Duplicated reads are usually a consequence of amplification of the DNA fragments in the lab. Therefore, they are not biologically meaningful.

We have now completed the mapping procedure. Let’s have a look at our mapping results:

```
samtools view reads_mapped_sorted_dedup.bam | less -S
```

(exit by pressing q)

We can also get a summary about the number of mapped reads. For this we use the `samtools idxstats` command (<http://www.htslib.org/doc/samtools-idxstats.html>):

```
samtools idxstats reads_mapped_sorted_dedup.bam
```

### 14.2.5 Genotyping

The next step we need to perform is genotyping, i.e. the identification of all SNPs that differentiate the sample from the reference. For this we use the *Genome Analysis Toolkit (GATK)* (DePristo et al. 2011 – <http://www.broadinstitute.org/gatk/>):

It uses the reference genome and the mapping as input and produces an output in *Variant Call Format (VCF)* ([https://en.wikipedia.org/wiki/Variant\\_Call\\_Format](https://en.wikipedia.org/wiki/Variant_Call_Format)).

Perform genotyping on the mapping file:

```
gatk3 -T UnifiedGenotyper -R ../YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output_mode EMIT...
```

Let's have a look...

```
cat mysnps.vcf | less -S
```

(exit by pressing q)

### 14.2.6 Mapping and Genotyping for the other Samples/Parameters

Let's now continue with mapping and genotyping for the other samples and parameter settings.

#### 14.2.6.1 Sample2 Lenient

```
cd ..
cd sample2_lenient

bwa aln -n 0.01 -l 16 ../YpestisC092.fa ../sample2.fastq.gz > reads_file.sai

bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ../YpestisC092.fa reads_file.sai ../sa...
```

```
samtools view -b -S reads_mapped.sam > reads_mapped.bam

samtools sort reads_mapped.bam > reads_mapped_sorted.bam

samtools index reads_mapped_sorted.bam

samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam
```

```
samtools index reads_mapped_sorted_dedup.bam
```

```
gatk3 -T UnifiedGenotyper -R ..\YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output
```

#### 14.2.6.2 Sample1 Strict

```
cd ..
```

```
cd sample1_strict
```

```
bwa aln -n 0.1 -l 32 ..\YpestisC092.fa ..\sample1.fastq.gz > reads_file.sai
```

```
bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ..\YpestisC092.fa reads_file
```

```
samtools view -b -S reads_mapped.sam > reads_mapped.bam
```

```
samtools sort reads_mapped.bam > reads_mapped_sorted.bam
```

```
samtools index reads_mapped_sorted.bam
```

```
samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam
```

```
samtools index reads_mapped_sorted_dedup.bam
```

```
gatk3 -T UnifiedGenotyper -R ..\YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output
```

#### 14.2.6.3 Sample2 Strict

```
cd ..
```

```
cd sample2_strict
```

```
bwa aln -n 0.1 -l 32 ..\YpestisC092.fa ..\sample2.fastq.gz > reads_file.sai
```

```
bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ..\YpestisC092.fa reads_file
```

```
samtools view -b -S reads_mapped.sam > reads_mapped.bam
```

```
samtools sort reads_mapped.bam > reads_mapped_sorted.bam
```

```
samtools index reads_mapped_sorted.bam
```

```
samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam
```

```
 samtools index reads_mapped_sorted_dedup.bam
```

```
 gatk3 -T UnifiedGenotyper -R ../YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output_mode EMIT_
```

#### 14.2.7 Comparing Genotypes

In order to combine the results from multiple samples and parameter settings we need to aggregate and comparatively analyse the information from all the *vcf* files. For this we will use the software *MultiVCFAnalyzer* (<https://github.com/alexherbig/MultiVCFAnalyzer>).

It produces various output files and summary statistics and can integrate gene annotations for SNP effect analysis as done by the program *SnpEff* (Cingolani et al. 2012 - <http://snpeff.sourceforge.net/>).

Run *MultiVCFAnalyzer* on all 4 files at once. First cd one level up (if you type ls you should see your 4 directories, reference, etc.):

```
 cd ..
```

Then make a new directory...

```
 mkdir vcf_out
```

...and run the programme:

```
 multivcfanalyzer NA YpestisC092.fa NA vcf_out F 30 3 0.9 0.9 NA sample1_lenient/mysnps.vcf sample1_
```

Let's have a look in the 'vcf\_out' directory (cd into it):

```
 cd vcf_out
```

Check the parameters we set earlier:

```
 less -S info.txt
```

(exit by pressing q)

Check results:

```
 less -S snpStatistics.tsv
```

(exit by pressing q)

The file content should look like this:

```

SNP statistics for 4 samples.
Quality Threshold: 30.0
Coverage Threshold: 3
Minimum SNP allele frequency: 0.9
sample SNP Calls (all) SNP Calls (het) coverage(fold) coverage(percent)
refCall allPos noCall discardedRefCall discardedVarCall filteredVarCall unhandle
sample1_lenient 213 0 16.38 92.69
4313387 4653728 293297 46103 728 0 0
sample1_strict 207 0 16.33 92.71
4314060 4653728 293403 45633 425 0 0
sample2_lenient 1274 0 9.01 83.69
3893600 4653728 453550 297471 7829 0 4
sample2_strict 1218 0 8.94 83.76
3896970 4653728 455450 295275 4815 0 0

```

First we find the most important parameter settings and then the table of results. The first column contains the dataset name and the second column the number of called SNPs. The genome coverage and the fraction of the genome covered with the used threshold can be found in columns 4 and 5, respectively. For example, sample1 had 207 SNP calls with strict parameters. The coverage is about 16-fold and about 93% of the genome are covered 3 fold or higher (The coverage threshold we set was 3).

#### 14.2.8 Exploring the Results

For visual exploration of mapping results so-called “Genome Browsers” are used. Here we will use the *Integrative Genomics Viewer (IGV)* (<https://software.broadinstitute.org/software/igv/>).

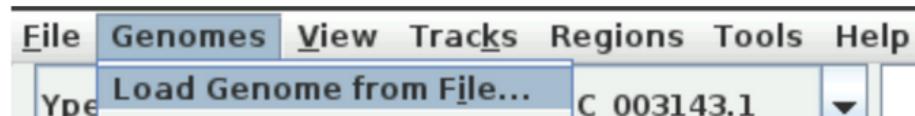
To open IGV, simply type the following command and the app will open:

```
igv
```

Note that you cannot use the terminal while IGV is open. If you want to use it anyways, open a second terminal via the bar on the bottom.

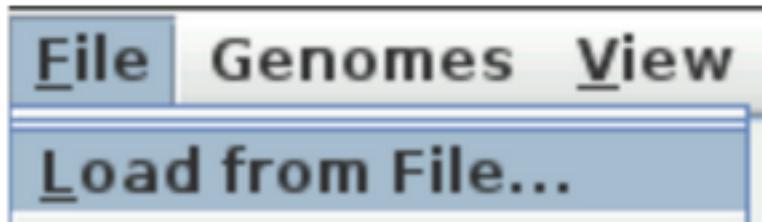
Load your reference (YpestisC092.fa):

→ Genomes → Load Genome from File



Load your *bam* files (do this 4 times, once for each mapping):

→ File → Load from File



Try to explore the mapping results yourself. Here are some questions to guide you. Please also have a look at the examples below.

What differences do you observe between the samples and parameters?

Differences in number of mapped reads, coverage, number of SNPs

Do you see any global patterns?

Which sample is more affected by changing the parameters?

Which of the two samples might be ancient, which is modern?

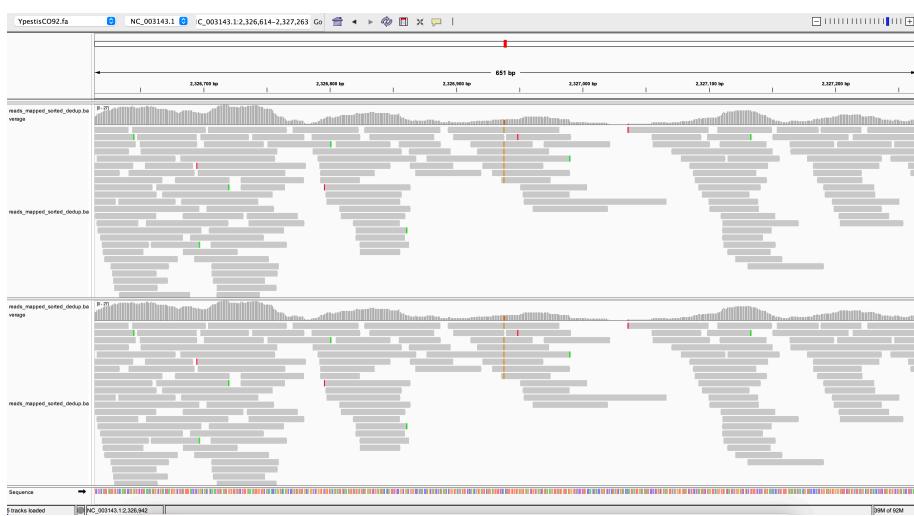
Let's examine some SNPs. Have a look at `snpTable.tsv`.

Can you identify SNPs that were called with lenient but not with strict parameters or vice versa?

Let's check out some of these in IGV. Do you observe certain patterns in these genomic regions?

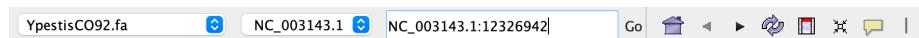
### 14.2.9 Examples

Please find here a few examples for exploration. To get a better visualization we have loaded here only `sample2_lenient` (top track) and `sample2_strict` (bottom track):



You can see all aligned reads in the current genomic region as stacks of grey arrows. In the middle of the image you see brown dashes in all of the reads. This is a SNP. You also see sporadically green or red dashes in some reads but not all of them at a given position. These sporadic differences are DNA damage such as we typically find it for ancient DNA.

For jumping to a specific coordinate you need to enter it into the coordinate field at the top:



E.g. if you enter 12326942 after the colon in the coordinate field and hit enter, you will jump to the same position as in the screenshot above.

Let's have a look at some positions.

For example position 36472:



In the middle of the image you see a SNP (T) that was called with strict parameters (bottom) but not with lenient parameters (top). But why would it not be called in the top track? It is not called because there are three reads that cover the same position, but do not contain the T. We can see that these reads have other difference to the reference at other positions. That's why they are not mapped with strict parameters. It is quite likely that they originate from a different species. This example demonstrates that sensitive mapping parameters might actually lead to a loss of certain SNP calls.

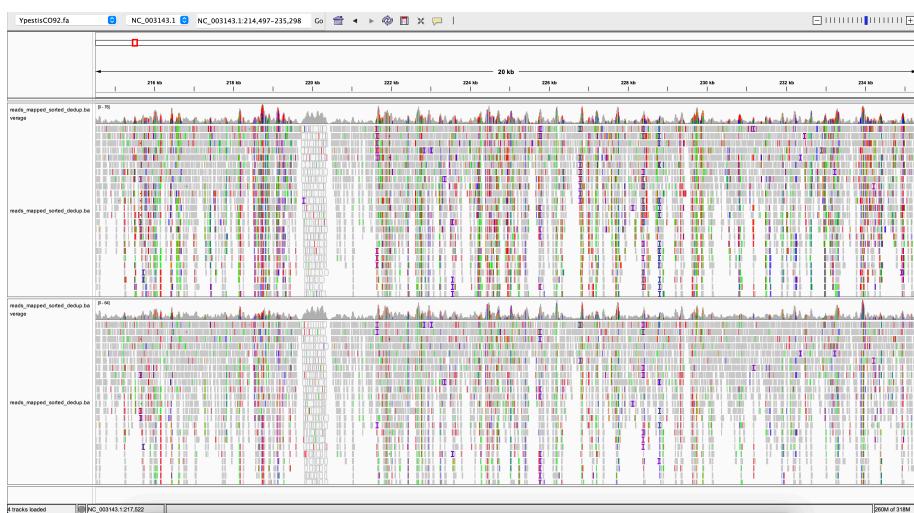
Does this mean that stricter parameters will always give us a clean mapping? Let's have a look at position 219200:



You might need to zoom out a bit using the slider in the upper right corner.

So, what is going on here? We see a lot of variation in most of the reads. This is reduced a bit with strict mapping parameters (bottom track) but the effect is still quite pronounced. Here, we see a region that seems to be conserved in other species as well, so we have a lot of mapping from other organisms. We can't compensate that with stricter mapping parameters and we would have to apply some filtering on genotype level to remove this variation from our genotyping. Removing false positive SNP calls is important as it would interfere with downstream analysis such as phylogenomics.

Such regions can be fairly large. For example, see this 20 kb region around position 224750:



**14.2.10 Conclusions**

- Mapping DNA sequencing reads to a reference genome is a complex procedure that requires multiple steps.
- Mapping results are the basis for genotyping, i.e. the detection of differences to the reference.
- The genotyping results can be aggregated from multiple samples and comparatively analysed e.g. in the context of phylogenomics.
- The chosen mapping parameters can have a strong influence on the results of any downstream analysis.
- This is particularly true when dealing with ancient DNA samples as they tend to contain DNA from multiple organisms. This can lead to mismapped reads and therefore incorrect genotypes, which can further influence downstream analyses.

# Chapter 15

## Introduction to Phylogenomics

### 15.1 Lecture

PDF version of these slides can be downloaded from [here](#).

### 15.2 Preparation

The data and conda environment .yaml file for this practical session can be downloaded from here: <https://doi.org/10.5281/zenodo.6983184>. See instructions on page.

Change into the session directory

```
cd /<path>/<to>/5b-phylogenomics/
```

The data in this folder should contain an alignment (snpAlignment\_session5.fasta) and a txt file with the ages of the samples that we are going to be working with in this session (samples.ages.txt)

Load the conda environment.

```
conda activate phylogenomics-functional
```

### 15.3 Basic concepts in phylogenomics

Before we start building trees, we are going to recap a few basic concepts in phylogenomics.

### 15.3.1 What is a phylogenetic tree

The Oxford Dictionary defines phylogenetics as “the branch of biology that deals with phylogeny, especially with the deduction of the historical relationships between groups of organisms”.

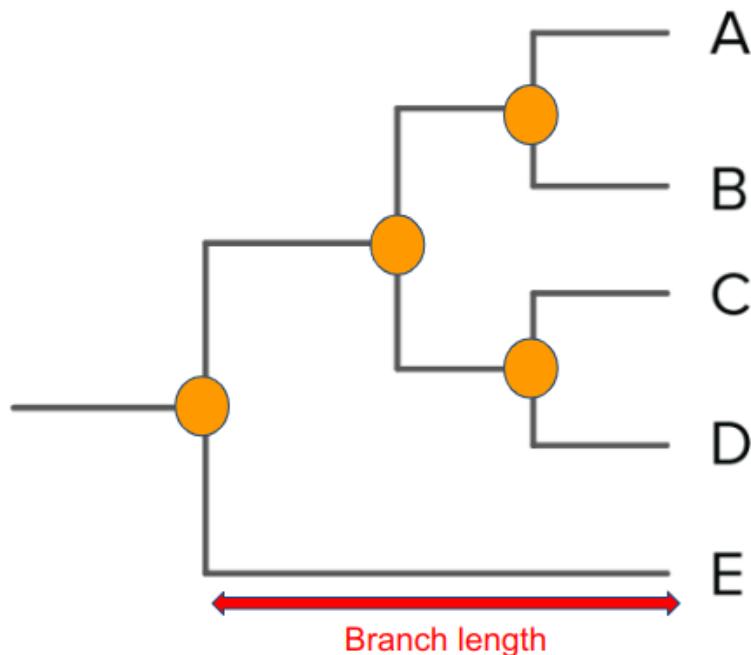
When we are building/computing a phylogeny we are inferring the relationship between organisms based on a set of homologous characters, which are characters that have been inherited from a common ancestor. Those characters can be morphological, which is common in paleontological studies, or molecular characters, which in our case they will be DNA sequences.

To display the relationships between the organism we use phylogenetic trees, which are composed of different elements.

### 15.3.2 Parts of a phylogenetic tree: Nodes and Branches

A phylogenetic tree has different parts:

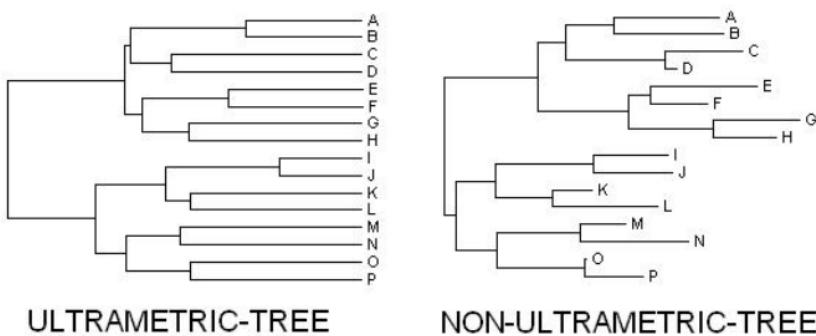
- The **tips**, which can also be called **leaves**, represent the sampled “individuals”. Each of the sequences that you used to reconstruct a tree will be displayed in your tree as tips. In the example, we used sequences A,B,C,D and E, and those appear at the tips of the tree.
- A **node** in the tree represents an ancestor (or ancestral sequences) shared by one or more tips. In the tree we see different nodes displayed, the most rightwards ones represent the ancestors between A and B; and C and D, and if we move toward the left we will see a node representing the ancestor of A,B,C and D, and the most leftwards node represent the common ancestor of all the sequences in our tree.
- **Branches** are the part of the tree that connects each node to other nodes or to the leaves. These represent evolutionary paths between nodes/leaves. The length of these branches is called **branch length** and it can represent different measures depending on the algorithm that you use to infer the phylogenetic tree, such as the number of changes (in the case of for example Maximum Parsimony tree, see Character-based phylogenetic methods), genetic/evolutionary distance (for Neighbour-Joining, see distance-based phylogenetic methods, or for Maximum Likelihood method, see Character-based phylogenetic methods), or the time between two taxa or nodes (for example, trees inferred by BEAST, see Bayesian phylogenetic inference using *BEAST2*).



### 15.3.3 Types of trees: Ultrametric vs. non-ultrametric

We can have different types of trees depending on how the distance to the root (represents the ancestor shared by all the taxa in the tree):

- In **ultrametric trees** the distance from the root to any of the tips in the tree is the **same**. One typical example of ultrametric tree will be a tree where the branch length represents time and all of our tips were samples in the present.
- In **non-ultrametric trees** the distance from the root to the tips differ from tip to tip. We will find this type of tree when we use algorithms where the branch lengths are calculated based on genetic distance/number of changes.



## 15.4 The start: DNA sequence alignment

### 15.4.1 DNA sequence alignment

For this course, our start would be a DNA sequence alignment, however phylogenies can be built using any other source of information that contain homologous characters, such as phenotypic characters inherited from the same ancestor, protein sequences, etc.

We start by collecting the DNA sequences of our organisms/individuals of interest, and now we need to make sure that the positions are homologous, which means that they were inherited from the same ancestor, to ensure that they share the same evolutionary history. A way to ensure that our positions are homologous is by **aligning** our DNA sequences.

There are different ways to produce an alignment from our sequences, and this will depend on the type of data that you have:

- Multiple Sequence Alignment (MSA) can be computed from complete genomes. This can be achieved with *MAFFT* or *Clustal Omega*. However, this will be computationally expensive to perform on large genomes.
- Reference based alignment: this consists of aligning your reads to a reference genome, and it was covered during the *Practical 4B: Genome mapping*.

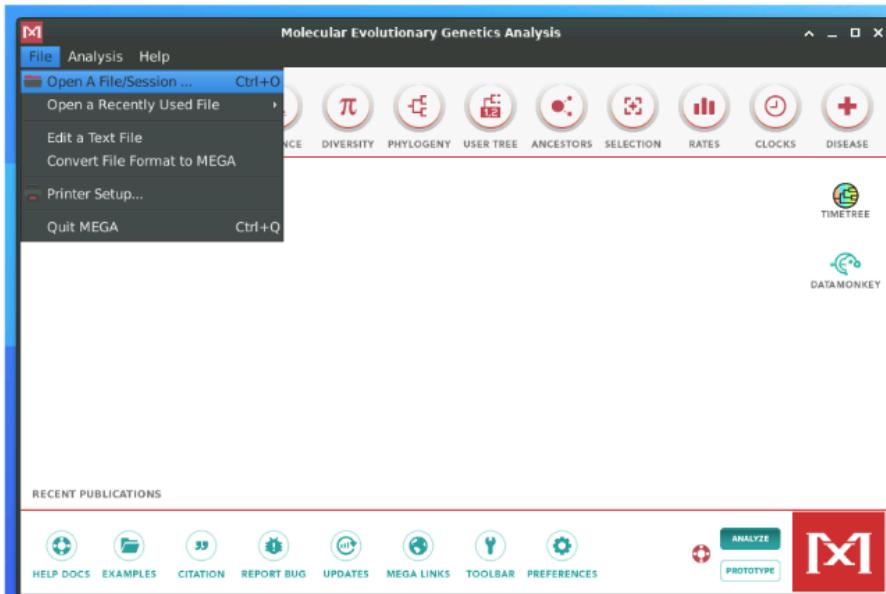
#### Note

For large genomic datasets, we often use Single Nucleotide Polymorphisms (SNPs) alignments, i.e. alignments that contain only variable genomic positions. This is to reduce the computational time since the variable positions are the most informative for reconstructing phylogenetic trees.

### 15.4.2 DNA sequence alignment

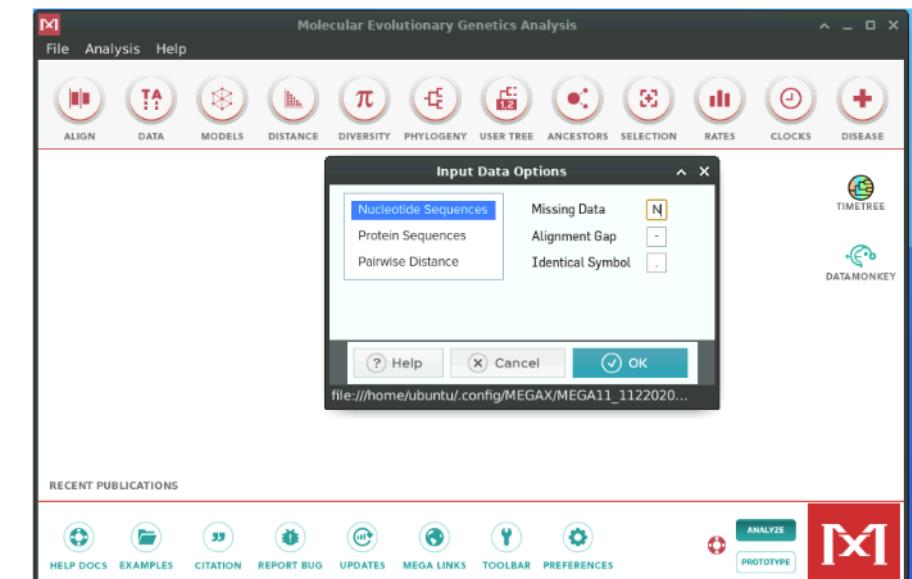
In this practical session, we will be working with an alignment produced as you learned in the *Practical 4B: Genome mapping*.

We are going to start by exploring the alignment in *MEGA*. So open the *MEGA* desktop application and load the alignment by clicking on File -> Open A File/Session -> Select the *snpAlignment\_session5.fasta*.



It will ask you what you want to do with the alignment. In *MEGA* you can also produce an alignment, however, since our sequences are already aligned we will press on *Analyze*.

Then we will select *Nucleotide Sequences* since we are working with a DNA alignment. Note that *MEGA* can also work with Protein Sequences as well as Pairwise Distance Matrix (which we will cover shortly). In the same window, we will change the character for *Missing Data* to N and click in *OK*.

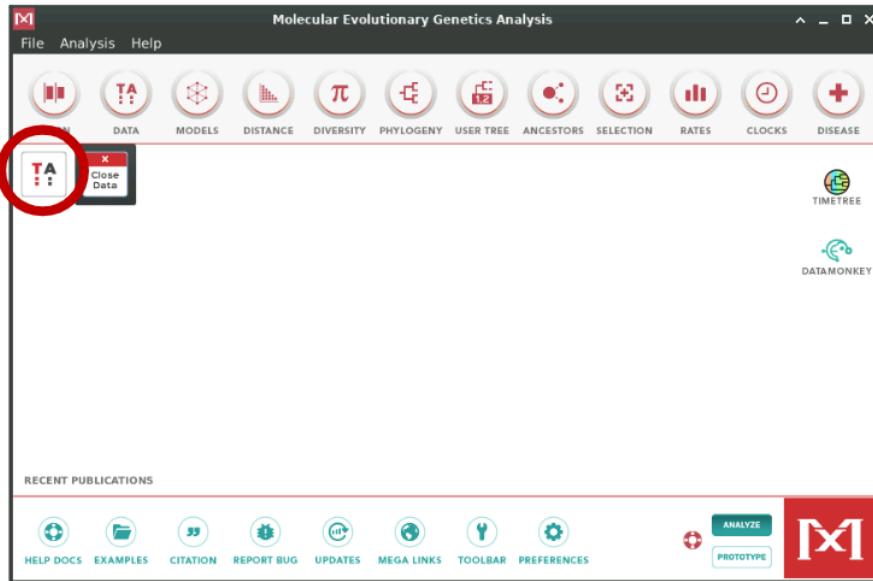


A window would open up asking if our alignment contains protein encoding sequences, and we will select No.

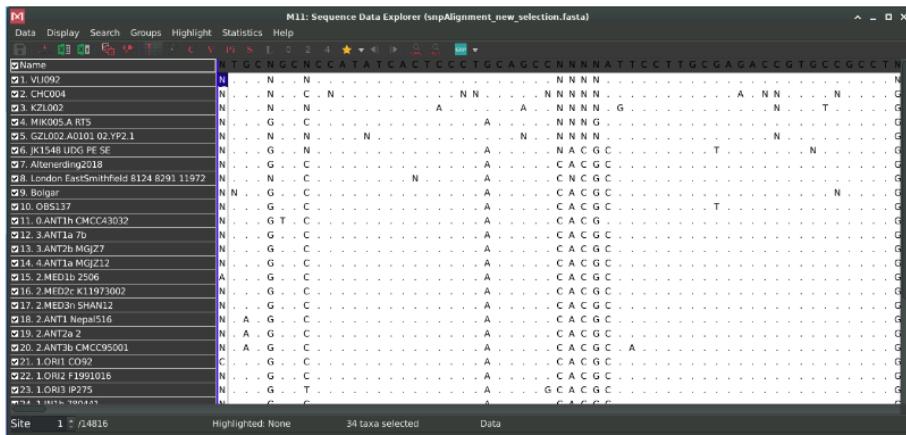
### 💡 Tip

If you had protein encoding sequences, you would have selected Yes. This will allow you to treat different positions with different evolutionary modes depending on their codon position. One can do this to take in account that the third codon position can change to different nucleotides without resulting in a different amino acid, while position one and two of the codon are more restricted.

To explore the alignment, you will then click on the box with TA



You will see an alignment containing sequences from the bacterial pathogen *Yersinia pestis*. Within the alignment, we have our sequences of interest (VLIo92, CHCoo4, KZLoo2) that date between 5000-2000 years Before Present (BP), and we want to know how they relate to the rest of the *Yersinia pestis* genomes in the alignment.



What do you think the dots represent?

- They represent positions that are same as the reference

What are the Ns in the sequences?

- They represent positions where we have missing data. We told MEGA to encode

missing positions as  $N$

How many sequences are we analysing?

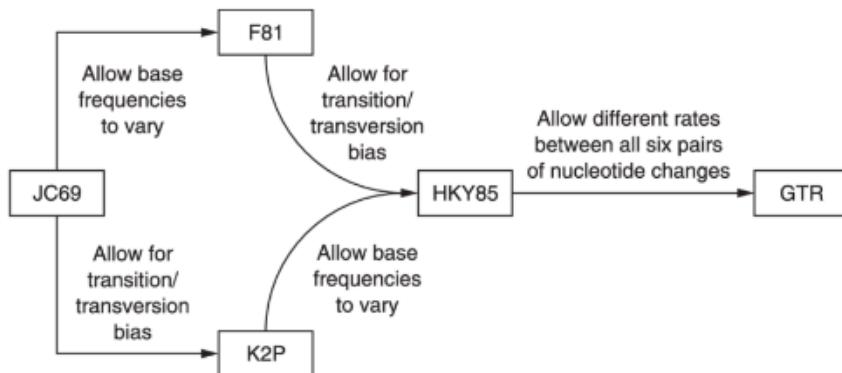
- We are analysing 33 sequences.

## 15.5 Distance-based phylogenetic methods

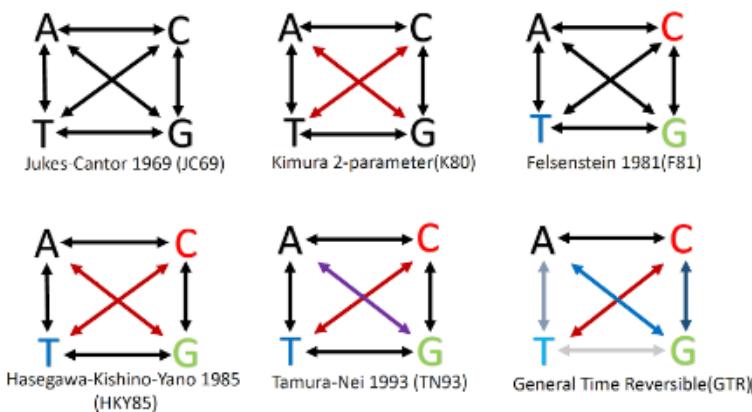
Distance-based methods calculate the tree from a pairwise distance matrix. The distances in the pairwise distance matrix can represent either:

- Number of differences between two sequences
- p-distance that is calculated as number of differences / total number of sites

For any of the phylogenetic methods, we can also take into account that different sites may evolve differently depending on their nucleotide composition. To take this into account, you will need to use **substitution models**. By applying different evolutionary models, one can take into account multiple consecutive mutations as well as different probabilities to observe a specific mutation given the specific character in this position. There are tools that allow you to choose the substitution model that it is more fitting to your specific alignment, such as jModelTest, ModelGenerator, etc.



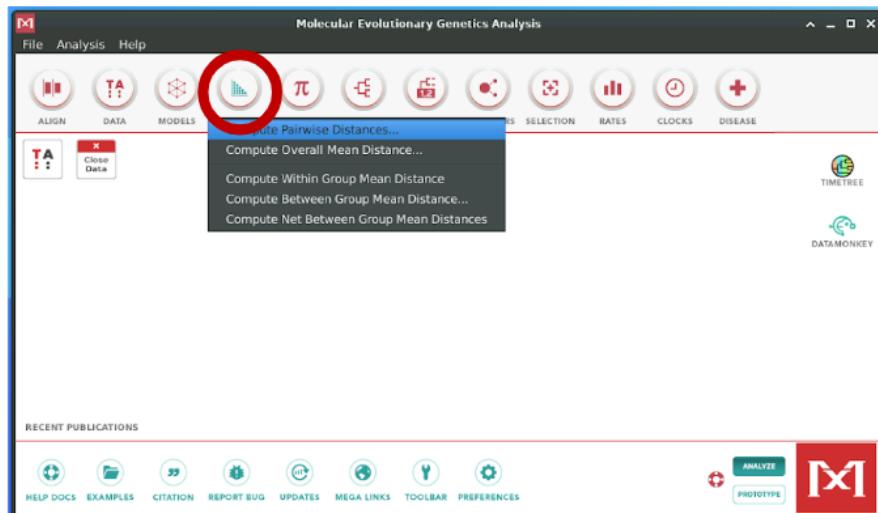
Egan A. N., Crandall K. A. (2006) Theory of Phylogenetic Estimation  
 Evolutionary Genetics: Concepts and Case Studies (pp.426-443)  
 Publisher: Oxford University Press  
 Editors: Charles W Fox, Jason B Wolf



### 15.5.1 Calculate a pairwise distance matrix

To calculate a pairwise distance matrix, one will have to check the number of differences between all the possible pair combination between the sequences in the alignment, and in the case of a p-distance normalise the number of differences by the total number of sites. All this differences will be then stored in a matrix.

MEGA also provides a function to calculate Pairwise Distances, for that you will have to click into the Distance symbol, and select *Compute Pairwise Distances*, the output will be a matrix with all the pairwise distances. This will be the first step you will do if you wanted to compute a distance-based phylogeny with methods such as UPGMA or Neighbour-Joining. We will be covering the second method.



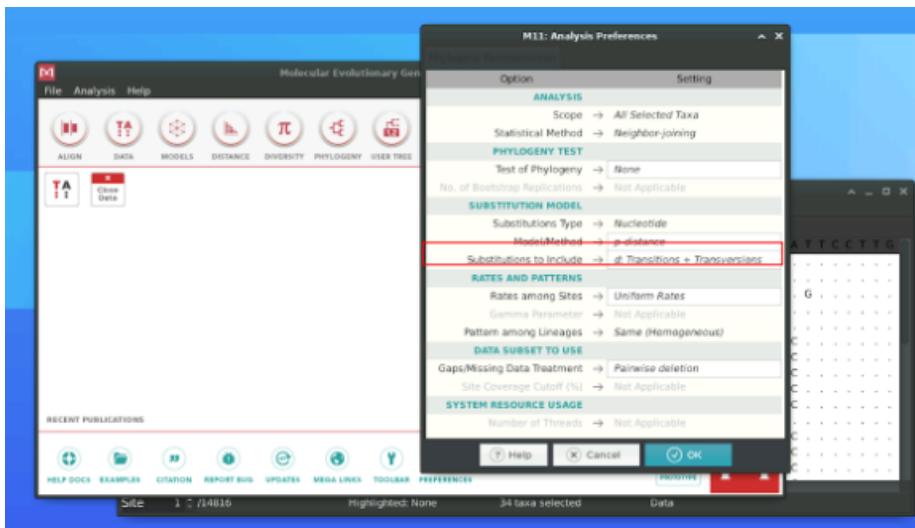
### 15.5.2 Calculating a Neighbour-Joining tree

In the slide you have an example on how a small Neighbour-Joining (NJ), with 6 taxa is calculated.

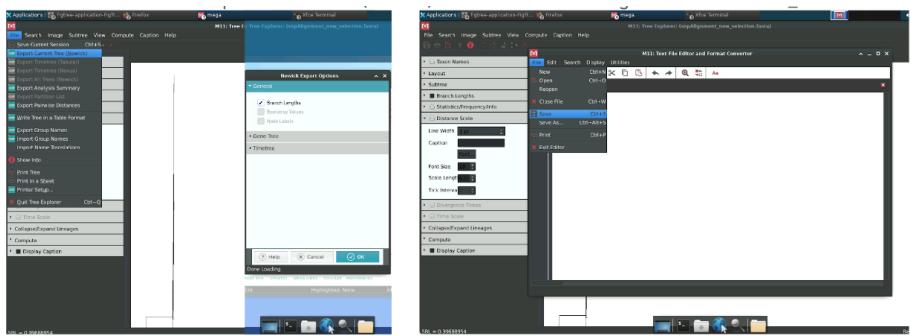
In essence, this method will be grouping taxa that have the shortest distance together first, and will be doing this iteratively until all the taxa/sequences included in your alignment have been placed in a tree. Luckily, you won't have to do this by hand since MEGA allows you to build a NJ tree.

### 15.5.3 Let's make our own NJ tree

For that go back to *MEGA* and click on the *Phylogeny* symbol and then select *Construct Neighbour Joining Tree*. In the window that would pop up, you will then chance the *Model/Method* to *p-distance* since we want to normalise by the total number of sites shared by sequences. Then press *OK* and a window with the calculated phylogenetic tree will pop up.

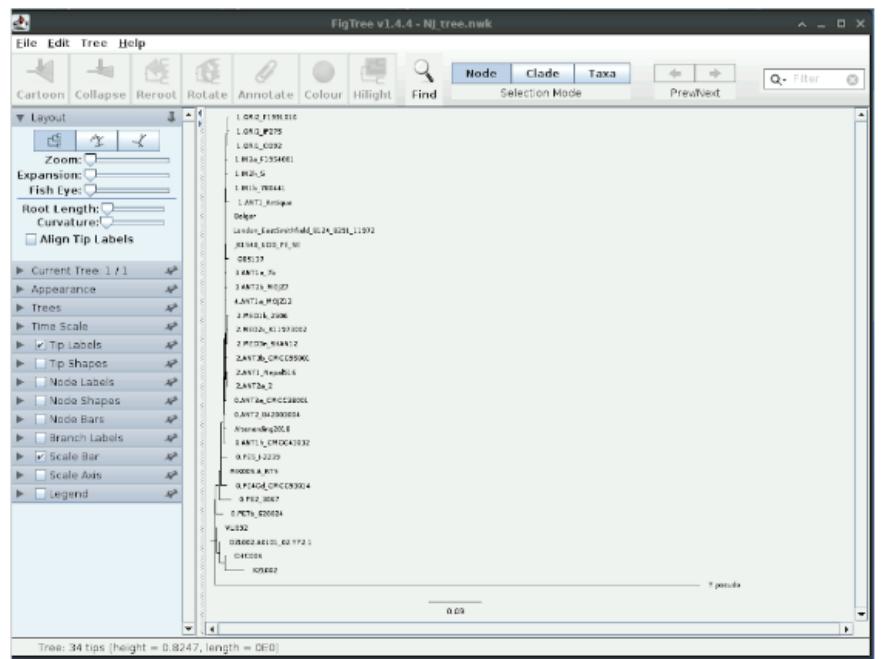


Since the tree is not easily visualised in *MEGA*, we will export it in newick format (an “standardised” format to write a tree in a computer-readable form) and explore our tree in *FigTree*. This tool has a better interface for visually manipulating trees and allows us to interact with the phylogenetic tree.



To do that you will click on *File*, then *Export current tree (Newick)* and click on *Branch Lengths* to include those in the newick annotation. When you press *OK*, a new window with the tree in newick format will pop up and you will then press *File -> Save* and saved it as *NJ\_tree.nwk* (If you are doing this for your own project, please give your files informative names).

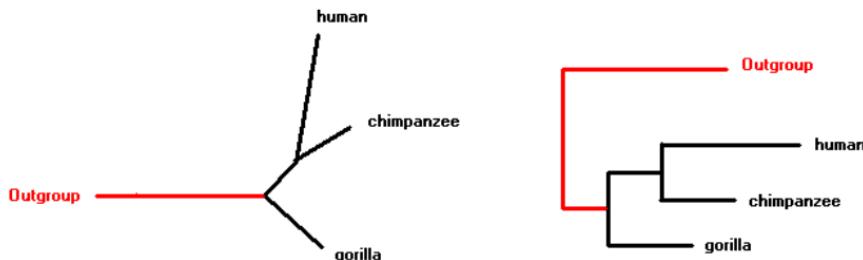
As said above, we will explore own NJ tree in *FigTree*. Open the software and then open the NJ tree by clicking on *File -> Open* and selecting the file with the NJ tree *NJ\_tree.nwk*



So, which type of tree is this? Before you answer this question, let me introduce you to another type of trees

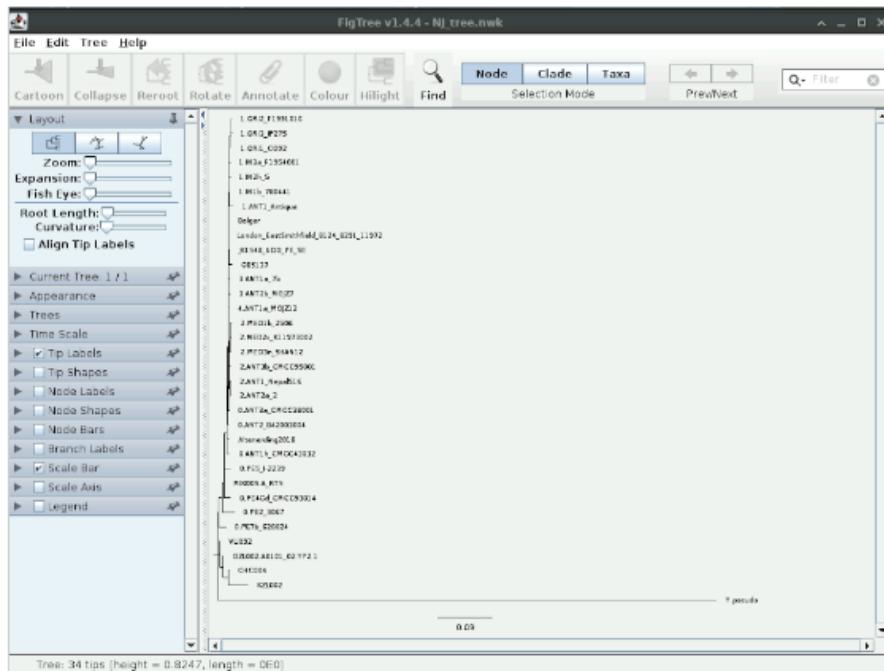
#### 15.5.4 Type of trees: Rooted vs Unrooted trees

- An *unrooted* tree does not contain a root. It displays the relationships between our sequences but not the direction in time. In the example, we can not tell if human or chimpanzee or gorilla are more ancestral to each other.
  - A *rooted* tree does contain a root. This root can be calculated based on the inclusion of an outgroup, a known sequenced to be older than any of the taxa in our tree, or by computational methods that can place the root by various methods, being one of such methods the Mid Point Rooting. A rooted tree represents the relationships and the direction in time. By rooting our example tree with the outgroup, we now can tell that gorilla is ancestral to both human and chimpanzees.

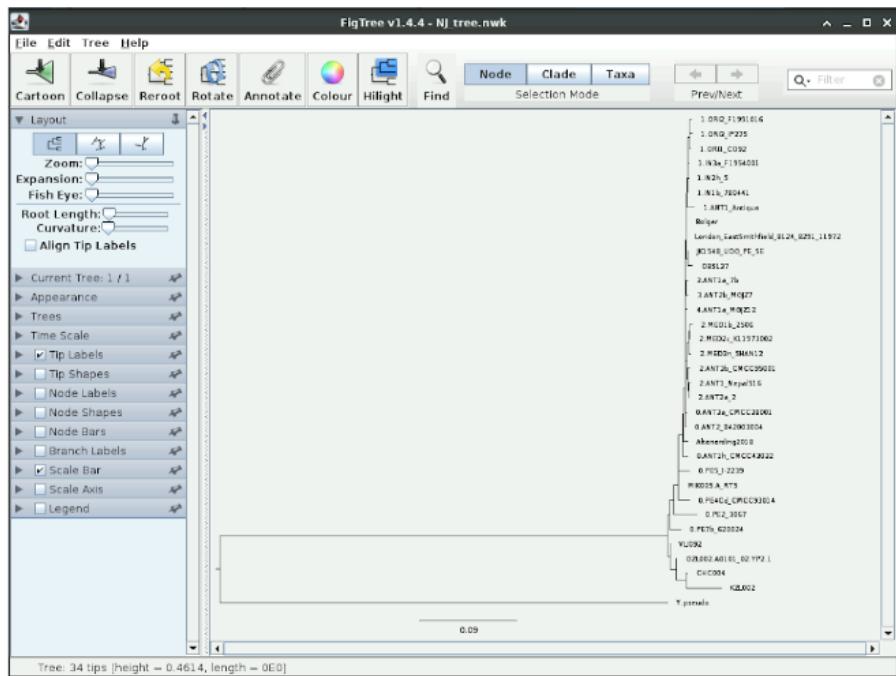


##### Let's make our own NJ tree

So if we go back to our question, which type of tree is this?



Even though a root is displayed by default, this is actually an unrooted tree. We know that *Yersinia pseudotuberculosis* (labelled here as *Y. pseudotuberculosis*) is an outgroup to *Yersinia pestis*. You can reroot the tree by selecting *Y.pseudotuberculosis* and pressing *Reroot*.



Now we have the correct tree.

How many leaves/tips has our tree?

- That's right, it is 33, the same number of sequences that was in our original SNP alignment.

Is it an ultrametric tree?

- No, we can see that root to tip distance is different between taxa. The branch lengths in the NJ tree will represent genetic distance between the different taxa.

Where are our taxa of interest? (KZLoo2, GZLoo2, CHCoo4 and VLio92)

- They all fall ancestral to the rest of *Yersinia pestis* in this tree.

Do they form a clade?

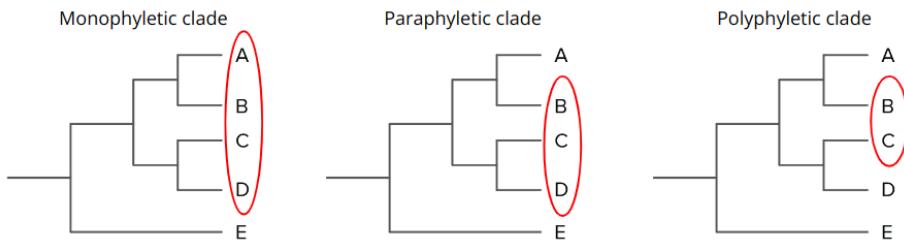
- All the genomes have a common ancestor and form a single branch, meaning that they form a clade.

Which type of clade?

- To answer this question, let's look at what types of clades we can have in a phylogenetic tree

### 15.5.5 Types of Clades

- A **Monophyletic clade** is a group of taxa that contain all the taxa that share a common ancestor.
- A **Paraphyletic clade** is a group of taxa including all the taxa with a common recent ancestor except one or more taxa. In the example, since A is missing from the clade selected, it won't be anymore a paraphyletic clade rather than a monophyletic. Another common example of paraphyletic clade would be selecting all the reptiles but excluding birds.
- A **Polyphyletic clade** is a group of taxa from different monophyletic clades.

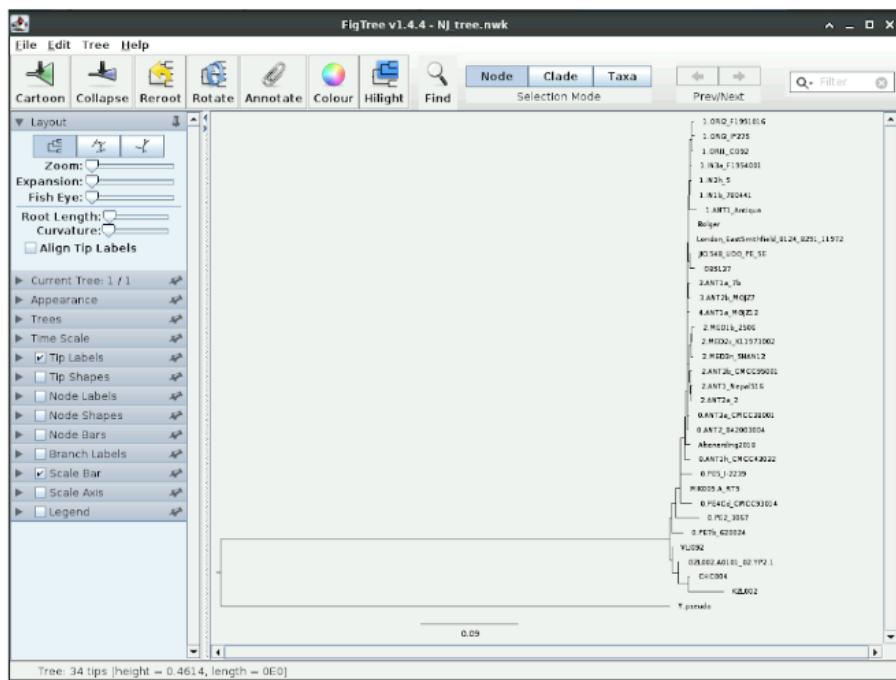


### 15.5.6 Lets make our own NJ tree

Now regarding our question, which type of clade form KZLoo2, GZLoo2, CHCoo4 and VLlo92?

Since they all share a common ancestor, they form a monophyletic clade.

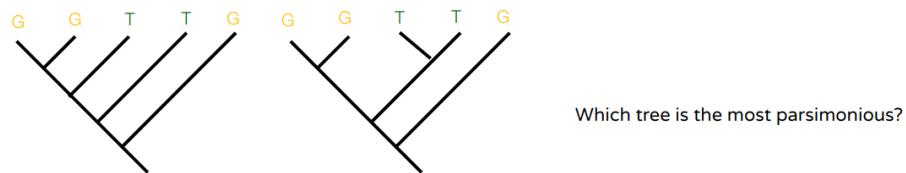
Until now we have learned how to explore a SNP alignment, the different part of the a phylogenetic tree and how to make a NJ tree. The NJ tree is based on pairwise distances, which is one of the simplest algorithms to build a tree and now you will see more complex algorithms for phylogenetic tree building.

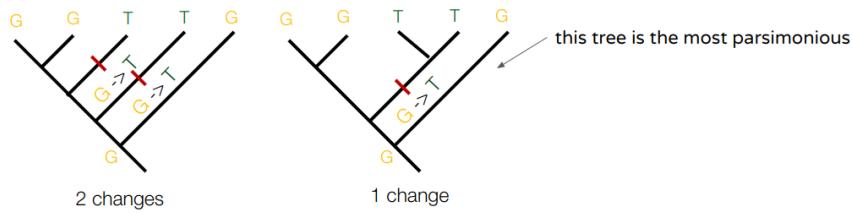


## 15.6 Character-based phylogenetic methods: maximum parsimony and probabilistic approaches

Character-based methods are not based on pairwise distances but rather model the complete evolution of each character (e.g. DNA nucleotides at each position) along the phylogenetic tree.

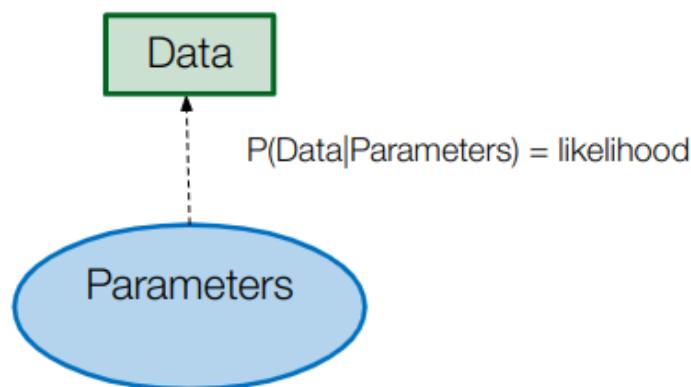
One of these methods is maximum parsimony and it consists in choosing the tree that underlies an evolutionary history with the least number of character changes.

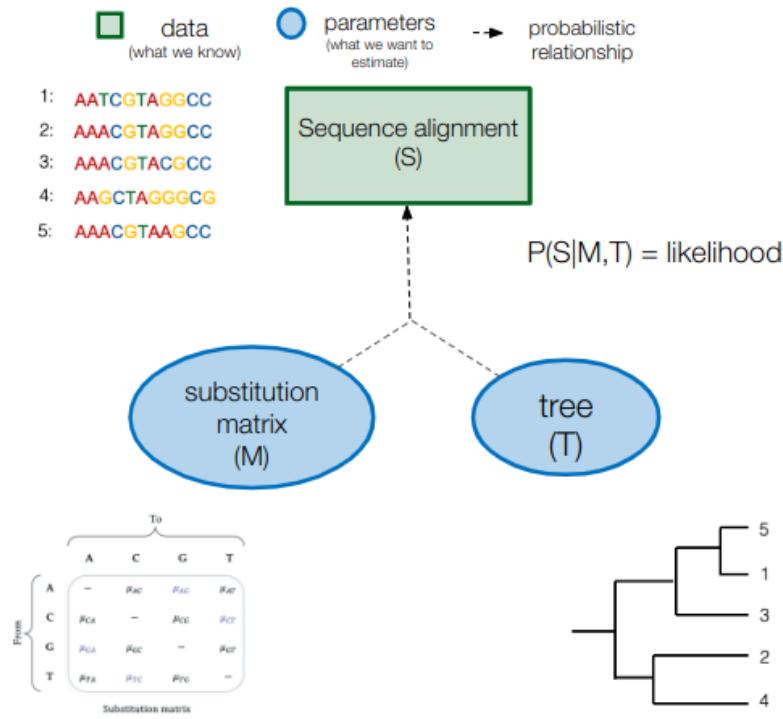




Other types of character-based methods which are more commonly used today are probabilistic methods. In general, these are statistical techniques that are based on probabilistic models under which the data that we observe is generated following a probability distribution depending on a set of parameters which we want to estimate.

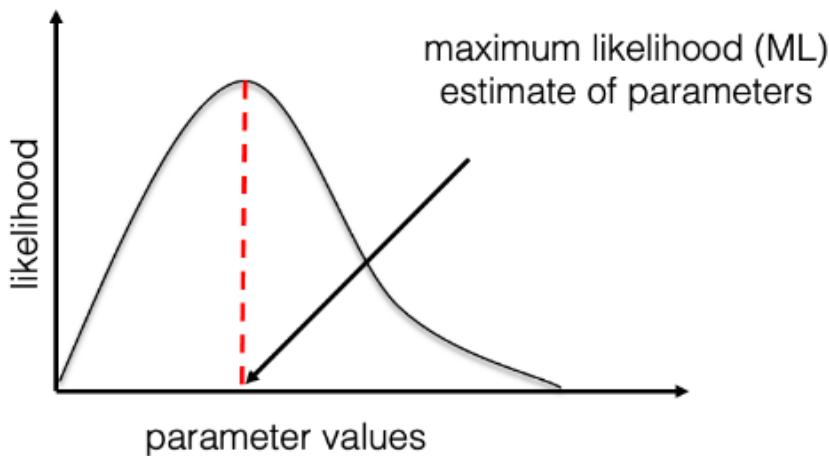
In a phylogenetic probabilistic model, the data is the sequence alignment and the parameters, are the substitution matrix and the phylogenetic tree. The probability of the data given the model parameters is called the likelihood.



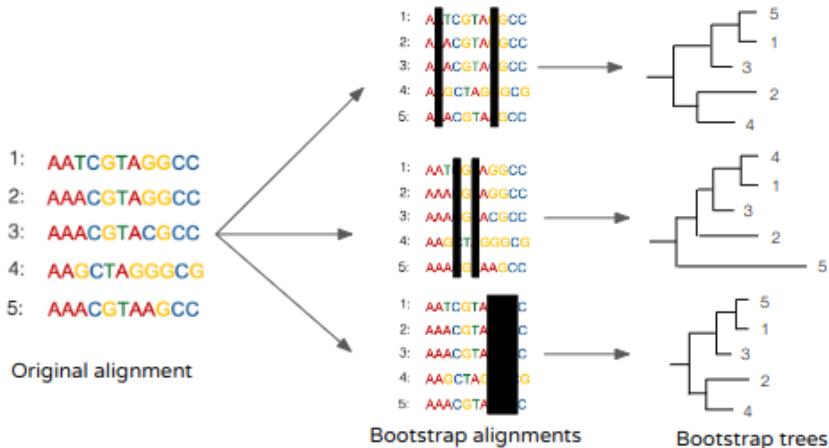


### 15.6.1 Maximum likelihood estimation and bootstrapping

One way we can make inferences from a probabilistic model is by finding the combination of parameters which maximises the likelihood. These parameter values are called maximum likelihood (ML) estimates. We are usually not able to compute the likelihood value for all possible combinations of parameters and have to rely on heuristic algorithms to find the maximum likelihood estimates.



The Maximum likelihood estimates are point estimates, i.e. single parameter values (for example, a tree), which does not allow to measure uncertainty. A classic method to measure the uncertainty of ML tree estimates is bootstrapping, which consists in repeatedly disturbing the alignment by masking sites from it and estimating a tree from each of these bootstrap alignments.



For each clade in the ML tree, a bootstrap support value is computed which corresponds to the proportion of bootstrap trees containing the clade. This gives an indication of how robustly the clade is supported by the data (i.e. whether it holds even after disturbing the dataset). Bootstrapping can be used to measure the topology un-

certainty of trees estimated with any inference method.

Here is a command to estimate an ML phylogenetic tree using *RAxML* (you may find the list of parameters in the *RAxML* manual):

```
raxmlHPC-PTHREADS -m GTRGAMMA -T 3 -f a -x 12345 -p 12345 -N autoMRE -s.snpAlignment_ses
```

Once the analysis has been completed, you can open the tree using *Figtree* (*RAxML\_bipartitions...* file, change “label” to “bootstrap support” at the prompt).

The tree estimated using this model is a substitution tree (branch lengths represent genetic distances in subst./site), and it is not oriented in time: this is an unrooted tree (displayed with a random root in *Figtree*). You can reroot the tree in *Figtree* using *Y. pseudotuberculosis* as an outgroup (click on the branch leading to *Y. pseudo* and then “Reroot”).

Where do our genomes of interest from the LNBA period fall (VLIo92, GZLoo2, CHCoo4, KZLoo2), compared to the rest of *Yersinia pestis* diversity? Is that placement well-supported? (look at the bootstrap support value: click on the “Node Labels” box and open the drop-down menu, change “Node ages” to “bootstrap support”)

You can notice that the phylogeny is difficult to visualize due to the long branch leading to *Y. pseudotuberculosis*. Having a very distant outgroup can also have deleterious effects on the estimated phylogeny (due to long branch attraction). We can construct a new phylogeny after removing the outgroup (go back to the alignment in mega, unclick *Y.pseudotuberculosis*, and export in fasta format), and then reroot the tree based on our previous knowledge: place the root on the branch leading to the LNBA genomes.

Then, export the rooted tree: file > export trees > “save as currently displayed”.

### 15.6.2 Estimating a time-tree using Bayesian phylogenetics

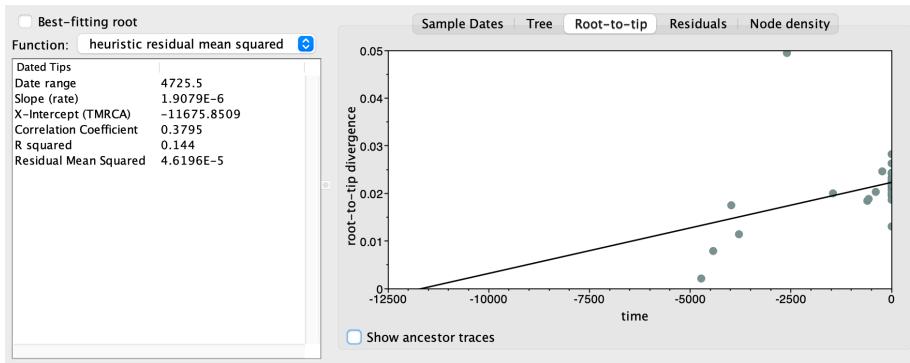
Now, we will try to use reconstruct a phylogeny in which the branch lengths do not represent a number of mutations but instead represent the time of evolution. To do so, we will use the sample ages ( $C_{14}$  dates) to calibrate the tree in time. This assumes a molecular clock hypothesis in which substitutions occur at a rate that is relatively constant in time so that the time of evolution can be estimated based on the number of substitutions.

#### 15.6.2.1 Temporal signal testing

It is a good practice to assess if the genetic sequences that we analyse do indeed behave like molecular clocks before trying to estimate a time tree. A classic test to do this is called root-to-tip regression, which consists in verifying that the oldest a sequence is, the closer it should be to the root because there was less time for mutations to accumulate before this sequence was sampled. The correlation between sample

age and distance to the root (root-to-tip regression) can be assessed using a rooted substitution tree and the program *tempest*:

- open tempest and load the re-rooted ML tree that we produced previously
- click on “import dates” in the “sample dates” tab, select the sample\_age.txt file, and then change to “dates specified as years before the present”
- look at the root-to-tip regression: is there a positive correlation?



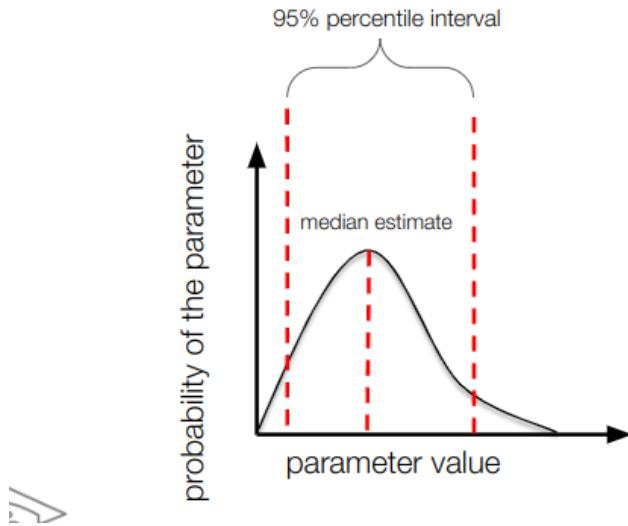
### 15.6.2.2 Bayesian phylogenetic inference using BEAST2

We will estimate a time tree from our alignment using Bayesian inference instead of maximum likelihood.

Bayesian inference is a type of inference which is based on a probability distribution that is different from the likelihood: the posterior probability. The posterior probability is the probability of the parameters given the data. The posterior distribution is easier to interpret than the likelihood because it contains all the information about the parameters: point estimates such as the median or the mean can be directly estimated from it, but also percentile intervals which can be used to measure uncertainty.

$$\mathbf{P(\text{Parameters}|\text{Data})} \propto P(\text{Data}|\text{Parameters}) \times P(\text{Parameters})$$

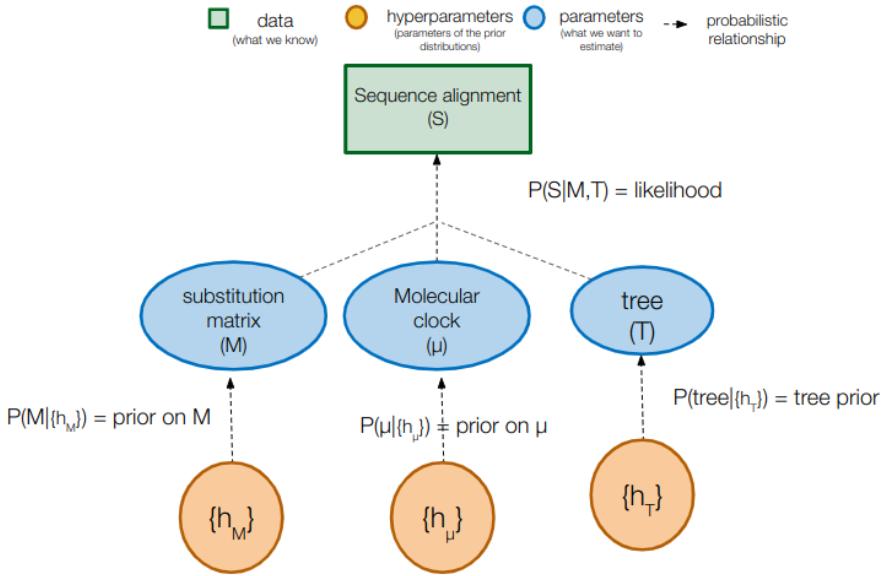
**Posterior**  $\propto$  Likelihood  $\times$  Prior



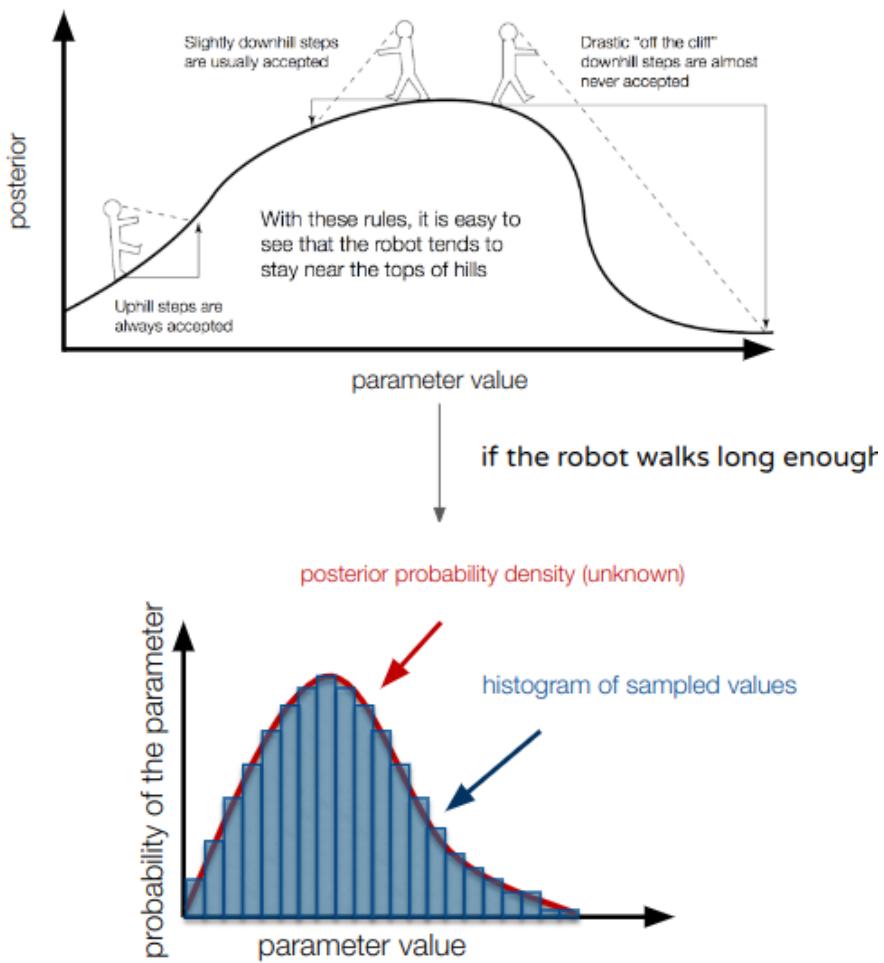
The Bayes theorem tells us that is proportional to the product of the likelihood and the “prior” probability of the data:

$$P(\theta|D) \propto P(D|\theta) \cdot P(\theta)$$

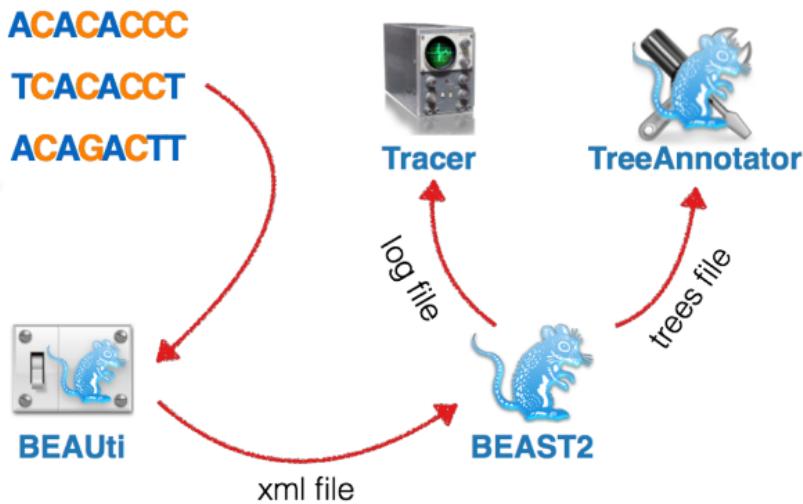
Therefore, for Bayesian inference, we need to complement our probabilistic model with prior distributions for all the parameters. Because we want to estimate a time tree, we also add another parameter: the molecular clock (average substitution rate in time units).



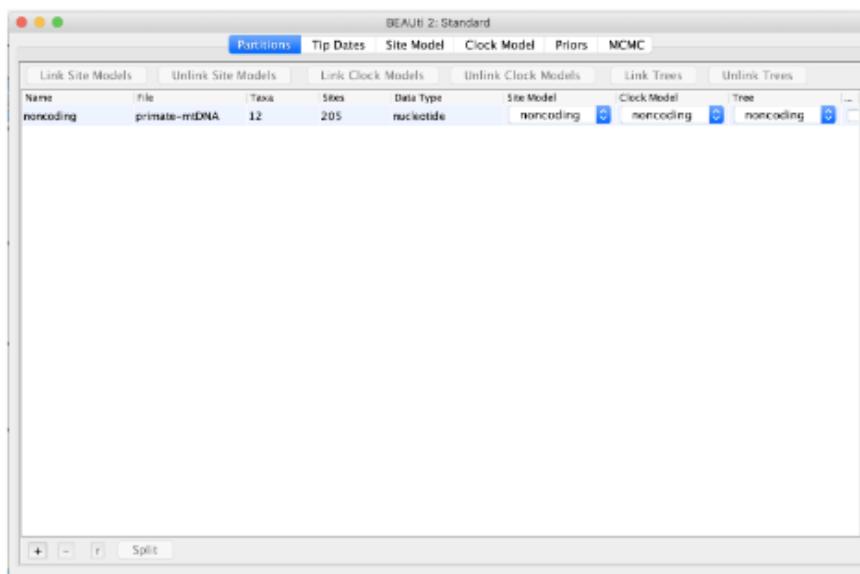
To characterize the full posterior distribution of each parameter, we would need in theory to compute the posterior probability for each possible combination of parameters. This is impossible, and we will instead use an algorithm called Markov chain Monte Carlo (MCMC) to approximate the posterior distribution. The MCMC is an algorithm which iteratively samples values of the parameters from the posterior distribution. Therefore, if the MCMC has run long enough, the (marginal) posterior distribution of the parameters can be approximated by a histogram of the sampled values.

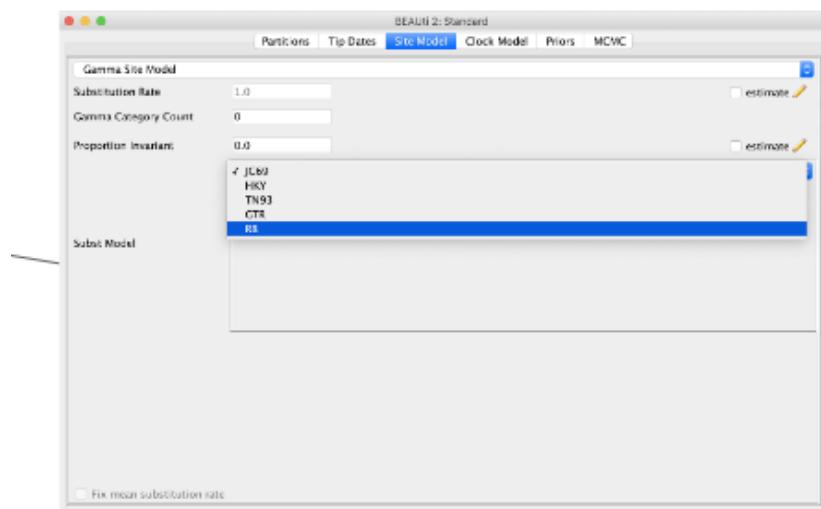
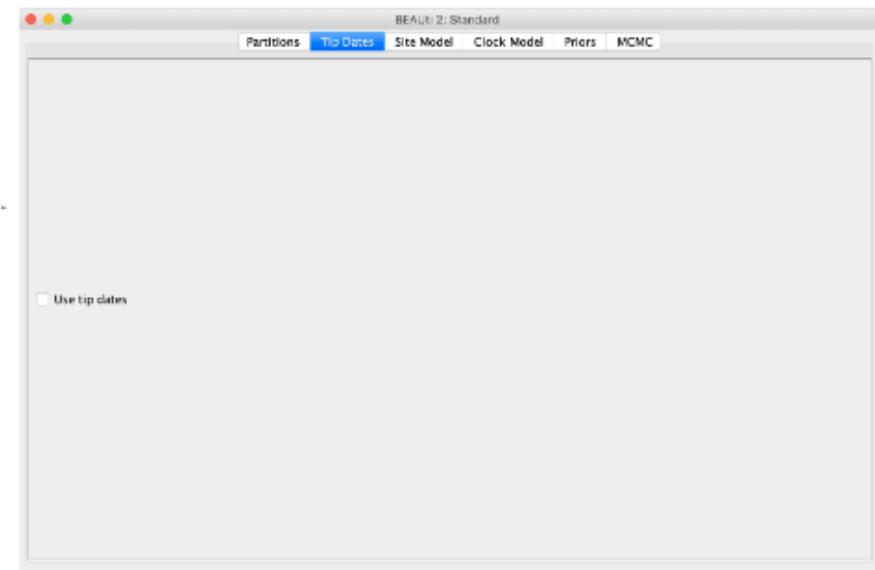


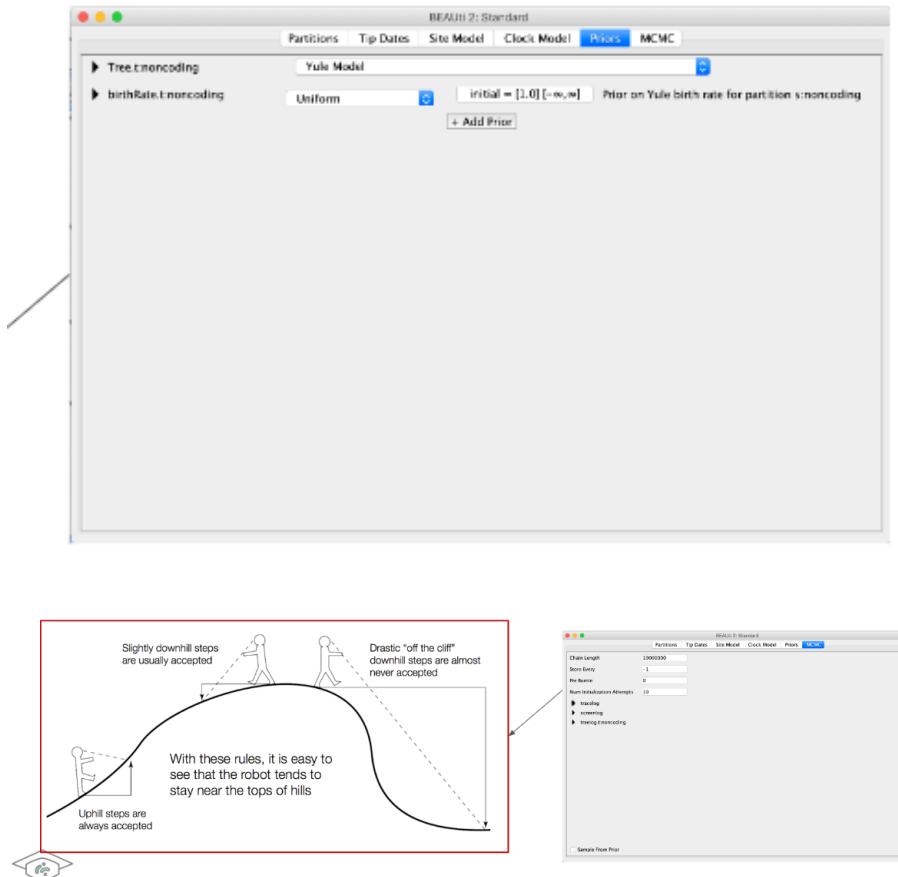
The different components of the *BEAST2* analysis can be set up in the program *BEAUti*:



- load the alignment in the “Partitions” tab
- set the sampling dates in the “Tip dates” tab
- choose the substitution model in the “Site model” tab
- choose the molecular clock model in the “Clock model” tab
- choose the prior distribution of parameters in the “Priors” tab
- set up the MCMC in the “MCMC” tab







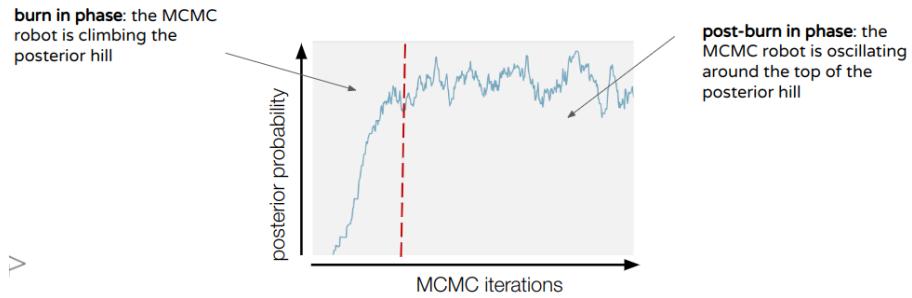
The “[taming the beast](#)” website has great tutorials to learn setting a *BEAST2* analysis. In particular, the “Introduction to BEAST2”, “Prior selection” and “Time-stamped data” are good starts.

Try running an analysis on the alignment without outgroup using the following:

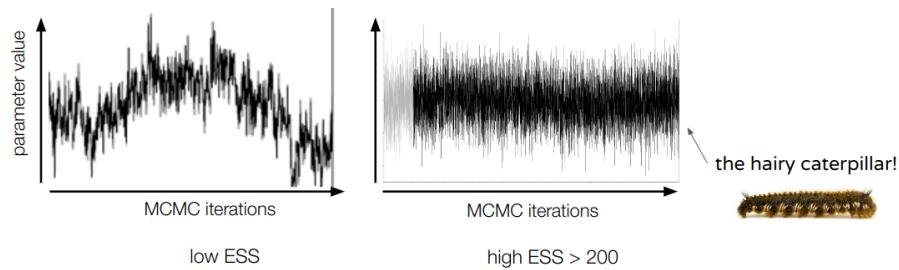
- don’t forget to specify the sample ages correctly
- use a GTR substitution matrix with a Gamma site model and 4 Gamma categories
- use a relaxed clock lognormal model with an initial value of 1E-4
- use a Bayesian Skyline Coalescent tree prior
- change the mean clock prior to a uniform distribution between 1E-6 and 1E-3 subst/site/year
- use 300M iterations for the MCMC chain, and log the parameters and trees each 10,000th iteration

Once the analysis is completed, assess the posterior distribution sampling and parameter estimates by loading the obtained log file into *Tracer*

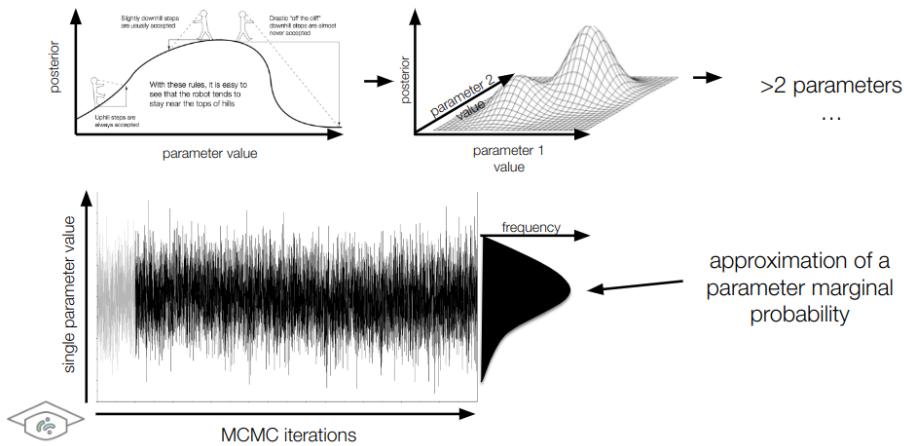
First, look at the trace of the posterior to check if the MCMC has passed the burn-in phase, and if you have remove all burn-in iterations



If so you can look at the trace and effective sample size (ESS) value of all parameters, to check that the MCMC has run long enough. The traces should look (more or less) like a “hairy caterpillar”, and a rule of thumb is that all ESS values should be above 200. If this is not the case, you should run the MCMC longer (BEAST2 has a -resume option that you can use to extend the MCMC sampling without starting everything from the beginning).

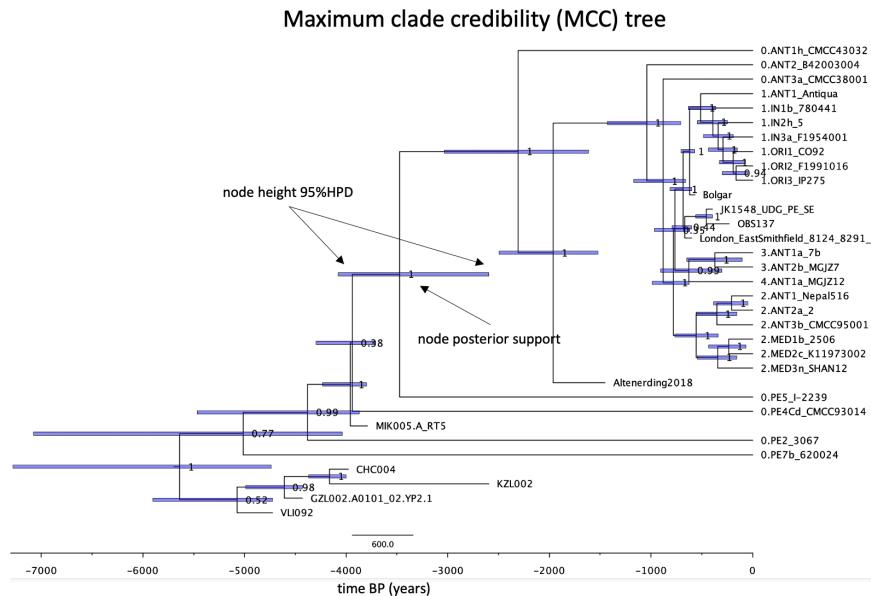


You can then look at the estimates of your parameter in the top-right panel (mean, median, 95% HPD interval, ...). Note that these are marginal estimates, i.e. integrated over all other parameters.



What is your estimate of the substitution (mean clock) rate?

You can then generate a maximum clade credibility (MCC) tree using *treeAnnotator*.



What is your mean estimate for the age of the common ancestor of all *Yersinia pestis* strains? To which parameter (displayed in beauti) does this corresponds?



## **Part V**

# **Ancient Metagenomic Resources**



Following the earlier chapters, you should now be familiar with the basic concepts of the practical aspects of the ancient metagenomic project. However, as with all bioinformaticians, we like to automate as much of our work as possible. In this chapter, we will introduce you to some of the dedicated resources to address this, covering both where and how to find ancient metagenomic data, but also how to speed up the analyses introduced earlier through automated pipelines.

## Accessing Ancient Metagenome Data

Finding relevant comparative data for your ancient metagenomic analysis is not trivial. While palaeogenomicists are very good at uploading their raw sequencing data to large sequencing data repositories such as the EBI’s ENA or NCBI’s SRA archives in standardised file formats, these files often have limited metadata. This often makes it difficult for researchers to search for and download relevant published data they wish to use to augment their own analysis.

AncientMetagenomeDir is a community project from the SPAAM community to make ancient metagenomic data more accessible. We curate a list of standardised metadata of all published ancient metagenomic samples and libraries, hosted on GitHub. In this chapter we will go through how to use the AncientMetagenomeDir repository and associated tools to find and download data for your own analyses. We will also discuss important things to consider when publishing your own data to make it more accessible for other researchers.

## Ancient Metagenomic Pipelines

Analyses in the field of ancient DNA are growing, both in terms of the number of samples processed and in the diversity of our research questions and analytical methods. Computational pipelines are a solution to the challenges of big data, helping researchers to perform analyses efficiently and in a reproducible fashion. Today we will introduce nf-core/eager, one of several pipelines designed specifically for the pre-processing, analysis, and authentication of ancient next-generation sequencing data.

In this chapter we will learn how to practically perform basic analyses with nf-core/eager, starting from raw data and performing preprocessing, alignment, and genotyping of several *Yersinia pestis*-positive samples. We will gain an appreciation of the diversity of analyses that can be performed within nf-core eager, as well as where to find additional information for customizing your own nf-core/eager runs. Finally, we will learn how to use nf-core/eager to evaluate the quality and authenticity of our ancient samples. After this session, you will be ready to strike out into the world of nf-core/eager and build your own analyses from scratch!



# Chapter 16

## Introduction to AncientMetagenomeDir

### 💡 Tip

For this chapter's exercises, if not already performed, you will need to create the **conda environment** from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate git-eager
```

### 16.1 Lecture

PDF version of these slides can be downloaded from [here](#).

### 16.2 Introduction

In most bioinformatic projects, we need to include publicly available comparative data to expand or compare our newly generated data with.

Including public data can benefit ancient metagenomic studies in a variety of ways. It can help increase our sample sizes (a common problem when dealing with rare archaeological samples) - thus providing stronger statistical power. Comparison with a range of previously published data of different preservational levels can allow an estimate on the quality of the new samples. When considering solely (re)using public data, we can consider that this can also spawn new ideas, projects, and meta analyses

to allow further deeper exploration of ancient metagenomic data (e.g., looking for correlations between various environmental factors and preservation).

Fortunately for us, genomicists and [particularly palaeogenomicists](#) have been very good at uploading raw sequencing data to well-established databases.

In the vast majority of cases you will be able to find publically available sequencing data on the [INSDC](#) association of databases, namely the [EBI's European Nucleotide Archive](#) (ENA), and [NCBI](#) or [DDBJ's](#) Sequence Read Archives (SRA). However, you may in some cases find ancient metagenomic data on institutional FTP servers, domain specific databases (e.g. [OAGR](#), [Zenodo](#), [Figshare](#), or [GitHub](#)).

But while the data is publicly available, we need to ask whether it is 'FAIR'.

### 16.3 Finding Ancient Metagenomic Data

[FAIR principles](#) were defined by researchers, librarians, and industry in 2016 to improve the quality of data uploads - primarily by making data uploads more 'machine readable'. FAIR standards for:

- Findable
- Accessible
- Interoperable
- Reproducible

And when we consider ancient metagenomic data, we are pretty close to this. Sequencing data is in most cases accessible (via the public databases like ENA, SRA), interoperable and reproducible because we use field standard formats such as FASTQ or BAM files. However *Findable* remains an issue.

This is because the *metadata* about each data file is dispersed over many places, and very often not with the data files themselves.

In this case I am referring to metadata such as: What is the sample's name? How old is it? Where is it from? Which enzymes were used for library construction? What sequencing machine was this library sequenced on?

To find this information about a given data file, you have to search many places (main text, supplementary information, the database itself), for different types of metadata (as authors report different things), and also in different formats (text, tables, figures).

This very heterogenous landscape makes it difficult for machines to index all this information (if at all), and thus means you cannot search for the data you want to use for your own research in online search engines.

### 16.4 AncientMetagenomeDir

This is where the SPAAM community project [AncientMetagenomeDir](#) comes in. AncientMetagenomeDir is a resource of lists of metadata of all publishing and publically

available ancient metagenomes and microbial genome-level enriched samples.

By aggregating and standardising metadata and accession codes of ancient metagenomic samples, the project aims to make it easier for people to find comparative data for their own projects, as well as help track the field over time and facilitate meta analyses.

Currently the project is split over three main tables: host-associated metagenomes (e.g. ancient microbiomes), host-associated single-genomes (e.g. ancient pathogens), and environmental metagenomes (e.g. lakebed cores or cave sediment sequences).

The repository already contains more than a thousand samples and span the entire globe and as far back as hundreds of thousands of years.

To make the lists of samples and their metadata as accessible and interoperable as possible, we utilise simple text (TSV - tab separate value) files - files that can be opened by pretty much all spreadsheet tools (e.g., Microsoft Office excel, LibreOffice Calc) and languages (R, Python etc.).

project_name	publication_year	publication_doi	site_name	latitude	longitude	geo_loc_name
Warinner2014	2014	10.1038/ng.2906	Dalheim	51.565	8.84	Germany
Warinner2014	2014	10.1038/ng.2906	Dalheim	51.565	8.84	Germany
Weyrich2017	2017	10.1038/nature21674	Gola Forest	7.657	-10.841	Sierra Leone
Weyrich2017	2017	10.1038/nature21674	El Sidrón Cave	43.386	-5.328	Spain
Weyrich2017	2017	10.1038/nature21674	El Sidrón Cave	43.386	-5.329	Spain
Weyrich2017	2017	10.1038/nature21674	Spy Cave	50.48	4.674	Belgium

Critically, by standardising the recorded all metadata across all publications this makes it much easier for researchers to filter for particular time periods, geographical regions, or sample types of their interest - and then use the also recorded accession numbers to efficiently download the data.

At their core all different AncientMetagenomeDir tables must have at 6 minimum metadata sets:

- Publication information (doi)
- Sample name(s)
- Geographic location (e.g. country, coordinates)
- Age
- Sample type (e.g. bone, sediment, etc.)
- Data Archive and accessions

Each table then has additional columns depending on the context (e.g. what time of microbiome is expected for host-associated metagenomes, or species name of the genome that was reconstructed).

The AncientMetagenomeDir project already has 3 releases, and will continued to be regularly updated as the community continues to submit new metadata of samples of new publications as they come out.

## 16.5 Further Improving Metadata Reporting in Ancient Metagenomics

However, for researchers, sample-level metadata likely will not include all the information that is needed to include and process public data in their own projects.

The SPAAM community have been busy over the last few months extending the types of metadata included in the AncientMetagenomeDir project, to include library level metadata.

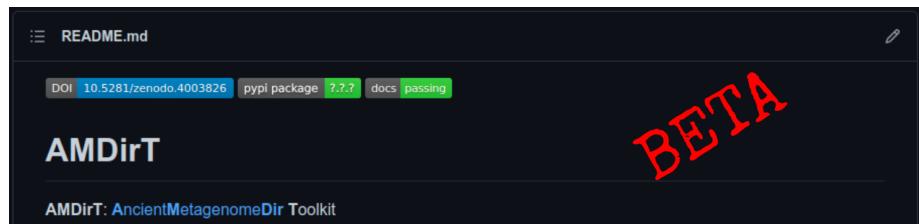
This metadata includes things such as whether a given set of data files contain sequencing data sequenced on which platform, whether the libraries have undergone damage treatment in the lab, or whether the uploaded data contains all or only mapped reads.

We have also started a new project - a MiXS checklist currently entitled ‘MINAS’ - which we aim to make *the* standard metadata reporting sheet for all ancient metagenomics and even for any ancient DNA sample. Such a checklist would be integrated into services such as the ENA or SRA, and therefore would standardise metadata *alongside* the raw data, and make ancient metagenomic data much more *findable* with search engines.

Finally, to make it easier for researchers who are not familiar with sequencing database infrastructure, we are in the process of building a new tool (something already in a usable state) called [AMDirT](#). This allows a web browser-based GUI to filter and select data, and produce scripts for you to download all the selected data (without having to go to the databases themselves).

This is something we are going to try out now!

## 16.6 Running AMDirT



First, we will need to activate a conda environment, and then install the latest development version of the tool for you.

**⚠ Warning**

This tutorial will require a web-browser! Make sure to run on your local laptop/PC or on a server with X11 forwarding

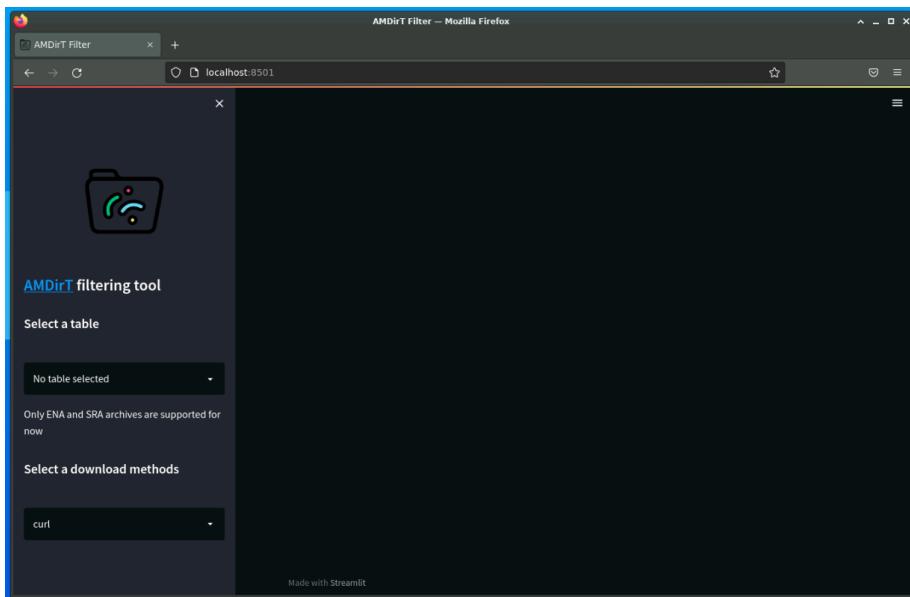
Open your terminal, and run the following two commands:

```
conda activate git-eager
pip install --upgrade --force-reinstall git+https://github.com/SPAAM-community/AMDirT.git@dev
```

Once that (hopefully) installs correctly, we can load the tool by running

**AMDirT** filter

Your web browser should now load, and you should see a two panel page.



Under **Select a table** use the dropdown menu to select ‘ancientsinglegenome-hostassociated’.

You should then see a table, pretty similar what you are familiar with with spreadsheet tools such as Microsoft Excel or LibreOffice calc.

Select samples to filter			
project_name	publication_year	publication_doi	site_name
<input type="checkbox"/> Schuenemann2013	2013	10.1126/science.1238286	Siguna
<input type="checkbox"/> Schuenemann2013	2013	10.1126/science.1238286	St. Jørgen cemetery, Odense
<input type="checkbox"/> Schuenemann2013	2013	10.1126/science.1238286	Refshale
<input type="checkbox"/> Schuenemann2013	2013	10.1126/science.1238286	St. Mary Magdalene leprosarium, Winchester
<input type="checkbox"/> Schuenemann2013	2013	10.1126/science.1238286	St. Mary Magdalene leprosarium, Winchester
<input type="checkbox"/> Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
<input type="checkbox"/> Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
<input type="checkbox"/> Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
<input type="checkbox"/> Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
<input type="checkbox"/> Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
<input type="checkbox"/> Schuenemann2018	2018	10.1371/journal.ppat.1006997	Szentes Kostöke
<input type="checkbox"/> Schuenemann2018	2018	10.1371/journal.ppat.1006997	Necropoli Vicenne Campochiaro

To navigate, you can scroll down to see more rows, and press shift and scroll to see more columns, or use click on a cell and use your arrow keys (↑, ↓, ←, →) to move around the table.

You can reorder columns by clicking on the column name, and also filter by pressing the little ‘burger’ icon that appears on the column header when you hover over a given column.

As an exercise, we will try filtering to a particular set of samples, then generate some download scripts, and download the files.

First, filter the **project\_name** column to ‘Kocher2021’.

project_name	publication_year	publication_doi	site_name
Kocher2021	Contains	0.1126/science.ab15658	Alto de Rodilla
Kocher2021	Kocher2021	0.1126/science.ab15658	Akbeit
Kocher2021	Contains	0.1126/science.ab15658	Alalakh
Kocher2021	Contains	0.1126/science.ab15658	Alalakh
Kocher2021	2021	0.1126/science.ab15658	Kaps
Kocher2021	2021	0.1126/science.ab15658	Kaps
Kocher2021	2021	0.1126/science.ab15658	Arslanetepe
Kocher2021	2021	0.1126/science.ab15658	Arslanetepe
Kocher2021	2021	0.1126/science.ab15658	Brandy's nad Labem
Kocher2021	2021	0.1126/science.ab15658	Boncuklu Hoyuk
Kocher2021	2021	0.1126/science.ab15658	Bolshoy Oleni Ostrov
Kocher2021	2021	0.1126/science.ab15658	Bolshoy Oleni Ostrov
Kocher2021	2021	0.1126/science.ab15658	Berele

Then scroll to the right, and filter the **geo\_loc\_name** to ‘United Kingdom’.

latitude	longitude	geo_loc_name	sample_name
52.08	0.18	United Kingdom	Contains
52.08	0.18	United Kingdom	United Kingdom
52.26	0.061	United Kingdom	Contains
52.9	0.544	United Kingdom	Filter...

You should be left with 4 rows.

Finally, scroll back to the first column and tick the boxes of these four samples.

The screenshot shows the AMDirT Filter tool running in Mozilla Firefox. The left sidebar has a logo and the text "AMDirT filtering tool". It includes sections for "Select a table" (set to "ancientsinglegenome-hostassociated") and "Select a download methods" (set to "curl"). A note says "Only ENA and SRA archives are supported for now". The main area displays a table titled "Displayed table: ancientsinglegenome-hostassociated". The table has columns: project\_name, publication\_year, publication\_doi, and site\_name. Four rows are shown, each with a checked checkbox in the first column:

project_name	publication_year	publication_doi	site_name
Kocher2021	2021	10.1126/science.abi5658	Hinxton
Kocher2021	2021	10.1126/science.abi5658	Hinxton
Kocher2021	2021	10.1126/science.abi5658	Oakington
Kocher2021	2021	10.1126/science.abi5658	Sedgeford

A "Validate selection" button is at the bottom of the table area.

Once you've selected the samples you want, you can press **Validate selection**. You should then see a series loading-spinner, and new buttons should appear!

The screenshot shows the AMDirT Filter tool after validating the sample selection. The main area now displays a message "4 samples selected" above three new buttons: "Download Curl sample download script", "Download nf-core/eager input TSV", and "Download Citations as BibTex". Below these buttons is a "Reset app" button.

You should have three main buttons:

- **Download Curl sample download script**
- **Download nf-core/eager input TSV**
- **Download Citations as BibText**

The first button is for generating a download script that will allow you to immediately download all sequencing data of the samples you selected. The second button is a pre-configured input file for use in the nf-core/eager ancient DNA pipeline, and finally, the third button generates a text file with (in most cases) all the citations of the data you downloaded, in a format accepted by most reference/citation managers.

It's important to note you are not necessarily restricted to [Curl](#) for downloading the data, or [nf-core/eager](#) for running the files. AMDirT aims to add support for whatever tools or pipelines requested by the community. For example, an already supported download alternative is with the [nf-core/fetchNGS](#) pipeline. You can select these using the drop-down menus on the left hand-side.

Press the three buttons to make sure you download the files. And once this is done, you can close the tab of the web browser, and in the terminal you can press `ctrl + c` to shutdown the tool.

## 16.7 Inspecting AMDirT Output

Lets look at the files that AMDirT has generated for you.

First you should `cd` into the directory that your web browser downloaded the files into (e.g. `cd ~/Downloads/`), then look inside the directory. You should see the following three files

```
$ ls
ancientMetagenomeDir_curl_download_script.sh
ancientMetagenomeDir_citations.bib
ancientMetagenomeDir_eager_input.csv
```

We can simple run `cat` on each file to look inside. If you run `cat` on the curl download script, you should see a series of `curl` commands with the correct ENA links for you for each of the samples you wish to download.

```
$ cat ancientMetagenomeDir_curl_download_script.sh
#!/usr/bin/env bash
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/009/ERR6053619/ERR6053619.fastq.gz -o ERR6053619...
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/008/ERR6053618/ERR6053618.fastq.gz -o ERR6053618...
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/005/ERR6053675/ERR6053675.fastq.gz -o ERR6053675...
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/006/ERR6053686/ERR6053686.fastq.gz -o ERR6053686...
```

By providing this script for you, AMDirT facilitates fast download of files of interest by replacing the one-by-one download commands for each sample with a *single* command!

```
$ bash ancientMetagenomeDir_curl_download_script.sh
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/009/ERR6053619/ERR6053619.fastq.gz -o
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/008/ERR6053618/ERR6053618.fastq.gz -o
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/005/ERR6053675/ERR6053675.fastq.gz -o
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/006/ERR6053686/ERR6053686.fastq.gz -o
```

Running this command should result in progress logs of the downloading of the data of the four selected samples!

Once the four samples are downloaded, AMDirT then facilitates fast processing of the data, as the *eager* script can be given directly to nf-core/eager as input. Importantly by including the library metadata (mentioned above), researchers can leverage the complex automated processing that nf-core/eager can perform when given such relevant metadata.

```
$ cat ancientMetagenomeDir_eager_input.csv
Sample_Name Library_ID Lane Colour_Chemistry SeqType Organism Strandedness
I0157 ERR6053618 0 4 SE Homo sapiens double unknown ERX5692504_ERR6053618.fastq.gz
I0161 ERR6053619 0 4 SE Homo sapiens double unknown ERX5692505_ERR6053619.fastq.gz
OAI017 ERR6053675 0 4 SE Homo sapiens double half ERX5692561_ERR6053675.fastq.gz
SED009 ERR6053686 0 4 SE Homo sapiens double half ERX5692572_ERR6053686.fastq.gz
```

Finally, we can look into the `citations` file which will provide you with the citation information of all the downloaded data and AncientMetagenomeDir itself.

### Warning

The contents of this file is reliant on indexing of publications on CrossRef. In some cases not all citations will be present, so this should be double checked!

```
$ cat ancientMetagenomeDir_citations.bib
@article{Fellows_Yates_2021,
 doi = {10.1038/s41597-021-00816-y},
 url = {https://doi.org/10.1038%2Fs41597-021-00816-y},
 year = 2021,
 month = {jan},
 publisher = {Springer Science and Business Media {LLC}},
 volume = {8},
 number = {1},
 author = {James A. Fellows Yates and Aida Andrades Valtue{\~n}a and {\'A}shild J. Becky Cribdon and Irina M. Velsko and Maxime Borry and Miriam J. Bravo-Lopez and Ant and Eleanor J. Green and Shreya L. Ramachandran and Peter D. Heintzman and Maria A. H\"ubner and Abigail S. Gancz and Jessica Hider and Aurora F. Allshouse and Valentina
```

```
title = {Community-curated and standardised metadata of published ancient metagenomic samples w
journal = {Scientific Data}
}
```

This file can be easily loaded into most reference managers and then have all the citations quickly added to your manuscripts.

## 16.8 Git Practise

A critical factor of AncientMetagenomeDir is that it is community-based. The community curates all new submissions to the repository, and this all occurs with Git.

The data is hosted and maintained on GitHub - new publications are evaluated on issues, submissions created on branches, made by pull requests, and PRs reviewed by other members of the community.

You can see the workflow in the image below from the AncientMetageomeDir [publication](#), and read more about the workflow on the AncientMetagenomeDir [wiki](#)

This means we can also use this repository to practise git!

Your task (with git terms removed):

1. Make a ‘copy’ the [jfy133/AncientMetagenomeDir](#) repository to your account
2. ‘Download’ the copied repo to your local machine
3. ‘Change’ to the dev branch
4. Modify ‘ancientsinglegenome-hostassociated\_samples.tsv’
  - Click [here](#) to get some example data to copy in to the end of the TSV file
5. ‘Send’ back to Git(Hub)
6. Open a ‘request’ adding changes to the original repo
  - Make sure to put ‘Summer school’ in the title of the ‘Request’

💡 Click me to reveal the correct terminology

1. **Fork** the [jfy133/AncientMetagenomeDir](#) repository to your account
2. **Clone** the copied repo to your local machine
3. **Switch** to the dev branch
4. Modify ‘ancientsinglegenome-hostassociated\_samples.tsv’
  - Click [here](#) to get some example data to copy in to the end of the TSV file
5. **Commit** and **Push** back to your **Fork** on Git(Hub)
6. Open a **Pull Request** adding changes to the original jfy133/AncientMetagenomeDir repo
  - Make sure to put ‘Summer school’ in the title of the pull request

## **16.9 Summary**

- Reporting of metadata messy! Consider when publishing your own work!
  - Use AncientMetagenomeDir as a template
- Use AncientMetagenomeDir and AMDirT (beta) to rapidly find public ancient metagenomic data
- Contribute to AncientMetagenomeDir with git
  - Community curated!

# Chapter 17

## Ancient Metagenomic Pipelines



### Tip

For this chapter's exercises, if not already performed, you will need to create the **conda** environment from the `yml` file in the following [archive](#), and activate the environment:

```
conda activate git-eager
```

### 17.1 Lecture

PDF version of these slides can be downloaded from [here](#).

### 17.2 Introduction

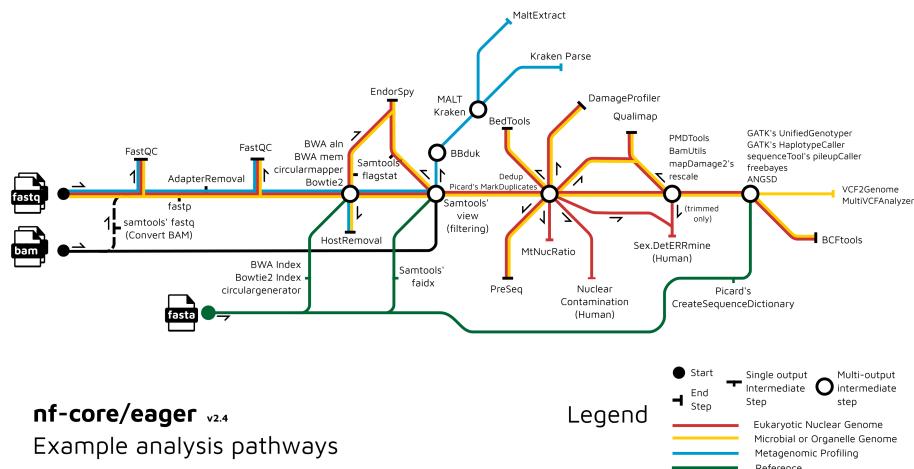
A **pipeline** is a series of linked computational steps, where the output of one process becomes the input of the next. Pipelines are critical for managing the huge quantities of data that are now being generated regularly as part of ancient DNA analyses. Today we will discuss one option for managing computational analyses of ancient next-generation sequencing datasets, [nf-core/eager](#). Keep in mind that other tools, like the [Paleomix](#) pipeline, can also be used for similar applications.

### 17.3 What is nf-core/eager?

nf-core/eager is a computational pipeline specifically designed for preprocessing and analysis of ancient DNA data. It is a reimplementation of the previously published EAGER (Efficient Ancient Genome Reconstruction) pipeline ([Peltzer et al. 2016](#)) using **Nextflow**. The nf-core/eager pipeline was designed with the following aims in mind:

- Portability**- In order for our analyses to be reproducible, others should be able to easily implement our computational pipelines. nf-core/eager is highly portable, providing easy access to pipeline tools and facilitating use across multiple platforms. nf-core eager utilizes Docker, Conda, and Singularity for containerization, enabling distribution of the pipeline in a self-contained bundle containing all the code, packages, and libraries needed to run it.
- Reproducibility**- nf-core/eager uses custom configuration profiles to specify both HPC-level parameters and analyses-specific options. These profiles can be shared alongside your publication, making it easier for others to reproduce your methodology!
- New Tools**- Finally, nf-core/eager includes additional, novel methods and tools for analysis of ancient DNA data that were not included in previous versions. This is especially good news for folks interested in microbial sciences, who can take advantage of new analytical pathways for metagenomic analysis and pathogen screening.

### 17.4 Steps in the pipeline



A detailed description of steps in the pipeline is available as part of nf-core/eager's extensive documentation. For more information, check out the usage documentation [here](#).

Briefly, nf-core/eager takes standard input file types that are shared across the ge-

nomics field, including raw fastq files, aligned reads in bam format, and a reference fasta. nf-core/eager can perform preprocessing of this raw data, including adapter clipping, read merging, and quality control of adapter-trimmed data. Note that input files can be specified using wildcards OR a standardized tsv format file; the latter facilitates streamlined integration of multiple data types within a single EAGER run! More on this later.

nf-core/eager facilitates mapping using a variety of field-standard alignment tools with configurable parameters. An exciting new addition in nf-core/eager also enables analysis of off-target host DNA for all of you metagenomics folks out there. Be sure to check out the functionality available for metagenomic profiling (blue route in the ‘tube map’ above).

nf-core/eager incorporates field-standard quality control tools designed for use with ancient DNA so that you can easily evaluate the success of your experiments. Multiple genotyping approaches and additional analyses are available depending on your input datatype, organism, and research questions. Importantly, all of these processes generate data that we need to compile and analyze in a coherent way. nf-core eager uses [MultiQC](#) to create an integrated html report that summarizes the output/results from each of the pipeline steps. Stay tuned for the practical portion of the walkthrough!

## 17.5 How to build an nf-core/eager command: A practical introduction

For the practical portion of the walkthrough, we will utilize sequencing data from four aDNA libraries, which you should have already downloaded from NCBI. If not, please see the **Preparation** section above.

These four libraries come from two ancient individuals, GLZoo2 and KZLoo2. GLZoo2 comes from the Neolithic Siberian site of Glazkovskoe predmestie and was radiocarbon dated to 3081-2913 calBCE. KZLoo2 is an Iron Age individual from Kazakhstan, radiocarbon dated to 2736-2457 calBCE. Both individuals were infected with the so-called ‘Stone Age Plague’ of *Yersinia pestis*, and libraries from these individuals were processed using hybridization capture to increase the number of *Y. pestis* sequences available for analysis.

Our aims in the following tutorial are to:

1. Preprocess the fastq files by trimming adapters and merging paired-end reads
2. Align reads to the *Y. pestis* reference and compute the endogenous DNA percentage
3. Filter the aligned reads to remove host DNA
4. Remove duplicate reads for accurate coverage estimation and genotyping
5. Merge data by sample and perform genotyping on the combined dataset
6. Review quality control data to evaluate the success of the previous steps

Let’s get started!

First, activate the conda environment that we downloaded during setup:

```
conda activate git-eager
```

Next, download the latest version of the nf-core/eager repo (or check for updates if you have a previously-installed version):

```
nextflow pull nf-core/eager
```

Finally, we will build our eager command:

```
nextflow run nf-core/eager \
 -r 2.4.5 -ds11 \
 -profile conda \
 --fasta ../reference/GCF_001293415.1_ASM129341v1_genomic.fna \
 --input ancientMetagenomeDir_eager_input.tsv \
 --run_bam_filtering --bam_unmapped_type fastq \
 --run_genotyping --genotyping_tool ug --gatk_ug_out_mode EMIT_ALL_SITES \
 --run_bcftools_stats
```

For full parameter documentation, click [here](#).

And now we wait...

## 17.6 Top Tips for nf-core/eager success

### 1. Screen sessions

Depending on your input data, infrastructure, and analyses, running nf-core/eager can take hours or even days. To avoid crashing due to loss of power or network connectivity, try running nf-core/eager in a screen or tmux session:

```
screen -R eager
```

### 2. Multiple ways to supply input data

In this tutorial, a tsv file to specify our input data files and formats. This is a powerful approach that allows nf-core eager to intelligently apply analyses to certain files only (e.g. merging for paired-end but not single-end libraries). Check out the contents of our tsv input file using the following command:

```
cat ancientMetagenomeDir_eager_input.tsv
```

Inputs can also be specified using wildcards, which can be useful for fast analyses with simple input data types (e.g. same sequencing configuration, file location, etc.).

```
nextflow run nf-core/eager -r 2.4.5 -ds11 -profile conda --fasta ../reference/GCF_001293415.1_
```

$$\dots$$

See the online nf-core/eager documentation for more details.

### 3. Get your MultiQC report via email

If you have GNU mail or sendmail set up on your system, you can add the following flag to send the MultiQC html to your email upon run completion:

```
--email "your_address@something.com"
```

4. Check out the EAGER GUI

For folks who might be less comfortable with the command line, check out the nf-core/eager [GUI](#)! The GUI also provides a full list of options with short explanations for those interested in learning more about what the pipeline can do.

5. When something fails, all is not lost!

When individual jobs fail, nf-core/eager will try to automatically resubmit that job with increased memory and CPUs (up to two times per job). When the whole pipeline crashes, you can save time and computational resources by resubmitting with the `-resume` flag. nf-core/eager will retrieve cached results from previous steps as long as the input is the same.

6. Monitor your pipeline in real time with the Nextflow Tower

Regular users may be interested in checking out the Nextflow Tower, a tool for monitoring the progress of Nextflow pipelines in real time. Check [here](#) for more information.

## 17.7 Questions to think about

1. Why is it important to use a pipeline for genomic analysis of ancient data?
2. How can the design of the nf-core/eager pipeline help researchers comply with the FAIR principles for management of scientific data?
3. What metrics do you use to evaluate the success/failure of ancient DNA sequencing experiments? How can these measures be evaluated when using nf-core/eager for data preprocessing and analysis?



# Summary

In summary, this book has no content whatsoever.

1 + 1

[1] 2



## **Part VI**

# **Appendices**



# **Chapter 18**

## **Resources**

### **18.1 Introduction to NGS Sequencing**

- <https://www.youtube.com/watch?v=fCd6B5HRaZ8>
- [https://emea.illumina.com/content/dam/illumina-marketing/documents/products/illumina\\_sequencing\\_introduction.pdf](https://emea.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf)



# References

- Dijk, Erwin L van, Hélène Auger, Yan Jaszczyszyn, and Claude Thermes. 2014. “Ten Years of Next-Generation Sequencing Technology.” *Trends in Genetics* 30 (9): 418–26. <https://doi.org/10.1016/j.tig.2014.07.001>.
- Kircher, Martin, Susanna Sawyer, and Matthias Meyer. 2012. “Double Indexing Overcomes Inaccuracies in Multiplex Sequencing on the Illumina Platform.” *Nucleic Acids Research* 40 (1): e3. <https://doi.org/10.1093/nar/gkr771>.
- Ma, Xiaotu, Ying Shao, Liqing Tian, Diane A Flasch, Heather L Mulder, Michael N Edmonson, Yu Liu, et al. 2019. “Analysis of Error Profiles in Deep Next-Generation Sequencing Data.” *Genome Biology* 20 (1): 50. <https://doi.org/10.1186/s13059-019-1659-6>.
- Meyer, Matthias, and Martin Kircher. 2010. “Illumina Sequencing Library Preparation for Highly Multiplexed Target Capture and Sequencing.” *Cold Spring Harbor Protocols* 2010 (6): db.prot5448. <https://doi.org/10.1101/pdb.prot5448>.
- Schuster, Stephan C. 2008. “Next-Generation Sequencing Transforms Today’s Biology.” *Nature Methods* 5 (1): 16–18. <https://doi.org/10.1038/nmeth1156>.
- Shendure, Jay, and Hanlee Ji. 2008. “Next-Generation DNA Sequencing.” *Nature Biotechnology* 26 (10): 1135–45. <https://doi.org/10.1038/nbt1486>.
- Sinha, Rahul, Geoff Stanley, Gunsagar Singh Gulati, Camille Ezran, Kyle Joseph Travaglini, Eric Wei, Charles Kwok Fai Chan, et al. 2017. “Index Switching Causes ‘Spreading-of-Signal’ Among Multiplexed Samples in Illumina HiSeq 4000 DNA Sequencing.” *bioRxiv*. <https://doi.org/10.1101/125724>.
- Slatko, Barton E, Andrew F Gardner, and Frederick M Ausubel. 2018. “Overview of Next-Generation Sequencing Technologies.” *Current Protocols in Molecular Biology / Edited by Frederick M. Ausubel ... [Et Al.]* 122 (1): e59. <https://doi.org/10.1002/cpmb.59>.
- Valk, Tom van der, Francesco Vezzi, Mattias Ormestad, Love Dalén, and Katerina Guschanski. 2019. “Index Hopping on the Illumina HiseqX Platform and Its Consequences for Ancient DNA Studies.” *Molecular Ecology Resources*, March. <https://doi.org/10.1111/1755-0998.13009>.



# Chapter 19

## Tools

This page lists all the software with links used and referred to in the practical chapters of the book.

### 19.1 Introduction to R and the Tidyverse

- [r](#)
- [r studio \(desktop\)](#)
- [tidyverse](#)

### 19.2 Introduction to Python and Pandas

- [python](#)
- [jupyter](#)

### 19.3 Introduction to Git(Hub)

- [git](#) (normally installed by default on all UNIX based operating systems e.g. Linux, OSX)

### 19.4 Functional Profiling

- [r](#)
- [r studio \(desktop\)](#)
- [tidyverse](#)
- [humann3](#)

## 19.5 *De novo* assembly

- fastp
- megahit
- bowtie2
- samtools
- bioawk
- diamond
- metabat2
- maxbin2
- concoct
- metawrap
- checkm-genome
- gunc
- pydamage
- prokka

## 19.6 Genome Mapping

- bwa
- igv
- gatk

## 19.7 Phylogenomics

- beast2\_
- tracer
- tempest
- mega

## 19.8 Ancient Metagenomic Pipelines

- nextflow
- nf-core tools
- nf-core/eager