Introduction to Ancient Metagenomics

2023-07-05

Table of contents

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 Andrades 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 inter-disciplinary colleagues.

Chapter 2

Acknowledgements

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

2.1 Financial Support

WSS
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 examime 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 patters 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 phylognenetic 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 `tidyr`
  - `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-h
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.frame`s and `tibble`s

```
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
```

- `tibble`s 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 `tibble`s

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
) +              # The plot consists of functions linked with +
geom_bar(        # "geoms" define the plot layers we want to draw
mapping = aes( # The aes() function maps variables to visual properties
    x = publication_year, # publication_year -> x-axis
    fill = community_type # community_type -> fill color
)
)
```

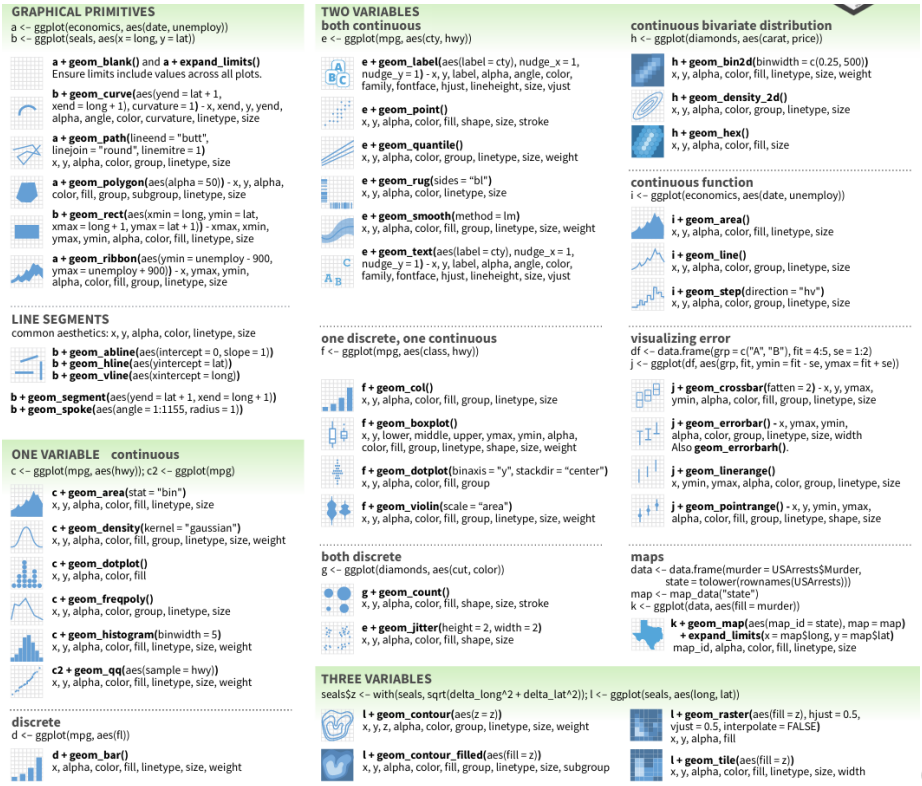
- `geom_*`: data + geometry (bars) + statistical transformation (sum)

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



(){width=55%}

- 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

```r
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")

```r
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

```r
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

```r
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_tibb
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 handeld more easily with the `forcats` package

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

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(                                              # add a column that
    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 columns
    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

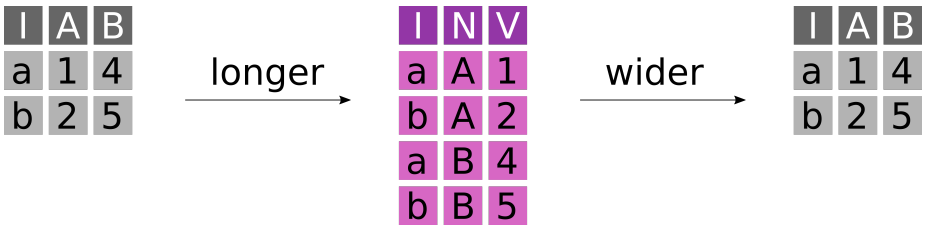
```
samples %>% dplyr::mutate(hemi = ifelse(latitude >= 0, "North", "South")) %$% h

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

1. Move the column `gear` to the first position of the mtcars dataset

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

2. 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")
)
```

3. 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/b187df6ebd23dfeb42935f
ancientmetagenome-hostassociated/libraries/ancientmetagenome-hostassociated_libraries.tsv"

libraries <- readr::read_tsv(library_table_url)
print(libraries, n = 3)
```

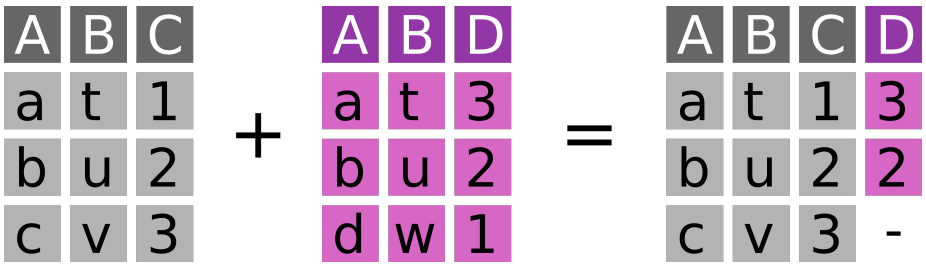
10.8.3 Meaningful subsets

```
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

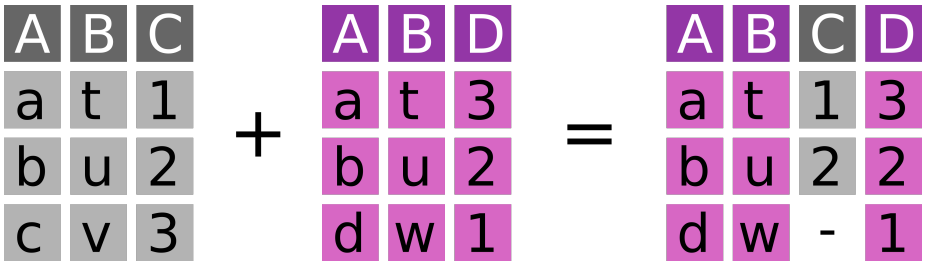



```
left <- dplyr::left_join(
x = samsub,                          # 1060 observations
y = libsub,                          # 1657 observations
by = c("project_name", "sample_name") # the key columns by which to join
)

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

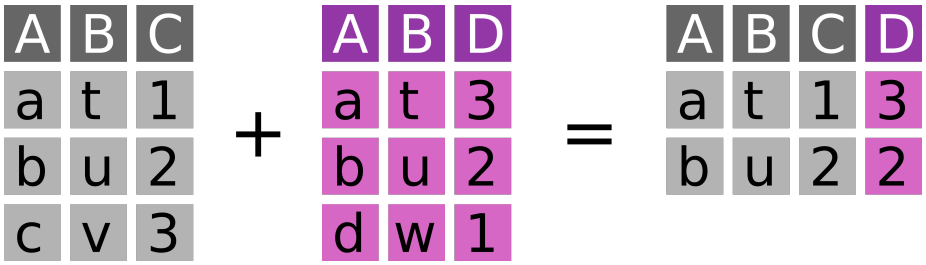



```
right <- dplyr::right_join(
  x = samsub,                              # 1060 observations
  y = libsub,                              # 1657 observations
  by = c("project_name", "sample_name")
)

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

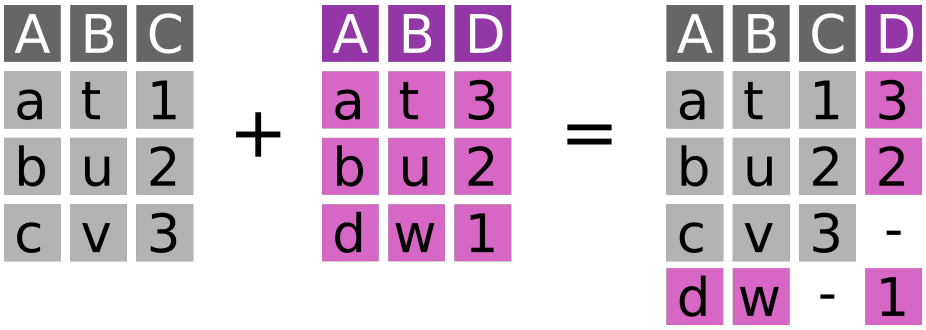



```
inner <- dplyr::inner_join(
x = samsub,                              # 1060 observations
y = libsub,                              # 1657 observations
by = c("project_name", "sample_name")
)

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

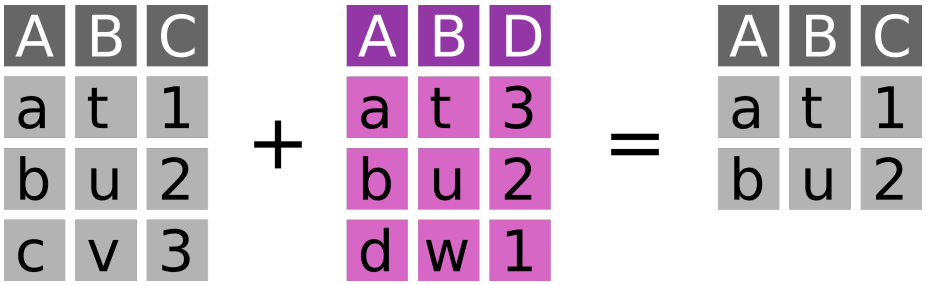



```
full <- dplyr::full_join(
  x = samsub,                              # 1060 observations
  y = libsub,                              # 1657 observations
  by = c("project_name", "sample_name")
)

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




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

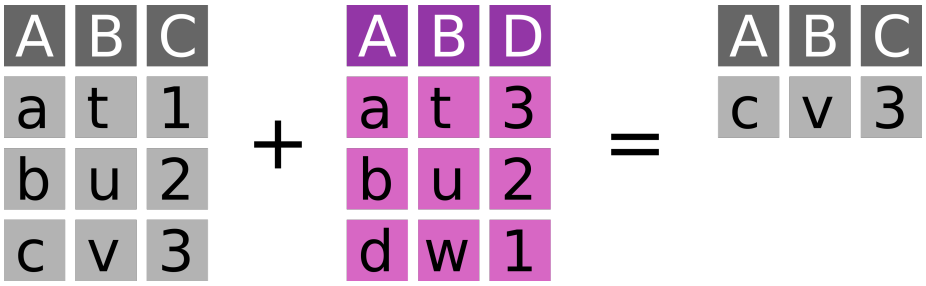
```
)
```

```
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




```
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 visulization with plotnine

Maxime Borry
SPAAM Summer School 2022

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

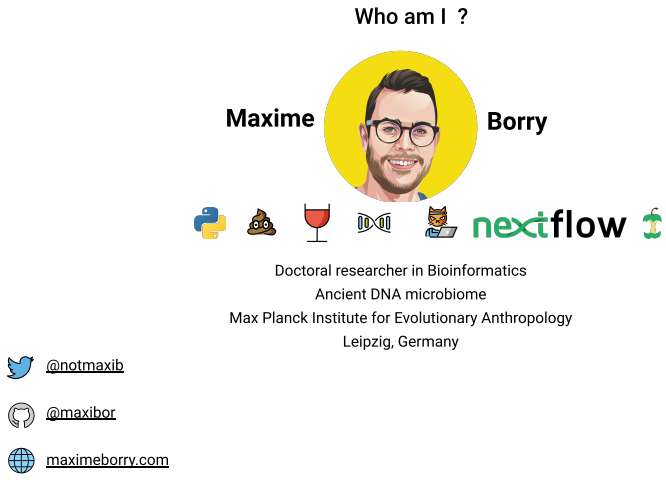


Figure 11.1: svg

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
```

- 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__
```

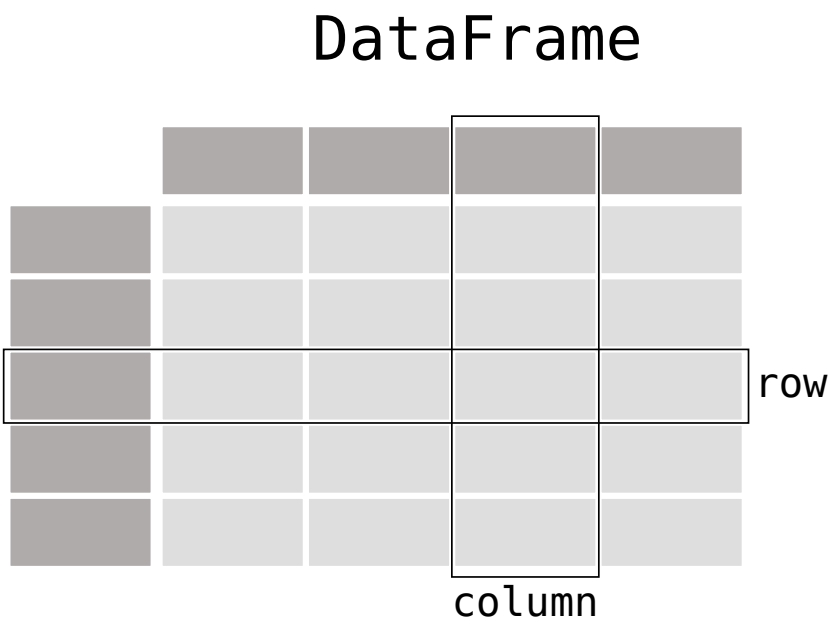
```
'1.23.1'
```

```
! conda list | grep plotnine
```

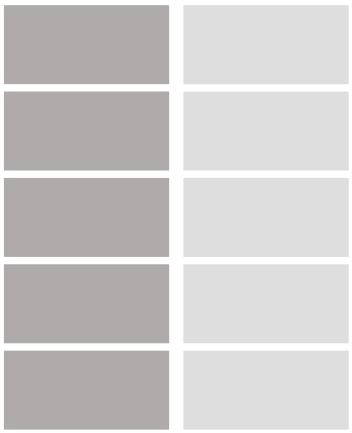
```
plotnine                  0.9.0              pyhd8ed1ab_0    conda-forge
```

11.7 2 - Foreword on Pandas

11.7.1 Pandas terminology



Series



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

11.8 3 - Reading data with Pandas

```
sample_table_url = "https://raw.githubusercontent.com/SPAAM-community/AncientMetagen
ancientmetagenome-hostassociated/samples/ancientmetagenome-hostassociated_samples.ts
library_table_url = "https://raw.githubusercontent.com/SPAAM-community/AncientMetage
ancientmetagenome-hostassociated/libraries/ancientmetagenome-hostassociated_librarie
```

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=
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=Fal
 date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None,
 compression: 'CompressionOptions' = 'infer', thousands=None, decimal: 'str' = '.',
```

```
lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None,
comment=None, encoding=None, encoding_errors: 'str | None' = 'strict', dialect=None,
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None, storage_options: 'StorageOptions' = Nor
   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>`_ .

        .. 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 ``'    '``) 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_accession     object
dtype: object
```

- `int64` is for integers
- `floating64` is for floating point precision numbers, also known as double in some other programing 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

Dom Square, Riga

56.949

24.104

Latvia

TZA3

Homo sapiens

400

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283100,ERS7283101

1059

Kazarina2021b

2021

10.1016/j.jasrep.2021.103213

St. Peter's Church, Riga

56.947

24.109

Latvia

TZA4

Homo sapiens

500

10.1016/j.jasrep.2021.103213

oral

tooth

ENA

PRJEB47251

ERS7283102,ERS7283103

Let's randomly inspect 5 rows

```
sample_df.sample(n=5)
```

project_name

publication_year

publication_doi

site_name

latitude

longitude

geo_loc_name

sample_name

sample_host

sample_age

sample_age_doi

community_type

material

archive

archive_project

archive_accession

413

Neukamm2020

2020

10.1186/s12915-020-00839-8

Abusir el-Meleq

29.240

31.100

Egypt

Abusir1576

Homo sapiens

2200

10.1038/ncomms15694

skeletal tissue

bone

ENA

PRJEB33848

ERS3635981

754

Rampelli2021

2021

10.1038/s42003-021-01689-y

El Salt

38.687

-0.508

Spain

V3

Homo sapiens neanderthalensis

44700

10.1038/s42003-021-01689-y

gut

sediment

ENA

PRJEB41665

ERS5428042

436

Neukamm2020
2020
10.1186/s12915-020-00839-8
Abusir el-Meleq
29.240
31.100
Egypt
Abusir1606
Homo sapiens
2600
10.1186/s12915-020-00839-8
skeletal tissue
bone
ENA
PRJEB33848
ERS3635928
474
Neukamm2020
2020
10.1186/s12915-020-00839-8
Abusir el-Meleq
29.240
31.100
Egypt
Abusir1654
Homo sapiens
2300
10.1038/ncomms15694
oral
tooth
ENA

PRJEB33848

ERS3635960

573

Philips2017

2017

10.1186/s12864-020-06810-9

Kowalewko

52.699

17.605

Poland

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

10.1038/nature21674

Stuttgart-Mühlhausen I

11

Weyrich2017

2017

10.1038/nature21674

Stuttgart-Mühlhausen I

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

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

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

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

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

project_name

publication_year

publication_doi

site_name

latitude

longitude

geo_loc_name

sample_name

sample_host

sample_age

sample_age_doi

community_type

material

archive

archive_project

archive_accession

1

Warinner2014

2014

10.1038/ng.2906

Dalheim

51.565

8.840

Germany

G12

Homo sapiens

900

10.1038/ng.2906

oral

dental calculus

SRA

PRJNA216965

SRS473747,SRS473746,SRS473748,SRS473749,SRS473750

2
Weyrich2017
2017
10.1038/nature21674
Gola Forest
7.657
-10.841
Sierra Leone
Chimp
Pan troglodytes
100
10.1038/nature21674
oral
dental calculus
SRA
PRJNA685265
SRS7890499
3
Weyrich2017
2017
10.1038/nature21674
El Sidrón Cave
43.386
-5.328
Spain
ElSidron1
Homo sapiens neanderthalensis
49000
10.1038/nature21674
oral
dental calculus

SRA
PRJNA685265
SRS7890498
4
Weyrich2017
2017
10.1038/nature21674
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 columm
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

oral

dental calculus

SRA

PRJNA685265

SRS7890498

4

2017

10.1038/nature21674

El Sidrón Cave

43.386

-5.329

Spain

ElSidron2

Homo sapiens neanderthalensis

49000

10.1038/nature21674

oral

dental calculus

SRA

PRJNA685265

SRS7890496

...

...

...

...

...

...

...

...

...

...

…

…

…

…

…

…

1055

2021

10.1016/j.jasrep.2021.103213

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

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
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False

False

1

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

2

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

3

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

4

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

1055

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

1056

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

1057

False

False

False

False
False
False
False
False
False
False
False
False
False
False
False
1058
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
1059

False

False

False

False

False

False

False

False

False

False

False

False

False

False

False

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

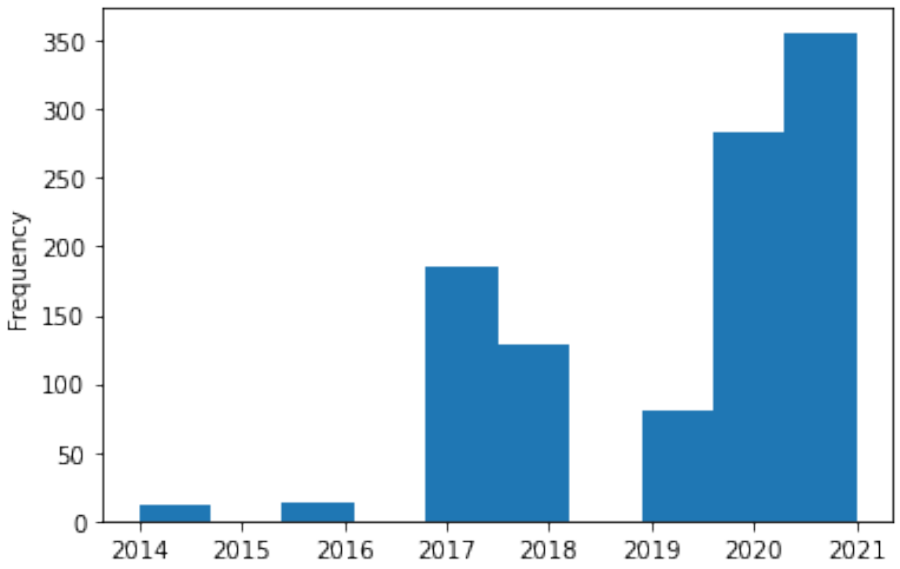


Figure 11.2: png

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_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

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 ellioti              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 or 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='heigh
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(co
```

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/algo
```

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_proj
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/…
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/...

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/...

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.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…

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

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'.
```

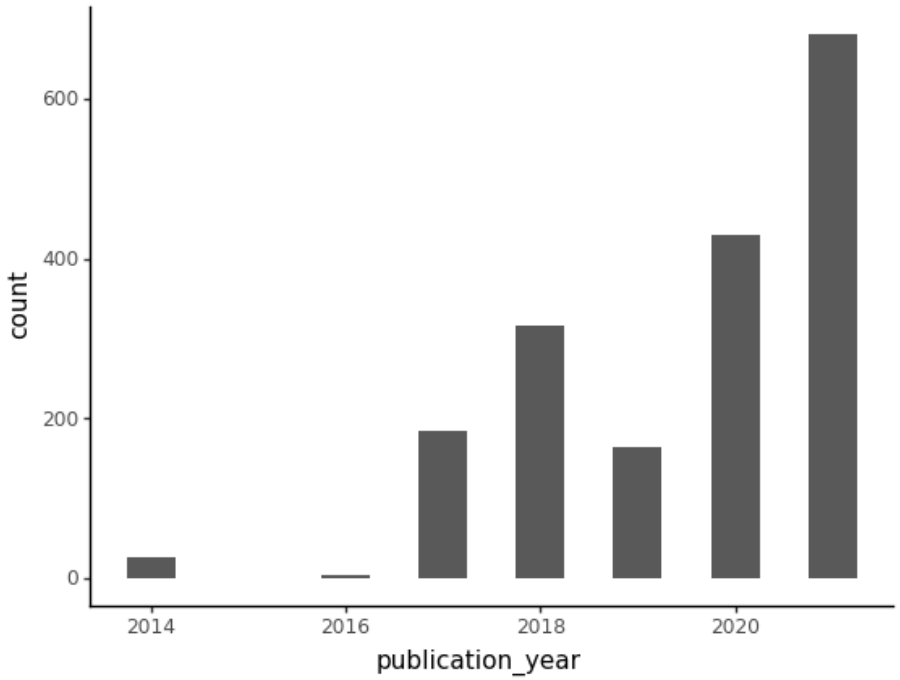


Figure 11.3: png

```
<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')
```

```
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()
```

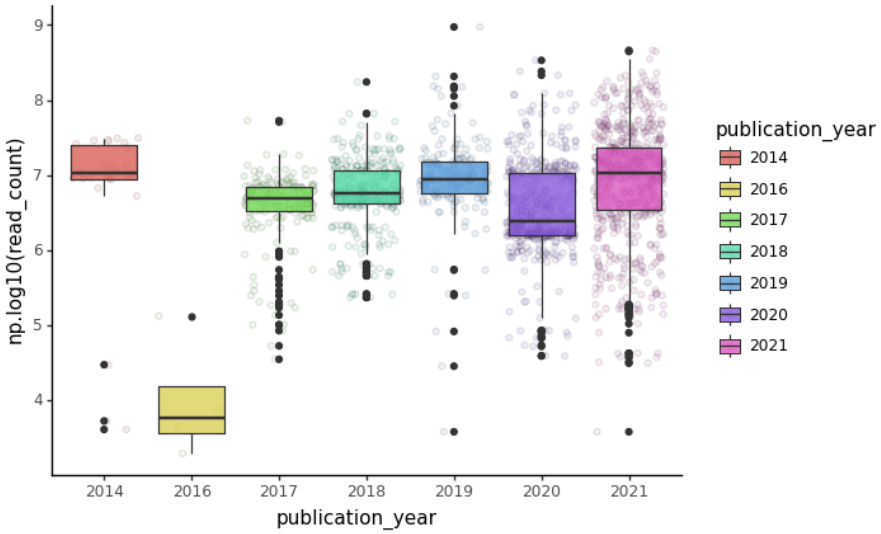


Figure 11.4: png

```
<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
```

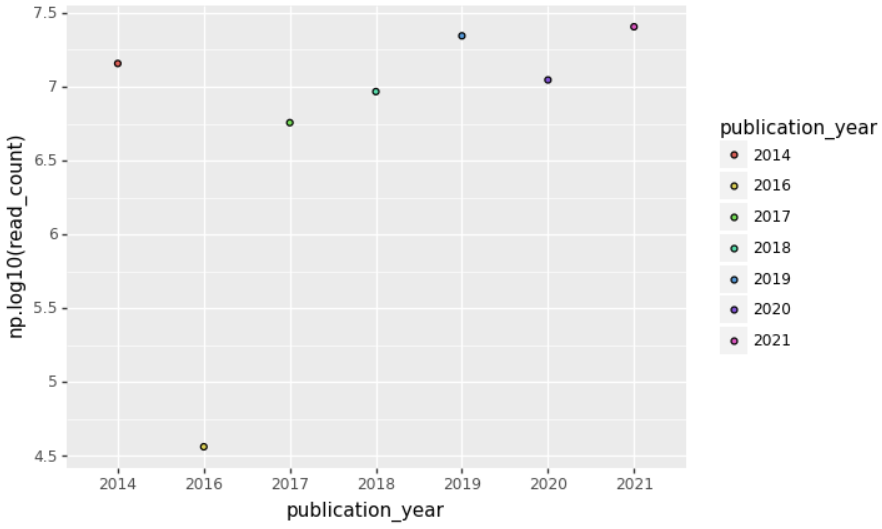


Figure 11.5: png

```
<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, colums 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('arch
```

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.

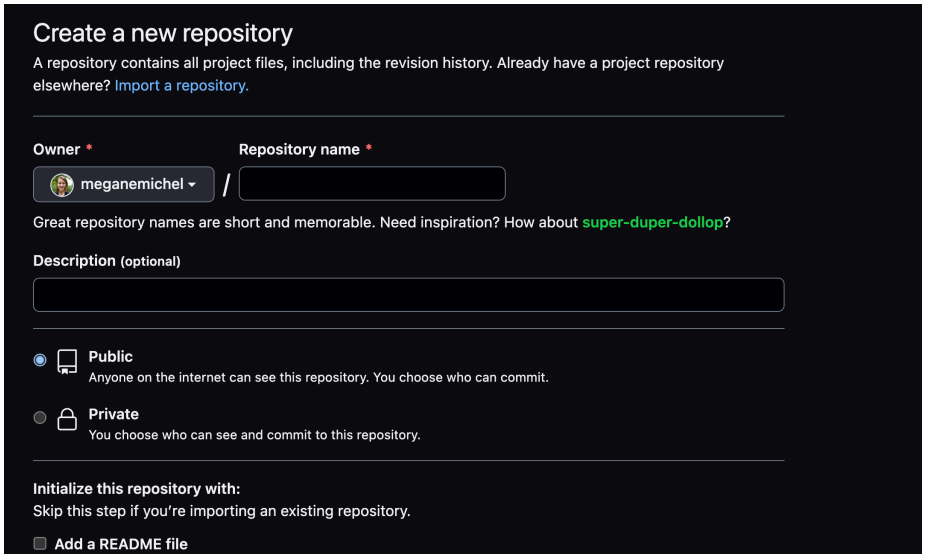
```
ssh -T git@github.com
```

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

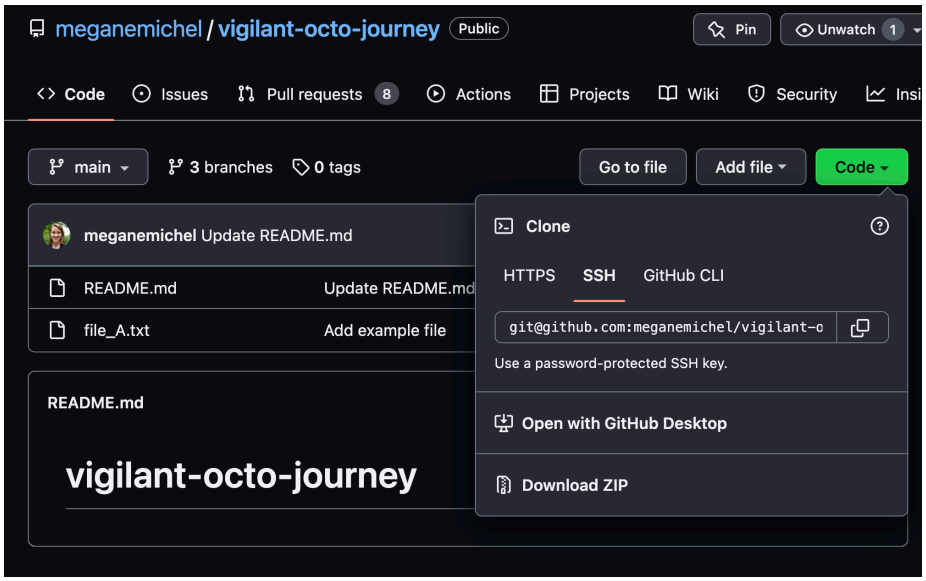
Now that you have set up your own SSH key, we can begin working on some version controlled data! Navigate to your github homepage and create a new

repository. You can choose any name for your new repo (including the default). Add a README file, then select Create Repository.



> **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"
```

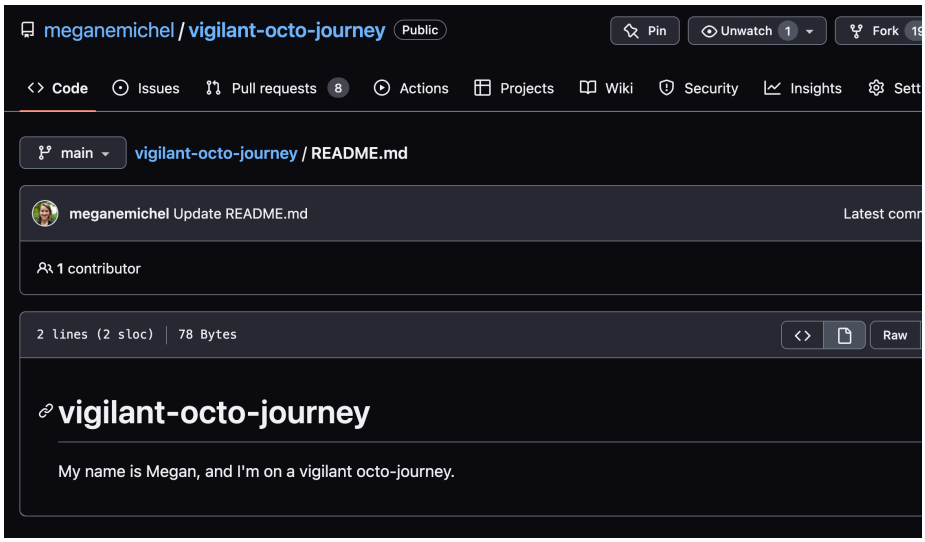
Finally, let's **push** our local commit back to our remote repository.

```
git push
```

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

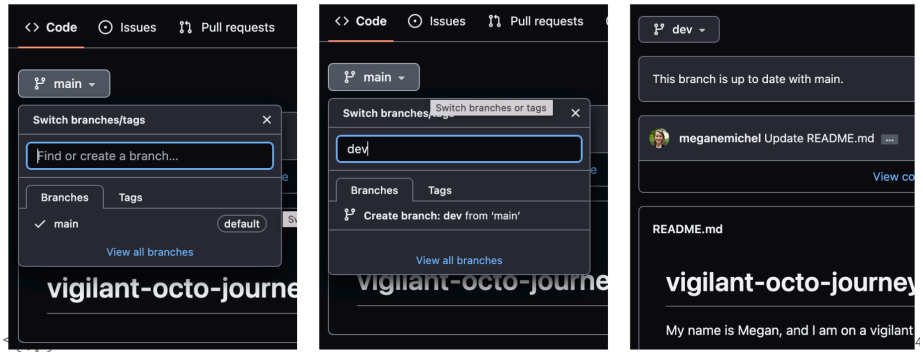
```
git pull
```

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 suggestinos 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 Metagenomic Resources

Chapter 13

Introduction to AncientMetagenomeDir

13.1 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 relevent published data they wish to use 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.

13.2 Lecture

PDF version of these slides can be downloaded from here.

13.3 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 publically available sequencing data on the INSDC association of databases, namely the EBI's European Nucleotide Archive (ENA), and NCBI or DDBJ's Sequence Read Archives (SRA). However, you may in some cases find ancient metagenomic data on institutional FTP servers, domain specific databases (e.g. OAGR), Zenodo, Figshare, or GitHub.

But while the data is publicly available, we need to ask whether it is 'FAIR'.

13.4 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 reserch in online search engines.

13.5 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	7.657	-10.841	Sierra Leone
Weyrich2017	2017	10.1038/nature21674	El Sidrón Cave	43.386	-5.328	Spain
Weyrich2017	2017	10.1038/nature21674	El Sidrón Cave	43.386	-5.329	Spain
Weyrich2017	2017	10.1038/nature21674	Spy Cave	50.48	4.674	Belgium

Criticially, 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 metagenoes, or species name of the genome that was reconstructed).

The AncientMetagenomeDir project already has 3 releases, and will continued to be regularly updated as the community continues to submit new metadata of samples of new publications as they come out.

13.6 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.

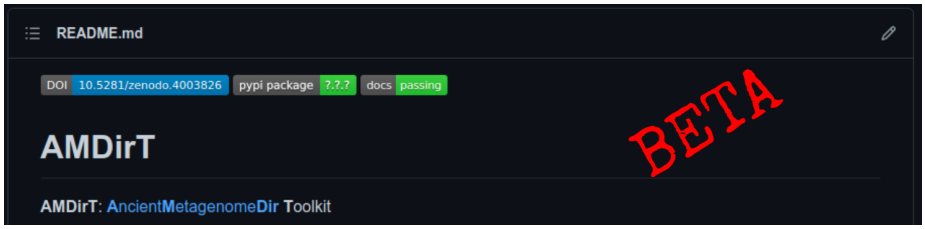
This metadata includes things such as whether a given set of data files contain sequencing data sequenced on which platform, whether the libraries have undergone damage treatment in the lab, or whether the uploaded data contains all or only mapped reads.

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 interegrated 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 infrastucture, we are in the process of building a new tool (something already in a usable state) called AMDirT. This allows a web browser-based GUI to filter and select data, and produce scripts for you to download all the selected data (without having to go to the databases themselves).

This is something we are going to try out now!

13.7 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

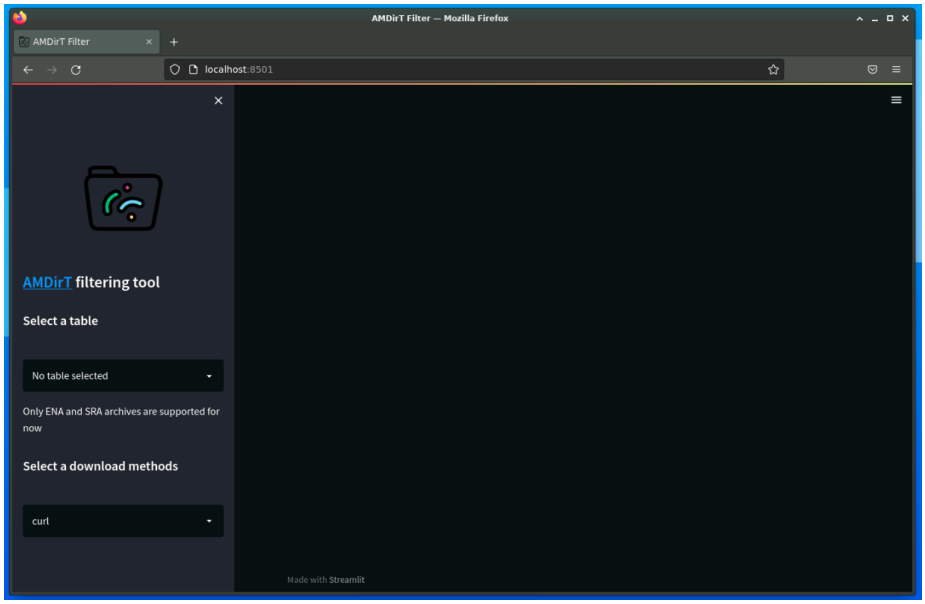
Open your terminal, and run the following two commands:

```
conda activate git-eager
pip install --upgrade --force-reinstall git+https://github.com/SPAAM-community/AMDirT.git@dev
```

Once that (hopefully) installs correctly, we can load the tool by running

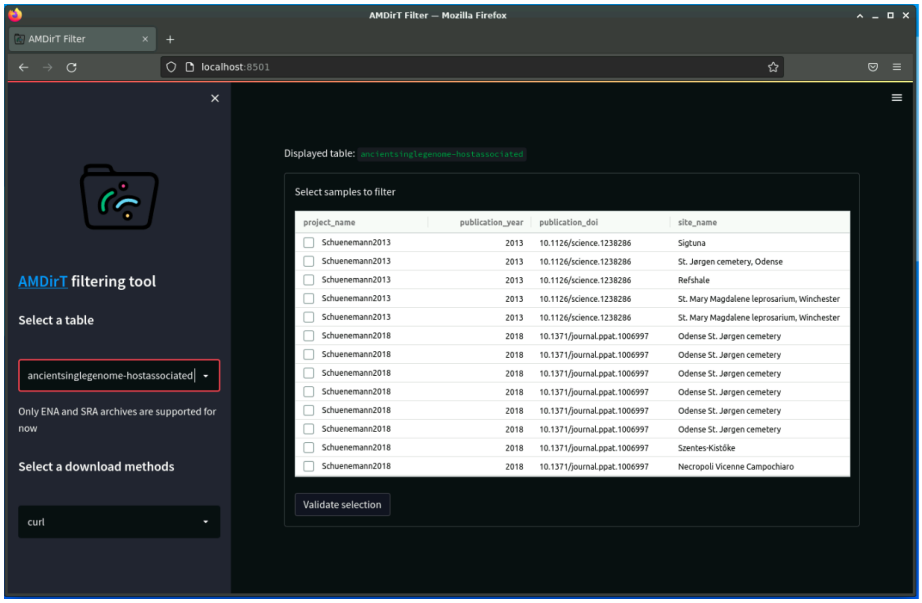
```
AMDirT filter
```

Your web browser should now load, and you should see a two panel page.



Under **Select a table** use the dropdown menu to select 'ancientsinglegenome-hostassociated'.

You should then see a table, pretty similar what you are familiar with with spreadsheet tools such as Microsoft Excel or LibreOffice calc.

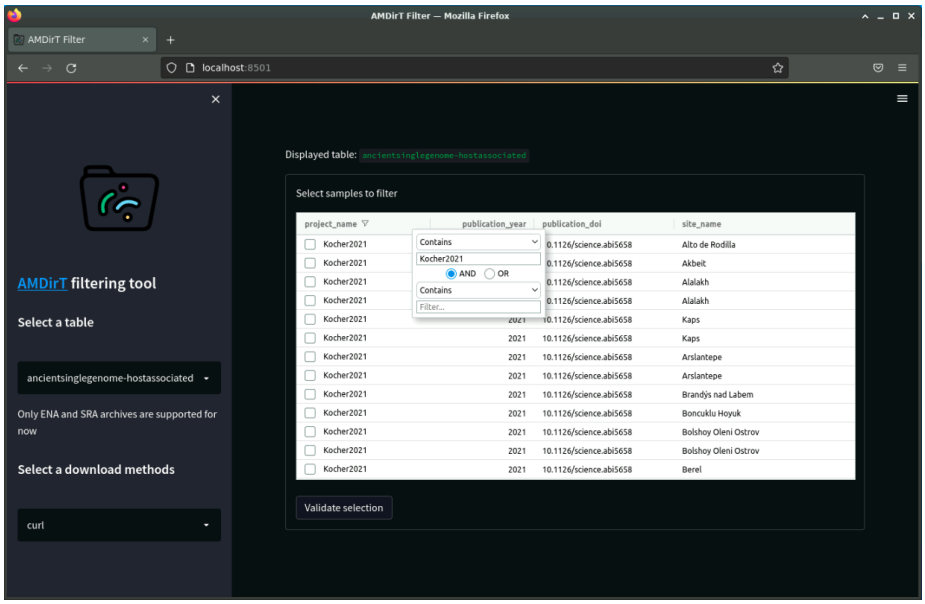


To navigate, you can scroll down to see more rows, and press shift and scroll to see more columns, or use click on a cell and use your arrow keys (, , ,) to move around the table.

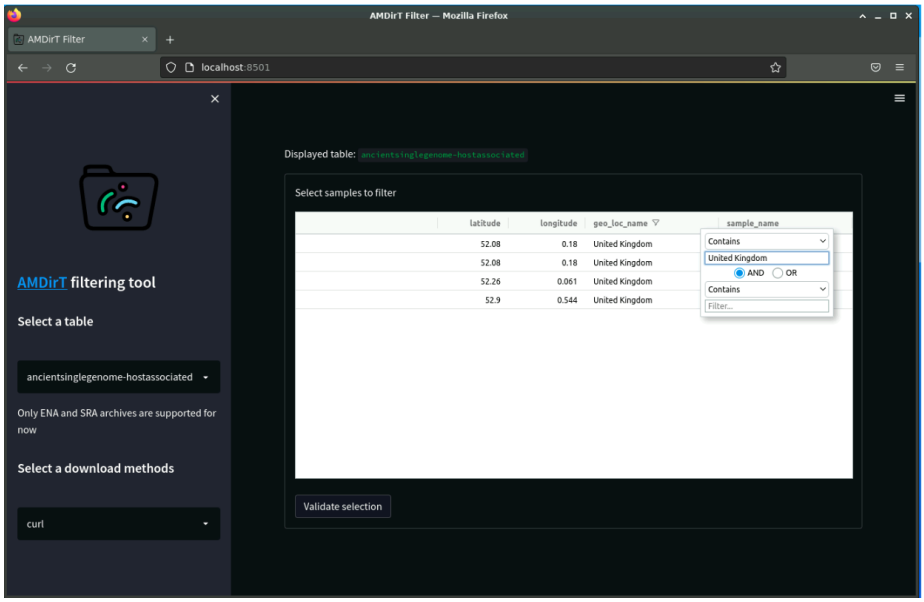
You can reorder columns by clicking on the column name, and also filter by pressing the little 'burger' icon that appears on the column header when you hover over a given column.

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'.

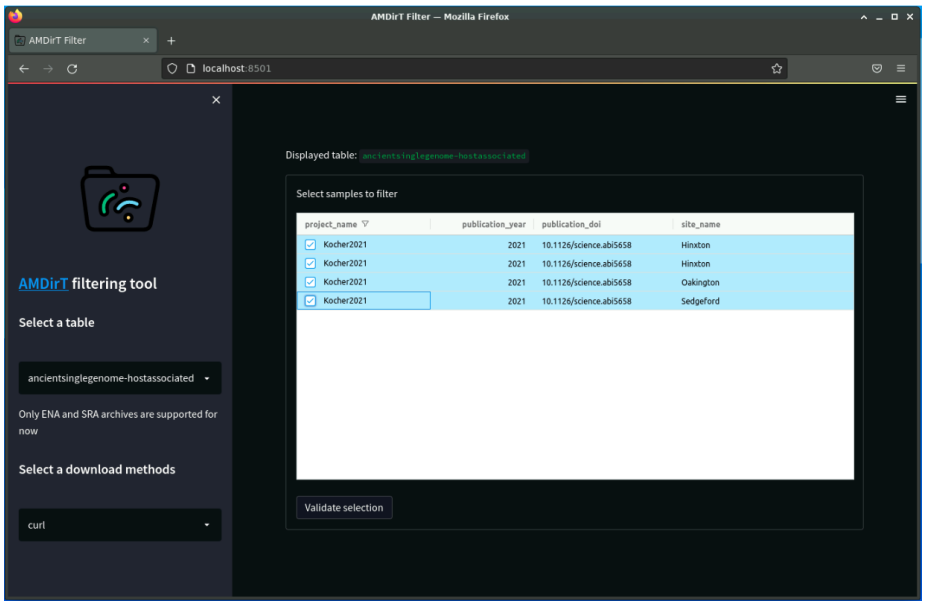


Then scroll to the right, and filter the **geo_loc_name** to 'United Kingdom'.

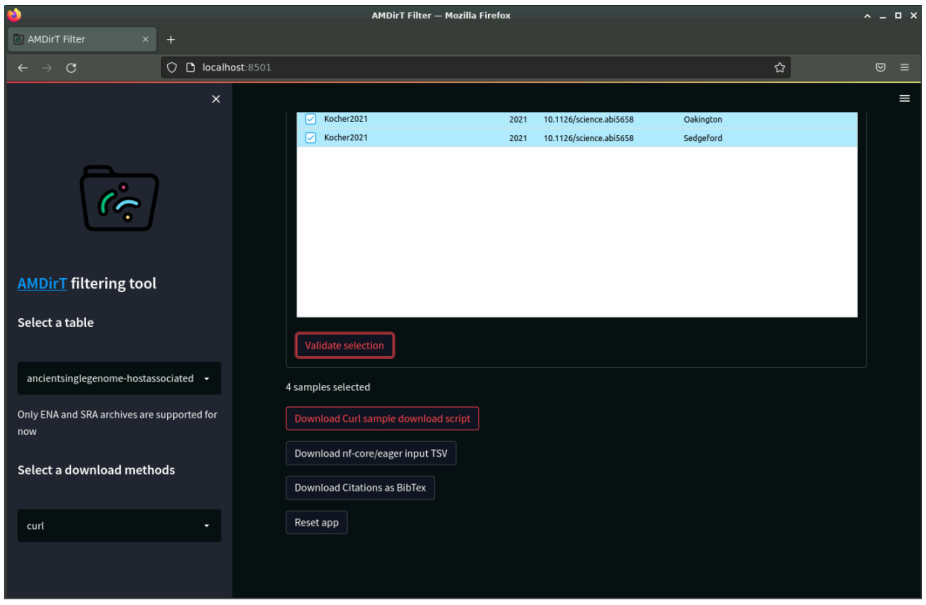


You should be left with 4 rows.

Finally, scroll back to the first column and tick the boxes of these four samples.



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!



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.

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

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_ERR605361
I0161   ERR6053619  0   4   SE  Homo sapiens    double  unknown ERX5692505_ERR605361
OAI017  ERR6053675  0   4   SE  Homo sapiens    double  half    ERX5692561_ERR605367
SED009  ERR6053686  0   4   SE  Homo sapiens    double  half    ERX5692572_ERR605368
```

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 {\AA}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
    Hübner and Abigail S. Gancz and Jessica Hider and Aurora F. Allshouse and Valent
    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.

13.9 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

13.10 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 14

Ancient Metagenomic Pipelines

14.1 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!

14.2 Lecture

PDF version of these slides can be downloaded from here.

14.3 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.

14.4 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 distrubition 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.

14.5 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 multpile 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!

14.6 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 from two ancient individuals, GLZ002 and KZL002. GLZ002 comes from the Neolithic Siberian site of Glazkovskoe predmestie adn 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 \ #Tells nextflow to execute the EAGER pipeline
-r 2.4.5 -ds1l \ #Specifies which pipeline and Nextflow versions to run for reproduc
-profile conda  \ #Profiles configure your analysis for specific computing environme
--fasta ../reference/GCF_001293415.1_ASM129341v1_genomic.fna \ #Specify reference in
--input ancientMetagenomeDir_eager_input.tsv \ #Specify input in tsv format or wildc
--run_bam_filtering --bam_unmapped_type fastq \ #Filter unmapped reads and save in f
--run_genotyping --genotyping_tool ug --gatk_ug_out_mode EMIT_ALL_SITES \ #Run genot
--run_bcftools_stats #Generate variant calling statistics
```

For full parameter documentation, click here.

And now we wait…

14.7 Top Tips for nf-core/eager success

1. Screen sessions

Depending on your input data, infrastructre, 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 -ds1l -profile conda --fasta ../reference/GCF_0012
--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-coreager 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.

14.8 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 princples 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 15

Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

Part IV

Appendicies

Chapter 16

Resources

16.1 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

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 17

Tools