

# Introduction to Ancient Metagenomics

2023-07-05



# Table of contents

<b>Introduction</b>	<b>1</b>
<b>1 Authors</b>	<b>3</b>
<b>2 Acknowledgements</b>	<b>19</b>
2.1 Financial Support . . . . .	19
2.2 Institutional Support . . . . .	19
2.3 Infrastructural Support . . . . .	20
<b>I Theory</b>	<b>21</b>
<b>3 Introduction to NGS Sequencing</b>	<b>23</b>
3.1 Introduction . . . . .	23
3.2 Lecture . . . . .	23
3.3 Readings . . . . .	23
3.4 Questions to think about . . . . .	24
<b>4 Introduction to Ancient DNA</b>	<b>25</b>
4.1 Introduction . . . . .	25
4.2 Lecture . . . . .	26
4.3 Questions to think about . . . . .	26
<b>5 Introduction to Metagenomics</b>	<b>27</b>
5.1 Introduction . . . . .	27
5.2 Lecture . . . . .	27
5.3 Questions to think about . . . . .	27
<b>6 Introduction to Microbial Genomics</b>	<b>29</b>
6.1 Introduction . . . . .	29
6.2 Lecture . . . . .	29
6.3 Questions to think about . . . . .	29

<b>7 Introduction to Evolutionary Biology</b>	<b>31</b>
7.1 Introduction . . . . .	31
<b>II Useful Skills</b>	<b>33</b>
<b>8 Bare Bones Bash</b>	<b>35</b>
8.1 Introduction . . . . .	35
<b>9 Lecture</b>	<b>37</b>
9.1 Session 1 . . . . .	37
9.2 Session 2 . . . . .	37
<b>10 Introduction to R and the Tidyverse</b>	<b>39</b>
10.1 Introduction . . . . .	39
10.2 Lecture . . . . .	40
10.3 The working environment . . . . .	40
10.4 Loading data into tibbles . . . . .	40
10.5 Plotting data in <b>tibbles</b> . . . . .	42
10.6 Conditional queries on tibbles . . . . .	45
10.7 Transforming and manipulating tibbles . . . . .	48
10.8 Combining tibbles with join operations . . . . .	52
<b>11 Introduction to Python and Pandas</b>	<b>59</b>
11.1 Abstract . . . . .	59
11.2 Lecture . . . . .	60
11.3 Introduction to data manipulation in Python with Pandas and visulization with plotnine . . . . .	60
11.4 Overview: . . . . .	61
11.5 0 - Foreword, working in a jupyter environment . . . . .	61
11.6 1 - Loading required libraries . . . . .	63
11.7 2 - Foreword on Pandas . . . . .	63
11.8 3 - Reading data with Pandas . . . . .	64
11.9 5 - Computing basic statistics . . . . .	109
11.106 - Filtering . . . . .	113
11.117 - GroupBy operations, and computing statistics on grouped values	132
11.128 - Reshaping data, from wide to long and back . . . . .	133
11.139 - Joining two different tables . . . . .	138
11.1410 - Visualizing some of the results with Plotnine . . . . .	148
11.1511 - Bonus, dealing with ill-formatted columns . . . . .	151
<b>12 Introduction to Git(Hub)</b>	<b>159</b>
12.1 Introduction . . . . .	159
12.2 Lecture . . . . .	159
12.3 SSH setup . . . . .	159
12.4 The only 6 commands you really need to know . . . . .	160

12.5 Working collaboratively . . . . .	163
12.6 Pull requests . . . . .	164
12.7 Questions to think about . . . . .	164
<b>III Ancient Metagenomics</b>	<b>165</b>
<b>13 Taxonomic Profiling, OTU Tables and Visualisation</b>	<b>169</b>
13.1 Abstract . . . . .	169
13.2 Lecture . . . . .	169
13.3 Download and Subsample . . . . .	169
13.4 Hands on introduction to ancient microbiome analysis . . . . .	172
<b>14 Introduction to <i>de novo</i> Genome Assembly</b>	<b>245</b>
14.1 Abstract . . . . .	245
14.2 Introduction . . . . .	245
<b>IV Ancient Genomics</b>	<b>247</b>
<b>15 Introduction to Genome Mapping</b>	<b>251</b>
15.1 Abstract . . . . .	251
15.2 Lecture . . . . .	252
15.3 Mapping to a Reference Genome . . . . .	252
<b>16 Introduction to Phylogenomics</b>	<b>263</b>
<b>17 Abstract</b>	<b>265</b>
<b>V Ancient Metagenomic Resources</b>	<b>267</b>
<b>18 Introduction to AncientMetagenomeDir</b>	<b>269</b>
18.1 Abstract . . . . .	269
18.2 Lecture . . . . .	269
18.3 Introduction . . . . .	269
18.4 Finding Ancient Metagenomic Data . . . . .	270
18.5 AncientMetagenomeDir . . . . .	271
18.6 Further Improving Metadata Reporting in Ancient Metagenomics	272
18.7 Running AMDirT . . . . .	273
18.8 Inspecting AMDirT Output . . . . .	277
18.9 Git Practise . . . . .	279
18.10 Summary . . . . .	280
<b>19 Ancient Metagenomic Pipelines</b>	<b>281</b>
19.1 Abstract . . . . .	281
19.2 Lecture . . . . .	281

19.3 Introduction . . . . .	282
19.4 What is nf-core/eager? . . . . .	282
19.5 Steps in the pipeline . . . . .	282
19.6 How to build an nf-core/eager command: A practical introduction	283
19.7 Top Tips for nf-core/eager success . . . . .	284
19.8 Questions to think about . . . . .	285
<b>20 Summary</b>	<b>287</b>
<b>VI Appendices</b>	<b>289</b>
<b>21 Resources</b>	<b>291</b>
21.1 Introduction to NGS Sequencing . . . . .	291
<b>References</b>	<b>293</b>
<b>22 Tools</b>	<b>295</b>

# 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*



# **Chapter 1**

## **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

---

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



**Thiseas 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.

---



## Chapter 2

# Acknowledgements

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

### 2.1 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)

### 2.2 Institutional Support



### **2.3 Infrastructural Support**



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) (031A532B, 031A533A, 031A533B, 031A534A, 031A535A, 031A537A, 031A537B, 031A537C, 031A537D, 031A538A). z

# **Part I**

# Theory



# Chapter 3

# Introduction to NGS Sequencing

## 3.1 Introduction

In this section, I 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 I will describe how the various technologies behind Illumina next-generation sequencing machines.

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

## 3.2 Lecture

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

## 3.3 Readings

### 3.3.1 Reviews

(Schuster 2008)

(Shendure and Ji 2008)

(Slatko, Gardner, and Ausubel 2018)

(Dijk et al. 2014)

### 3.3.2 Sequencing Library Construction

(Kircher, Sawyer, and Meyer 2012)

(Meyer and Kircher 2010)

### 3.3.3 Errors and Considerations

(Ma et al. 2019)

(Sinha et al. 2017)

(Valk et al. 2019)

## 3.4 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 4

## Introduction to Ancient DNA

### 4.1 Introduction

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 “smile 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.

## 4.2 Lecture

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

## 4.3 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 5

## Introduction to Metagenomics

### 5.1 Introduction

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.

### 5.2 Lecture

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

### 5.3 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 6

# Introduction to Microbial Genomics

## 6.1 Introduction

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 lecture I will provide 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.

## 6.2 Lecture

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

## 6.3 Questions to think about



# Chapter 7

## Introduction to Evolutionary Biology

### 7.1 Introduction

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.

#### 7.1.1 Lecture

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



## **Part II**

# **Useful Skills**



# Chapter 8

## Bare Bones Bash

### 8.1 Introduction

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.



# **Chapter 9**

## **Lecture**

### **9.1 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 [BareBonesBash website](#)

### **9.2 Session 2**

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

[Intro to Bash](<https://spaam-community.github.io/wss-summer-school/assets/slides/2022/1bc-barebonesbash/bbb2/session2.html> “:include :type=iframe width=100% height=400px)

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

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



# Chapter 10

## Introduction to R and the Tidyverse

### 10.1 Introduction

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

#### 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.

## 10.2 Lecture

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

## 10.3 The working environment

### 10.3.1 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 `tidyrr`
  - `forcats` will be briefly mentioned
  - `purrr` and `stringr` are left out

## 10.4 Loading data into tibbles

### 10.4.1 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

### 10.4.2 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
  ...
)
```

### 10.4.3 What does `readr` produce? The `tibble`

```
sample_table_path <- "/vol/volume/3b-1-introduction-to-r-and-the-tidyverse/ancientmetagenome-hostassociated/samples/ancientmetagenome-hostassociated_samples.tsv"
sample_table_url <-
  "https://raw.githubusercontent.com/SPAAM-community/AncientMetagenomeDir/b187df6ebd23dfeb42935f/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)
```

### 10.4.4 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`

```

samples[1,1]                      # Access the first row and column
samples[1,]                         # Access the first row
samples[,1]                         # 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 col

```

## 10.5 Plotting data in `tibbles`

### 10.5.1 `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
  ) +
  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)

### 10.5.2 `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))

```

### 10.5.3 ggplot2 features many geoms

### GRAPHICAL PRIMITIVES

**a <- geom\_bar(aes(x=unemploy))**  
**b <- ggplot(seals, aes(x = long, y = lat))**

- a + geom\_blank() and a + expand\_limits()**  
 Ensure limits include values across all plots.
- b + geom\_curve(aes(yend = lat + 1,**  
 $xend = long + 1), curvature = 1)$ , xend, yend, y, alpha, angle, color, curvature, linetype, size
- a + geom\_point(aes(label = "bent"))**, linejoin = "round", linemtire = 1)
- a, x, y, alpha, color, group, linetype, size**
- a + geom\_polygon(aes(alpha = 50))** - x, y, alpha, color, fill, group, subgroup, linetype, size
- b + geom\_rect(aes(min = long, ymin = lat,**  
 $xmax = long + 1, ymax = lat + 1))$ , xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size
- a + geom\_ribbon(aes(ymin = unemploy - 900,**  
 $ymax = unemploy + 900))$  - x, y, max, min, alpha, color, fill, group, linetype, size

---

### LINE SEGMENTS

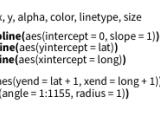
common aesthetics: x, y, alpha, color, linetype, size

- 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\_spose(aes(angle = 1:1155, radius = 1))**

---

### ONE VARIABLE continuous

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

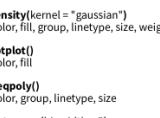


- c + geom\_area(stat = "bin")** - x, y, alpha, color, fill, linetype, size
- c + geom\_density(kernel = "gaussian")** - x, y, alpha, color, fill, group, linetype, size, weight
- c + geom\_dotplot()** - x, y, alpha, color, fill
- c + geom\_freqpoly()** - x, y, alpha, color, group, linetype, size
- c + geom\_histogram(binwidth = 5)** - x, y, alpha, color, fill, linetype, size, weight
- c2 + geom\_qq(aes(sample = hwy))** - x, y, alpha, color, fill, linetype, size, weight

---

### discrete

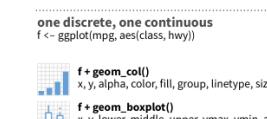
d <- ggplot(mpg, aes(flt))



- d + geom\_bar()** - x, alpha, color, fill, linetype, size, weight

### TWO VARIABLES both continuous

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

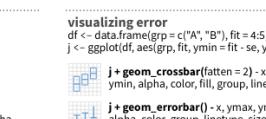


- e + geom\_label(aes(label = cty, nudge\_x = 1, nudge\_y = 1))** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust
- e + geom\_point()** - x, y, alpha, color, fill, shape, size, stroke
- e + geom\_shape()** - x, y, alpha, color, group, linetype, size, weight
- e + geom\_rug(sides = "bl")** - x, y, alpha, color, linetype, size
- e + geom\_smooth(method = "lm")** - x, y, alpha, color, fill, group, linetype, size, weight
- e + geom\_text(aes(label = cty, nudge\_x = 1, nudge\_y = 1))** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

---

### one discrete, one continuous

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

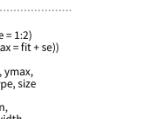


- f + geom\_col()** - x, y, alpha, color, fill, group, linetype, size
- f + geom\_boxplot()** - x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight
- f + geom\_dotplot(binaxis = "y", stackdir = "center")** - x, y, alpha, color, fill, group
- f + geom\_violin(scale = "area")** - x, y, alpha, color, fill, group, linetype, size, weight

---

### both discrete

g <- ggplot(diamonds, aes(cut, color))



- g + geom\_count()** - x, y, alpha, color, fill, shape, size, stroke
- e + geom\_jitter(height = 2, width = 2)** - x, y, alpha, color, fill, shape, size

---

### THREE VARIABLES

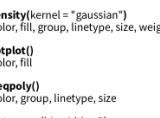
sealsSz <- with(seals, sqrt(delta\_long^2 + delta\_lat^2)); l <- ggplot(seals, aes(long, lat))



- l + geom\_contour(aes(z = z))** - x, y, z, alpha, color, group, linetype, size, weight
- l + geom\_contour\_filled(aes(fill = z))** - x, y, alpha, color, fill, group, linetype, size, subgroup
- l + geom\_raster(aes(fill = z))** - hjust = 0.5, vjust = 0.5, interpolate = FALSE
- l + geom\_tile(aes(fill = z))** - x, y, alpha, color, fill, linetype, size, width

### 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
- h + geom\_density\_2d()** - x, y, alpha, color, group, linetype, size
- h + geom\_hex()** - x, y, alpha, color, fill, size

---

### continuous function

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



- i + geom\_area()** - x, y, alpha, color, fill, linetype, size
- i + geom\_line()** - x, y, alpha, color, group, linetype, size
- i + geom\_step(direction = "hv")** - x, y, alpha, color, group, linetype, size

---

### visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1.2)

j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))



- j + geom\_crossbar(fatten = 2)** - x, y, max, ymin, alpha, color, fill, group, linetype, size, width
- j + geom\_errorbar()** - x, y, max, ymin, alpha, color, group, linetype, size, width  
Also **geom\_errorbarh()**.
- j + geom\_linerange()** - x, ymin, ymax, alpha, color, group, linetype, size
- j + geom\_pointrange()** - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

---

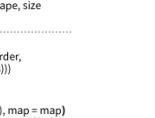
### maps

data <- data.frame(murder = USArrests\$Murder,  
 state = tolower(rownames(USArrests)))

map <- map\_data("state", match = TRUE)

k <- ggplot(data, aes(fill = murder))

**k + geom\_map(aes(map\_id = state, map = map))**  
 $+ \text{expand\_limits}(x = \text{map\$long}, y = \text{map\$lat})$



- map\_id** - alpha, color, fill, linetype, size

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

#### 10.5.4 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

### 10.5.5 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

### 10.5.6 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")
```

### 10.5.7 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

### 10.5.8 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))
```

### 10.5.9 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

### 10.5.10 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)))
```

## 10.6 Conditional queries on tibbles

### 10.6.1 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
```

### 10.6.2 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
```

### 10.6.3 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 pro
```

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

### 10.6.4 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
  )
```

### 10.6.5 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)
```

### 10.6.6 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`)

### 10.6.7 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()
```

## 10.7 Transforming and manipulating tibbles

### 10.7.1 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 column
```

- 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$cfo <- forcats::fct_reorder(sa2$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
```

### 10.7.2 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")
```

### 10.7.3 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

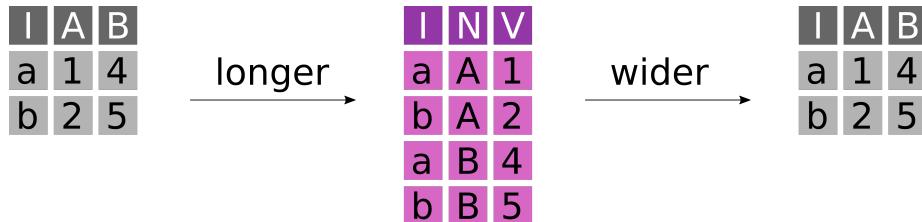
```

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

```

#### 10.7.4 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

#### 10.7.5 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”

### 10.7.6 Making a wide dataset long with `pivot_longer`

```
carsales_long <- carsales %>% tidyr::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
```

### 10.7.7 Making a long dataset wide with `pivot_wider`

```
carsales_wide <- carsales_long %>% tidyr::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)

### 10.7.8 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*.

### 10.7.9 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)
```

## 10.8 Combining tibbles with join operations

### 10.8.1 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`

### 10.8.2 A second dataset

```
library_table_path <- "/vol/volume/3b-1-introduction-to-r-and-the-tidyverse/ancientmetagenome-  
library_table_url <-  
"https://raw.githubusercontent.com/SPAAM-community/AncientMetagenomeDir/b187df6ebd23dfab42935f  
ancientmetagenome-hostassociated/libraries/ancientmetagenome-hostassociated_libraries.tsv"  
  
libraries <- readr::read_tsv(library_table_url)  
print(libraries, n = 3)
```

### 10.8.3 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)
```

### 10.8.4 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

### 10.8.5 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

### 10.8.6 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		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

### 10.8.7 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	t	1	3
b	u	2	2
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

### 10.8.8 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					

+ =

a	t	1
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

### 10.8.9 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

### 10.8.10 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

### 10.8.11 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 11

## Introduction to Python and Pandas

### 11.1 Abstract

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

#### 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 neither R nor Python.

## 11.2 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.

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
```

## 11.3 Introduction to data manipulation in Python with Pandas and visualization with plotnine

Maxime Borry  
SPAAM Summer School 2022

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

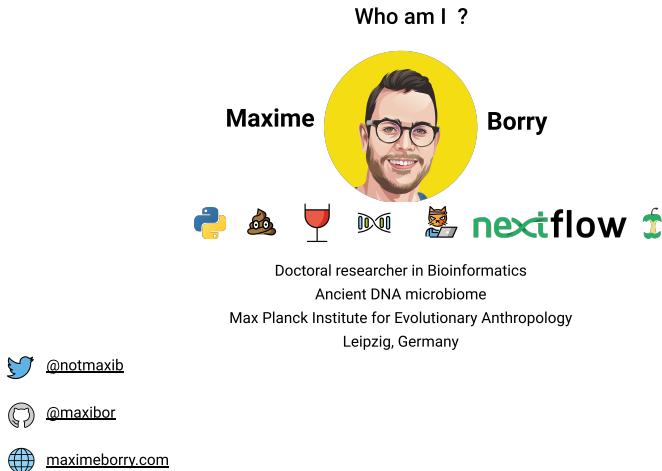


Figure 11.1: svg

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

## 11.4 Overview:

- 0 - 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

## 11.5 0 - Foreword, working in a jupyter environment

### 11.5.1 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



! [] ([https://maximeborry.com/authors/maxime/avatar\\_hu4dc3c23d5a8c195732bbca11d7ce61](https://maximeborry.com/authors/maxime/avatar_hu4dc3c23d5a8c195732bbca11d7ce61))

- 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

## 11.6 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'

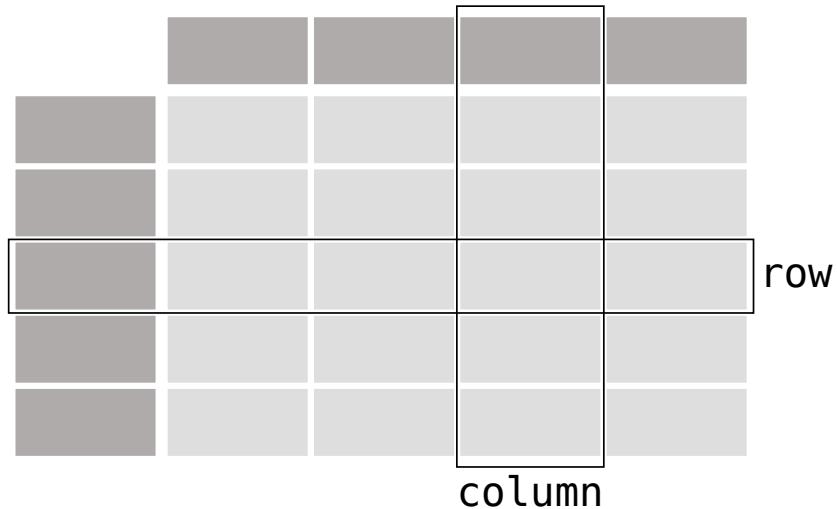
! conda list | grep plotnine

plotnine          0.9.0           pyhd8ed1ab_0    conda-forge
```

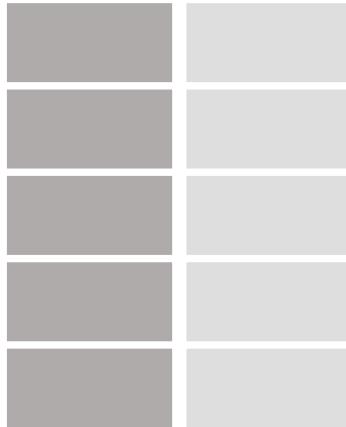
## 11.7 2 - Foreword on Pandas

### 11.7.1 Pandas terminology

## DataFrame



# Series



The pandas getting started tutorial: [pandas.pydata.org/docs/getting\\_started](https://pandas.pydata.org/docs/getting_started)

## 11.8 3 - Reading data with Pandas

```
sample_table_url = "https://raw.githubusercontent.com/SPAAM-community/AncientMetagenome-HostAssociatedSamples/ancientmetagenome-hostassociated/samples/ancientmetagenome-hostassociated_samples.tsv"
library_table_url = "https://raw.githubusercontent.com/SPAAM-community/AncientMetagenome-HostAssociatedLibraries/ancientmetagenome-hostassociated/libraries/ancientmetagenome-hostassociated_libraries.tsv"
```

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=None, 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' = '.', encoding: 'str' = 'utf-8', low_memory=True, memory_map=False, float_precision=None, na_rep='NaN', keep_name=False, quoting=0, doublequote=True, escapechar=None, lineterminator=None, quotechar='"', skipfooter=0, skiprows=0, compression_opts=None, storage_options=None, **kwargs)
```

```

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']]``
    for ``['bar', 'foo']`` order.

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>](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
    <https://docs.python.org/3/library/codecs.html#error-handlers>`_ .
    .. 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 ``\t``) 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

### 11.8.1 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_accession'],  
      dtype='object')
```

### 11.8.2 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

### 11.8.3 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  
0  
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  
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

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\_accession  
1055  
Kazarina2021b  
2021  
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

T3

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

T9

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral  
tooth  
ENA  
PRJEB47251  
ERS7283098,ERS7283099  
1058  
Kazarina2021b  
2021  
[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)  
Dom Square, Riga  
56.949  
24.104  
Latvia  
TZA3  
Homo sapiens  
400  
[10.1016/j.jasrep.2021.103213](https://doi.org/10.1016/j.jasrep.2021.103213)  
oral  
tooth  
ENA  
PRJEB47251  
ERS7283100,ERS7283101  
1059  
Kazarina2021b  
2021  
[10.1016/j.jasrep.2021.103213](https://doi.org/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
PCA0040
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
```

```
11.1038/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  
El Sidrón Cave  
43.386  
-5.329  
Spain  
ElSidron2  
Homo sapiens neanderthalensis  
49000  
10.1038/nature21674  
oral  
dental calculus  
SRA  
PRJNA685265  
SRS7890496  
5  
Weyrich2017  
2017  
10.1038/nature21674  
Spy Cave  
50.480  
4.674  
Belgium  
Spy1  
Homo sapiens neanderthalensis  
35800

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890491

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

1055

Kazarina2021b

2021

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

T3

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  
T9  
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  
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\_accession  
0  
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  
Gola Forest  
7.657  
-10.841  
Sierra Leone  
Chimp  
Pan troglodytes  
100  
10.1038/nature21674  
oral  
dental calculus  
SRA  
PRJNA685265  
SRS7890499  
3  
2017  
10.1038/nature21674  
El Sidrón Cave  
43.386  
-5.328  
Spain  
ElSidron1  
Homo sapiens neanderthalensis  
49000  
10.1038/nature21674



...  
...  
...  
...  
...  
...  
1055  
2021  
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  
2021  
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
```



False

1

False

2

False

3

False

4

False

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

1055

False

False

False

False

False

False



False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

1058

False

False

False

False

1059

```
False  
1060 rows × 16 columns
```

```
# 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

## 11.9 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'>

```

## 11.10 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

```

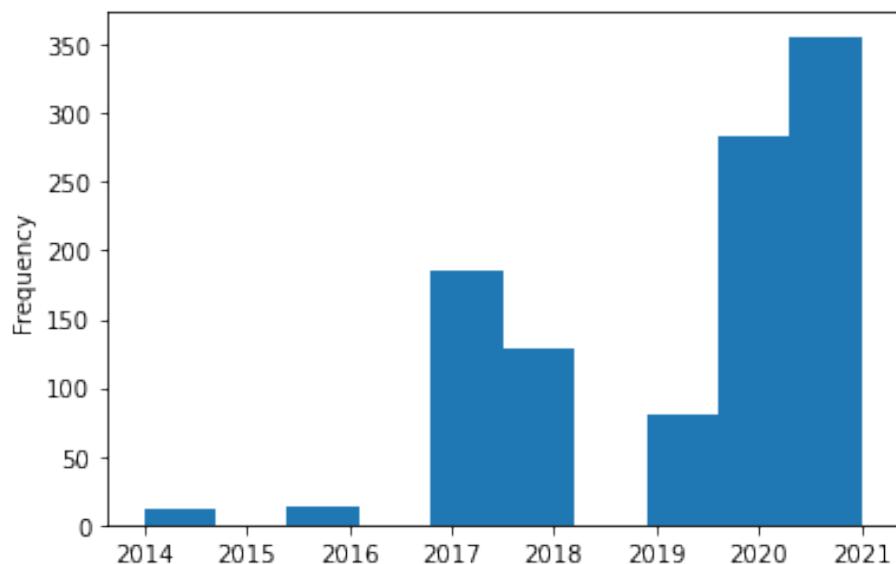


Figure 11.2: png

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  
0

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  
TP02,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  
0  
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
0
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  
TP02,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
0
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
```

## 11.11 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
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 ellioti       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
```

## 11.12 8 - Reshaping data, from wide to long and back

### 11.12.1 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

0

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')
tidy_df
```

individual

birthyear

height

0

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  
0  
John  
150  
155  
157  
160  
1  
Jack  
149  
153  
154  
155  
  
pd.wide_to_long(wide_df, ['year'], i='individual', j='birthyear', sep="-").rename(cc  
  
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

### 11.12.2 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/algos
birthyear
1991
1992
1993
1994
individual
Jack
149
153
154
155
John
150
155
157
160
```

## 11.13 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 dataframe, 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', 'archive_pro  
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  
0  
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/...  
9c40c43b5d455e760ae8db924347f0b2  
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/)  
dec1507f742de109529638bf00e0732f  
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/...](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/...](ftp.sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957741/)  
e02e3549ddd3ba6dc278a7f573c07321  
877360302

4

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.84

Germany

G12

Homo sapiens

900

...

none

NaN

Illumina HiSeq 2000

SINGLE

WGS

8978974

SRR957742

[ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957742/...](ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR957/SRR957742/)

b7c620b8ee165c08bee204529341ca5b

690614774

...

...

...

...

...

...

...

...

...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
...  
1802  
Maixner2021  
2021  
10.1016/j.cub.2021.09.031  
Edlersbergwerk - oben, Hallstatt  
47.560  
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...

542787c645b0aeebe15c66cc926d3f69;0bc58d56be3c3...

86783041;98100690

1803

Maixner2021

2021

10.1016/j.cub.2021.09.031

Edlersbergwerk - oben, Hallstatt

47.560

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

47.560  
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...](ftp.sra.ebi.ac.uk/vol1/fastq/ERR576/001/)  
57fc575d32db14f1d5c1ed7f6a106e91;4f57b9d978b53...  
51852071;56288763  
1805  
Maixner2021  
2021  
10.1016/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  
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

## 11.14 10 - Visualizing some of the results with Plotnine

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'.
```

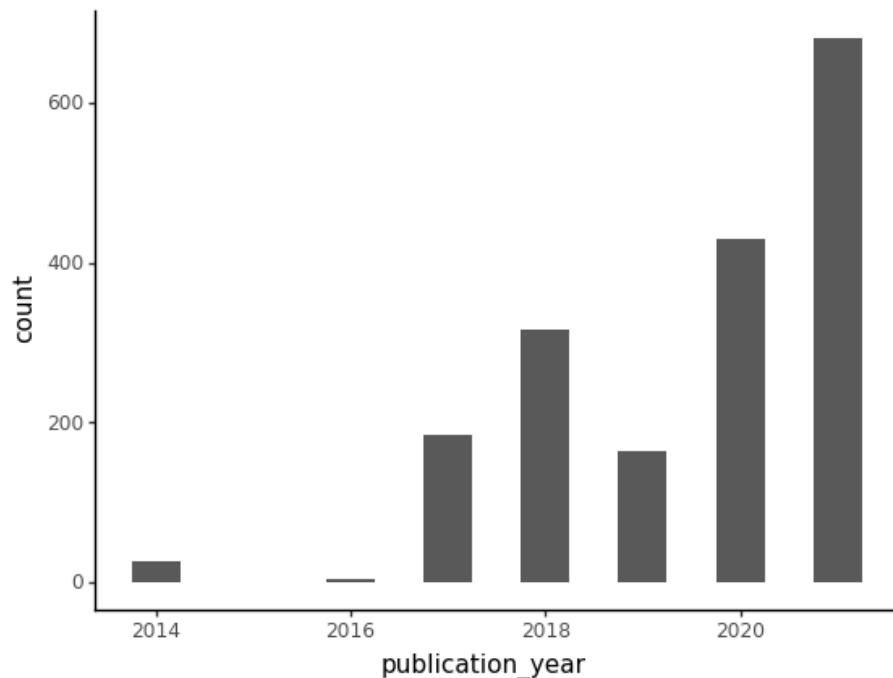


Figure 11.3: png

```
<ggplot: (366051178)>
```

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')
```

## 11.14. 10 - VISUALIZING SOME OF THE RESULTS WITH PLOTNINE149

```
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()
```

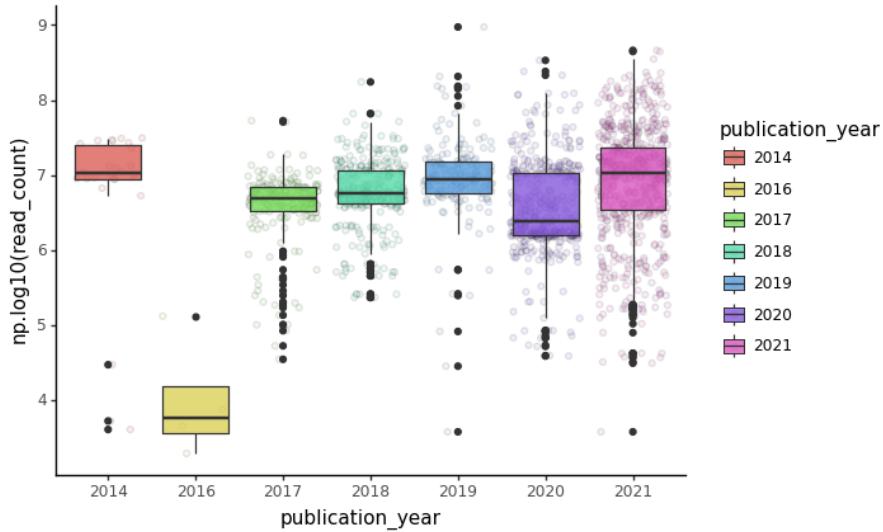


Figure 11.4: png

```
<ggplot: (366112582)>
```

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

```
avg_read_count_by_year = merged_df.groupby('publication_year')['read_count'].mean().to_frame()  
avg_read_count_by_year  
  
publication_year  
read_count  
0  
2014  
1.437173e+07  
1  
2016  
3.653450e+04  
2
```

```
2017  
5.712598e+06  
3  
2018  
9.273287e+06  
4  
2019  
2.211632e+07  
5  
2020  
1.111819e+07  
6  
2021  
2.547655e+07
```

```
ggplot(avg_read_count_by_year, aes(x='publication_year', y='np.log10(read_count)', f
```

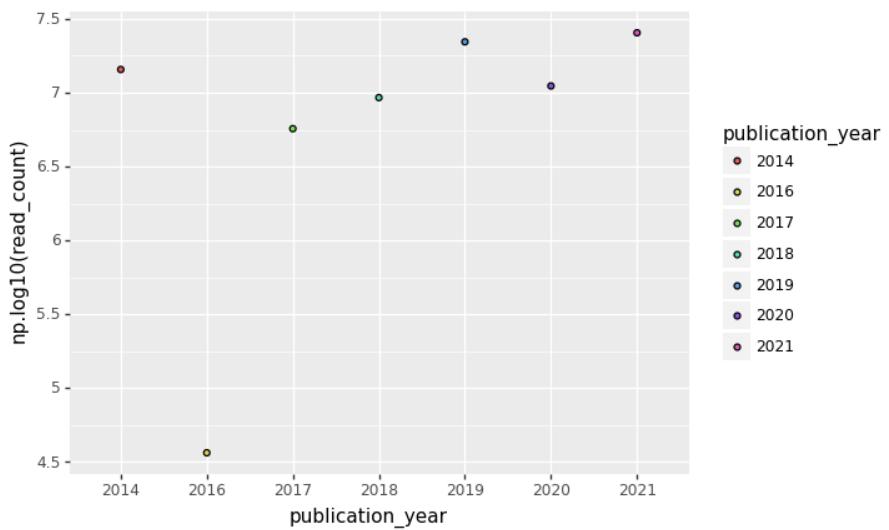


Figure 11.5: png

```
<ggplot: (366206706)>
```

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

## 11.15 11 - Bonus, dealing with ill-formatted columns

Sometimes, columns can contains 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('archiv
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
0
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  
0  
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  
0  
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

0

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 12

# Introduction to Git(Hub)

## 12.1 Introduction

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.

## 12.2 Lecture

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

## 12.3 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 you 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>.

## 12.4 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.

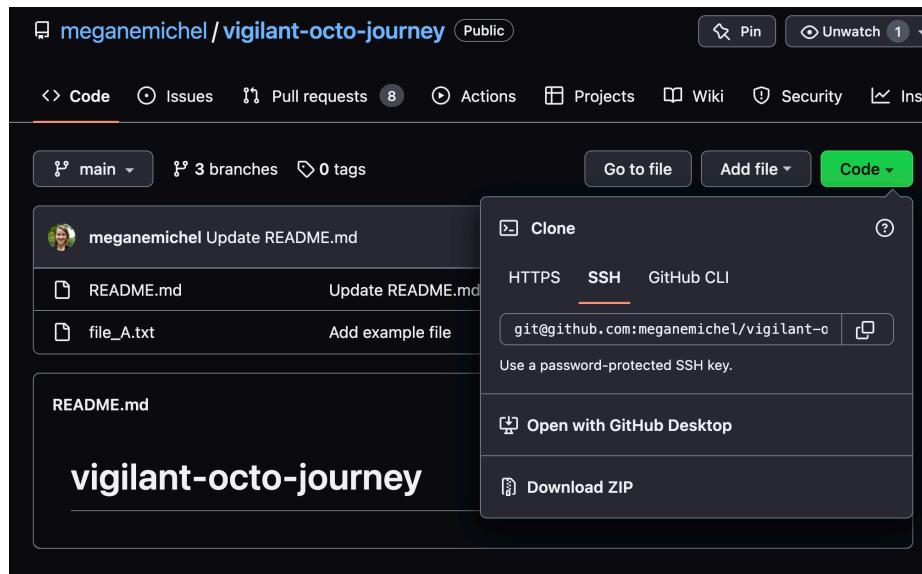
Initialize this repository with:  
Skip this step if you're importing an existing repository.

Add a README file

**i 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"
```

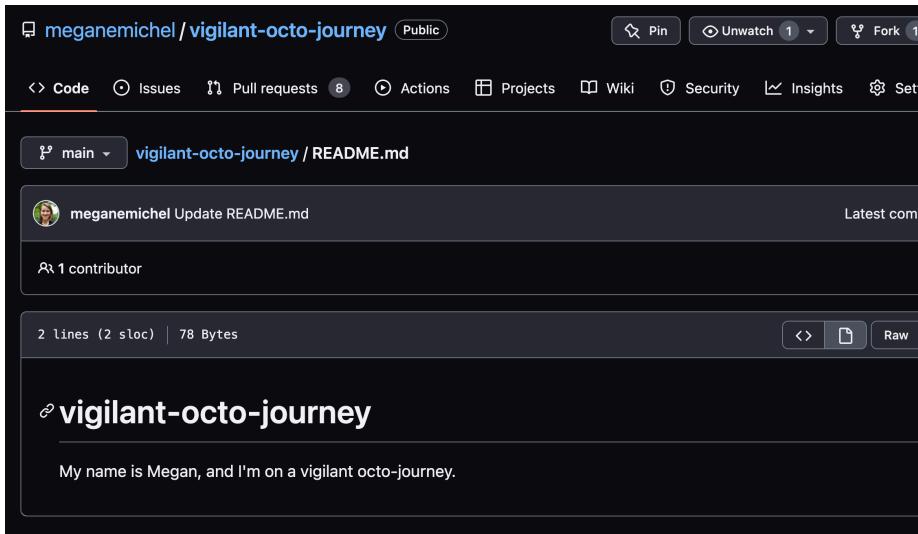
Finally, let's **push** our local commit back to our remote repository.

```
git push
```

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

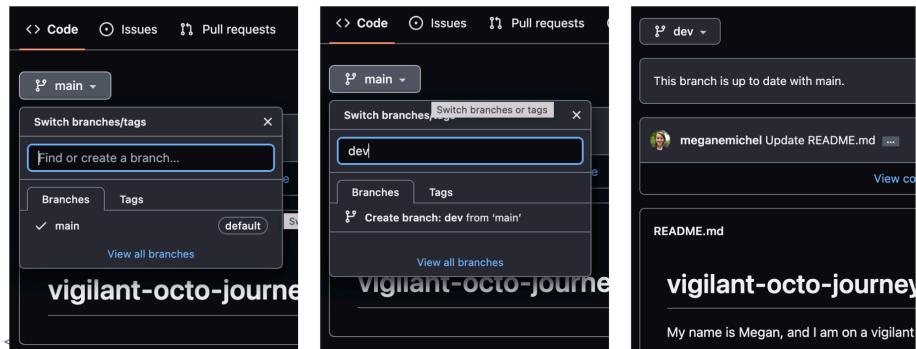
```
git pull
```

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.



## 12.5 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!

## 12.6 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>.

## 12.7 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?).



# Chapter 13

## Taxonomic Profiling, OTU Tables and Visualisation

### 13.1 Abstract

### 13.2 Lecture

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

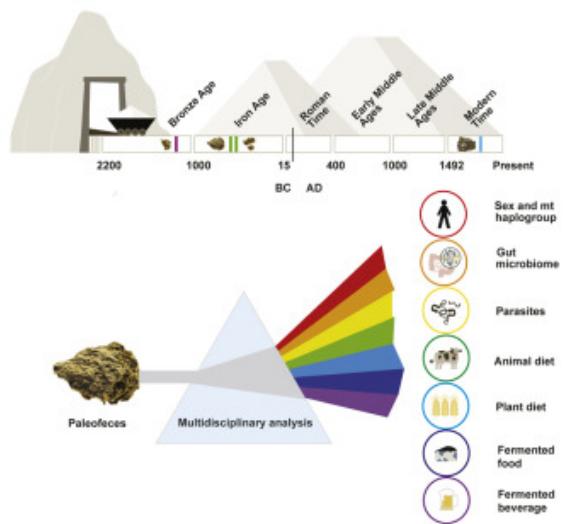
This session is run using a Jupyter notebook. This can be found [here](#). However, it will already be installed on compute nodes during the summer school.

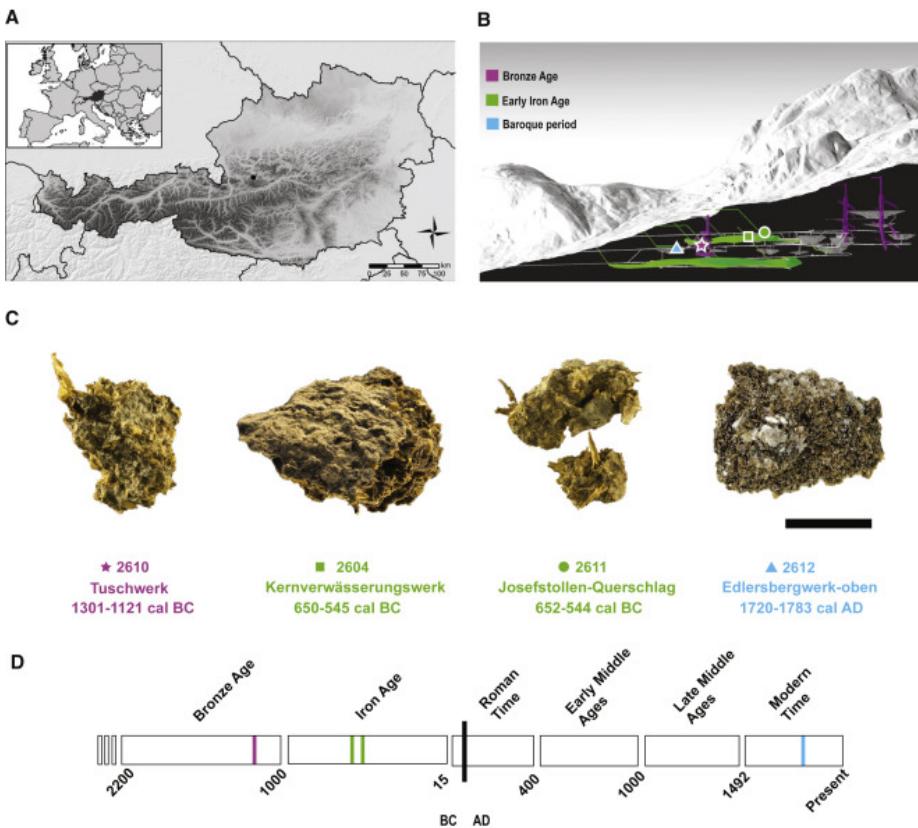
We highly recommend viewing this walkthrough via the Jupyter notebook above! The output of commands on the website for this walkthrough are displayed in their own code blocks - be wary of what you copy-paste!

### 13.3 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](#)





### 13.3.1 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
```

## 13.4 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

### 13.4.1 0. 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

### 13.4.2 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.gz' --fasta 'human_genome.fasta' --hostremoval_input_fastq
```

### 13.4.3 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
   Default is to discard it. (string [=])
  --unpaired2                            for PE input, if read2 passed QC but read1 not, it will be
   If --unpaired2 is same as --unpaired1 (default mode), both
   written to this same file. (string [=])
  --overlapped_out                       for each read pair, output the overlapped region if it has
   base. (string [=])
  --failed_out                           specify the file to store reads that cannot pass the filter
  -m, --merge                             for paired-end input, merge each pair of reads into a single
   overlapped. The merged reads will be written
   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
   controlled by --merge. In the merging mode, specify the file name to store merged
   reads. (string [=])
  --merged_out                           indicate the input is using phred64 scoring (it'll be converted
   so the output will still be phred33)
  --include_unmerged                     compression level for gzip output (1 ~ 9). 1 is fastest, 9 is slowest
  -6, --phred64                          input from STDIN. If the STDIN is interleaved paired-end FASTQ
   stream passing-filters reads to STDOUT. This option will merge
   FASTQ output for paired-end output. Disabled by default.
  -z, --compression                      --stdin
   --stdout
```

```
--interleaved_in
--reads_to_process
--dont_overwrite
--fix_mgi_id
-V, --verbose
-A, --disable_adapter_trimming
-a, --adapter_sequence
    --adapter_sequence_r2
    --adapter_fasta
    --detect_adapter_for_pe
-f, --trim_front1
-t, --trim_tail1
-b, --max_len1
-F, --trim_front2
-T, --trim_tail2
-B, --max_len2
-D, --dedup
    --dup_calc_accuracy
    --dont_eval_duplication
-g, --trim_poly_g
    --poly_g_min_len
-G, --disable_trim_poly_g
-x, --trim_poly_x
    --poly_x_min_len
-5, --cut_front
-3, --cut_tail
-r, --cut_right
-W, --cut_window_size
-M, --cut_mean_quality
    --cut_front_window_size
    --cut_front_mean_quality
    --cut_tail_window_size
    --cut_tail_mean_quality
    --cut_right_window_size
    --cut_right_mean_quality
```

indicate that <in1> is an interleaved FASTQ which  
Disabled by default.

specify how many reads/pairs to be processed. Do  
don't overwrite existing files. Overwriting is  
the MGI FASTQ ID format is not compatible with r  
output verbose log information (i.e. when every  
adapter trimming is enabled by default. If this  
the adapter for read1. For SE data, if not speci  
For PE data, this is used if R1/R2 are found no  
the adapter for read2 (PE data only). This is us  
If not specified, it will be the same as <adapt  
specify a FASTA file to trim both read1 and read  
by default, the auto-detection for adapter is fo  
option to enable it for PE data.

trimming how many bases in front for read1, defa  
trimming how many bases in tail for read1, defa  
if read1 is longer than max\_len1, then trim read  
long as max\_len1. Default 0 means no limitation

trimming how many bases in front for read2. If :  
trimming how many bases in tail for read2. If i:  
if read2 is longer than max\_len2, then trim read  
Default 0 means no limitation. If it's not spe  
enable deduplication to drop the duplicated read  
accuracy level to calculate duplication (1~6), 1  
Default 1 for no-dedup mode, and 3 for dedup mode  
don't evaluate duplication rate to save time and  
force polyG tail trimming, by default trimming :  
the minimum length to detect polyG in the read :  
disable polyG tail trimming, by default trimming  
enable polyX trimming in 3' ends.

the minimum length to detect polyX in the read :  
move a sliding window from front (5') to tail, o  
its mean quality < threshold, stop otherwise.

move a sliding window from tail (3') to front, o  
its mean quality < threshold, stop otherwise.

move a sliding window from front to tail, if mea  
< threshold, drop the bases in the window and t  
the window size option shared by cut\_front, cut\_

the mean quality requirement option shared by cu  
Range: 1~36 default: 20 (Q20) (int [=20])

the window size option of cut\_front, default to  
the mean quality requirement option for cut\_fron  
the window size option of cut\_tail, default to c  
the mean quality requirement option for cut\_tai  
the window size option of cut\_right, default to  
the mean quality requirement option for cut\_rig

```

-Q, --disable_quality_filtering
-q, --qualified_quality_phred
-u, --unqualified_percent_limit
-n, --n_base_limit
-e, --average_qual

-L, --disable_length_filtering
-l, --length_required
--length_limit
-y, --low_complexity_filter

-Y, --complexity_threshold
--filter_by_index1
--filter_by_index2
--filter_by_index_threshold
-c, --correction
--overlap_len_require
--overlap_diff_limit
--overlap_diff_percent_limit

-U, --umi
--umi_loc
--umi_len
--umi_prefix
--umi_skip
-p, --overrepresentation_analysis
-P, --overrepresentation_sampling

-j, --json
-h, --html
-R, --report_title
-w, --thread
-s, --split
-S, --split_by_lines
-d, --split_prefix_digits
--cut_by_quality5
--cut_by_quality3
--cut_by_quality_aggressive
--discard_unmerged
-?, --help

```

quality filtering is enabled by default. If this option is set to 0, quality filtering is disabled. The quality value that a base is qualified. Default 15 means that if one read's number of N base is >n\_base\_limit, then this read will be discarded. If one read's average quality score <avg\_qual, then this read will be discarded. Default 0 means no requirement (int [=0])

length filtering is enabled by default. If this option is set to 0, reads shorter than length\_required will be discarded, default 0. If one read's length longer than length\_limit will be discarded, default 0. This will enable low complexity filter. The complexity is defined as the difference between the current base and the previous base that is different from its next base (base[i] != base[i+1]). The threshold for low complexity filter (0~100). Default is 10. If this option is specified, it must be a file that contains a list of barcodes of index1 to be used for index filtering. If this option is specified, it must be a file that contains a list of barcodes of index2 to be used for index filtering. The allowed difference of index barcode for index filtering is 1. This will enable base correction in overlapped regions (only for PE reads). The minimum length to detect overlapped region of PE reads is 30 by default. (int [=30])

adapter trimming and correction. 30 by default. (int [=30])

the maximum number of mismatched bases to detect overlapping. This will affect overlap analysis based PE merge, adapter trimming and correction. 30 by default. (int [=30])

the maximum percentage of mismatched bases to detect overlap. This will affect overlap analysis based PE merge, adapter trimming and correction. 30 by default. (int [=30])

enable unique molecular identifier (UMI) preprocessing. If this option is specified, it must be a file that contains a list of barcodes of index1/index2/read1/read2. If the UMI is in read1/read2, its length should be provided. If specified, an underline will be used to connect prefix and suffix. For example, prefix=UMI, UMI=AATTCTG, final=UMI\_AATTCTG). No prefix by default. If the UMI is in read1/read2, fastp can skip several bases at once. This will enable overrepresented sequence analysis.

one in (--overrepresentation\_sampling) reads will be computed. Smaller is faster, default is 10000. (int [=10000])

analysis (1~10000), smaller is slower, default is 20. (int [=20])

the json format report file name (string [=fastp.json])

the html format report file name (string [=fastp.html])

should be quoted with ' or ", default is "fastp report" (string [=fastp report])

worker thread number, default is 3 (int [=3])

split output by limiting total split file number with this option. This will be added to output name ( 0001.out.fq, 0002.out.fq...)

split output by limiting lines of each file with this option. This will be added to output name ( 0001.out.fq, 0002.out.fq...), disable padding. The digits for the sequential number padding (1~10), default is 4. 0 to disable padding (int [=4])

DEPRECATED, use --cut\_front instead.

DEPRECATED, use --cut\_tail instead.

DEPRECATED, use --cut\_right instead.

DEPRECATED, no effect now, see the introduction for merging.

print this message

```

%%bash
fastp \
--in1 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.fwd.fq_subsample_10000
--in2 ../data/subsampled/ERR5766177_PE.mapped.hostremoved.fwd.fq_subsample_10000
--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 --mer
--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):

METAg genomic PHylogenetic ANalysis for metagenomic taxonomic profiling.

AUTHORS: Francesco Beghini (francesco.beghini@unitn.it), Nicola Segata (nicola.segata@unitn.it), D  
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.

---

===== 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 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 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 file you should provide the metagenome size:  
\$ metaphlan metagenome.bowtie2.bz2 --nproc 5 --input\_type bowtie2out --nreads 520000 --size
  - \* 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 composed of multiple files (but you need to specify the --bowtie2out parameter):  
\$ metaphlan metagenome\_1.fastq,metagenome\_2.fastq --bowtie2out metagenome.bowtie2.bz2
- 

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

MetaPhlAn introduces the capability of characterizing organisms at the strain level using aggregated marker information. Such capability comes with several slightly different features that 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 strains of interest. This operation can be performed quickly as it exploits the --bowtie2out option.

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.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.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.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 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 run of MetaPhlAn. If the input file is missing, the script assumes that the input is provided via 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 DEFAULT_DB_FOLDER variable in the shell.
                           [default /Users/maxime/mambaforge/envs/summer_school_microbiome]
-x INDEX, --index INDEX
                           Specify the id of the database version to use. If "latest", MetaPhlAn will automatically download it.
                           If an index name is provided, MetaPhlAn will try to use it, if it exists.
                           If the database files are not found on the local MetaPhlAn instance, it 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 allows you to use a specific executable even when it is not in the system PATH or the system's PATH environment variable.
--bowtie2_build BOWTIE2_BUILD
                           Full path to the bowtie2-build command to use, deafult assumes 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 default]
```

## 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 abundance [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
                        [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
                                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-zero count
                        * marker_ab_table: normalized marker counts (only when > 0.0 and normalized)
                        * marker_counts: non-normalized marker counts [use with extreme caution]
                        * marker_pres_table: list of markers present in the sample (threshold at 0.0)
                        * clade_specific_strain_tracker: list of markers present for a specific clade
                        [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
                        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_sp

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 'Sample'
  --use_group_representative
                           Use a species as representative for species groups.
  --sample_id value     Specify the sample ID for this analysis. Defaults to 'Metaphlan'
  -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
  --biom biom_output, --biom_output_file biom_output
                           If requesting biom file output: The name of the output file in
  --mdelim mdelim, --metadata_delimiter_char mdelim
                           Delimiter for bug metadata: - defaults to pipe. e.g. the pipe |

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 needed
  --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
                           '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 ...
--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
```

### 13.4.4 4. Visualizing the taxonomic profile

#### 13.4.4.1 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

- 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

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

#### 13.4.4.2 4.2 Visualizing metaphlan taxonomic profile with Krona

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

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

### 13.4.5 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](#)

#### 13.4.6 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 informations.

```
! pip install taxopy
```

```
Requirement already satisfied: taxopy in /Users/maxime/mambaforge/envs/summer_school_m
```

```

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_CHOCOPhlan_201901
#/home/maxime_borry/.conda/envs/maxime/envs/summer_school_microbiome/bin/metaphlan ../results/fas
--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','additional_species'])

ancient_data.head()

clade_name
NCBI_tax_id
relative_abundance
additional_species
0
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 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 corres
    else:

```

## 190 CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION

```
        return taxopy.Taxon(12908, taxo_db) # otherwise, return the taxon associated with the taxid
    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
0
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__Actinobacteria
```

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 191

```
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
0
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...

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 193

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, Prevellaceae,...  
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
```

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 195

[Ruminococcus bromii, Ruminococcus, Oscillosp...

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').rename(columns={
```

```
    'relative_abundance': 'abundance'}
```

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').r  
  
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()

de028ad4-7ae6-11e9-a106-68b59976a384
PNP_Main_283
PNP_Validation_55
G80275
PNP_Main_363
SAMEA7045572
SAMEA7045355
HD-13
EGAR00001420773_9002000001423910
SID5428-4
...
A46_02_1FE
TZ_87532
A94_01_1FE
KHG_7
LDK_4
KHG_9
A48_01_1FE
KHG_1
TZ_81781
A09_01_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

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 199

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

200 *CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION*

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  
...

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 201

```
0.05011  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
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_  
  
modern_species.head()  
  
de028ad4-7ae6-11e9-a106-68b59976a384  
PNP_Main_283  
PNP_Validation_55  
G80275  
PNP_Main_363  
SAMEA7045572  
SAMEA7045355  
HD-13  
EGAR00001420773_9002000001423910  
SID5428-4  
...  
A46_02_1FE  
TZ_87532  
A94_01_1FE  
KHG_7  
LDK_4
```

202CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION

KHG\_9  
A48\_01\_1FE  
KHG\_1  
TZ\_81781  
A09\_01\_1FE  
Bacteroides vulgatus  
0.0  
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



204 CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION

0.0  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.00000  
0.0  
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

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 205

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='outer')
all_phylums = ancient_phylums.merge(modern_phylums, left_index=True, right_index=True, how='outer')
```

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
0
ShaoY_2019
de028ad4-7ae6-11e9-a106-68b59976a384
C01528_ba
stool
no
control
healthy
```

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 207

0.0  
4.0  
newborn  
...  
NaN  
1  
ZeeviD\_2015  
PNP\_Main\_283  
PNP\_Main\_283  
stool  
no  
control  
healthy  
NaN  
NaN  
adult  
...  
NaN  
NaN  
NaN  
NaN  
NaN

NaN  
NaN  
NaN  
NaN  
NaN  
2  
ZeeviD\_2015  
PNP\_Validation\_55  
PNP\_Validation\_55  
stool  
no  
control  
healthy  
NaN  
NaN  
adult  
...  
NaN  
3  
VatanenT\_2016  
G80275  
T014806

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 209

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

```

NaN
5 rows × 130 columns

```

### 13.4.7 7. Comparing ancient and modern samples

#### 13.4.7.1 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
    .to_frame(name='group').set_index(metadata['sample_id']) # rename the column to
    .reset_index()
    .append({'sample_id': 'ERR5766177', 'group': 'ancient'}, ignore_index=True) # add
)
group_info

/var/folders/1c/11qb09f15jddsh65f6xv1n_r0000gp/T/ipykernel_40830/27419655.py:2:
FutureWarning: The frame.append method is deprecated and will be removed from pandas in
Use pandas.concat instead.
    metadata['non_westernized']

sample_id
group
0
de028ad4-7ae6-11e9-a106-68b59976a384
westernized

```

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 21

1

PNP\_Main\_283

westernized

2

PNP\_Validation\_55

westernized

3

G80275

westernized

4

PNP\_Main\_363

westernized

...

...

...

196

A48\_01\_1FE

non\_westernized

197

KHG\_1

non\_westernized

198

TZ\_81781

non\_westernized

199

A09\_01\_1FE

non\_westernized

200

ERR5766177

ancient

201 rows × 2 columns

## 212CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION

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
```

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 213

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

0.0

0.0

0.00000

ERR5766177

ancient

0

0.25809

0.0

0.0

0.00000

0.0

0.0

0.0

214 *CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION*

0.0  
0.0  
0.0  
...  
0.0  
0.00000  
0.0  
4.49959  
0.00000  
0.0  
0.0  
0.00000  
de028ad4-7ae6-11e9-a106-68b59976a384  
westernized  
1  
10.27631  
0.0  
0.0  
17.44332  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
...  
0.0  
0.01486  
0.0  
0.77098  
0.00000

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 215

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  
0.0  
...  
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  
0.0  
0.0  
0.0  
0.0  
0.00000  
0.0  
0.19699  
0.00000  
0.0  
0.0  
0.00000  
KGH\_9  
non-westernized

196

5.47750

0.0

0.0

14.81828

0.0

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

3.64655

0.09964

0.0

0.0

0.00000

A48\_01\_1FE

non\_westernized

197

36.77145

0.0

0.0

10.13908

0.0

0.0

0.0

0.0

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 219

0.0

0.0

...

0.0

0.00000

0.0

17.64151

0.00000

0.0

0.0

0.00000

KHG\_1

non\_westernized

198

1.16426

0.0

0.0

57.14031

0.0

0.0

0.0

0.0

0.0

...

0.0

0.00000

0.0

0.30580

0.70467

0.0

220CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION

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  
0.0

56.20177  
0.00000

0.0  
0.0  
0.00000

A09\_01\_1FE

non\_westernized

201 rows × 24 columns

Now, we need it in the tidy format

```
tidy_phylums = (  
  all_phylums
```

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 221

```
.transpose()  
.merge(group_info, left_index=True, right_on='sample_id')  
.melt(id_vars=['sample_id', 'group'], value_name='relative_abundance', var_name='Phylum',  
)
```

Finally, we only want to keep the mean relative abundance for each phylum

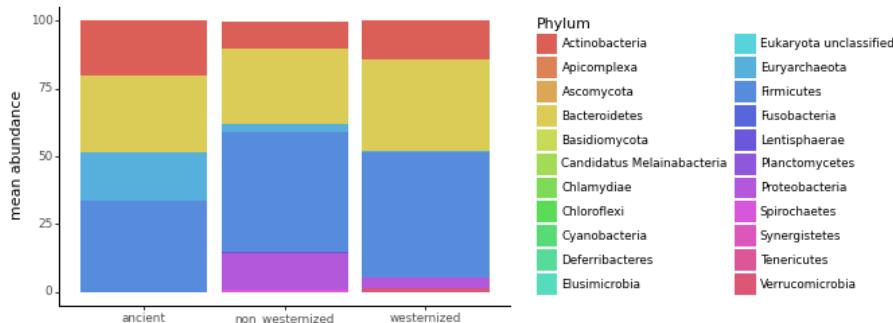
```
tidy_phylums = tidy_phylums.groupby(['group', 'Phylum']).mean().reset_index()
```

```
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)>
```

#### 13.4.7.2 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
de028ad4-7ae6-11e9-a106-68b59976a384
0.798112
0.251280
11
PNP_Main_283
5.092878
0.954159
118
PNP_Validation_55
3.670162
0.812438
72
G80275
```

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 223

3.831358

0.876712

66

...

...

...

...

KHG\_9

3.884285

0.861683

87

A48\_01\_1FE

4.377755

0.930024

53

KHG\_1

3.733834

0.875335

108

TZ\_81781

2.881856

0.719491

44

A09\_01\_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='sa
    .set_index('sample_id')
    .rename(columns={'non_westernized':'group'})
)
alpha_diversity['group'] = alpha_diversity['group'].replace({'yes':'non_westernized'})
```

alpha\_diversity

shannon

simpson

richness

group

sample\_id

ERR5766177

3.032945

0.844769

21

ERR5766177

de028ad4-7ae6-11e9-a106-68b59976a384

0.798112

0.251280

11

westernized

PNP\_Main\_283

5.092878

0.954159

118

westernized

PNP\_Validation\_55

3.670162

0.812438

### 13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 225

72

westernized

G80275

3.831358

0.876712

66

westernized

...

...

...

...

...

KHG\_9

3.884285

0.861683

87

non\_westernized

A48\_01\_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
```

```
A09_01_1FE
```

```
2.982322
```

```
0.719962
```

```
75
```

```
non_westernized
```

```
201 rows × 4 columns
```

```
alpha_diversity = alpha_diversity.melt(id_vars='group', value_name='alpha diversity')
```

```
alpha_diversity
```

```
group
```

```
diversity_index
```

```
alpha diversity
```

```
sample_id
```

```
ERR5766177
```

```
ERR5766177
```

```
shannon
```

```
3.032945
```

```
de028ad4-7ae6-11e9-a106-68b59976a384
```

```
westernized
```

```
shannon
```

```
0.798112
```

```
PNP_Main_283
```

```
westernized
```

```
shannon
```

```
5.092878
```

```
PNP_Validation_55
```

```
westernized
```

```
shannon
```

```
3.670162
```

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 227

G80275

westernized

shannon

3.831358

...

...

...

...

KHG\_9

non\_westernized

richness

87.000000

A48\_01\_1FE

non\_westernized

richness

53.000000

KHG\_1

non\_westernized

richness

108.000000

TZ\_81781

non\_westernized

richness

44.000000

A09\_01\_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':'#5B9BD5'})
g += theme(subplots_adjust={'wspace': 0.15})
g

```

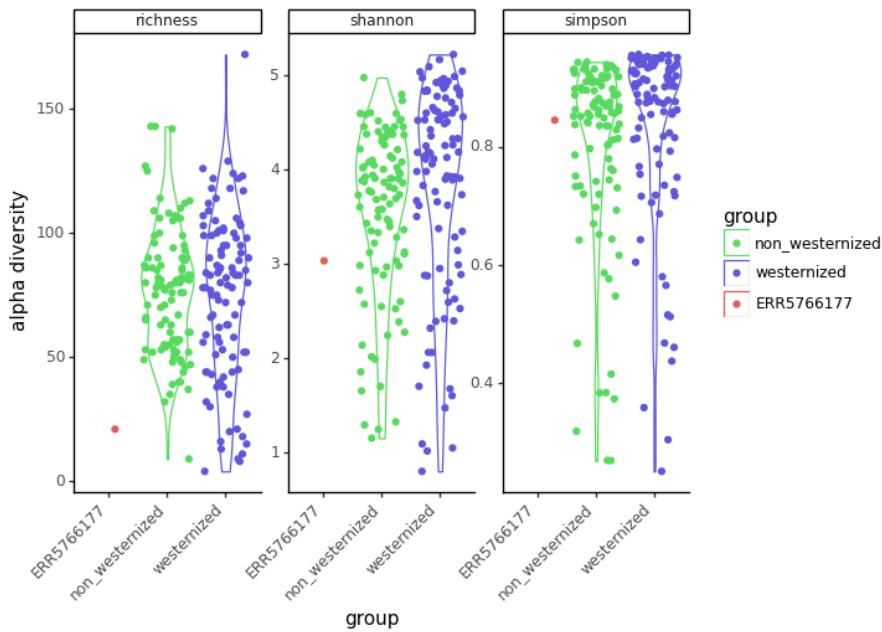


Figure 13.1: png

&lt;ggplot: (407407577)&gt;

**Pause and think:** Why do we observe a smaller species richness and diversity in our sample ?

#### 13.4.7.3 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:103: UserWarning: The result contains negative eigenvalues. Please compare their magnitude with the magnitude of the positive ones. If the negative ones are smaller, it's probably safe to ignore them, but if they are large in magnitude, 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

230 *CHAPTER 13. TAXONOMIC PROFILING, OTU TABLES AND VISUALISATION*

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

0.0

0.0

0.0

0.0

0.0

0.0

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 231

0.0  
0.0  
0.0  
de028ad4-7ae6-11e9-a106-68b59976a384  
-0.099355  
0.145224  
-0.191676  
0.127626  
0.119754  
-0.132209  
-0.097382  
0.036728  
0.081294  
-0.056686  
...  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0  
PNP\_Main\_283  
-0.214108  
-0.147466  
0.116027  
0.090059  
0.076644







13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 235

...

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

0.0

A48\_01\_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

0.0

0.0

0.0

0.0

0.0

0.0

0.0
0.0
KHG_1
-0.100009
0.167885
0.009915
0.076842
-0.405582
-0.039111
-0.006421
-0.009774
-0.072252
0.150000
...
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
TZ_81781
0.405716
-0.139297
-0.075026
-0.079716
-0.053264
-0.119271

13.4. HANDS ON INTRODUCTION TO ANCIENT MICROBIOME ANALYSIS 237

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

A09\_01\_1FE

0.089101

0.471135

0.069629

-0.125644

-0.036793

0.115151

0.060507

-0.000912

-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') .re

ggplot(var_explained, aes(x='PC', y='variance explained', group=1)) \
+ geom_point() \
+ geom_line() \
+ theme_classic()
```

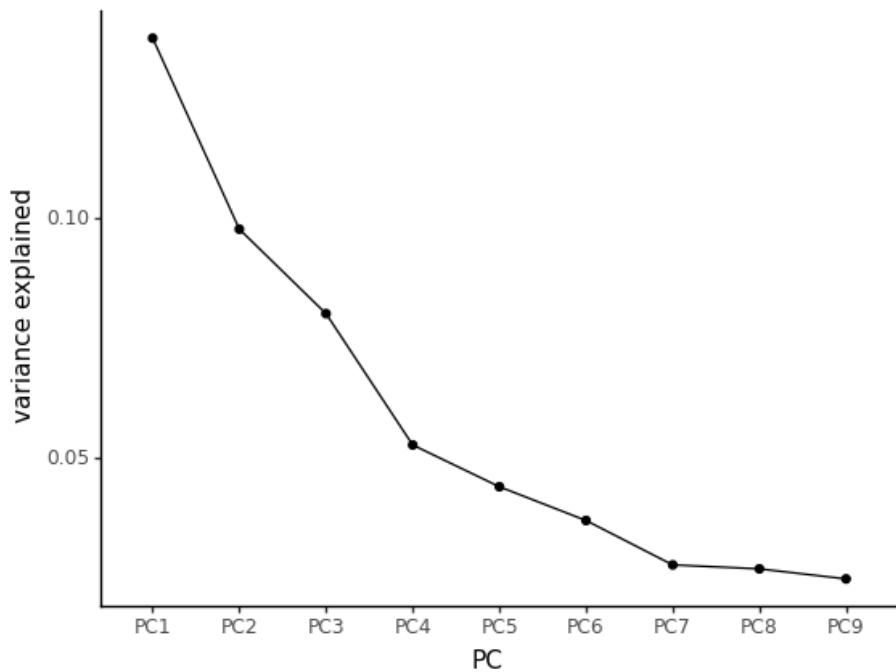


Figure 13.2: png

```
<ggplot: (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_id',
           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

```
ggplot(pcoa_embed, aes(x='PC1', y='PC2', color='group')) \
+ geom_point() \
+ theme_classic() \
+ scale_color_manual({'ERR5766177': '#DB5F57', 'westernized': '#5F57DB', 'non_westernized': '#57DB57'})
```

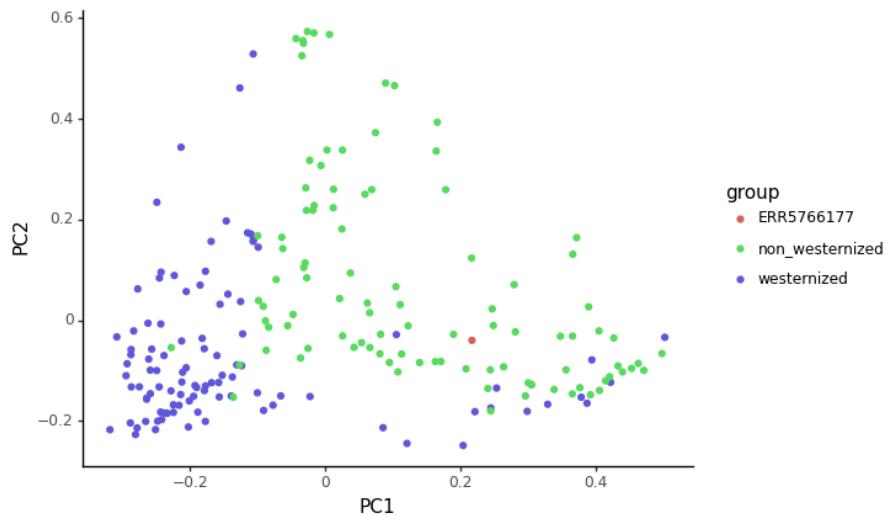


Figure 13.3: png

```
<ggplot: (407572134)>
```

```
ggplot(pcoa_embed, aes(x='PC1', y='PC3', color='group')) +
  geom_point() +
  theme_classic() +
  scale_color_manual({'ERR5766177':'#DB5F57','westernized': '#5F57DB','non_westernized'}
```

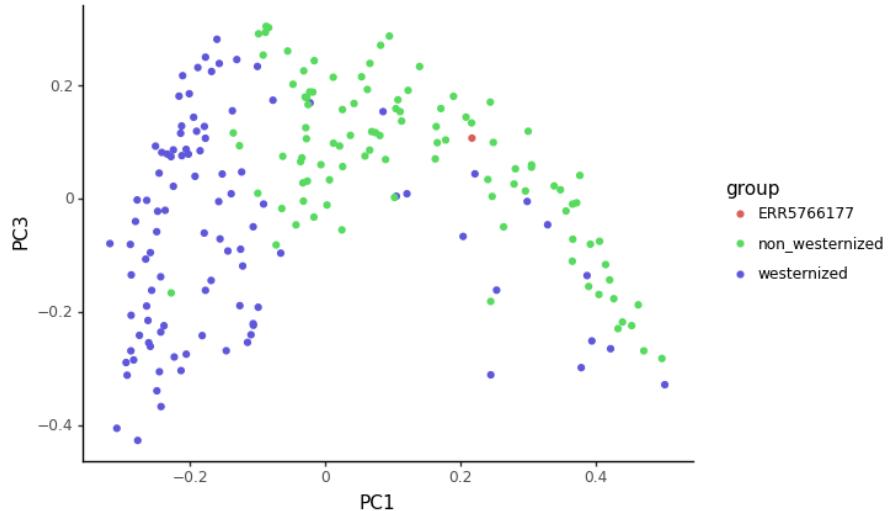


Figure 13.4: png

&lt;ggplot: (407612651)&gt;

Then with a 3d plot

```
import plotly.express as px

fig = px.scatter_3d(pcoa_embed, x="PC1", y="PC2", z="PC3",
                     color = "group",
                     color_discrete_map={'ERR5766177':'#DB5F57','westernized': '#5F57DB',
   'non_westernized': '#5F57DB'},
                     hover_name="sample")
fig.show()
```

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'})

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()
)

<seaborn.matrix.ClusterGrid at 0x185b56100>

```

### 13.4.8 8. Additional steps

#### 13.4.8.1 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 SourceTracker

```

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_
```

Converting to **biom** format

```
!biom convert -i ../results/sourcetracker2/all_species_counts.tsv \
-o ../results/sourcetracker2/all_species_counts.biom \
```

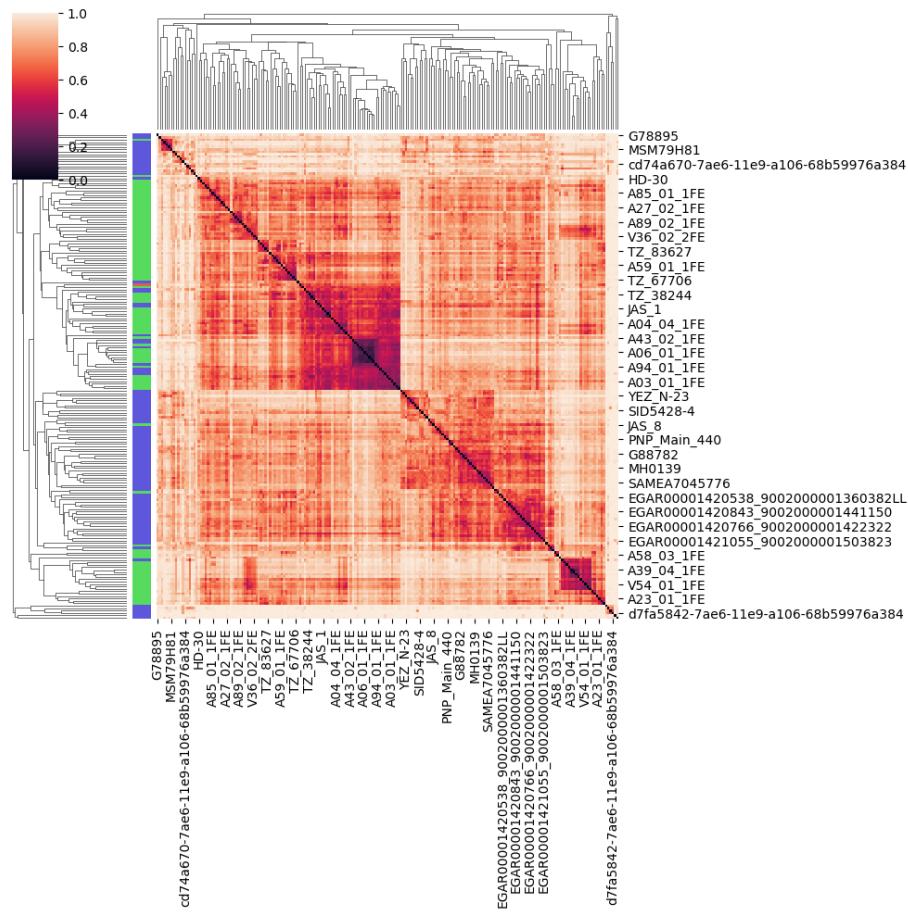


Figure 13.5: png

```
--table-type="Taxon table" --to-hdf5
```

Converting the metadata to Sourcetracker format

```
st2_metadata = metadata[['sample_id', 'non_westernized']].rename(columns={'non_westernized': 'Env'})  
st2_metadata['Env'] = st2_metadata['Env'].replace({'yes': 'non_westernized', 'no': 'westernized'})  
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/l1qb09f15jddsh65f6xv1n_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 pa  
  
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](#))

#### 13.4.8.2 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 14

## Introduction to *de novo* Genome Assembly

### 14.1 Abstract

*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 session, 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.

#### 14.1.1 Lecture

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

### 14.2 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.



# Chapter 15

## Introduction to Genome Mapping

### 15.1 Abstract

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 session 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 down-stream analyses such as phylogenetics.

## 15.2 Lecture

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

### 15.3 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.

#### 15.3.1 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.

#### 15.3.2 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
```

### 15.3.3 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
```

#### 15.3.4 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.sai > reads_mapped.sam
```

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
```

### 15.3.5 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
```

Let's have a look...

```
cat mysnps.vcf | less -S
```

(exit by pressing q)

### 15.3.6 Mapping and Genotyping for the other Samples/Parameters

Let's now continue with mapping and genotyping for the other samples and parameter settings.

#### 15.3.6.1 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

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
```

#### 15.3.6.2 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.sai

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

```

#### 15.3.6.3 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.sai

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

```

#### 15.3.7 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/mysn
```

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 unha
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).

### 15.3.8 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/>).

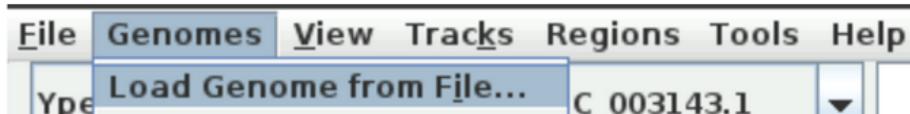
To open IGV, simply type the following command and the app will open:

```
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.

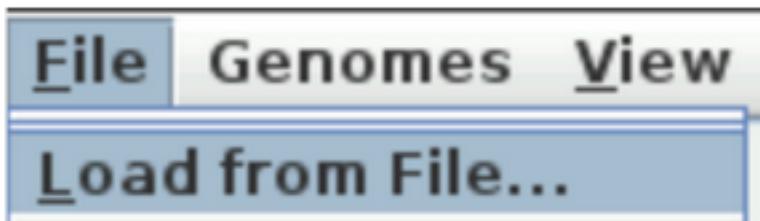
Load your reference (`YpestisC092.fa`):

→ *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?

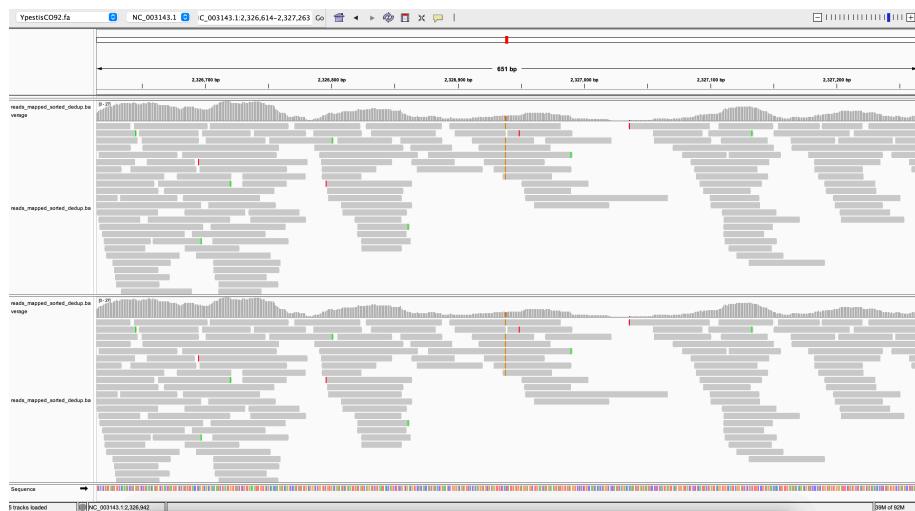
Let's examine some SNPs. Have a look at `snpTable.tsv`.

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?

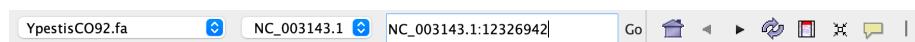
### 15.3.9 Examples

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.

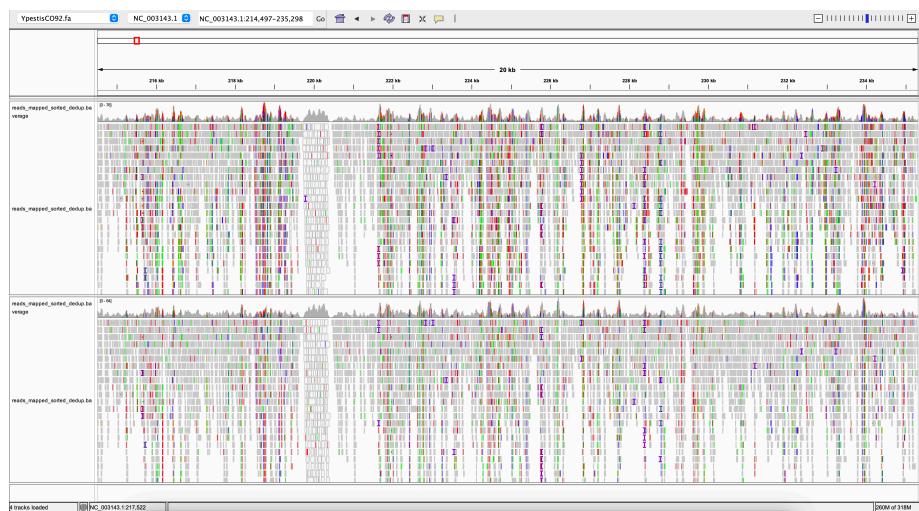
Does this mean that stricter parameters will always give us a clean mapping? Let's have a look at position 219200:



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:



### 15.3.10 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 16

# Introduction to Phylogenomics



# Chapter 17

## Abstract

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 session, 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.



# Part V

## Ancient Metagenomic Resources



# Chapter 18

## Introduction to AncientMetagenomeDir

### 18.1 Abstract

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.

### 18.2 Lecture

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

### 18.3 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 publicly 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’.

## 18.4 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.

## 18.5 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	7.857	-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	-5.328	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 continue to be regularly updated as the community continues to submit new metadata of samples of new publications as they come out.

## 18.6 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.

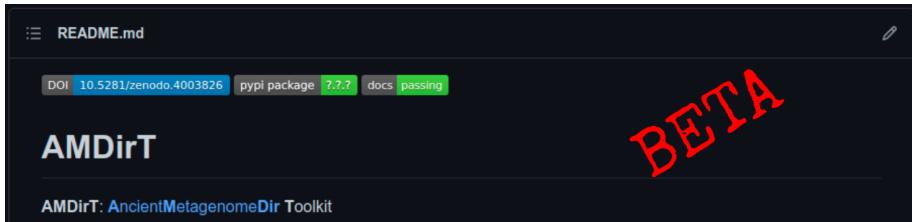
This metadata includes things such as whether a given set of data files contain sequencing data sequenced on which platform, whether the libraries have undergone damage treatment in the lab, or whether the uploaded data contains all or only mapped reads.

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 [AMDiT](#). 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!

## 18.7 Running AMDirT



First, we will need to activate a conda environment, and then install the latest development version of the tool for you.

this tutorial will require a web-browser! Make sure to run on your local laptop/PC or on a server with X11 forwarding

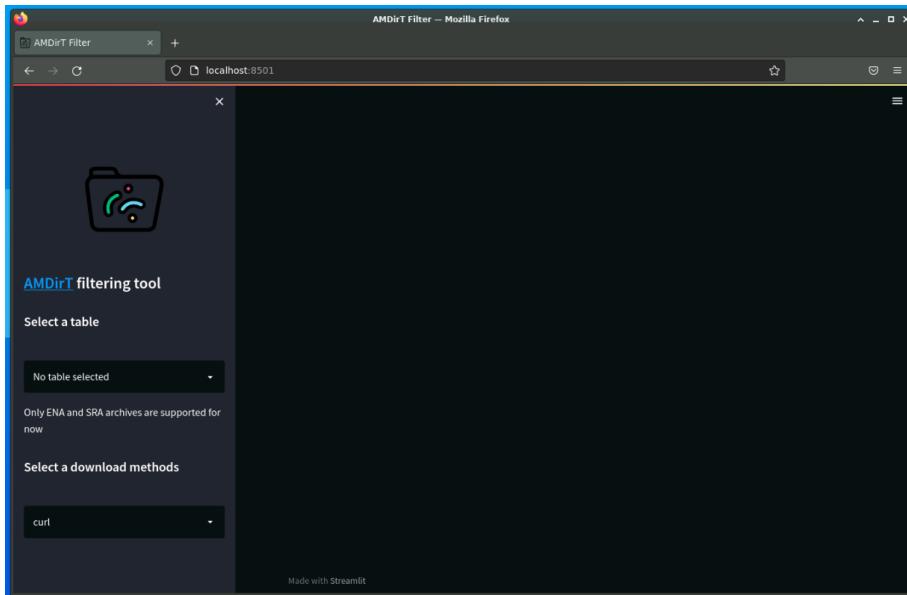
Open your terminal, and run the following two commands:

```
conda activate git-eager
pip install --upgrade --force-reinstall git+https://github.com/SPAAAM-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.

The screenshot shows the AMDirT Filter web application running in Mozilla Firefox. The URL is localhost:8501. The main interface has a sidebar on the left with a logo, the title 'AMDirT filtering tool', a 'Select a table' dropdown set to 'ancientsinglegenome-hostassociated' (which is highlighted with a red box), and a 'Select a download methods' dropdown set to 'curl'. The main content area is titled 'Displayed table: ancientsinglegenome-hostassociated'. It shows a table with columns: project\_name, publication\_year, publication\_doi, and site\_name. The table lists 16 rows of data, mostly from Schuenemann et al. (2013) and 2018, across various sites like Siguna, St. Jørgen cemetery, Odense, and Winchester. A 'Validate selection' button is at the bottom of the table.

project_name	publication_year	publication_doi	site_name
Schuenemann2013	2013	10.1126/science.1238286	Siguna
Schuenemann2013	2013	10.1126/science.1238286	St. Jørgen cemetery, Odense
Schuenemann2013	2013	10.1126/science.1238286	Reßhale
Schuenemann2013	2013	10.1126/science.1238286	St. Mary Magdalene leprosarium, Winchester
Schuenemann2013	2013	10.1126/science.1238286	St. Mary Magdalene leprosarium, Winchester
Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
Schuenemann2018	2018	10.1371/journal.ppat.1006997	Odense St. Jørgen cemetery
Schuenemann2018	2018	10.1371/journal.ppat.1006997	Szentendre-Kőszike
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	2021	0.1126/science.ab15658	Alto de Rodilla
Kocher2021	2021	0.1126/science.ab15658	Akbeit
Kocher2021	2021	0.1126/science.ab15658	Alalakh
Kocher2021	2021	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	
52.08	0.18	United Kingdom	
52.26	0.061	United Kingdom	
52.9	0.544	United Kingdom	

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 main window title is "AMDit Filter - Mozilla Firefox" and the address bar shows "localhost:8501". On the left, there's a sidebar with a logo, the text "AMDit filtering tool", a dropdown menu set to "ancientsinglegenome-hostassociated", and a note stating "Only ENA and SRA archives are supported for now". Below that is a "Select a download methods" dropdown set to "curl". The right side displays a table titled "Displayed table: ancientsinglegenome-hostassociated". The table has columns: project\_name, publication\_year, publication\_doi, and site\_name. It lists four entries for "Kocher2021" with the following details:

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 located 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!

This screenshot shows the same AMDirT Filter interface after validating the sample selection. The "Validate selection" button is now highlighted with a red border. Below it, a message says "4 samples selected". Underneath, there are three new buttons: "Download Curl sample download script", "Download nf-core/eager input TSV", and "Download Citations as BibTex". At the bottom of the interface 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.

## 18.8 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.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/008/ERR6053618/ERR6053618.fastq.gz -o ERR6053618.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/005/ERR6053675/ERR6053675.fastq.gz -o ERR6053675.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/006/ERR6053686/ERR6053686.fastq.gz -o ERR6053686.fastq.gz
```

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
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/008/ERR6053618/ERR6053618.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/005/ERR6053675/ERR6053675.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/006/ERR6053686/ERR6053686.fastq.gz
```

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
I0161 ERR6053619 0 4 SE Homo sapiens double unknown ERX5692505_ERR6053619
OAI017 ERR6053675 0 4 SE Homo sapiens double half ERX5692561_ERR6053675
SED009 ERR6053686 0 4 SE Homo sapiens double half ERX5692572_ERR6053686
```

Finally, we can look into the `citations` file which will provide you with the citation information of all the downloaded data and AncientMetagenomeDir itself.

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
Becky Cribdon and Irina M. Velsko and Maxime Borry and Miriam J. Bravo-Lopez and
and Eleanor J. Green and Shreya L. Ramachandran and Peter D. Heintzman and Maria
H\"ubner and Abigail S. Gancz and Jessica Hider and Aurora F. Allshouse and Valentine
title = {Community-curated and standardised metadata of published ancient metage-
```

```
journal = {Scientific Data}  
}
```

This file can be easily loaded into most reference managers and then have all the citations quickly added to your manuscripts.

## 18.9 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

## 18.10 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 19

## Ancient Metagenomic Pipelines

### 19.1 Abstract

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 preprocessing, 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!

### 19.2 Lecture

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

### 19.3 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.

### 19.4 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:

1. **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.
2. **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!
3. **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.

### 19.5 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 genomics 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!

## 19.6 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, GLZ002 and KZL002. GLZ002 comes from the Neolithic Siberian site of Glazkovskoe predmestie and was radiocarbon dated to 3081-2913 calBCE. KZL002 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 #Generate variant calling statistics
```

For full parameter documentation, click [here](#).

And now we wait...

## 19.7 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 crashed 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_ASM129341v1_genomic.fna \
--input "data/*fastq.gz" <...>
```

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.

## 19.8 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?



# Chapter 20

## Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```



## **Part VI**

# **Appendices**



# Chapter 21

## Resources

### 21.1 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 22**

### **Tools**

