

# Introduction to Ancient Metagenomics

# Table of contents

<b>Introduction</b>	<b>1</b>
<b>Citing this book</b>	<b>2</b>
<b>Authors</b>	<b>3</b>
<b>Acknowledgements</b>	<b>21</b>
Financial Support . . . . .	21
Institutional Support . . . . .	21
Infrastructural Support . . . . .	22
<b>Before you Start</b>	<b>23</b>
Creating a conda environment . . . . .	24
Additional Software . . . . .	24
<b>I. Theory</b>	<b>27</b>
Lectures . . . . .	28
Introduction to NGS Sequencing . . . . .	28
Introduction to Ancient DNA . . . . .	28
Introduction to Metagenomics . . . . .	29
Introduction to Microbial Genomics . . . . .	29
Introduction to Evolutionary Biology . . . . .	30
<b>1. Introduction to NGS Sequencing</b>	<b>31</b>
1.1. Lecture . . . . .	31
1.2. Introduction . . . . .	31
1.3. Basic structure of DNA . . . . .	31
1.4. DNA replication . . . . .	33
1.5. Extracting DNA . . . . .	34
1.5.1. Modern DNA . . . . .	34
1.5.2. Ancient DNA . . . . .	34
1.6. DNA sequencing . . . . .	34
1.6.1. Sanger sequencing . . . . .	35

*Table of contents*

1.6.2. Next generation sequencing . . . . .	39
1.7. Illumina sequencing . . . . .	39
1.7.1. Adapters . . . . .	40
1.7.2. Clustering . . . . .	40
1.7.3. Sequencing-by-synthesis . . . . .	41
1.7.4. Colour chemistry . . . . .	42
1.7.5. Paired-end sequencing . . . . .	43
1.7.6. Demultiplexing . . . . .	43
1.7.7. FASTQ File . . . . .	44
1.8. Sequencing recap . . . . .	45
1.9. Sequencing and considerations for ancient metagenomics . . . . .	45
1.9.1. Low DNA preservation . . . . .	46
1.9.2. Index hopping . . . . .	46
1.9.3. Sequencing errors . . . . .	47
1.9.4. Dirty Genomes . . . . .	47
1.9.5. Low Sequence Diversity . . . . .	48
1.9.6. Q and A . . . . .	49
1.10. Readings . . . . .	51
1.10.1. Reviews . . . . .	51
1.10.2. Sequencing Library Construction . . . . .	51
1.10.3. Errors and Considerations . . . . .	52
1.11. Questions to think about . . . . .	52
1.12. References . . . . .	52
<b>2. Introduction to Ancient DNA</b>	<b>53</b>
2.1. Lecture . . . . .	53
2.2. Questions to think about . . . . .	53
<b>3. Introduction to Metagenomics</b>	<b>54</b>
3.1. Introduction . . . . .	54
3.2. Lecture . . . . .	54
3.3. Questions to think about . . . . .	54
<b>4. Introduction to Microbial Genomics</b>	<b>55</b>
4.1. Lecture . . . . .	55
4.2. Introduction . . . . .	55
4.3. Larger Genomic Elements . . . . .	56
4.4. Sampling & Sequencing . . . . .	57
4.5. Validating the Presence of Organisms of Interest . . . . .	60
4.6. Genomes . . . . .	61
4.6.1. Recombination . . . . .	63

*Table of contents*

4.6.2. Pangenomes . . . . .	63
4.6.3. SNP Effects . . . . .	64
4.6.4. Virulence Associated Intervals . . . . .	65
4.7. Coinfections, Multi-strain Infections etc. . . . .	66
4.8. Resources . . . . .	66
4.9. Questions to think about . . . . .	67
<b>5. Introduction to Evolutionary Biology</b>	<b>68</b>
5.0.1. Lecture . . . . .	68
<b>II. Useful Skills</b>	<b>69</b>
Introduction to the Command Line (Bare Bones Bash) . . . . .	70
Introduction to R . . . . .	70
Introduction to Python . . . . .	71
Introduction to Git and GitHub . . . . .	71
<b>6. Introduction to the Command Line</b>	<b>72</b>
6.1. Lecture . . . . .	72
6.1.1. Session 1 . . . . .	72
6.1.2. Introduction . . . . .	75
6.1.3. The 5 commandments of Bare Bones Bash . . . . .	75
6.1.4. What is a terminal? . . . . .	76
6.1.5. Understanding the command prompt . . . . .	76
6.1.6. Absolute vs Relative paths . . . . .	78
6.1.7. Basic commands . . . . .	80
6.1.8. Datastreams, piping, and redirects . . . . .	81
6.1.9. Help text . . . . .	84
6.1.10. Variables . . . . .	85
6.1.11. Quotes matter! . . . . .	87
6.1.12. Find . . . . .	87
6.1.13. For loops . . . . .	89
6.1.14. How to Google like a pro . . . . .	90
6.1.15. (optional) Clean-up . . . . .	91
6.1.16. Conclusion . . . . .	93
<b>7. Introduction to R and the Tidyverse</b>	<b>94</b>
7.1. Lecture . . . . .	94
7.2. The working environment . . . . .	95
7.2.1. R, RStudio, the tidyverse and penguins . . . . .	95

## *Table of contents*

7.3.	Loading data into tibbles . . . . .	96
7.3.1.	Reading data with <code>readr</code> . . . . .	96
7.3.2.	How does the interface of <code>read_csv</code> work? . . . . .	97
7.3.3.	What does <code>readr</code> produce? The <code>tibble!</code> . . . . .	97
7.3.4.	How to look at a <code>tibble</code> ? . . . . .	98
7.4.	Plotting data in <code>tibbles</code> . . . . .	99
7.4.1.	<code>ggplot2</code> and the “grammar of graphics” . . . . .	99
7.4.2.	<code>scales</code> control the behaviour of visual elements . . . . .	100
7.4.3.	Colour <code>scales</code> . . . . .	103
7.4.4.	More variables! Defining plot matrices via <code>facets</code> . . . . .	104
7.4.5.	Setting purely aesthetic settings with <code>theme</code> . . . . .	107
7.4.6.	Ordering elements in a plot with <code>factors</code> . . . . .	108
7.4.7.	Exercise . . . . .	110
7.5.	Conditional queries on tibbles . . . . .	112
7.5.1.	Selecting columns and filtering rows with <code>select</code> and <code>filter</code> . . . . .	112
7.5.2.	Chaining functions together with the pipe <code>%&gt;%</code> . . . . .	114
7.5.3.	Summary statistics in <code>base R</code> . . . . .	115
7.5.4.	Group-wise summaries with <code>group_by</code> and <code>summarise</code> . . . . .	117
7.5.5.	Sorting and slicing tibbles with <code>arrange</code> and <code>slice</code> . . . . .	118
7.5.6.	Exercise . . . . .	120
7.6.	Transforming and manipulating tibbles . . . . .	121
7.6.1.	Renaming and reordering columns with <code>rename</code> and <code>relocate</code> . . . . .	121
7.6.2.	Conditional operations with <code>ifelse</code> , <code>case_when</code> and <code>case_match</code> . . . . .	124
7.6.3.	Switching between long and wide data with <code>pivot_longer</code> and <code>pivot_wider</code> . . . . .	126
7.6.4.	Exercise . . . . .	128
7.7.	Combining tibbles with join operations . . . . .	130
7.7.1.	Types of joins . . . . .	130
7.7.2.	Left join with <code>left_join</code> . . . . .	131
7.7.3.	Right join with <code>right_join</code> . . . . .	132
7.7.4.	Inner join with <code>inner_join</code> . . . . .	133
7.7.5.	Full join with <code>full_join</code> . . . . .	134
7.7.6.	Semi join with <code>semi_join</code> . . . . .	135
7.7.7.	Anti join with <code>anti_join</code> . . . . .	136
7.7.8.	Exercise . . . . .	137
7.8.	References . . . . .	138
<b>8.</b>	<b>Introduction to Python and Pandas</b>	<b>139</b>
8.1.	Lecture . . . . .	139
8.2.	Introduction to data manipulation in Python with Pandas and visualisation	140
8.3.	Table of content: . . . . .	140

## Table of contents

8.4.	Working in a jupyter environment . . . . .	140
8.5.	Pandas . . . . .	143
8.5.1.	Getting started . . . . .	143
8.5.2.	Pandas data structures . . . . .	143
8.6.	Reading data with Pandas . . . . .	145
8.7.	Data exploration . . . . .	146
8.7.1.	Columns . . . . .	147
8.7.2.	Inspecting the DataFrame . . . . .	148
8.7.3.	Accessing rows and columns . . . . .	149
8.7.4.	Conditional subsetting . . . . .	151
8.8.	Describing a DataFrame . . . . .	154
8.9.	Dealing with missing data . . . . .	156
8.9.1.	Grouping data . . . . .	158
8.10.	Combining data . . . . .	158
8.10.1.	Concatenation . . . . .	158
8.10.2.	Merging . . . . .	160
8.11.	Data visualization . . . . .	162
8.11.1.	Histogram . . . . .	162
8.11.2.	Bar plot . . . . .	164
8.11.3.	Scatter plot . . . . .	167
8.12.	Plotnine . . . . .	169
<b>9.</b>	<b>Introduction to Git(Hub)</b>	<b>171</b>
9.1.	Introduction . . . . .	171
9.2.	Lecture . . . . .	171
9.3.	SSH setup . . . . .	171
9.4.	The only 6 commands you really need to know . . . . .	173
9.5.	Working collaboratively . . . . .	175
9.6.	Pull requests . . . . .	176
9.7.	Questions to think about . . . . .	176
<b>III.</b>	<b>Ancient Metagenomics</b>	<b>177</b>
	Taxonomic Profiling . . . . .	178
	Functional Profiling . . . . .	178
	<i>De novo</i> Assembly . . . . .	178
<b>10.</b>	<b>Taxonomic Profiling, OTU Tables and Visualisation</b>	<b>180</b>
10.1.	Introduction . . . . .	180
10.2.	Chapter Overview . . . . .	185
10.2.1.	Download and Subsample . . . . .	185

## Table of contents

10.3. Data pre-processing . . . . .	188
10.4. Adapter sequence trimming and low-quality bases trimming . . . . .	189
10.5. Taxonomic profiling with Metaphlan . . . . .	194
10.6. Visualizing the taxonomic profile . . . . .	201
10.6.1. Visualizing metaphlan taxonomic profile with Pavian . . . . .	201
10.6.2. Visualizing metaphlan taxonomic profile with Krona . . . . .	202
10.7. Getting modern comparative reference data . . . . .	203
10.8. Loading the ancient sample taxonomic profile . . . . .	204
10.9. Bringing together ancient and modern samples . . . . .	219
10.9.1. Time to merge ! . . . . .	220
10.10. Comparing ancient and modern samples . . . . .	222
10.10.1. Taxonomic composition . . . . .	222
10.11. Let's make some plots . . . . .	225
10.12. Ecological diversity . . . . .	225
10.12.1. Alpha diversity . . . . .	225
10.12.2. Beta diversity . . . . .	230
10.13. References . . . . .	236
<b>11. Functional Profiling</b>	<b>238</b>
11.1. Lecture . . . . .	238
11.2. Preparation . . . . .	238
11.3. HUMAnN3 Pathways . . . . .	239
11.4. humann3 tables . . . . .	240
11.5. Sample Clustering with PCA . . . . .	247
11.5.1. Pathway abundance analyses . . . . .	247
11.5.2. Species contributions to pathways . . . . .	248
11.5.3. Final Visualisation . . . . .	254
<b>12. Introduction to <i>de novo</i> Genome Assembly</b>	<b>255</b>
12.0.1. Lecture . . . . .	255
12.1. Introduction . . . . .	255
12.2. Practical course . . . . .	258
12.2.1. Sample overview . . . . .	258
12.2.2. Preparing the sequencing data for <i>de novo</i> assembly . . . . .	259
12.2.3. <i>De novo</i> assembly . . . . .	263
12.2.4. Aligning the short-read data against the contigs . . . . .	267
12.2.5. Reconstructing metagenome-assembled genomes . . . . .	268
12.2.6. Taxonomic assignment of contigs . . . . .	275
12.2.7. Taxonomic assignment of MAGs . . . . .	278
12.2.8. Evaluating the amount of ancient DNA damage . . . . .	280
12.2.9. Annotating genomes for function . . . . .	282

*Table of contents*

12.2.10 Summary . . . . .	283
<b>13. Authentication and Decontamination</b>	<b>285</b>
<b>14. Introduction</b>	<b>286</b>
<b>15. Simulated ancient metagenomic data</b>	<b>289</b>
<b>16. Genomic hit confirmation</b>	<b>291</b>
16.1. Modern validation criteria . . . . .	291
16.1.1. Depth vs breadth and evenness of coverage . . . . .	291
16.1.2. Breadth of coverage via KrakenUniq . . . . .	292
16.1.3. Evenness of coverage via Samtools . . . . .	297
16.1.4. Alignment quality . . . . .	300
16.1.5. Affinity to reference . . . . .	303
16.2. Ancient-specific validation criteria . . . . .	304
16.2.1. Ancient status . . . . .	305
<b>17. Microbiome contamination correction</b>	<b>310</b>
17.1. Decontamination . . . . .	310
17.2. Microbial source tracking . . . . .	318
17.3. Summary . . . . .	326
17.4. Questions to think about . . . . .	327
17.5. Readings . . . . .	327
17.6. Resources . . . . .	327
<b>IV. Ancient Genomics</b>	<b>329</b>
Genome Mapping . . . . .	330
Phylogenomics . . . . .	330
<b>18. Genome Mapping</b>	<b>332</b>
18.1. Lecture . . . . .	332
18.2. Mapping to a Reference Genome . . . . .	332
18.2.1. Preparation . . . . .	333
18.2.2. Reference Genome . . . . .	333
18.2.3. Mapping Parameters . . . . .	334
18.2.4. Mapping Sample1 . . . . .	335
18.2.5. Genotyping . . . . .	336
18.2.6. Mapping and Genotyping for the other Samples/Parameters . . . . .	337
18.2.7. Comparing Genotypes . . . . .	339
18.2.8. Exploring the Results . . . . .	340

*Table of contents*

18.2.9. Examples . . . . .	341
18.2.10. Conclusions . . . . .	345
<b>19. Introduction to Phylogenomics</b>	<b>346</b>
19.1. Lecture . . . . .	346
19.2. Practical . . . . .	346
19.2.1. Preparation . . . . .	346
19.2.2. Visualize the sequence alignment . . . . .	347
19.2.3. Distance-based phylogeny: Neighbour Joining . . . . .	352
19.2.4. Probabilistic methods: Maximum Likelihood and Bayesian inference	358
<b>V. Ancient Metagenomic Resources</b>	<b>385</b>
Accessing Ancient Metagenome Data . . . . .	386
Ancient Metagenomic Pipelines . . . . .	386
<b>20. Accessing Ancient Metagenomic Data</b>	<b>388</b>
20.1. Lecture . . . . .	388
20.2. Introduction . . . . .	388
20.3. Finding Ancient Metagenomic Data . . . . .	389
20.4. AncientMetagenomeDir . . . . .	390
20.5. AMDirT . . . . .	391
20.5.1. Running AMDirT viewer . . . . .	391
20.5.2. Inspecting AMDirT viewer Output . . . . .	398
20.5.3. AMDirT convert . . . . .	400
20.6. Git Practise . . . . .	401
20.7. Summary . . . . .	403
20.8. Resources . . . . .	403
20.9. References . . . . .	403
<b>21. Ancient Metagenomic Pipelines</b>	<b>404</b>
21.1. Lecture . . . . .	404
21.2. Introduction . . . . .	404
21.3. Workflow managers . . . . .	405
21.4. What is nf-core/eager? . . . . .	406
21.4.1. Running nf-core/eager . . . . .	408
21.4.2. Top Tips for nf-core/eager success . . . . .	414
21.4.3. nf-core/eager output . . . . .	415
21.4.4. Clean up . . . . .	422
21.5. What is aMeta? . . . . .	422
21.5.1. Running aMeta . . . . .	423

## *Table of contents*

21.5.2. Setting up aMeta manually . . . . .	424
21.5.3. aMeta configuration . . . . .	426
21.5.4. Prepare and run aMeta . . . . .	427
21.5.5. aMeta output . . . . .	427
21.5.6. Clean up . . . . .	429
21.6. What is nf-core/mag? . . . . .	429
21.6.1. Running nf-core/mag . . . . .	432
21.6.2. Configuring Nextflow pipelines . . . . .	435
21.6.3. nf-core/mag output . . . . .	437
21.6.4. Clean up . . . . .	439
21.7. Questions to think about . . . . .	439
21.8. References . . . . .	439
<b>Summary</b>	<b>440</b>
<b>VI. Appendices</b>	<b>441</b>
<b>22. Resources</b>	<b>442</b>
22.1. Introduction to NGS Sequencing . . . . .	442
<b>23. Tools</b>	<b>443</b>
23.1. Introduction to R and the Tidyverse . . . . .	443
23.2. Introduction to Python and Pandas . . . . .	443
23.3. Introduction to Git(Hub) . . . . .	443
23.4. Functional Profiling . . . . .	443
23.5. <i>De novo</i> assembly . . . . .	444
23.6. Genome Mapping . . . . .	444
23.7. Phylogenomics . . . . .	444
23.8. Ancient Metagenomic Pipelines . . . . .	444

# 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.

## Warning

The PDF/ePUB version of the book is currently still under construction, and is likely misformatted and missing much of the content. It is not recommended for use.

*All material was originally developed for the SPAAM Summer School: Introduction to Ancient Metagenomics*

# Citing this book

The source material for this book is located on GitHub:

<https://github.com/SPAAAM-community/intro-to-ancient-metagenomics-book>.

If you wish to cite this book, please use the following bibliographic information

James A. Fellows Yates, Christina Warinner, Alina Hiß, Arthur Kocher, Clemens Schmid, Irina Velsko, Maxime Borry, Megan Michel, Nikolay Oskolkov, Sebastian Duchene, Thisseas Lamnidis, Aida Andrades Valtueña, Alexander Herbig, & Alexander Hübner. (2023). Introduction to Ancient Metagenomics. In Introduction to Ancient Metagenomics (Version 2022). Zenodo. DOI: 10.5281/zenodo.8027281

This work is licensed under a Creative Commons Attribution 4.0 International License.

# **Authors**

The creation of this text book was developed through a series of summer schools run by the SPAAM community, and financially supported by the Werner Siemens-Stiftung. The have contributed to the development of this textbook.

*Authors*

---

2022-2023



**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.

*Authors*

---



2022-2023

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

*Authors*

---

2022-2023



**Aida**

**Andrade**

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

*Authors*

---

2022-2023



**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.

## *Authors*

---

2022-2023



### **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.

*Authors*

---

2022-2023



**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.

## *Authors*

---

2022-2023



**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.

*Authors*

---

2022-2023



**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.

## *Authors*

---

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.

## *Authors*

---



2022-2023

### **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.

*Authors*

---

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.

*Authors*

---



2022-2023

**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.

*Authors*

---

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.

## *Authors*

---

2022-2023



### **Thisseas**

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

*Authors*

---

2023



**Kevin Nota**

has a PhD in molecular paleoecology from Uppsala University. Currently he is a postdoc in the Max Planck Research Group for Ancient Environmental Genomics. His main research interest is in population genomics from ancient environmental samples.

*Authors*

---



2023

Meriam Guellil is an expert in ancient microbial phylogenomics and metagenomics, particularly of human pathogens. She is particularly interested in the study of diseases that are invisible in the archaeological and osteological record, and the study of their evolution throughout human history. Her previous research includes studies on microbial species such as *Yersinia pestis*, *Haemophilus influenzae*, *Borrelia recurrentis* and *Herpes simplex 1*.

## *Authors*

---

2023



Robin Warner is a MSc bioinformatics student at the Leipzig University. He is currently writing his master's thesis in the Max Planck Research Group for Ancient Environmental Genomics about the comparison of ancient sedimentary DNA capture methods and shotgun sequencing.

---

# Acknowledgements

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

## Financial Support



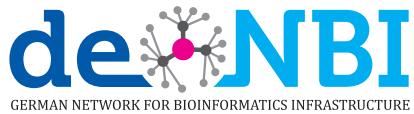
The content of this textbook was developed from the SPAAM Summer School: Introduction to Ancient Metagenomics summer school series, sponsored by the Werner Siemens-Stiftung (Grant: Paleobiotechnology, awarded to Pierre Stallforth, Hans-Knöll Institute, and Christina Warinner, Max Planck Institute for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

## Institutional Support



*Infrastructural Support*

## **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

# Before you Start

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

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

## Warning

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

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

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

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

## *Creating a conda environment*

Before loading the environment for the exercises, the environment will need to be installed using the `yml` with the instructions below, and then activated. A list of the software in each chapter's environment can be found in the Appendix.

### **Creating a conda environment**

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

1. Download the `conda` env on
2. Within the resulting directory run the following `conda` command to install the software into it's dedicated environment

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

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

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

```
conda env list
```

4. To load the relevant environment, you can run

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

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

```
conda deactivate
```

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



Tip

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

### **Additional Software**

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

## Additional Software

- Docker (installation method will vary depending on your OS)
  - Standard install
  - Linux-nerd install

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /e
sudo chmod a+r /etc/apt/keyrings/docker.gpg
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
$(./etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin do
## May need to do a reboot or something here
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
sudo reboot ## will kick you out, but it'll be back in a minute or two
```

- rename (if not already installed, e.g. on OSX)

```
sudo apt install rename
```

- *De novo* assembly

- MetaWrap

```
conda create -n metawrap-env python=2.7
conda activate metawrap-env
conda install biopython bwa maxbin2=2.2.7 metabat2 samtools=1.9
cd ~/bin/
git clone https://github.com/bxlab/metaWRAP.git
echo "export PATH=$PATH:~/bin/metawrap/bin" >> ~/.bashrc
```

- Functional Profiling

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

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

- Phylogenomics

## *Additional Software*

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

```
export tempest='bash /home/ubuntu/bin/TemEst_v1.5.3/bin/tempest'
```

- MEGAX (v11.0.11)

- Authentication and Decontamination

- Sourcetracker

```
cd /<path>/<to>/authentication-decontamination/sourcetracker/
git clone https://github.com/danknights/sourcetracker.git

## patch due to R update
sed -i '538,539s/^/#/' sourcetracker/src/SourceTracker.r
```

- decOM

```
cd /<path>/<to>/authentication-decontamination/decom
git clone https://github.com/CamilaDuitama/decOM.git
cd decOM
conda env create -n decOM --file environment.yml
conda deactivate
conda activate decOM

export PATH=/absolute/path/to/decOM:${PATH}
```

- Ancient Metagenomic Pipelines

```
```bash
cd /<path>/<to>/ancient-metagenomic-pipelines/
git clone https://github.com/NBISweden/aMeta
cd aMeta
conda env create -f workflow/envs/environment.yaml
conda activate aMeta
```

```

# **Part I.**

# **Theory**

## Lectures

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

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

## Lectures

### Introduction to NGS Sequencing

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

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

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

### Introduction to Ancient DNA

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

We next cover genome basics and take an in-depth look at the way DNA degrades over time. We detail the fundamentals of DNA damage, including the specific chemical processes that lead to DNA fragmentation and C->T miscoding lesions. We then demystify the DNA damage “smiley plot” and explain the how the plot’s symmetry or asymmetry is related to the specific enzymes used to repair DNA during library construction. We discuss how DNA damage is and is not clock-like, how to interpret and troubleshoot DNA damage plots, and how DNA damage patterns can be used to authenticate ancient samples, specific taxa, and

## *Lectures*

even sequences. We cover laboratory strategies for removing or reducing damage for greater accuracy for genotype calling, and we discuss the pros and cons of single-stranded library protocols. We then take a closer look at proofreading and non-proofreading polymerases and note key steps in NGS library preparation during which enzyme selection is critical in ancient DNA studies.

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

### **Introduction to Metagenomics**

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

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

### **Introduction to Microbial Genomics**

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

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

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

## **Introduction to Evolutionary Biology**

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

The first section will introduce phylogenetic trees and a set of core terms and concepts for their interpretation. Next, it will focus on some of the most popular approaches to inferring phylogenetic trees; those based on genetic distance, maximum likelihood, and Bayesian inference. These methods carry important considerations regarding the process that generated the data, computational capability, and data quality, all of which will be discussed here. Finally, we will direct our attention to examples of analyses of ancient and modern pathogens (e.g. *Yersinia pestis*, Hepatitis B virus, SARS-CoV-2) and critically assess appropriate choice of models and methods.

# 1. Introduction to NGS Sequencing

## 1.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

! Important

Text is a raw transcription of the lecture above - extensive editing is still underway.  
Readability will be low.

## 1.2. Introduction

We will make a start. So in this session, introduction to NGS data, I want to very briefly recap the basics of DNA for a good reason. Then introduce what DNA sequence is and how that works, and then also explain how Illumina NGS data, sequencing data is generated. The reason why I'm focusing on Illumina is because this is what the vast, very vast majority of ancient DNA will be sequenced on for reasons which I will explain later.

## 1.3. Basic structure of DNA

So first, to actually understand how sequencing works, we need to look at, just do a very, very briefly recap of what DNA is. So DNA, as you probably, everybody knows, is a double helix molecule which is stored in every single cell in your body.

And this double helix is actually made up of four main components called nucleotides, which you can see here (Figure 1.1), and they consist of two strands in which these four nucleotides will come together and bind together in a particular order. And these four nucleotides are made up of two groups, pyrimidines and purines. So pyrimidines are cytosines and thiamines, and purines are guanines and adenines.

## 1. Introduction to NGS Sequencing

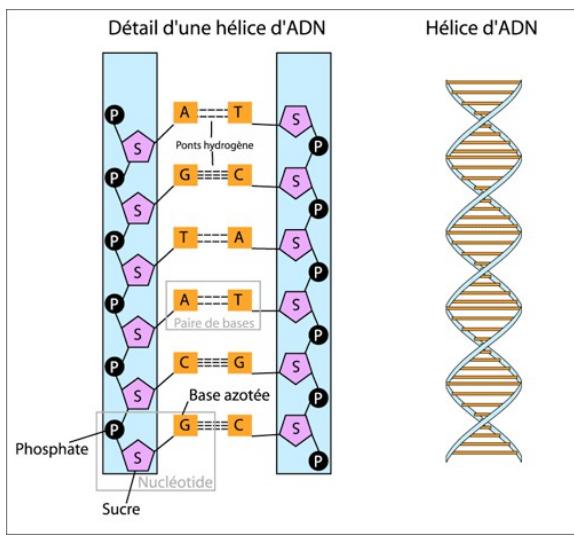


Figure 1.1.: 2D molecular diagram of DNA helix, with sugar-phosphate backbone and amine groups labelled indicated. Source: Pradana Aumars, CC BY-SA 4.0, via Wikimedia Commons

## 1. Introduction to NGS Sequencing

And this base pairing, which brings the two strands of the DNA together, always consists of one pyrimidine and one pyrimidine. I always remember which group go together, and so I remember it as C always goes with G, think CGI, and then go A with T. What's some of the best CGI you've ever seen? It's from Star Wars, so think AT-AT Walker (Figure 1.2), Antid Walker, you have been able to remember that. So what this means, and because they go together, it always means they're complementary. So whenever you find a C on one strand of the DNA molecule, you will see a G on the other and vice versa, and again, A with a T on the other. So depending on which strand you are reading, you can always get the order of the base on the other strand because of this complementary base pairing.

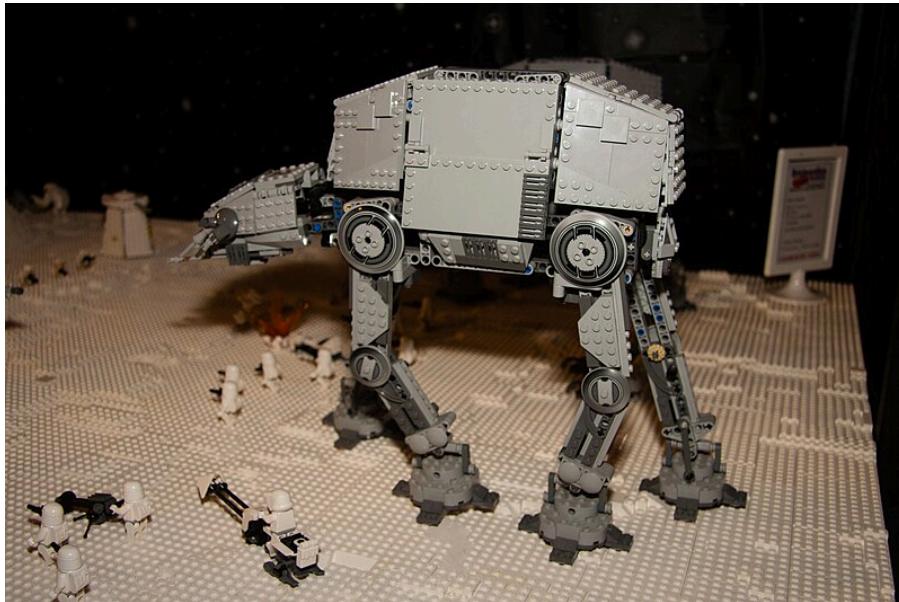


Figure 1.2.: Lego construction of AT AT walker from Star Wars. Source: Tim Moreillon, CC BY-SA 2.0 Generic, via Wikimedia Commons

## 1.4. DNA replication

And this is important because this is essentially how replication occurs, so when you basically make a copy of the DNA strand. And in very simple terms, what we do is, or what the body does, is unwinds the multiple various macro structures, including double helix. You then separate the strands into two, so you have, all of your nucleotides are basically exposed, so you don't have them binding together. And then you get an enzyme called DNA polymerase, which basically attaches onto the strands that's reading along it, and when it finds an exposed nucleotide, it will say, well, okay, I recognize, for example,

## *1. Introduction to NGS Sequencing*

this is a C that's being exposed, and it waits to basically pick up a free nucleotide floating around in the cellular gunk, and then basically allows it to bind together in that position. Then it basically will move along to the next exposed nucleotide, see what it is, let's say it's an A, and then it will basically wait to find the T and fix it on the strand, and eventually another enzyme, I believe, comes along, I can't remember the name, I didn't name this, comes along and basically repairs the backbone, like, no, it doesn't matter, prepares the backbone to basically then have your two, strands from your original strand. And just remember basically having this enzyme picking up free nucleotides and adding it to the new strand, because this is the important thing the sequencing is within a minute.

## **1.5. Extracting DNA**

### **1.5.1. Modern DNA**

As a reminder, how we get DNA, so when you do this, you basically get your sample, you then have to break down the cells, the cell walls, and membranes, and then you basically have to destroy a lot of stuff inside the cell, there's not the DNAs, things like RNAs, proteins, and things like this, because this can basically inhibit your DNA replication downstream in your molecular steps. You then separate out the DNA from the rest of the broken stuff, which you can then pull out (Figure 1.3). And normally in modern DNA, this can actually look like a long, this sort of spaghetti-like thing.

### **1.5.2. Ancient DNA**

However, ancient DNA is a bit different. The process is the same, you basically have to break down the tissue, in this case it's, let's say, bone, so you have to demineralize it to release all of the biomolecules, and you have to degrade all the other stuff, but the DNA molecules are also degraded, so they're already fragmented, so they're very short, they're very damaged, so they have modified nucleotides, and they also have contamination. So basically, your small fragments DNA is sitting in a super-modern DNA, so a lot of these things will be covered in more detail in other later sessions, but you have to remember that they're damaged, they're old, they're very, very short.

## **1.6. DNA sequencing**

And this brings us on to DNA sequencing, which is essentially the conversion of the chemical, that's better, chemical nucleotides of a DNA molecule to the human-readable ACTG

## 1. Introduction to NGS Sequencing

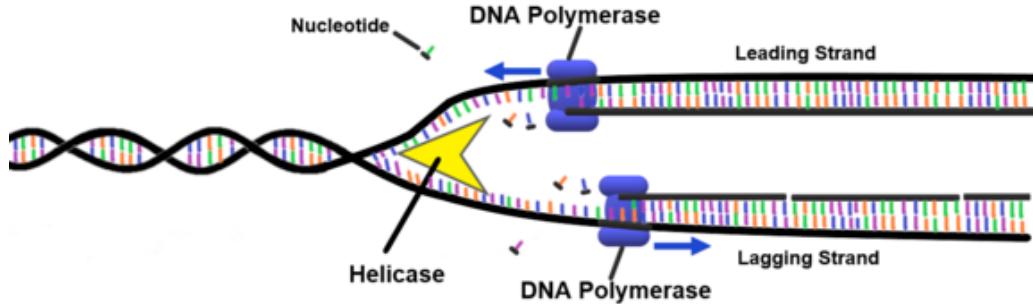


Figure 1.3.: Schematic diagram of DNA replication, with a helicase unwinding DNA strand with DNA polymerases on the leading and lagging strand incorporating free nucleotides. Source: Christine Imiller, CC BY-SA 4.0 Generic, via Wikimedia Commons

on your computer screen, and what we basically do all of our analyses on in genetics and genomics. And the way this works is pretty common across most methods, which is you replicate a strand, as I described a minute ago, but instead of adding just a standard nucleotide, you add a fluorophore-modified nucleotide, so a fluorophore is a little molecule which essentially, when you excite it somehow, will emit a color.

And in the case of DNA, you can have four different nucleotides, and each one will emit a different color Figure 1.4. And so quite often, the way you excite the fluorophore is firing a laser, which then emits the light, and you record the color. And so when you're basically adding your nucleotide each time, you fire a laser, take a picture of the color being emitted, and then you know which base it is, and then you repeat on the next base, and next base, and next base.

### 1.6.1. Sanger sequencing

So historically, the first, let's say, mass production sequencing method was called Sanger sequencing (Figure 1.5). So what this involved was taking a DNA molecule, making lots and lots of copies of it, but also fragmenting it in a random manner, so all of the, oh sorry, fragmenting it, no. I'm getting confused, one step ahead. Sorry, taking a DNA

1. Introduction to NGS Sequencing

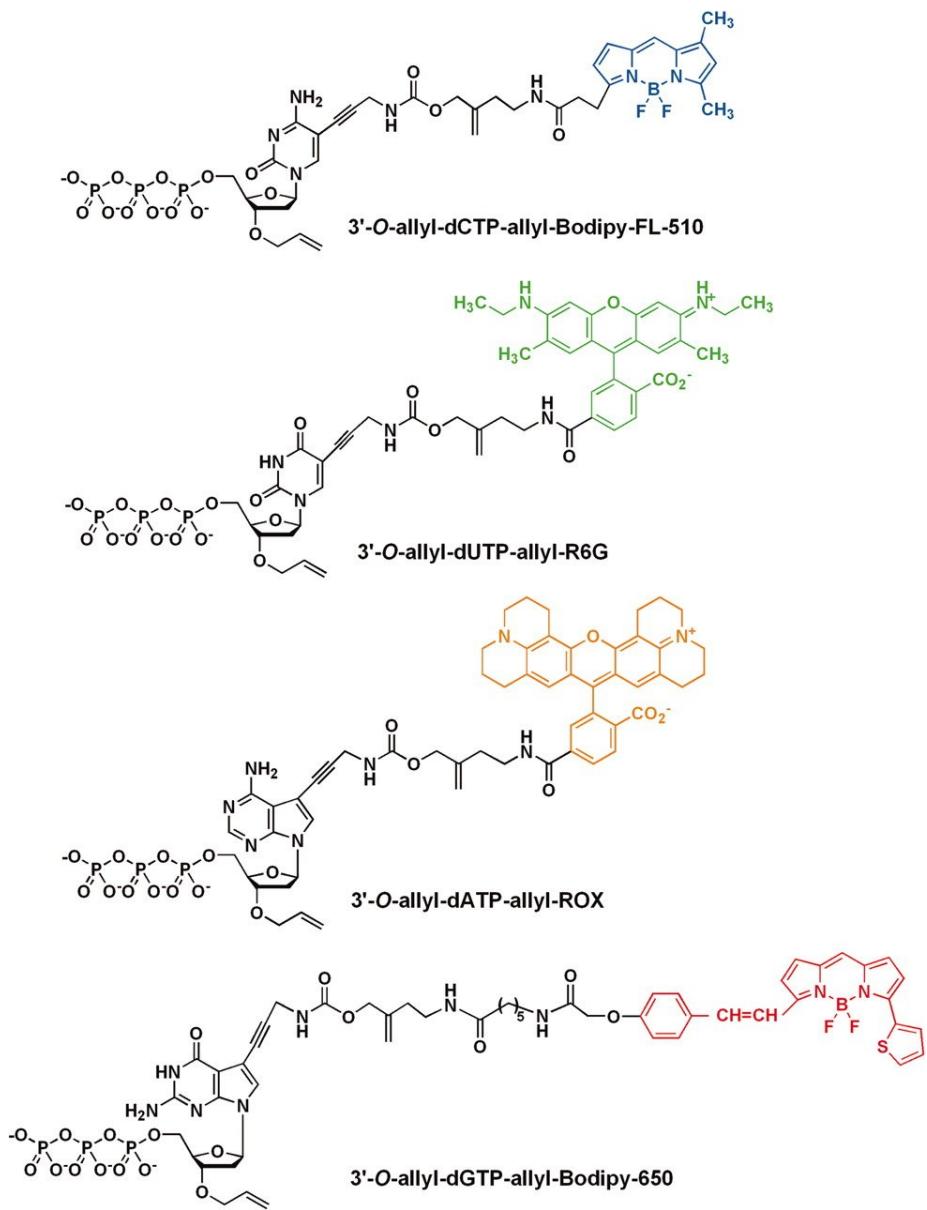


Figure 1.4.: Molecular diagram of the four DNA nucleotides with coloured fluorophore amine groups. Source: [@Ju2006-cl]

## 1. Introduction to NGS Sequencing

molecule, making lots of copies of it. Then you start extending the molecule, but instead of adding just no standard nucleotides, you mix in a few special modified nucleotides, which include essentially a blocker. And what this means is that once the polymerase gets to this particular blocking nucleotide, it will not extend it any further. However, as you added a mixture of standard nucleotides and also these blocker nucleotides, your DNA molecules will be extended to somewhat a random length each time, because you have many, many different copies.

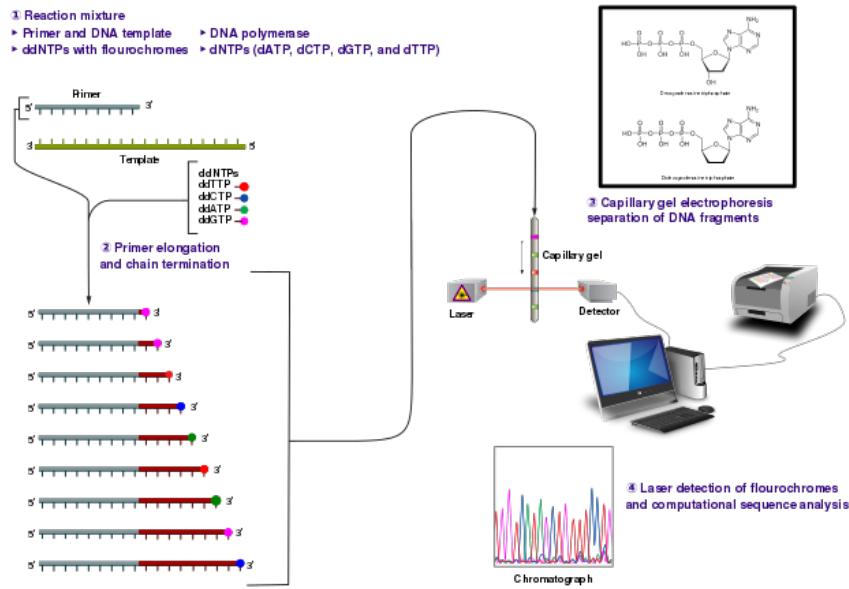


Figure 1.5.: Diagram of sanger sequencing, with a primer, template and free blocking nucleotides being added to ends of molecules at random lengths, and then being sent through a size-selecting capillary with a laser detector to identify which of the four blocking nucleotides are passing through the capillary and thus ready in a chromatograph. Source: Estevezj, CC-BY-SA 3.0 Unported via Wikimedia Commons

In the end, you will essentially have the entirety of your original molecule covered, as you can sort of see in this sort of step-like manner here. And what would happen once you basically have your randomly extended molecules, but with these blockers, you would then send it through a capillary gel, which basically separates out the DNA molecules based on its length, and then you would fire a laser. And the important thing about these blocking nucleotides is also they were essentially fluorophores, so when you fire the laser, it'll emit a light. And as you basically have your DNA molecules going through the capillary gel, so the shorter ones going faster through the gel because of resistance, and the longer ones going slower, you can basically record the order of the molecules going through the capillary gel.

## 1. Introduction to NGS Sequencing

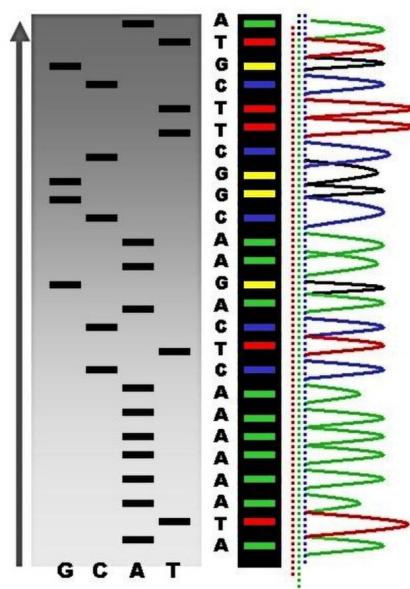


Figure 1.6.: Example of gel-based chromatogram with four columns with bands indicating which colour is the blocking nucleotide of that particular sized DNA molecule, and on the size an intensity graph of each colour given off by each fluorophore to indicate which colour it is. Source: Morse Phoque (Abizar), CC-BY-SA 3.0 Unported via Wikimedia Commons

## *1. Introduction to NGS Sequencing*

And then according to the light, you can basically see the different colors, and then with that, you can basically count which base is in your DNA molecule, as it goes through the capillary gel, and basically reconstruct the sequence by this method (Figure 1.6). However, this is, the approach was actually not so good for high-throughput DNA, so trying to reconstruct whole genomes, a lot of the original human genome was generated with this, but this too years, was extremely expensive, and it's also very, very resource-hungry, you have to use a lot of preliminary, you have to use a lot of DNA, which again, when we deal with ancient DNA, which you get very small amounts, because it's very degraded, it doesn't really work.

### **1.6.2. Next generation sequencing**

And then in about 2005, next-generation sequencing, which is a bit of a misnomer, to be honest, came along, where you can see sequence billions and billions of DNA molecules at once. It was very fast and cheap, and very much revolutionized genetics, and pushed us into a new era of genomics. And the market leader was, and still is, is a company called Illumina. There are others called PacBio and IonTorrent, but really Illumina is the one that pretty much everybody uses nowadays for at least short read sequencing. And as I'm sure you're all aware, this is sort of, these machines are more second generation now, we have new machines like Oxford Nanopore, which basically do very long reads, PacBio to a certain extent, and that's sort of more a third generation that we're entering right now.

Oh, poop. Oh, dear, so unfortunately, my pretty picture has been, a video has been deleted, but what I want you to imagine is a big black window with lots of colored points. And when I press play, on all of these points, they're going to start changing colors, going from red to green to yellow to blue, and this happening thousands of times at once across this big black screen. And this is essentially the process of sequencing.

## **1.7. Illumina sequencing**

So what this black window that you should have been seeing would have been is something called a flow cell. This is a glass slide, and on this glass slide, it's embedded sort of bound to the base of this, is lots of short DNA sequences, synthetic DNA sequences called oligos. And what you do is essentially take your DNA, and you basically inject it into the flow cell, and your DNA will spread across the flow cell, and start binding to this lawn. So imagine, literally imagine like a grass lawn on the base of this glass slide, and basically all of your DNA molecules are attaching to these random synthetic nucleotides, oligos.

## *1. Introduction to NGS Sequencing*

### **1.7.1. Adapters**

But the question is, how is your DNA actually binding onto this lawn, and not basically getting washed away as your solution flows through the flow cell? What you do before injecting into the flow cell is convert your DNA samples into something called a library. A library is an adapters, and see these are basically the complement sequence of the oligos, of these synthetic oligos, which allow you to bind to the lawn flow cell. But in addition to the adapters, you add onto DNA sequence, things called priming sites, so this is where the enzymes actually bind on to start copying the DNA.

And also, when you're sequencing multiple samples at once on the same sequencing run, you can add things called indices, or known as barcodes, which are basically sample specific. So it allows you to mix multiple samples at once into one sequencing run, sequencing all at the same time, later on separate them out. So this is ultimately a slightly simplified version of a Illuminate DNA construct. So the X, X, X, X, X in the middle, this is your target. So this is actually your DNA sequence from your sample.

And then at both ends, you essentially have a target primer. So this is where your polymerase will bind onto to start, then actually going, reading into your target, or your insert is another phrase that you can call it. Then prior to the target primer, you also have an index. So this is actually your sample specific barcode. This is actually added typically onto the adapter, and so this happens in the, no, sorry. And the library construction, sorry. And then also you have the adapter and index primer right at the beginning. So this is both what binds onto your flow cell, but also acts as a primer for actually reading the index because you also need a sequence index to know which DNA molecule, to which sample DNA molecule it's coming from.

### **1.7.2. Clustering**

So once you've done this, in Illuminate sequencing, you have the problem that the fluorophores will be adding to your DNA molecule. The eliterated is not enough for a camera to pick up. It's much, much, much too small. You're dealing with these very, very small biomolecules. And so what you have to do is once your DNA molecule is bound to your flow cell, sort of some randomly, you start to make lots and lots and lots of copies. So when you have lots of copies, that means that all the copies will make the same nucleotide, sorry, the same color at once, and make the emissions strong enough that a camera in the sequencer can actually take a picture and record a lot base it is.

And the procedure making these copies is called clustering. So what this consists of is you have your DNA molecule, which is bound to one of the synthetic oligos on the lawn. And you basically do some of the bridge amplification where you basically bend over the DNA

## *1. Introduction to NGS Sequencing*

molecule. Sorry, sorry, I forgot. Your DNA molecule is single stranded at the moment. So when you bind this over, bend this over so that the reverse complement will bind to the other oligo, a primer can bind on and start reading across the molecule to reconstruct your double stranded DNA. So you basically get the complement or reverse strand of your DNA molecule to double stranded DNA. Then you had a primer to basically cut at one end or each end of the DNA molecule on the two different sort of forward and reverse adapters to result in two single stranded again DNA molecules. So basically the reverse complement of the DNA molecule, but you have one, sorry, two strands. Then you do the same thing again, bend it over, reconstruct the full double stranded molecule again, separate it out and do this many, many, many, many, many different times. And basically you have loads and loads and loads of copies of the same molecule in the same tiny little cluster, which is why you could go clustering in the same point in the flow cell.

Then prior to sequencing, you actually will cut all of one of these types off. So let's say you'd only be left with the purple ones to make sure you only have the same sequence. So not the reverse complements, but just the same sequence in one spot on your flow cell. Sorry, one second. Basically this onto the actual sequencing process.

### **1.7.3. Sequencing-by-synthesis**

Like I mentioned before with the Sanger sequencing and replication, what you do is you basically have your single stranded molecule, you'd had a prime, the primer, the priming site, sorry, priming site at the enzyme for the polymerase. And then you start to add free floating modified nucleotides, which have these fluorophores which will basically emit your light. So there is a slight difference in Sanger sequencing, however. So once you start adding, your polymerase goes along, it starts adding a free nucleotide and then you have the fluorophore. Now this fluorophore does block the next nucleotide being added on. So this makes sure that on all of the molecules in your cluster, and you're only adding the correct nucleotide at one point. You then defy the laser, so the light is emitted, you take a photo.

However, the difference here between Sanger sequencing and why actually Illumina sequencing or the sequencing by synthesis is much more resource-sufficient is you can actually cut off this fluorophore and then basically repeat the process again. So rather than basically having your DNA molecule having one use of single use, you can sort of recycle the same DNA molecule to basically then add the next nucleotide. So this case will be, let's say this blue one, and I'll attach that on. Fire a laser, take a photo, remove the fluorophore, and then back again. And basically you can reconstruct the entire sequence of the molecule without having to basically throw away the DNA molecule once you've taken the picture

## *1. Introduction to NGS Sequencing*

of just that single nucleotide. And this normally happens at the Illumina sequencing somewhere between either categories of 50, polyester and helicopter. Normally happens at 50, 75times. And these are also known as cycles.

Okay, this is sort of what you should have been seeing like earlier in this animation. You should see a big black square with all these colored dots. So the black, black square is your flow cell and all these colored dots are these different clusters, each representing a single DNA molecule which we copy lots and lots and lots of times. And so what you can imagine sort of like in like a video or a film where basically you've got hundreds and hundreds of thousands of single shots just sped up over time, you add the first, so you have your flow cell, your DNA cluster is bound to the flow cell. You add the nucleotides, you fire the laser. And when you fire the laser, you should see basically a color being emitted. So in this case, green here and here, this is another DNA molecule, which is yellow, blue. And you take the picture and then you know that a green is only attached to teas. And so basically you see a tea molecule being picked up. So you record a tea, you remove the fluorophore, you add, wash those away, add the new fluorophores, the next base pair, the nucleotide will be bound onto the molecule, you fire the laser, it will be this time a blue. And this means you have a G. Then you again, cut the fluorophore off, add the new ones, fire the laser, emits a red. And so this is how you basically reconstruct all of these molecules. And again, the flow cell has millions and millions of these points, of these colored dots. And this is why you can see so many different DNA molecules all at once. Now, again, you have in your head these four colors before because you've seen the animation, also this picture, but there is differences to this. So there's actually two different, with the luminous sequencing, there's two different methods of actually emitting light. One is called the four channel system, or four color chemistry, where each nucleotide has a distinct color.

### **1.7.4. Colour chemistry**

But there's also on particularly the next seek and no seek machines, a slight different system, where they called, they tried to make it a bit cheaper by only using two colors or two dyes. And the approach they take there is that, again, a T is green, red is, C is a red. However, in this case, an A is actually two colors mixed at once. So if you emit that both, so if the machine picks up two colors, two wavelengths are being emitted, that is an A. However, if there was no color being emitted at all in that cluster, the machine reaches as a G, or a no detected dye. So this is very important for some caveats or things you have to consider when processing your data a bit later on.

## *1. Introduction to NGS Sequencing*

### **1.7.5. Paired-end sequencing**

So something you have to consider though is we're dealing with biology. It is not like, I don't know, chemistry or even sort of physics, where things are perfect and wonderful. Over time, areas start happening, we're not perfect. And essentially the imaging reagents start getting tired, the polymerases start adding mistakes, or don't bind on properly and more areas will occur. So sometimes your nucleotides will not bind, meaning you'll skip a base or you'll get multiple nucleotides being added once and you go forward one. And essentially within your cluster, the DNA molecules being sort of replicated and emitting light will get desynced. So the color will get a bit blurry and less clear.

And so first the machine does calculate some sort of base quality. So it captures the probability that it thinks it captured the right color, so the right nucleotide of each photo. But to the point where if it has no idea, it will call a dead base call, which will be reported as an N. And this became more and more of a problem, particularly in the early days of sequencing. And so people thought, how can we improve or correct such sort of errors? And the idea that came up was paired end sequencing.

So what this means is you do one sequencing in one direction, then you flip the entire read over and then read it from the other end. So whereas you have to consider that when you're in the forward direction, over time you're getting more and more errors. So the further you get into the molecule, the less confident you are. And so the more errors are gonna be. You can then turn it over and start the whole thing from scratch with fresh reagents. By going from the reverse end forward, you basically can correct the mistakes that were occurring at the end of the forward sequencing, but get the high quality calls from the other end from the beginning. So you can sort of, that's a bad explanation. But you can sort of see here, you read it once in this direction. So you read the DNA insert and the prime, the index. You turn it over and you do the same thing, but from the beginning. So you basically sequence the same DNA molecule twice. An added bonus of this is also you get more cycles. So if your DNA molecule is a single molecule, it's actually longer than the cycles. It's 50% fair, five base pairs. By going from the other end, you can capture any DNA nucleotides which you're missing from the forward sequencing. This is not so necessarily relevant to DNA. We will typically very short DNA molecules, but in some cases that will apply.

### **1.7.6. Demultiplexing**

Then it comes on to biological to computational sequence. How do we take these sort of very raw nucleotide sequences and put it into a format that a computer can read? This typically happens in a step of demultiplexing. It's actually very rare that you yourself as students will have to deal with this or do with this. Typically this is done by sequencing

## *1. Introduction to NGS Sequencing*

centers or by your lab team. But what this essentially consists of is normally or rather often nowadays, there are two steps in which you're stuck together, which colloquially are no demultiplexing.

The first step is called base calling. This is where you basically take your photo files of every single nucleotide and convert this into an ACTNG. So taking image files, putting into a text file. But also in most cases, we have multiple sequence, multiple samples at once. And all the samples have these barcodes or indices. And we need to actually separate these out. So you know that all of these DNA molecules come from this sample or these DNA sample, sorry, DNA molecules from this sample. And this happens in this demultiplexing step. So essentially a computer program will read in each DNA molecule whether it finds such a barcode across from one sample, let's say an example here, the red one that corresponds to sample one and the reverse here. And then the machine will basically sort or order the DNA molecules accordingly. So all of the DNA molecules which have these combinations and indices will put into sample one, sample two. So the blue and yellow will go here and so on. So this is something you, again, rarely will have to do yourself, but it's something just to keep in mind.

### **1.7.7. FASTQ File**

And the output of this demultiplexing step is something called a fastq file. So this is a text-based format for storing biological sequences sometimes of cases nucleotide sequences, but also with the base quality scores. So these are the things where the computer tries to estimate the probability that it thinks that the nucleotide call was correct. And both are encoded basically with ASCII text characters. Doesn't really matter what they are, if you're not familiar with that. So this is a very, very small example. These files can be gigabytes in size, so huge, huge text files typically compressed, but still a gigabyte start to being compressed.

But what they, all they are made of, of basically a repeating set of four different lines. You have the first line up here, and this is called the ID line. This stores multiple information from the sequencer. So the sequencing, or the sequencing machine ID, then a run number, a flow cell ID, so each flow cell will have an ID from the manufacturer. And then you have essentially a bunch of coordinates going on here, which basically tells you from which cluster on their flow cell has the DNA molecule come from. Then you can have a bunch of extra information. This is often quite random depending on the sequencing and what they put in here, but often people put things like barcodes, maybe sort of how many errors were allowed during gene multiplexing, because you could do a certain enough filtering during that step. On the second line, you have the DNA sequence itself. So in this case, it starts with an N, this is dead-based call. This is actually quite often common, because this is when the camera is still calibrating itself. And so the first base is often a bit rubbish, but

## *1. Introduction to NGS Sequencing*

then the rest of the molecule, as you can see, sorry, the sequence here is A, C, T, and G, so it's usually a new molecule. You'll then have a plus, which is a separator. And the fourth line of this repeating set of four is then the base quality scores. So these are random, sort of a random set of characters, which I'll explain in a second. But basically this tells you the confidence of that, how good we think that that nucleotide call was.

And then you can basically see the same thing here on the next line, and it repeats as follows. So you can see, for example, this number is a bit different, because it's from a slightly different coordinate cluster. So these quality scores, they are not uniform, I have to say. So it depends on both the age of the machine and what the manufacturer selected. But typically they will look something like this, where there's a fixed order in the ASCII characters. And each one will correspond to a different probability of Fred's score. I won't explain exactly what that is, it's a bit mazzy. But essentially what it means is that the higher the character along this score, the more confident you are of the base call, because the probability that it's incorrect is sort of low.

## **1.8. Sequencing recap**

So to recap, DNA molecules are essentially made of nucleotides, A, C, T, and Gs. We have two strands, which is complementary based pairings, C, G, CGI, AT, Atatoka, Star Wars, it's great. Modern DNA is very long, but AT DNA is very short. And this is very good for NGS sequencing, where we do this massive multiplexing. So where rather than trying to sequence few very long DNA molecules, we sequence lots and lots and lots of very short ones at very high accuracy, which we can then reconstruct the long sequences later on, if you've got a good DNA. So the main steps are adding adapters to create something called a library, which allows your DNA molecules to bind to the glass slide, the flow cell, or something called a lawn. You then basically make a new strand, each cycle, you basically add a fluorescent nucleotide, which you can fire a laser, AT, which emits a colour, you can take a photo, and by basically recording, the order of the colours being emitted in a single point in the flow cell, you can reconstruct the DNA sequence. This de-syncing of clusters results in lower based quality scores over time, so you can also improve this by paired end sequencing, where you basically sequence from one end, and then do turn it around, then you sequence from the other end with fresh reagents.

## **1.9. Sequencing and considerations for ancient metagenomics**

So for the last section of this lecture, I want to give you a few ideas of things you should consider when you're dealing with DNA for ancient genomics. Some of those are applicable

## *1. Introduction to NGS Sequencing*

to modern genomics too, but it's things I find that, through my career, people forget about in some cases.

### **1.9.1. Low DNA preservation**

So firstly is low DNA preservation. So when you're dealing with an ancient DNA, your samples are very old, and only have very little DNA in the sample, and during library preparation, you may have to do lots and lots and lots of amplification, make lots and lots and lots of copies of your DNA to make a sufficient amount to actually put sequencing. And this is important, and it will be discussed later during the week, but this is important because during library construction, you can actually inflate your counts in terms of DNA molecules that come from a particular taxon, micro-brothaxon, for example, and basically skew your estimate of which species are in your sample or not, or were in your sample because they're now dead. Also, by overamplifying your DNA molecules, you reduce the number of sequencers you actually get out there. By having these duplicate molecules, you're not actually providing any more extra information about your DNA library, and your sequencing flow cell only has a fixed number of sequencing slots, which basically, if you've overamplified your library, will basically fill up your slots and you will not sequence as many unique reads, and so the amount of information you're getting out of your DNA molecule can be problematic. So this is actually quite, I've jumped ahead quite a bit in terms of detail here, but this is, I wanted to have the slides here for you to go back to and recap later.

### **1.9.2. Index hopping**

Another thing during sequencing, you have to consider is something called index hopping. So this is a problem, particularly with Illumina Sequencer, or people are aware of with Illumina sequencing, and it's a challenge when you're doing multiplex sequencing. When you're sequencing lots and lots of samples at once, and you have to have these indexes or barcodes, which allow you to identify this DNA molecule comes from this sample. This seems to happen more often on a type of flow cell, which you find on Illumina HiSeq X and NovaSeq machines, and is ultimately caused by free floating index primers. And why that is a problem, is that if you do not sufficiently clean up your primers, during the clustering process, you can accidentally start adding on or switching barcodes between DNA molecules. This does happen at a relatively low rate, but it does happen. And so what it means is that essentially you switch the barcodes and you may accidentally assign a DNA molecule from one sample into another sample when you're doing demultiplexing. And so let's say you're dealing with a microbiome sample, and you have lots of, let's say, unless you have an oral microbiome sample and a gut microbiome sample, you may start seeing, for example,

## *1. Introduction to NGS Sequencing*

oral species popping up in your, or sort of oral species which are only found in the mouth, ending up in your gut samples, which may be a bit weird. This can also be a particular problem if you're doing microbial genomics, so doing, let's say, pathogen reconstruction, if you're working on that. And you mix capture results with your shotgun samples because then you may start picking up the high amount of capture results in your shotgun samples where you're doing screening. So you may start getting false positives there. There's quite a few papers on this, also in the context of ancient DNA, so van der Waag has got quite a good paper to understand that and also had to correct such, or estimate the level of this in your studies.

### **1.9.3. Sequencing errors**

They go back to the sequencing, you have to consider your sequencing errors. So if you don't sufficiently quality control and check for these errors happening on DNA sequencing, you may actually start incorporating errors into your analysis downstream. So for example, what may happen is if you have a low base quality score, the machine may have picked up the wrong base or the wrong nucleotide. And this means that your DNA molecule or sequence, when you compare to reference genome databases or reference genomes, may start going to the wrong place or match the wrong reference genome because you have the wrong nucleotides, the wrong sequence. Which can be a problem. Also, it can reduce your chance of getting sufficient overlap during the assembly, which is where you basically stick together all of the overlapping DNA molecules together to try and reconstruct the entire molecule. This is something that Alex will present on Thursday, like Subna. And also if you're doing variant calling for phylogenomics and you have very low coverage, this may also start increasing, increasing the add errors and you will do the wrong SNP call, which means that basically your, the relationship between your genomes in your tree, for example, will be skewed. And so it's always very important to check for such errors and check that your sequencing run was high quality.

### **1.9.4. Dirty Genomes**

You also have to consider, so this is again sort of jumping ahead, but there's a reason why I'm putting in this presentation, is dirty genomes. So unfortunately, there are many reference genomes which are very dirty. So dirty, I mean, for example, having a lot of adapters still in there. And this means you have the problem where if you yourself have not sufficiently cleaned up your DNA library to remove the adapters and remove also pre-processing, you will start seeing weird results. For example, I very, very highly expect that if any of you screen against the NTPI-NT nucleotide database, you will start seeing carp everywhere. And the reason why is because people somehow got onto the NCBI a

## *1. Introduction to NGS Sequencing*

whole carp genome without removing any of their adapters. So there's adapters sequences everywhere in the genome. And so whenever you have an adapter in your library, this will basically align to the carp genome and then you'll get carp, which particularly if you're trying to look at diet, for example, or ancient diet in, like say microbiome studies, where you look at some calculus, you may start seeing carp even if you're, I don't know, from, got samples from, I don't know, Chile or somewhere where carp is not expected to find. You also often will find this with zebrafish. So often many, many, many, I think it's Darius Varini or something like that, which makes no sense because all of these fish comes from one lake in Africa, but you see it everywhere in your metronome examples. And again, it's because of dirty genomes where they think of adapters or vectors in the genome.

### **1.9.5. Low Sequence Diversity**

Another thing is low sequence diversity. So what I mean by this is mononucleotide reads, like GGGGG, or dinoucleotide repeats. This is not so much of an error necessarily when you're doing metagenomics, but when you come into genomics, this can be a problem. So the problem with such DNA molecules, like this one here, GGGGG, is they're very unspecific. They give you no information. That can come from any species everywhere because they are very common across all genomes. So the problem here is that firstly, it slows down your processing because basically you are aligning against, or comparing of DNA sequence against many, many, many different genomes to ultimately say, I don't know which one it comes from, which is unnecessary. And also in some cases, it can inflate counts at higher nodes when you're doing an LCA. This will be described later on. But this can be very common. So if you remember this two-color chemistry which I mentioned earlier, where you don't have one color per base, but rather if there is no color emitted, the machine uses a G, particularly with an ek-seq and no-seq data, you will have a lot of these Gs, particularly as we're dealing with ancient DNA, which is very short. When you have very short ancient DNA and you don't reach all of your cycles, so let's say you're doinbase pair cycles, but your DNA molecules are onlbase pairs, you will basically get to the end of your DNA molecules and not add anything else. So you start getting these very long tails of Gs at the end of your molecules. And if you don't remove these, this will make it very difficult to correctly align your DNA molecule to a reference genome or sequence. So be aware to look for these and remove these. Also because it will speed up your processing. So to recap the considerations, a lot of this will make more sense later on in the other sessions, but consider your duplication rates. You check for lots of copies of same DNA molecule in your library. It's a good idea to check for index hopping, so making sure that the index combinations that you have in your library are correct and you've sorted your samples correctly. Always check for sequencing error and remove low quality bases if possible. Check for adapters, so to make sure you don't start finding CARP everywhere. And also it's a good idea to look for low-seq and diversity reads. For example,

## *1. Introduction to NGS Sequencing*

particularly if you put next-seq or no-seq data, because it just slows down your results and you get lower quality text-long assignment.

### **1.9.6. Q and A**

#### **1.9.6.1. How to design barcodes**

Okay. How to design the barcodes. That's a question from UD. That is quite tricky because there's a lot of considerations you have to make when making sure there's a balance and they're not too similar to each other. Often manufacturers have tools which allow you to basically generate this. I believe they also to sentence and have standard sets which they can also send you that you request. So you yourself do not have to necessarily design these. It depends on your lab. So often I'd say speak to your sequencing center if you have them. Because they all have advice. All check manufacturers, I think most like Agilent and Illumina will also basically have such things for you then. Yeah, sometimes they make it a little bit defined but you can't find them. And if you read the Meyer and Chercher article for buildinnew libraries, it would be provided in that set. Yeah. Could everyone hear Tina then? No? Okay. So she said that often the manufacturer make it a bit hard to find such functionality on their websites and stuff, but you can often do that. But also if you read the sort of classic paper by Meyer and Chercher 2011, 12, they actually have the set of barcodes that you can use yourself. So I think that link, that paper is on the website somewhere but we can also share with you.

#### **1.9.6.2. How many indices can you use**

How many indices can you use? So this is a good question. This depends on your strategy and is slowly changing over time. So you can actually choose one index if you want, which is at the beginning of your, this is from Laura, which is the beginning of your molecule. What has been recommended and what the Meyer and Chercher paper introduces double indexing where you have two at the end. And these are attached to your adapters. What people are commonly doing now is actually adding additional barcodes called inline barcodes. So these are very short sequences, about seven base pairs I think, which you actually attach immediately to the DNA molecule before library preparation. So after extraction before library preparation. And this actually helps you with, sorry, with correcting for index hopping. So if you read the van der Waal paper, which I mentioned earlier, we can send the link again later. They also describe how they use these internal barcodes to separate out. So for example, you can get from a manufacturer about 100, let's say barcodes, but you will normally per library can have somewhere between one to four separate identifiers.

## *1. Introduction to NGS Sequencing*

### **1.9.6.3. Why Gs called more often in two colour chemistry sequencing**

So I have a question regarding the gene calling for Gs, right? You mentioned that Gs are a characteristic of ancient DNA or sequencing errors. Why is it specifically Gs that are called more often rather than the other bases? The reason, okay, so that is because your DNA molecule is very short. So let's say your DNA sequence is onlbase pairs long, but your sequencing cycles, so the number of cycles of imaging your machine is gonna do is let's sabase pairs. Once you've got through thbase pairs, there is nothing to sequence anymore. Your lights are not going to emit anything. So when you're on nobody-connected data, if no, sorry, machines, if no light is emitted, it reads it as a G. And you have to remember that the machine is not going to stop imaging once that one DNA molecule is finished. The DNA, the sequencing machine will keep taking photos until it's reached to the number of cycles you've set, whether in this case, 75. So once you've exhausted your DNA molecule, there's nothing to sequence, nothing to image anymore. So basically the machine will just keep picking up G for every remaining cycle of the run. Does that make sense? Yes. Yes, it does. But still I don't get why G and not like AT. Why is it specifically the space? Because on nobody can make sick data, whatever reason they've decided Gs means nothing. There's no color. Once it runs out of the...

Yeah, so basically the problem with ancient DNA is we often have very short reads. So you might do say two bsequencing, which is very common, but you might have a read that's onlbases long. And so once it kind of runs out of DNA, it will just, it won't sequence anymore. So there'll be no more fluorophores. And because the NovaSeq and the next you can interpret that as a G, you'll just get these polyG tails, but actually it just means no more data. And another thing to... Could you make it up for yourself? And I think this may also happen in modern data as well. If you've fragmented your modern data too short, you'll also get that. It's just that in the complex of ancient DNA, the reason why I said that is because we are naturally already very, very short because of the degradation. And I'll also say this is, if you're used to doing modern DNA, this is where it's really different because let's say you're sequencing a regular library. What you would normally do for modern DNA is you have genomic DNA, which is huge. So for your microbial DNA, each genome is something likmillion bases long, and that's way too big for an aluminum machine. So what you would do is you would shear it either enzymatically or by sonication, usually to an average size of aboubases. And then you do your alumina sequencing usually two by 150. So you kind of measure one side, then you measure the other side, and you get a total obases sequenced out of thbase pair read. You never run out of DNA. You never actually get to the end of the molecule when you're sequencing. And so for most people that do modern DNA sequencing, they've never dealt with this problem before because they never see it because they're always sequencing a DNA molecule longer than what their sequencing chemistry can actually do. For ancient DNA, it's very different. We actually don't shear. We take advantage of the fact that because our DNA is short, we don't have to shear. And

## *1. Introduction to NGS Sequencing*

because we don't shear, it actually allows us to exclude some of the modern contamination because any modern DNA that's in there will be so long that it won't build a proper library and it won't be sequenced. And so it kind of helps us clear out some of the modern DNA that might be present. And so we will only sequence the short DNA sequences, which are more likely to actually be ancient. But the problem there is we're dealing with the real size of the ancient DNA, which might be bases, 30 basesbasesbases. We don't have necessarily the kind of consistency you would have if you were intentionally shearing modern DNA. So we do have some sequences that are very short.

### **1.9.6.4. What is full genome sequencing**

Okay, so Liasat asked, when we're talking about whole genome sequencing, so WGS and full genome sequencing, FGS, is it the same? I've never heard of FGS. So yes, I would say it probably is.

### **1.9.6.5. Tools for generating indices**

And then Jaime asks, is, I hope I'm saying that right. Is this kind of program publicly available? Is this to? Sorry? It was the index checker to make sure your pool is not. It was the index checker, yes. So again, lots of tools online, I think, basically, to make sure there's no overlap. Normally the manufacturer will offer such thing.

## **1.10. Readings**

### **1.10.1. Reviews**

[@Schuster2008-qx]

[@Shendure2008-fh]

[@Slatko2018-hg]

[@Van\_Dijk2014-ep]

### **1.10.2. Sequencing Library Construction**

[@Kircher2012-fg]

[@Meyer2010-qc]

## *1. Introduction to NGS Sequencing*

### **1.10.3. Errors and Considerations**

[@Ma2019-lg]

[@Sinha2017-zo]

[@Van\_der\_Valk2019-to]

### **1.11. 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?

### **1.12. References**

## **2. Introduction to Ancient DNA**

### **2.1. Lecture**

Lecture slides and video from the 2022 edition of the summer school.

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

### **2.2. Questions to think about**

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

## **3. Introduction to Metagenomics**

### **3.1. Introduction**

### **3.2. Lecture**

Lecture slides and video from the 2022 edition of the summer school.

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

### **3.3. Questions to think about**

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

# 4. Introduction to Microbial Genomics

## 4.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.



Slides and videos are from previous version of summer school and may not match text

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

## 4.2. Introduction

Microbial genomics is the identification and study of microbial genomes, their structure and composition. Microbial species come from diverse groups of organisms but are generally defined as organisms which are not visible to the naked eye. Most prominent amongst them are **bacteria**, which are single-celled prokaryotes with either circular and linear dsDNA genomes (up to ~14 Mbp). **Archaea** are mostly mutualistic or commensal single-celled prokaryotes with circular dsDNA genomes (~0.5 to ~5.8 Mbp). The often references group of **protozoa**, is a not phylogenetically defined grouping (often used in databases). They are a wide variety of free-feeding single-celled eukaryotes from the protists group with larger genomes (~2.9 to ~160 Mbp).

Finally, **viruses** are understood as microbial organisms, even though they are not technically considered as “organisms”. Viruses lack the ability to live or replicate independently of host cells. They are mostly defined as infectious agents and have smaller linear or circular ssDNA, dsDNA & RNA genomes (2 kb to over 1 Mb). Finally, another category of viruses are **retroviruses**. These will usually be RNA viruses which can integrate into the host genome by converting to DNA. But there are also DNA viruses, which integrate into the human genome. There the virus can either be latent and be later triggered to activate, continuously produce virions or lose the ability to produce virions and become part of the host genome. An integrated genome is called a *provirus* and vertically inherited proviral sequences are called *endogenous retroviruses* (ERVs).

#### 4. Introduction to Microbial Genomics

There are several types of microbial organisms found within ancient DNA samples of animal hosts.

- (a) **Pathogens:** infectious microorganisms or agents which cause disease in the infected host. Usually specialized organisms with delimited ecological niches.
- (b) **Commensals:** non-infectious microorganisms or agents which live within a host/environment without causing harm and can be beneficial.
- (c) **Environmental microorganisms:** Environmental microbial organisms not endogenous to the host ante-mortem, which e.g. stem from the depositional environment, the storage environment or the lab.

However, it should be noted that not all cases are quite as clear-cut as there are wide varieties of microbial lifestyles. Some organisms can be considered pathogenic in one sampling location and commensal in the next (*pathobionts*). Some organisms can also be considered harmful, when they are represented in very large amounts or in combination with other organisms. The sampling location can therefore be important information when studying the health of hosts.

### 4.3. Larger Genomic Elements

When using reference sequences from public depositories such as NCBI you will often be confronted with multiple intervals which represent the chromosome(s) and additional extrachromosomal sequences associated with a species or strain. A **chromosome** is understood as the main genetic element. It contains the core genome with all essential genetic elements, which usually remain the same across a species.

**Plasmids**, on the other hand, are extrachromosomal DNA replicons. They are replicating and acquiring/losing genetic material independently of the chromosome and can be present in high copy numbers. Plasmids can be vertically or horizontally transferred. The number of plasmids of a bacterium varies a lot (0 to 30+), and can vary within a species. They can carry virulence genes and genes which can give selective advantages. Plasmids generally have much smaller circular genomes (1 to over 400 Kbp). It should be noted that they are under-represented in databases, meaning that they are often ignored during the assembly/sequencing process and thus not present in all reference sequences, even if they could otherwise have been there.

Besides plasmids, **bacteriophages** (or phages) are also important extrachromosomal (and sometimes chromosomally integrated) genetic material associated with bacterial genomes. Although they are not considered as part of the genome, and therefore part of reference sequences, if they are not integrated in the host cell genome. Bacteriophages are dsDNA

#### 4. Introduction to Microbial Genomics

viruses of small size that infect bacteria. As phages have mostly dsDNA genomes, they are also well suited for the recovery using standard aDNA techniques. They can be found everywhere and carry gene sets of diverse size and composition. They enter the cytoplasm of bacteria, where they can then replicate. There are a huge number of phages, and some are specific to certain bacterial genera or species, which can be useful for identifying taxa or phylogenetic clades. When bacteriophage genomes integrate into the host cell chromosomal genome (e.g. through horizontal gene transfer) or exist as a plasmid within the bacterial cell, they are called **prophages**. They can make up a large fraction of the pan-genome of plastic species.

### 4.4. Sampling & Sequencing

For ancient DNA samples, screenings for organisms of interest is usually performed on “shotgun” sequencing data, which can then be either further gnomically analysed following species identification or the library will be enriched for organisms identified during screening. Prior to sequencing, laboratory workflows, and particularly library building setups, can have a major impact on your final output (e.g. if you are interested in RNA/ssDNA viruses but have dsDNA libraries).

Each pathogen is subject to tissue tropism, meaning that each species will have a range of cells or tissue types it will preferentially or exclusively proliferate and replicate in. This phenomenon is called **tissue tropism**. In some instances, the pathogen can also become latent within said tissues, meaning it will remain within the host tissue without causing disease. Organisms which cause **bacteremia** or **viremia**, meaning they are present in the bloodstream, will not be as restricted and are generally considered easier to detect. Although this can be limited by the disease phenotype or the stage of infection (e.g. *Haemophilus influenzae*).

Accordingly, what organism you can find is highly dependent on the sample and where it was taken from. For example, if the organism of interest enters the bloodstream, teeth, which are vascularized and are excellent DNA archives, will probably be a good choice for sampling. Some pathogens (e.g. *Mycobacterium leprae*) are present in higher quantities within lesions caused by their infections, making those lesions better suited for sampling. In the case of infections, which are unlikely to be retrievable from hard tissue, calcified nodules can be an interesting type of sample. Bone will generally be less well suited, with some exceptions. Integrated viruses and retroviruses will be likelier to be found in samples types with higher host DNA, such as the petrous bone or the ossicles. And early childhood infection can also be found within low remodelling cortical bone.

#### 4. Introduction to Microbial Genomics

##### Some Myths to Dispel...

- Microbial content and pathogen content is NOT directly correlated to human DNA content. Human DNA content is not a measure for overall sample preservation. You can have really bad samples for human DNA but get a full microbial genome from the same sample!
- You can also find microbial signatures in petrous bones etc. it will just be a different range of organisms. Everything is worth getting screened!

The typical number of sequences recovered for an organism will depend on a range of factors which can only rarely be predicted even in samples for which osteological/historical record show a clear association with a pathogen. Major factors are: overall sample preservation, depth of sequencing, abundance of the organism peri-mortem, disease phenotype, genome size & composition, “noise” from other organism, etc.

Overall, it is considered a good result if your target organism makes out around 0.1% of shotgun datasets, but in many cases it will be way less. However, depending on the organism, the amount of data you need to confidently classify it will be very different, mostly due to sequence conservation and genome size and complexity. While a bacterium is generally easier to detect, their large genome size and sequence conservation, makes them harder to verify. On the other hand, viruses, having much smaller genomes, are harder to detect, especially at low sequencing depth, but generally easier to verify within a margin of uncertainty.

For some applications in microbial genomics, unselective/biased sequencing can be key, and for this shotgun sequencing is a powerful tool at our disposal since it allows for the indiscriminate sequencing of all DNA found within the sample. Particularly in cases of organisms with large pangenomes, where you might be interested in looking for a large variety of intervals, which would be very costly to do using target enrichment. Or in cases where novel insights and genomes become relevant following the design of a target enrichment kit. Additionally, it allows for the simultaneous analysis of both microbial and host DNA. Understanding the composition of the microbial community of your sample can also be highly relevant with regard to identifying co-infectants, reconstructing disease histories and excluding gene intervals which could also stem from commensals or contaminants.

##### Shotgun sequencing for microbial genomics:

###### Pro(s):

- Non-targeted taxa can be found.
- Allows the analysis of DNA from both microbial organisms and host simultane-

#### 4. Introduction to Microbial Genomics

ously.

- Allows you to detect untargeted co-infections.
- Allows you to understand the composition of the microbial community (can be relevant for genomic analysis).
- No knowledge of the taxon genomic diversity is needed beforehand.
- Enables the long time use of the data with ever growing databases.

##### Con(s):

- Can be much more expensive (depending on the sample).
- Overall less effective at generating adequate/high coverage data.

However, microbial species make up only very small fractions of genomic libraries. So small that it can either be impossible or very expensive to try to assemble a genome using only shotgun data in most circumstances. This is usually where target enrichment, or capture, comes into play. Often, the baits required for capture are custom designed for each study. The design of such kits necessities appropriate knowledge of the genetic diversity to be expected during analysis. In most cases, a mixture of shotgun and target enrichment can be most effective, especially with regards to authentication if enrichment is performed on UDG/Half-UDG genomic libraries.

##### Target enrichment for microbial genomics:

##### Pro(s):

- Much cheaper per genome, especially in samples with bad preservation.
- Effective at generating high coverage genomes.
- Allows for the generation of large number of genomes effectively.
- Great for core-genome reconstructions.

##### Con(s):

- Necessitates knowledge of the genetic diversity relevant to your research questions during the design phase.
- Any sequence not included, within a margin of variation, will not be captured.
- Will not generate any data with regards to the host or the rest of the microbial community (in fact that is the point).
- For some species, capturing the pangenome would be extremely expensive provided it doesn't fully exceed kit sizes.

## 4.5. Validating the Presence of Organisms of Interest

Following analysis of your data using appropriate databases and taxonomic classifiers, it is time to validate your hit. The results of a classifier should not be your end result. Properly validating the presence of a species is key to any genomic analysis! An identification by classifiers is not a sufficient validation on its own, as it can be very tricky to work around false-positive hits and missing database entries. Databases are biased towards pathogenic species, as they represent the bulk of research. Closely non-pathogenic species will either be represented less or not at all, which leads to high read assignments to the LCA (Lowest Common Ancestor) and pathogenic taxa. You should also consider whether it makes biological sense for the species you found to be detected in the sample's tissue and based on your laboratory workflow.

For ancient DNA, the validation will often consist of three steps:

- (1) Authentication of the data as aDNA using deamination signatures.
- (2) Comparative or competitive mappings to relevant reference sequences (target organism and closely related environmental/commensal species).
- (3) Identification of intervals specific to the target species or sub-species.

### 💡 Tip

It can also be a good idea in some cases to double check the modern data you are using during your analysis. There are cases in which the metadata doesn't match reality.

Increased coverage in selected intervals in an alignment is often caused by sequence conservation. Meaning that you could have the same number of reads mapping to a genome but in the first case all reads are mapping to 5% of the genome with high depth of coverage, whether in the other one you will see low coverage across the whole sequence. The first one is likely to be a false positive, and your detection is based on intervals from either a commensal or an environmental organism which have very high sequence similarity to the ones in your reference genome. This can be further worsened by low complexity intervals. These peaks will usually also reflect in the edit distance, as such tiling will also result in more sequence mismatches. While this can be reduced with higher mapping stringency, this in turn will probably decrease your coverage significantly and in some cases this can cause a reference bias. However, in most cases, these intervals will not pass the mapping quality filter.

#### 4. Introduction to Microbial Genomics

##### Definition

**Conserved Intervals:** Regions of genomes common to organisms from the same taxonomic units (can affect all taxonomic ranks and child taxa, and very distantly related organisms). They will show high sequence similarity and cause noise/contamination within the analysis, particularly for ancient DNA.

##### 💡 Tip

In many cases, running a range comparative and/or competitive mappings to closely related species is advised in order to exclude a misidentification and the presence of multiple species with similar sets of genes in your dataset (e.g. *Neisseria meningitidis*). It can also be useful to use multiple reference sequences per species if it has multiple phylogenetically strongly delimited clades (e.g. *H. influenzae*)

Viruses are less affected, as they do not carry house-keeping genes, and only have a very small usually highly specialized set of genes, which makes them easier to validate. However, viral sequencing of non-pathogenic species has only recently started to become more popular due to metagenomic studies, i.e. non-pathogenic taxa are even more under-represented, and many viral genomes have low complexity repeats over large portions of their sequence.

Finally, intervals or mutations specific to the target organism should be identified and investigated. This can range from plasmids, genes, phages to specific mutations. Often a combination of these factors is required to confidently validate the presence of the organism. This step should not be overlooked as this will not always be clearly visible in a phylogeny, depending on the composition of the alignment used.

## 4.6. Genomes

##### ℹ️ Strains VS Genomes

- A **strain** is a genetic variant or subtype of a species or sub-species. They usually exhibit significant genetic differences to other strains of the same taxonomic unit. The level of change required for significance can vary.
- A **genome** can be the exact copy of an already sequenced strain.

**The case of aDNA:** Since our samples are so old, most of our genome constitute new strains. However, caution is advised. Particularly with clonal species (e.g. Black Death genomes).

#### 4. Introduction to Microbial Genomics

There are many different microbial species, and they all have very different evolutionary dynamics and genomes. This means that depending on which species you are working on, you might have to approach genomic analysis differently. Additionally, the research questions which could be answered with your data will/can also be rather different. This sections will summarise some of the core principles underlying many possible research questions.

Mapping is an important tool in ancient DNA to investigate the presence and absence of genomic intervals. However, mapping can be impeded by the reference sequence itself. Gene duplication, low complexity sequences and GC Skew (over- or under-abundance of GC in intervals) can impact the mappability (and mapping evenness) of sequencing reads to the reference sequence and their mapping quality score. Which in turn might be problematic during analysis. This can be expressed in mappability estimates, which estimate how likely it is for short reads to map to a sequence interval based on its composition and uniqueness.

Applying a mapping quality filter by default after a bwa aln mapping, will not only cause all bad quality reads to be removed from the alignment, but also any read which could align to multiple sections of your reference genome. This means that most likely every duplicated interval/gene, specific or conserved, and low complexity regions will also be removed! Mapping quality filters are important tools, but you should make sure you understand how much of the genome is actually covered (e.g. terminal repeats in viral genomes) and when to apply them. The same applies for competitive mappings using bwa aln!

##### Definition

**Sequence Complexity:** Defined as the observed vocabulary usage for word size. Meaning, how complex is the sequence of nucleotides within an interval. Often given as values calculated using either entropy or DUST algorithms. E.g.:

AAAAAAAAAAAAAA » *LOWEST COMPLEXITY*  
ATGATGATGATGATGATGATG » *LOW COMPLEXITY*  
ATGTTTCGAGGCATGATAACCGTATG » *COMPLEX SEQUENCE*

##### Definition

**GC Content:** Is usually given as a percentage of guanine and cytosine bases within the sequence. Can impact DNA stability, amplification, sequencing, and capture if the GC content is too high or too low. Too high or too low GC-content will also lead to a loss in sequence complexity. E.g.:

GCCCCCCCCGGAATGGACCCGCGCCT » *80% GC*  
ATTGGAACCTAATTATATAGCAA » *20% GC*

#### 4. Introduction to Microbial Genomics

##### 4.6.1. Recombination

Bacteria and viruses are haploid and have small genomes, making some things much easier, but... They like to mix and match! Microbial organisms will exchange genetic material using a range of biological mechanisms (conjugation, transduction, and/or transformation) this leads to increased genetic diversity via **horizontal gene transfer**. Some parts of the genome will be more heavily affected by this exchange than others. This constant exchange of genetic material can happen while maintaining genome size, meaning that while genes are gained, others are lost. How much **recombination** a genome will undergo, is highly dependent on the species or sometimes the phylogenetic clade. These dynamics cause recombination breakpoints in reference based alignments where SNP counts will increase, which can impede phylogenetic analysis (e.g.: by causing elongated branches).

However, some species do not recombine at all or do so only very rarely. These species have **clonal genomes**, meaning they transfer their entire genome vertically, most the of time, without significant changes to their pangenome or the sequence itself. Actually, many of the species which have been extensively studied using aDNA, fall within this group (e.g. *Y. pestis*, *M. leprae*).

Most microbial organisms change their genomes via recombination and gene loss/gain, while maintaining genome size and retaining their core genome. Virulence is then often defined by gaining or losing virulence factors. However, some increase their virulence by reducing their genome and becoming highly specialized (e.g. *Borrelia recurrentis*) in what is called **reductive evolution**. In these cases, it is of interest to include closely related phylogenetically “basal” species (pathogenic or not), which might inform you on genes/plasmids the modern genomes have lost, but the ancestral genomes might have still retained.

##### 4.6.2. Pangenomes

As mentioned, microbial organisms can be very plastic and exchange genomic material. This can lead to a wide variety of genes being represented within a single species. Pangenomes represent the sum of all genomic intervals represented within the genomes of one species. And thus, characterising the genome of a new strain can be a lot more complex than sticking to a single reference sequence.

Pangenomes are made up of three groups:

- **Core Genes:** a set of essential genes common to all strains of a species.

#### 4. Introduction to Microbial Genomics

- **Accessory/Shell Genes:** a set of genes common amongst some strains of a species which encode for supplementary or modified biochemical functions. In some cases these will also be virulence genes.
- **Singleton/Unique/Cloud Genes:** genes, which are specific to single strains of a species. Unlikely to be recovered and identified using ancient DNA.

We also differentiate between open and closed pangenomes. In open pangenomes, the gene set available to the bacterium constantly expands by acquisition and loss of genes via horizontal gene transfer from its environment, for the purpose of adaptation (e.g. environmental adaptation, metabolism, virulence and antibiotic resistance). All while maintaining genome size and vertical transmission of the core genome. Closed pangenome have limited exchanges of genetic material. This is often the case in highly specialized organisms. So while they are much more plastic than strictly clonal species, the available genetic pool for exchange will be smaller.

#### 4.6.3. SNP Effects

Single nucleotide polymorphisms (SNPs) can heavily impact the function of genes and the phenotype of an organism when they happen to affect and change the amino acid sequence of coding elements of a genome. This effect is frequently used in ancient DNA to either predict the presence/absence of significant mutation, which could have changed the phenotype of a disease.

##### Definitions

**Non-synonymous (missense) mutation:** SNPs that alters the amino acid sequence of a protein by changing one base of a triplet group. E.g.: GAT>GGT == Asp(D)>Gly(G)

**Frameshift:** Changes in the amino acid sequence caused by insertions or deletions of nucleotides in coding sequences which are not multiples of three.

**Start/Stop Codon:** Nucleotide triplet which signals the beginning and end of translation.

**Stop/Start-gain mutation:** Mutation which leads to a change in the amino acid sequence and leads to a premature stop/start of translation.

**Start/Stop-loss mutation:** Mutation which leads to a change in the amino acid sequence and leads to the loss of a start/stop codon. Leads to the reduction or elimination of protein production.

The impact of such changes is predicted by tracking changes in the translation of coding sequences using a reference sequence and SNP/MNP/Indel calls. The problem when using

#### 4. Introduction to Microbial Genomics

ancient DNA, is that genome coverage is often uneven, and it isn't rare to lack full coverage of intervals of interest or lack resolution to correctly call indels across the reference. Larger genomic insertions and recombination are also not accounted for when a genome is reconstructed solely using a reference based approach. This means that while SNP effects can indeed be very interesting and potentially highly significant, they should be understood as prediction based on the available coverage. It is therefore important to report coverage over the entire coding and promoter intervals, when discussing SNP effect.

Important exception are known and heavily conserved mutations, which for example are involved in **pseudogenization**. Pseudogenization is the process through which genes lose their function due to mutations but remain in the genome without being expressed or without having a function. It is a mechanism underlying gene loss. They are significant in aDNA research because we can capture "active" versions of pseudogenes and potentially date their loss of function. It is an critical part of the genome reduction process of highly specialised bacteria.

##### 4.6.4. Virulence Associated Intervals

One of the question ancient DNA research is interested in is the evolution of virulence in pathogenic species. The location and nature of virulence factors can vary. An increase in virulence can be caused by gene gain (on chromosomes or plasmids), plasmids, mutations, or changes to complex gene mechanisms (e.g. immune evasion). To understand the underlying processes of virulence adaptation has been one of the recurring questions in the field. Virulence intervals vary but can be found within the literature and in some cases in curated databases.

###### 💡 Tip

- Are there loci associated with increased virulence known to the literature?
- Are there phylogenetic groups associated with increased virulence?

Not all species have evolutionary dynamics which allow us to detect a temporal signal or recognise geographic structure within their phylogenies. However, phylogenetic clades can also inform us on other aspects of their evolution (virulence, ecological niche etc.). When investigating virulence, you should be checking for the presence of chromosomal Virulence factors (genes, mutations), currently this is mostly done using either as a Presence/Absence matrix or a cluster analysis. As well as investigating the presence/absence of plasmids and plasmid mediated virulence factors and functional mechanisms, and relevant SNPs.

Beyond virulence, some genes, and gene combinations can also inform us on changes in disease phenotype and microbial adaptation to host or ecological niche.

## 4.7. Coinfections, Multi-strain Infections etc.

The detection and study of co-infection should also be considered. While it may not always be possible to reconstruct full genomes for each pathogen, the knowledge of the presence of multiple additional infectious agents alone can be very informative. Additionally, the detection of multi-strain infections (in high coverage genomes), should also be considered, especially in the case of multi-focal infections for which multiple samples are available.

### Reconstructing disease phenotype & etiopathology:

- Where was the genome isolated from?
  - How can this relate to the disease phenotype?
- Do we have knowledge of potential coinfection? Strain differences?
- Do we have knowledge of the extent of affected tissues within the host body?
- Can we conclude whether it is likely to be a chronic or an acute infection?
- In what demographic cohort would the individual(s) belong to?
  - What could this tell you about the course of the disease or the likelihood of acute infections?
- Does the host genomes show any clinically relevant variants which could impact how the pathogen would affect them?

**FINALLY:** Integrate the data in a synthesis of all available data types. E.g.: osteology, archaeology, isotopes etc.

## 4.8. Resources

Some useful websites:

- GenBank NCBI Database: <https://www.ncbi.nlm.nih.gov/genbank/>
- The European Nucleotide Archive (ENA): <https://www.ebi.ac.uk/ena/browser/home>
- The virulence factor database (VFDB): <http://www.mgc.ac.cn/VFs/main.htm>
- Viral Neighbour Genomes In The Assembly Resource (NCBI): <https://www.ncbi.nlm.nih.gov/genome/viruses/about/assemblies/>
- NCBI Taxonomic Browser: <https://www.ncbi.nlm.nih.gov/taxonomy>
- BacDiv: <https://bacdive.dsmz.de/>
- Bacterial And Viral Bioinformatics Resource Center: <https://www.bv-brc.org/>

## 4.9. Questions to think about

Get to know your genome!

- Is the genome circular or linear?
- Does the species carry any plasmids?
- How genetically diverse are the genomes of the species?
- Does the species share large portions of its genome with closely related environmental or commensal microbial organisms? If yes could they also be present in the data?

Get to know your species!

- Is the species clonal or heavily recombinant? Overall? Within clades?
- Is the pangenome open or closed? How large is it?
- Has the genome undergone a reductive evolution?
- Is the genome plastic but maintains genome size?
- Are there significant genomic/phenotypic differences across clades?
- Is your samples grouping within modern diversity or basal to it?

Depending on the organism, available data and databases will be very different...

- What data is available?
- What type of data is available?
- What metadata is available?

Reconstructing disease phenotype & etiopathology:

- Where was the genome isolated from?
  - How can this relate to the disease phenotype?
- Do we have knowledge of potential coinfection? Strain differences?
- Do we have knowledge of the extent of affected tissues within the host body?
- Can we conclude whether it is likely to be a chronic or an acute infection?
- In what demographic cohort would the individual(s) belong to?
  - What could this tell you about the course of the disease or the likelihood of acute infections?
- Does the host genomes show any clinically relevant variants which could impact how the pathogen would affect them?

## **5. Introduction to Evolutionary Biology**

### **5.0.1. Lecture**

Lecture slides and video from the 2022 edition of the summer school.

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

## **Part II.**

# **Useful Skills**

## *Introduction to the Command Line (Bare Bones Bash)*

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

## **Introduction to the Command Line (Bare Bones Bash)**

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

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

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

## **Introduction to R**

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

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

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

## **Introduction to Python**

While R has traditionally been the language of choice for statistical programming for many years, Python has taken away some of the hegemony thanks to its numerous available libraries for machine and deep learning. With its ever increasing collection of libraries for statistics and bioinformatics, Python has now become one of the most used languages in the bioinformatics community.

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

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

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

## **Introduction to Git and GitHub**

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

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

# 6. Introduction to the Command Line

## 💡 Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate bare-bones-bash
```

## 6.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

### 6.1.1. Session 1

This is the Brief version of a more Broad two-part session that we gave in 2022, which can be found [here](#).

The teaching material for the Boundless Bare Bones Bash (Basic + Boosted) can be found in the Bare Bones Bash website.



Figure 6.1.: BareBonesBash Logo

The original Bare Bones Bash material was created by Aida Andrades Valtueña, James Fellows Yates, and Thisseas C. Lamnidis.

## 6. Introduction to the Command Line



The author portraits were designed by Zandra Fagernäs.



All material is provided under a Creative Commons Attribution-ShareAlike 4.0 International License.

### 💡 TL;DR: uncollapse me!

Below is a quick reference guide to the commands discussed in this tutorial. To understand actually what each command does, carry on reading below! For a complete run of all these commands AND MORE(!!), consider following the full Bare Bones Bash walkthroughs here.

| command | description   | example          | common flags or arguments      |
|---------|---|------------------|--------------------------------|
| pwd     | print working directory                               | pwd              |                                |
| ls      | list contents of directory                            | ls               | -l (long info)                 |
| mkdir   | make directory  | mkdir pen        |                                |
| cd      | change directory                                      | cd ~/pen         | ~ (home dir), - (previous dir) |
| ssh     | log into a remote server                              | ssh @.com        | -Y (allows graphical windows)  |
| mv      | move something to a new location (& rename if needed) | mv pen pineapple |                                |

## 6. Introduction to the Command Line

|        |  |                                       |                                |
|--------|--|---------------------------------------|--------------------------------|
| rmdir  | remove a directory                                   | rmdir pineapple                       |                                |
| wget   | download something from an URL                       | wget www.pineapple.com/pen.txt        | -i (use input file)            |
| cat    | print contents of a file to screen                   | cat pen.txt                           |                                |
| gzip   | a tool for dealing with gzip files                   | gzip pen.txt                          | -l (show info)                 |
| zcat   | print contents of a gzipped file to screen           | zcat pen.txt.gz                       |                                |
| whatis | get a short description of a program                 | whatis zcat                           |                                |
| man    | print the man(ual) page of a command                 | man zcat                              |                                |
| head   | print first X number of lines of a file to screen    | head -n 20 pineapple.txt              | -n (number of lines to show)   |
|        | pipe, a way to pass output of one command to another | cat pineapple.txt   head              |                                |
| tail   | print last X number of lines of a file to screen     | tail -n 20 pineapple.txt              | -n (number of lines to show)   |
| less   | print file to screen, but allow scrolling            | less pineapple.txt                    |                                |
| wc     | tool to count words, lines or bytes of a files       | wc -l pineapple.txt                   | -l (number of lines not words) |
| grep   | print to screen lines in a file matching a pattern   | grep pineapple.txt   grep pen         |                                |
| ln     | make a (sym)link between a file and a new location   | ln -s pineapple.txt pineapple_pen.txt | -s (make <i>symbolic</i> link) |
| nano   | user-friendly terminal-based text editor             | nano pineapple_pen.txt                |                                |

## 6. Introduction to the Command Line

|         |   |  |  |
|---------|---|--|--|
| rm      | more general<br>'remove' command,<br>including files                                  | rm pineapple_pen.txt   | -r (to remove directories)   |
| \$VAR   | Dollar sign + text<br>indicates the name<br>of a variable                             | \$PPAP   |  |
| echo    | prints string to<br>screen  | echo "\$PPAP"  |  |
| for     | begins 'for' loop,<br>requires 'in', 'do'<br>and 'done'                               | for p in apple pineapple;<br>do echo "\$p\$PPAP";<br>done applePen<br>pineapplePen |  |
| find    | search for files or<br>directories  | find -name 'pen'   | -type f (search only for<br>files) -name '*JPG'<br>(search for file names<br>matching the pattern) |
| "\$var" | use double quotes<br>to use contents of<br>variable                                   | pen=apple && echo<br>"\$pen"   |  |
| <>2>    | redirects the<br>standard<br>input/output/error<br>stream respectively<br>into a file | cat <file.txt<br>>file_copy.txt<br>2>cat_file.err                                  |  |

### 6.1.2. Introduction

The aim of this tutorial is to make you familiar with using bash everyday... for the rest of your life! More specifically, we want to do this in the context of bioinformatics. We will start with how to navigate around a filesystem in the terminal, download sequencing files, and then to manipulate these. Within these sections we will also show you simple tips and tricks to make your life generally easier.

This tutorial is designed so you follow along on any machine with a UNIX terminal (no warranty provided).

### 6.1.3. The 5 commandments of Bare Bones Bash

The Bare Bones Bash philosophy of learning to code follows five simple commandments:

## 6. Introduction to the Command Line

---

|   |   |
|---|---|
| 1) Be lazy!                               | Desire for shortcuts motivates you to explore more!           |
| 2) Google The Hive-Mind knows everything! | 99% of the time, someone else has already had the same issue. |
| 3) Document everything you do!            | Make future you happy!  |
| 4) There will ALWAYS be a typo!           | Don't get disheartened, even best programmers make mistakes!  |
| 5) Don't be afraid of you freedom!        | Explore! Try out things!                                      |

---

### 💡 Pro Tip

Remember: No one writes code that works first time, or without looking at Stack-Overflow sooner or later.

#### 6.1.4. What is a terminal?

A terminal is simply a fancy window that allows you to access the command-line interface of a computer or server (Figure 6.2).

The command-line itself is how you can work on the computer with just text.

`bash` (bourne again shell) is one of the most popular languages used in the terminal.

#### 6.1.5. Understanding the command prompt

After opening the terminal what you will normally see is a blank screen with a ‘command prompt’, like the one shown above. This typically consists of your username, the device name, a colon, a directory path and ends with a dollar symbol. Like so:

```
<username>@<device_name>:~$
```

The command prompt is **never** involved in any command, it is just there to ensure you know who and where you are. When copying a command you should **NOT** copy the command prompt.

Often times, when looking for commands online, the commands ran will be prefaced with a `$`. This is a stand-in for the command prompt. When adding multi-line commands, it is also common to preface the additional lines with a `>`. When copying such commands it is therefore important to remove these characters from the start of each line (if present).

## 6. Introduction to the Command Line

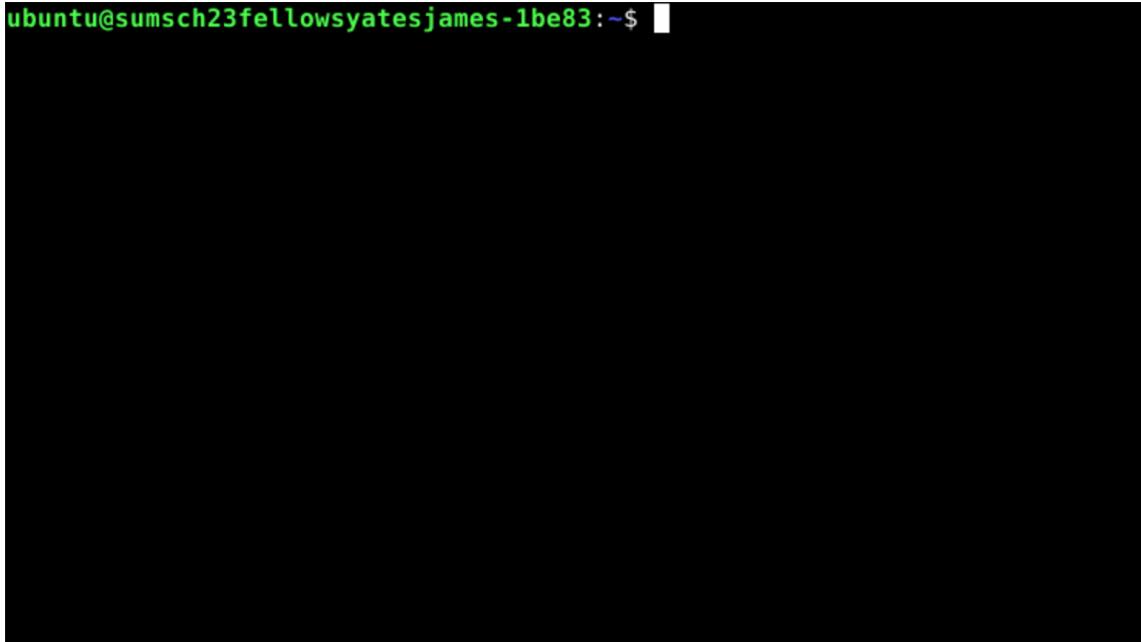


Figure 6.2.: An example command prompt

Finally, in this tutorial, the symbols <> are used to show things that will/should be replaced by another value. For example, in Thisseas' command prompt <username> will be replaced by `lammidis`, as that is his username.

Now back to your prompt: It tells you that you are in the directory `~`. The directory `~`, stands for your **home** directory. Note that this shorthand will point to a different place, depending on the machine and the user.

If you want to know what the shorthand means, (here comes your first command!) you can type in `pwd`, which stands for “print working directory”. Your “working directory” is whichever directory you are currently in.

```
pwd
```

```
/home/ubuntu
```

This prints the entire “filepath” of the directory i.e. the route from the “root” (the deepest directory of the machine), through every subdirectory, leading to your particular working directory.

## *6. Introduction to the Command Line*

### **6.1.6. Absolute vs Relative paths**

Filepaths (a.k.a. “paths”), come in two flavours. Let’s talk a bit about them!

- An **absolute** path will start with the deepest directory in the machine shown as a /. Paths starting with ~ are also absolute paths, since ~ translates to an absolute path of your specific home directory. That is the directory path you see in the output of the `pwd` command you just ran.
- Alternatively a **relative** path always begins from your working directory (i.e. your current directory). Often this type of path will begin with one (./) or two (../) dots followed by a forward slash, **but not always**. In the syntax of relative pathways . means “the current directory” and .. means “the parent directory” (or the ‘one above’).

#### **6.1.6.1. A real life analogy for paths**

You have just arrived to Leipzig for a summer school that is taking place at MPI-EVA. After some questionable navigation, you find yourself at the Bayerische Bahnhof. Tired and disheartened, you decide to ask for help.

You see a friendly-looking metalhead (Figure 6.3), and decide to ask them for directions!



Figure 6.3.: A friendly-looking metalhead.

I’m Happy to help, but I only give directions in **absolute paths!**  
From Leipzig Main Station, you should take Querstraße southward.

## 6. Introduction to the Command Line

Continue straight and take Nürnberg Str. southward until you reach Str. des 18 Oktober.

Finally take Str. des 18 Oktober. moving southeast until you reach MPI-EVA!

### 💡 Absolute paths

The directions above are equivalent to an absolute path, because they will **ALWAYS** take you to MPI-EVA, but you can only apply these directions **if** you start from Leipzig Main Station!

Examples of absolute paths:

```
/home/ubuntu  
/Leipzig_Main_Station/Querstraße/Nürnberg_Str/Str_18_-  
Oktober/Deutscher_Platz/MPI-EVA
```

Not sure how to get back to Leipzig Main Station to apply those directions, you decide to ask someone else for directions...

Lucky for you, a friendly looking local is passing by (Figure 6.4)!

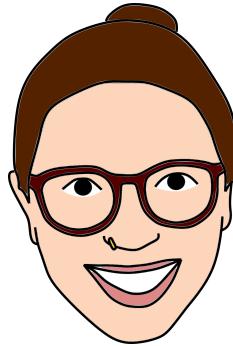


Figure 6.4.: A friendly-looking local.

You're currently on Str. des 18 Oktober. Walk straight that way, past the tram tracks, and you will find Deutscher Platz. You will see MPI-EVA to your right!

### 💡 Relative paths

These directions are equivalent to a relative path! They are easy to follow, but only work when you happen to start at the position you were in when you first got the directions!

Examples of relative paths:

```
./my_directory/my_file.txt
```

## 6. Introduction to the Command Line

```
../Str_18_Oktober/Deutscher_Platz/MPI-EVA
```

### 6.1.7. Basic commands

We will now explore some basic commands that you will use to explore folders and interact with files:

- list directory contents:

```
ls
```

Output should look like:

```
Desktop    Downloads    Pictures    Templates    bin    snap  
Documents  Music       Public      Videos      cache  thinclient_drives
```

 Note

We will use this format to show you commands and their corresponding output in the terminal (if any) for the rest of this chapter.

- make a directory:

```
mkdir barebonesbash
```

- move (or rename) files and directories

```
mv barebonesbash BareBonesBash
```

- change directories

```
cd BareBonesBash
```

- Download (**wget get**) a remote file to your computer

```
wget git.io/Boosted-BBB-meta
```

- copy a file or directory to a new location

## 6. Introduction to the Command Line

```
cp Boosted-BBB-meta Boosted-BBB-meta.tsv
```

- remove (delete) files

```
rm Boosted-BBB-meta
```

- Concatenate file contents to screen

```
cat Boosted-BBB-meta.tsv
```

- See only the **first/last** 10 lines of a file

```
head -n 10 Boosted-BBB-meta.tsv
```

```
tail -n 10 Boosted-BBB-meta.tsv
```

**i** Note

This is because the start of a cat is its head and the end of the cat is its tail (The great humour of computer scientists)

- Look at the contents of a file interactively (**less** than the complete file, press **q** to quit)

```
less Boosted-BBB-meta.tsv
```

- word count the number of lines (-l) in a file

```
wc -l Boosted-BBB-meta.tsv
```

15 Boosted-BBB-meta.tsv

### 6.1.8. Datastreams, piping, and redirects

Each of the commands you learned above is a small program with a very specialised functionality. Programs come in many forms and can be written in various programming languages, but most of them share some features. Specifically, most programs take some data in and spit some data out! Here's how that works, conceptually:

## 6. Introduction to the Command Line

### 6.1.8.1. Datastreams

Computer programs can take in and spit out data from different *streams* (Figure 6.5). By default there are 3 such data streams.

- **stdin** : the **standard input**
- **stdout**: the **standard output**
- **stderr**: the **standard error**



Figure 6.5.: Diagram showing stdin going into the program, and two output streams from the program: stderr and stdout.

#### 💡 Pro Tip

Each programme also has an ‘exit code’, which can tell you if execution completed with/without errors. You will rarely see these in the wild.

Typically, the **stdin** is where the input data comes in.

The **stdout** is the actual output of the command. In some cases this gets printed to the screen, but most often this is the information that needs to be saved in an output file.

The **stderr** is the datastream where errors and warnings go. This gets printed to your terminal to let you know when something is not going according to plan (Figure 6.6)!

```
ubuntu@sumsch23fellowyatesjames-1be83:~/BareBonesBash$ datastreams_demo.sh
This is the standard output (stdout).
This is the standard error (stderr).
```

Figure 6.6.: This program (in the form of a bash script executed in the terminal) takes no input, and prints one line to the stdout and one line to the stderr.

## 6. Introduction to the Command Line

### 6.1.8.2. Piping

A “pipe” (`|`) is a really useful feature of `bash` that lets you chain together multiple commands! When two commands are chained together with a pipe, the `stdout` of the first command becomes the `stdin` of the second (Figure 6.7)! The `stderr` is still printed on your screen, so you can always know when things fail.

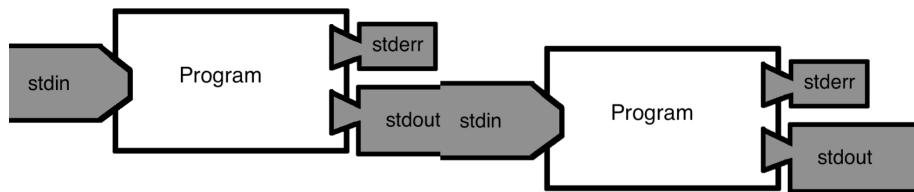


Figure 6.7.: Diagram showing `stdin` going into the program, and two output streams from the program: `stderr` and `stdout`, with the `stdout` becoming the `stdin` of a second program.

Example:

```
head -n 10 Boosted-BBB-meta.tsv | tail -n1
```

```
netsukeJapan      C      Artwork
```

The above command will only show the 10th line of `Boosted-BBB-meta.tsv`. The way it works is that `head` will take the first 10 lines of the file. These lines are then passed on to `tail` which will keep only the last of those lines.

### 6.1.8.3. Redirects

Much like streams of water in the real world, datastreams can be redirected.

This way you can save the `stdout` of a program (or even the `stderr`) into a file for later!

- `stdin` can be redirected with `<`.
  - An arrow pointing TO your program name!
- `stdout` can be redirected with `>`.
  - An arrow pointing AWAY your program name!
- `stderr` can be redirected with `2>`.

## 6. Introduction to the Command Line

- Because it is the secondary output stream.

### 💡 Pro Tip

It is also possible to combine streams, but we won't get into that here.

Example:

```
head -n 10 Boosted-BBB-meta.tsv | tail -n1 > line10.txt
```

This will create a new file called `line10.txt` within your work directory. Using `cat` on this file will BLOW YOUR MIND - (GONE WRONG)!

(Don't forget to like and subscribe!)

```
cat line10.txt
```

netsukeJapan C Artwork

```
ubuntu@sumsch23fellowsyatesjames-1be83:~/BareBonesBash$ datastreams_demo.sh
This is the standard output (stdout).
This is the standard error (stderr).
ubuntu@sumsch23fellowsyatesjames-1be83:~/BareBonesBash$ datastreams_demo.sh >output.txt
This is the standard error (stderr).
ubuntu@sumsch23fellowsyatesjames-1be83:~/BareBonesBash$ cat output.txt
This is the standard output (stdout).
ubuntu@sumsch23fellowsyatesjames-1be83:~/BareBonesBash$ datastreams_demo.sh >output.txt 2>runtime.log
ubuntu@sumsch23fellowsyatesjames-1be83:~/BareBonesBash$ cat runtime.log
This is the standard error (stderr).
```

Figure 6.8.: Here you can see an example of redirecting the output of the `datastreams_demo.sh` program from before. Redirecting the stdout with `>` only prints the stderr to the screen, and saves the stdout into `output.txt`. Additionally, we can redirect the stderr with `2>` into `runtime.log`, and then nothing is printed onto the screen.

### 6.1.9. Help text

You don't always have to google for documentation! Many programs come with in-built help text, or access to online manuals right from your terminal!

- You can get a **one sentence summary** of what a tool does with `whatis`

## 6. Introduction to the Command Line

```
whatis cat
```

```
cat(1) - concatenate files and print on the standard
         output
```

- While `man` gives you **access to online manuals** for each tool (exit with `q`)

```
man cat
```

### 6.1.10. Variables

Variables are a central concept of all programming. In short, a variable is a named container whose contents you can expand at will or change.

You can assign variables (tell the computer what do you want it to contain) with `=` and pull their contents with `$`

The easiest way to see the contents of a variable is using `echo`!

```
echo "This is my home directory: $HOME"
```

This is my home directory: /home/ubuntu

`$HOME` is a variable of the type called **environment variables**, which are set the moment you open your terminal or log into a server, they ensure the system works as intended and should not be change unless you are **very** sure of why.

#### Environment Variables

Environment variables in bash are typically named in all capital letters. It is a good idea to avoid using only capital letters for your variable names, so you avoid accidentally overwriting any environment variables.

But as mentioned, you can store in a variable anything you want, so let's see a few examples:

First let's try to store a number:

```
GreekFood=4          #Here, 'GreekFood' is a number.
echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

Greek food is 4 people who want to know what heaven tastes like.

## 6. Introduction to the Command Line

### Note

The `#` is used to add comments to your code. Comments are annotations that you write in your code to understand what it is doing but that the computer does not run. Very useful for when your future self or another person looks at your code

Now let's store a word ("string"):

```
GreekFood=delicious    #We overwrite that number with a word (i.e. a 'string').  
echo "Everyone says that Greek food is $GreekFood."
```

Everyone says that Greek food is delicious.

You can also store more than a single word (that is still a "string"):

```
GreekFood="Greek wine" #We can overwrite 'GreekFood' again,  
## but when there is a space in our string, we need quotations.  
echo "The only thing better than Greek food is $GreekFood!"
```

The only thing better than Greek food is Greek wine!

Since variables can be reset to whatever you want, you can also store a number again:

```
GreekFood=7 #And, of course, we can overwrite with a number again too.  
echo "I have been to Greece $GreekFood times already this year, for the food and wine!"
```

I have been to Greece 7 times already this year, for the food and wine!

### Overwriting variables

In these examples you have seen how the same variable has been overwritten, this means that you can only access the last content that you stored in the variable. All the previous contents that a variable may have had are inaccessible as soon as the same variable is given a new value.

## 6. Introduction to the Command Line

### 6.1.11. Quotes matter!

In bash, there is a big difference between a single quote ' and a double quote "!

- The contents of single quotes, are passed on as they are.
- Inside double quotes, contents are *interpreted*!

In some cases the difference doesn't matter:

```
echo "I like Greek Food"
echo 'I like Greek Food'
```

```
I like Greek Food
I like Greek Food
```

In other cases it makes all the difference:

```
Arr="Banana"
echo 'Pirates say $Arr'
echo "Minions say $Arr"
```

```
Pirates say $Arr
Minions say Banana
```

#### Why does it make a difference in the second example?

This is because in the second example we are using a variable. We have assigned `Banana` to the variable `$Arr`. As mentioned above, when single (') quotes are used the computer just prints what it receives without caring that `$Arr` is a variable.

In the `echo` with the double ("") quotes we are telling the computer to extract the value from the variable `$Arr` and that is why we see the store value (`Banana`) in the printed output in the terminal.

### 6.1.12. Find

You can also ask your computer where you have put your files, in case you forgot. To do this you can use `find!` The `find` command has the following syntax:

## 6. Introduction to the Command Line

```
## Don't run! Fake example
find /your/folder/ -type f -name 'your_file.txt'
```

- **First** part of the **find** command: *the place to look from*
  - e.g. . to indicate ‘here’
  - Could also use ~/
  - Could use absolute path e.g. /home/james/
- **Second** part of the **find** command: *what type of things to look for?*
  - Use **-type** to define the filetype:
    - \* file
    - \* directory
- **Third** part of the **find** command: *what to look in?*
  - Use **-name** to say ‘look in **names** of things’
- **Finally** after **-name** we give the the ‘strings’ to search for
  - Use wildcards (\*) for maximum laziness!

Now let’s put into practise what you have learnt about **find**.

For that you will download a messy folder from a collaborator, remember to check you are in the BareBonesBash folder!:

```
wget git.io/Boosted-BBB-images -O Boosted-BBB.zip
```

We realise that this is a compressed file, and more precisely is a zip file (extension **.zip**). In order to access to its content we will need to “unzip” it first. For that we can use the command **unzip**:

```
unzip Boosted-BBB.zip
```

We know that our collaborator has shared with us some pictures from animals that we need to use for our research, and according to your collaborator they are marked with **JPG**. We first try to check the contents of the directory to find them quickly.

```
ls Boosted-BBB
```

## 6. Introduction to the Command Line

```
And      Digging      Friday      Leave      Only      Where      Young
Anybody  Everything   Getting     Looking    Ooh       With       Youre
Dancing   Feel        Having     Night     Watch     You
```

Wow, what a mess! How would you retrieve all the files? Thanks to your wonderful teachers you have learnt how to use `find` and can simply run:

```
find Boosted-BBB -type f -name '*JPG*'
```

```
Boosted-BBB/Having/the/time/of/your/life/bubobubo.JPG.MP3.TXT
Boosted-BBB/With/a/bit/of/rock/music/exhibitRoyal.JPG.MP3.TXT
Boosted-BBB/Friday/night/and/the/lights/are/low/fanta.JPG.MP3.TXT
Boosted-BBB/Everything/is/fine/nomnom.JPG.MP3.TXT
Boosted-BBB/Getting/in/the/swing/giacomo.JPG.MP3.TXT
Boosted-BBB/Youre/in/the/mood/for/a/dance/snore.JPG.MP3.TXT
Boosted-BBB/Digging/the/dancing/queen/excited.JPG.MP3.TXT
Boosted-BBB/Anybody/could/be/that/guy/alopochenaegyptiacaArnhem.JPG.MP3.TXT
Boosted-BBB/And/when/you/get/the/chance/stretch.JPG.MP3.TXT
Boosted-BBB/Looking/out/for/angry.JPG.MP3.TXT
Boosted-BBB/Feel/the/beat/from/the/tambourine/oh/yeah/netsukeJapan.JPG.MP3.TXT
Boosted-BBB/Watch/that/scene/licorne.JPG.MP3.TXT
Boosted-BBB/You/can/weimanager.JPG.MP3.TXT
Boosted-BBB/Night/is/young/and/the/musics/high/bydgoszczForest.JPG.MP3.TXT
Boosted-BBB/Ooh/see/that/girl/pompeii.JPG.MP3.TXT
```

After `-name` we have written '`*JPG*`', this tells to `find` to search for any file that contains `JPG` in any part of its name, indicated by the `*`. The `*` are what are known as **wildcards**. To learn more on how to use them, please refer to the more complete material for this tutorial.

Now you have all the paths of the files that you will need!

### 6.1.13. For loops

Until now we have seen how to run single commands on a file. But, what about when you need to **repeat** a command multiple times on a list of things, for example a list of files?

To repeat an action (command) for a set of things (list, e.g. files) one needs to employ the concept of a loop. One of the most commonly used loops, is the **for** loop.

## 6. Introduction to the Command Line

A **for** loop allows us to go through a list of things and perform some actions. Let's see an example:

```
Variable=Yes
for i in Greece Spain Britain; do
    echo "Does $i have lovely food? $Variable"
done
```

```
Does Greece have lovely food? Yes
Does Spain have lovely food? Yes
Does Britain have lovely food? Yes
```

The for loop went through the list **Greece Spain Britain** and printed a statement with each item in the list. What happens if we change the order of the list to **Britain Greece Spain**?

```
Variable=Yes
for i in Britain Greece Spain; do
    echo "Does $i have lovely food? $Variable"
done
```

```
Does Britain have lovely food? Yes
Does Greece have lovely food? Yes
Does Spain have lovely food? Yes
```

We see that changing the order of the list will affect the output, this is because the **for** loop will go through the list in a sequential manner.

We can also add more elements to the list, and the **for** loop will continue until it reaches the end of the list.

### 6.1.14. How to Google like a pro

One of the most important skills you develop when coding and/or using the command line is **how to phrase your questions** so you can get relevant answers out of your search engine.

As Deep Thought put it in the Hitchhiker's Guide to the Galaxy:

Only when you know the question will you know what the answer means.

## 6. Introduction to the Command Line

Here are some quick tips to get you started:

- ALWAYS include the name of the language in your query.
  - **BAD:** “How to cat” (Figure 6.9)
  - **GOOD:** “How to cat bash” (Figure 6.10)
- BROADEN your question!
  - **BAD:** “How to set **X** to 4 in bash?”
  - **GOOD:** “How to set a **variable** to an **integer** in bash?”
- When you are more familiar, use fancy programmer lingo to make google think you know what you are talking about.

### **i** All the cool hackers say:

- **string** and not **text**.
- **float** and not **decimal**.

**Note:** some of these terms can be language specific.

### 6.1.15. (optional) Clean-up

It is extremely important to ALWAYS keep your directories clean from random clutter. This lowers the chances you will get lost in your directories, but also ensures you can stay lazy, since tab completion will not keep finding similarly named files. So let's clean up your home directory by removing all the clutter we downloaded and worked with today. The command below will remove the `~/BareBonesBash` directory **as well as all of its contents**.

```
cd ~      ## We shouldn't delete a directory while we are still in it. (It is possible though)
rm -r ~/BareBonesBash
```

### 💡 Pro Tip

Always be VERY careful when using `rm -r`. Check 3x that the path you are specifying is exactly what you want to delete and nothing more before pressing ENTER!

## 6. Introduction to the Command Line

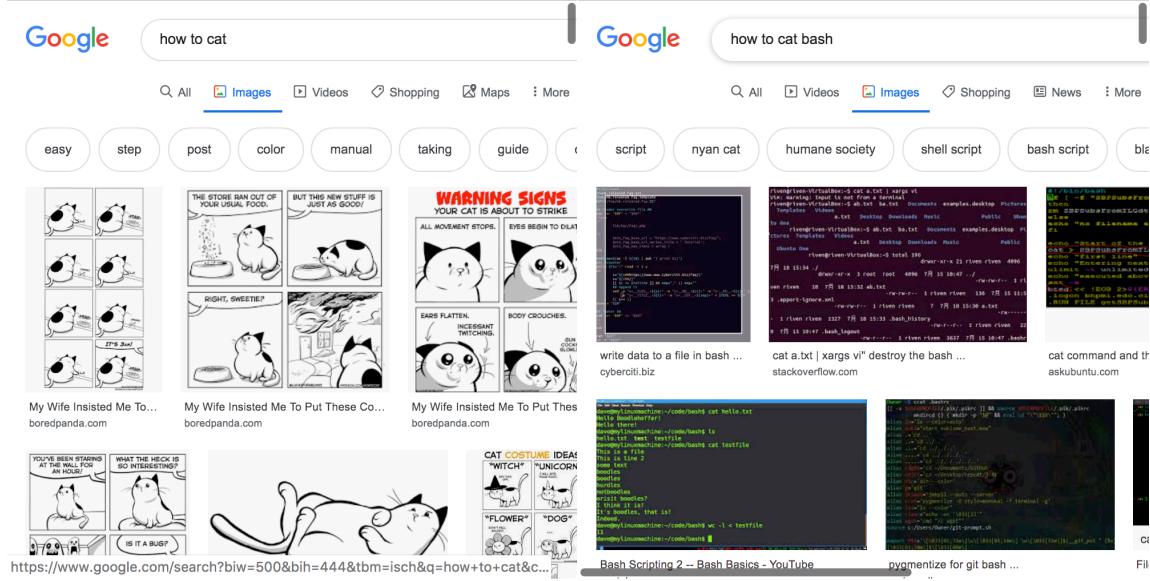


Figure 6.9.: How to cat the wrong way (i.e., don't include the language in your Google search and get loads of cat pictures)

Figure 6.10.: How to cat the BASH way (i.e., include the the language in your Google search and you get lots of terminal pictures! Much better, right? ...right?)

## *6. Introduction to the Command Line*

### **6.1.16. Conclusion**

You should now know the basics of working on the command line, like:

- What a command prompt is
- How to navigate around the filesystem via the command line
- How to view the contents of a file
- How to remove files and directories
- What a datastream is, and how they can be redirected
- How to chain commands together
- What a variable is, how to assign them and how to unpack them
- How to construct a simple for loop
- How to google more efficiently!

If you would like to know more about the magic of bash, you can find more commands as well as and more advanced bash concepts in the BareBonesBash website.

# 7. Introduction to R and the Tidyverse

## Note

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

## Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate r-tidyverse
```

## 7.1. Lecture

Lecture slides and video from the 2022 edition of the summer school. Note that the 2023 version features a different, more general dataset. The up-to-date, written tutorial below is the recommended version.

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

## 7.2. The working environment

### 7.2.1. R, RStudio, the tidyverse and penguins

R [@RCoreTeam2023] is a fully featured programming language, but it excels as an environment for (statistical) data analysis (<https://www.r-project.org>) RStudio [@RstudioTeam] is an integrated development environment (IDE) for R (and other languages) (<https://www.rstudio.com/products/rstudio>)

The tidyverse [@Wickham2019-ot] is a powerful 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 tutorial works with tidyverse ~v2.0. We will learn about the packages `readr`, `tibble`, `ggplot2`, `dplyr`, `magrittr` and `tidyrr`. `forcats` will be briefly mentioned, but `purrr` and `stringr` are left out.

The `palmerpenguins` package [@Horst2020] provides a neat example dataset to learn data exploration and visualization in R (<https://allisonhorst.github.io/palmerpenguins>)

#### Environment preparation code (self-guided)

When you are going through the textbook in your own time, to set up your working environment for this tutorial, please carry out the following steps.

1. Install R and RStudio for your operating system according to one of the many instructions online (e.g. [here](#))
2. Download the .qmd file underlying this webpage [here](#) and copy it to a new directory
3. Open RStudio and create a new project in this directory with `File -> New Project... -> Existing directory`
4. Open this very file from the RStudio project, so that you can easily follow along
5. Run the `Environment preparation code` code to install necessary R package dependencies and prepare the required data used below

## 7. Introduction to R and the Tidyverse

```
# installing necessary R packages
install.packages(c("tidyverse", "palmerpenguins"))

# preparing data
# (at the end of this tutorial
# you will understand this code)
library(magrittr)
set.seed(5678)

peng_prepended <- palmerpenguins::penguins %>%
  dplyr::filter(
    !dplyr::if_any(
      .cols = tidyselect::everything(),
      .fns = is.na
    )
  ) %>%
  tibble::add_column(., id = 1:nrow(.), .before = "species")

peng_prepended %>%
  dplyr::slice_sample(n = 300) %>%
  dplyr::arrange(id) %>%
  dplyr::select(-bill_length_mm, -bill_depth_mm) %>%
  readr::write_csv("penguins.csv")

peng_prepended %>%
  dplyr::slice_sample(n = 300) %>%
  dplyr::arrange(id) %>%
  dplyr::select(id, bill_length_mm, bill_depth_mm) %>%
  readr::write_csv("penguin_bills.csv")
```

### 7.3. Loading data into tibbles

#### 7.3.1. Reading data with `readr`

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

## 7. Introduction to R and the Tidyverse

```
readr::read_csv() # .csv files -> see penguins.csv  
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
```

### 7.3.2. How does the interface of `read_csv` work?

We can learn more about an R function with the `? operator.`

To open a help file for a specific function run `?<function_name>` (e.g. `?readr::read_csv`) in the R console. `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  
  ...  
)
```

### 7.3.3. What does `readr` produce? The tibble!

To read a .csv file (here "penguins.csv") into a variable (here `peng_auto`) run.

```
peng_auto <- readr::read_csv("penguins.csv")
```

```
Rows: 300 Columns: 7  
-- Column specification -----  
Delimiter: ","  
chr (3): species, island, sex
```

## 7. Introduction to R and the Tidyverse

```
dbl (4): id, flipper_length_mm, body_mass_g, year  
  
i Use `spec()` to retrieve the full column specification for this data.  
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

`readr` first prints some information on the number and type of rows and columns it discovered in the file.

It automatically detects column types – but you can also define them manually.

```
peng <- readr::read_csv(  
  "penguins.csv",  
  col_types = "icccddcc" # this string encodes the desired types for each column  
)
```

It then returns an in-memory representation of this data, a `tibble`.

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

### 7.3.4. How to look at a tibble?

Typing the name of any object into the R console will print an overview to the console.

```
peng
```

```
# A tibble: 300 x 7  
  id species island   flipper_length_mm body_mass_g sex   year  
  <int> <chr>   <chr>           <dbl>       <dbl> <chr> <chr>  
1     1 Adelie  Torgersen        181        3750 male  2007  
2     2 Adelie  Torgersen        186        3800 female 2007  
3     4 Adelie  Torgersen        193        3450 female 2007  
4     5 Adelie  Torgersen        190        3650 male  2007  
5     6 Adelie  Torgersen        181        3625 female 2007  
6     7 Adelie  Torgersen        195        4675 male  2007  
7     9 Adelie  Torgersen        191        3800 male  2007  
8    10 Adelie  Torgersen        198        4400 male  2007  
9    11 Adelie  Torgersen        185        3700 female 2007  
10   12 Adelie  Torgersen        195        3450 female 2007  
# i 290 more rows
```

## 7. Introduction to R and the Tidyverse

But there are various other ways to inspect the content of a `tibble`

```
str(peng) # a structural overview of an R object  
summary(peng) # a human-readable summary of an R object  
View(peng) # open RStudio's interactive data browser
```

## 7.4. Plotting data in tibbles

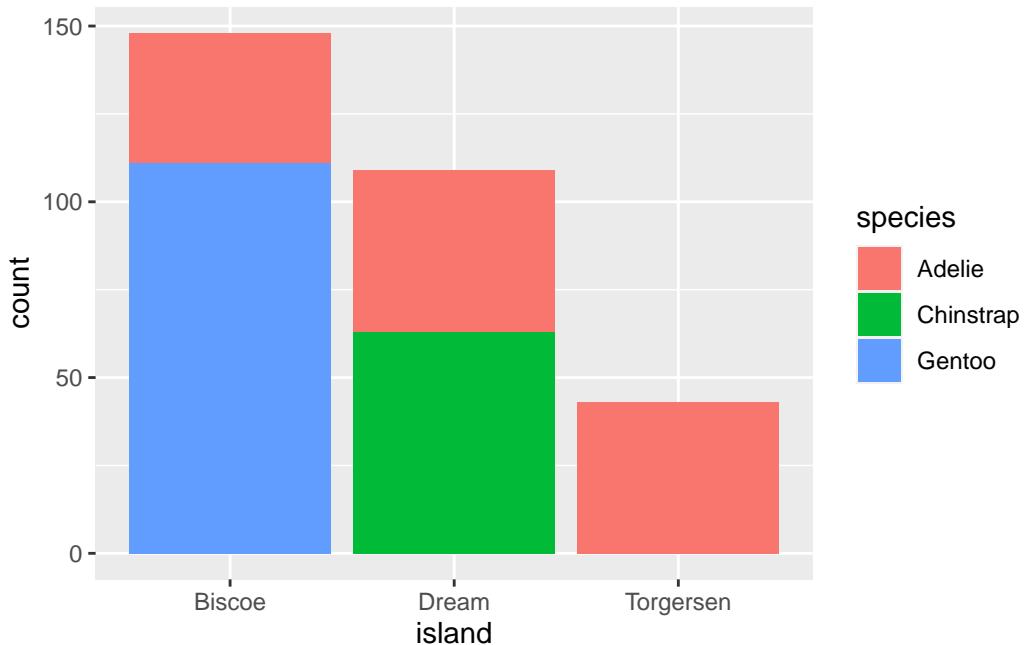
### 7.4.1. ggplot2 and the “grammar of graphics”

To understand and present data, we usually have to visualize it.

`ggplot2` is an R package that offers an unusual, but powerful and logical interface for this task [ @Wickham2016]. 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 = peng # This function can also take the input tibble  
) + # The plot consists of individual 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 = island, # publication_year -> x-axis  
      fill = species # community_type -> fill color  
    )  
  )
```

## 7. Introduction to R and the Tidyverse



A `geom_*` combines data (here `peng`), a geometry (here vertical, stacked bars) and a statistical transformation (here counting the number of penguins per island and species). `ggplot2` features many such geoms: A good overview is provided by this cheatsheet: <https://rstudio.github.io/cheatsheets/html/data-visualization.html>.

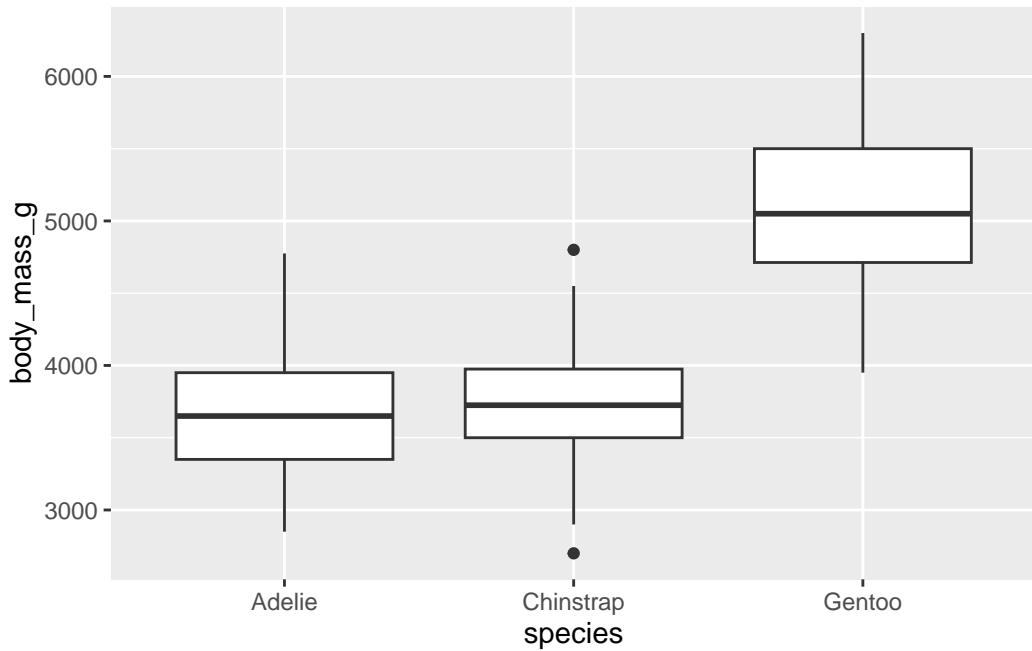
Beyond geoms, a ggplot plot can be further specified with (among others) `scales`, `facets` and `themes`.

### 7.4.2. scales control the behaviour of visual elements

Here is another plot to demonstrate this: Boxplots of penguin weight per species.

```
ggplot(peng) +  
  geom_boxplot(aes(x = species, y = body_mass_g))
```

## 7. Introduction to R and the Tidyverse



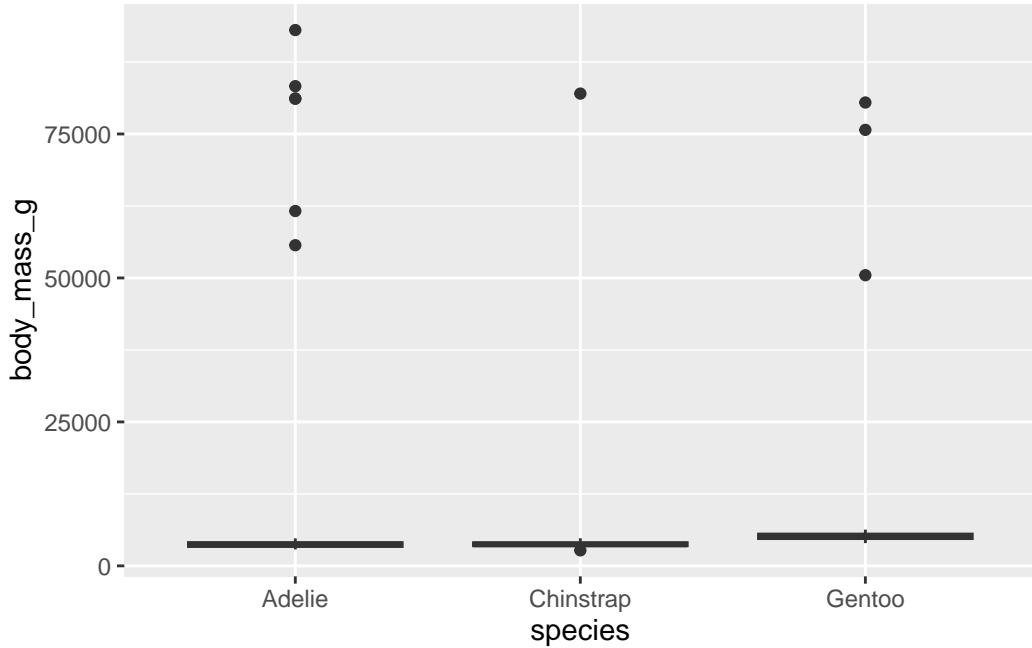
Let's assume we had some extreme outliers in this dataset. To simulate this, we replace some random weights with extreme values.

```
set.seed(1234) # we set a seed for reproducible randomness
peng_out <- peng
peng_out$body_mass_g[sample(1:nrow(peng_out), 10)] <- 50000 + 50000 * runif(10)
```

Now we plot the dataset with these “outliers”.

```
ggplot(peng_out) +
  geom_boxplot(aes(x = species, y = body_mass_g))
```

## 7. Introduction to R and the Tidyverse

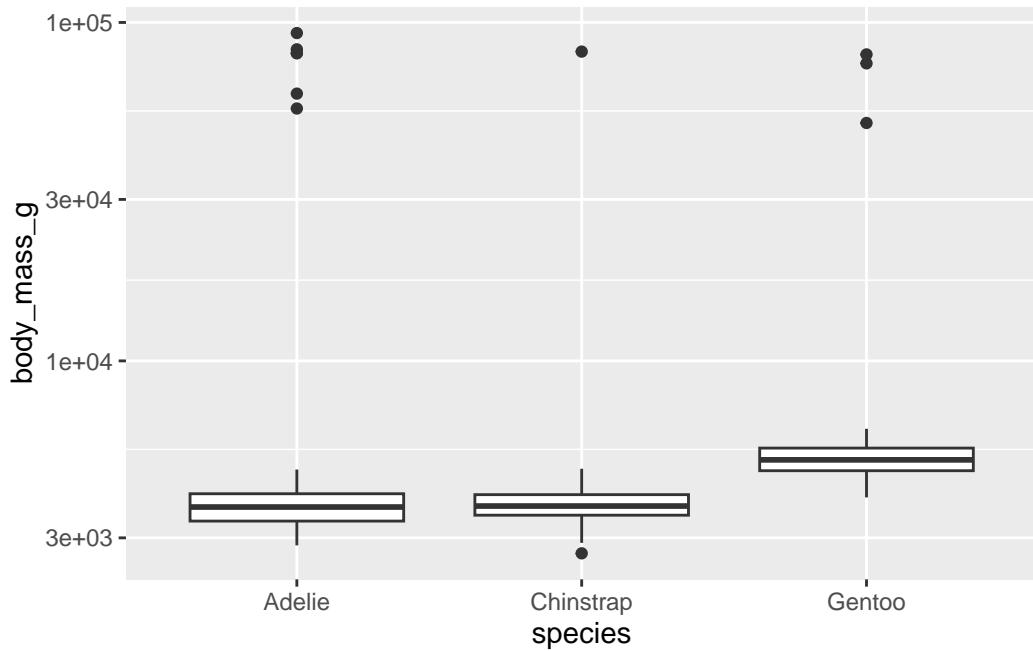


This is not well readable, because the extreme outliers dictate the scale of the y-axis. A 50+kg penguin is a scary thought and we would probably remove these outliers, but let's assume they are valid observation we want to include in the plot.

To mitigate this issue we can change the **scale** of different visual elements - e.g. the y-axis.

```
ggplot(peng_out) +  
  geom_boxplot(aes(x = species, y = body_mass_g)) +  
  scale_y_log10() # adding the log-scale improves readability
```

## 7. Introduction to R and the Tidyverse



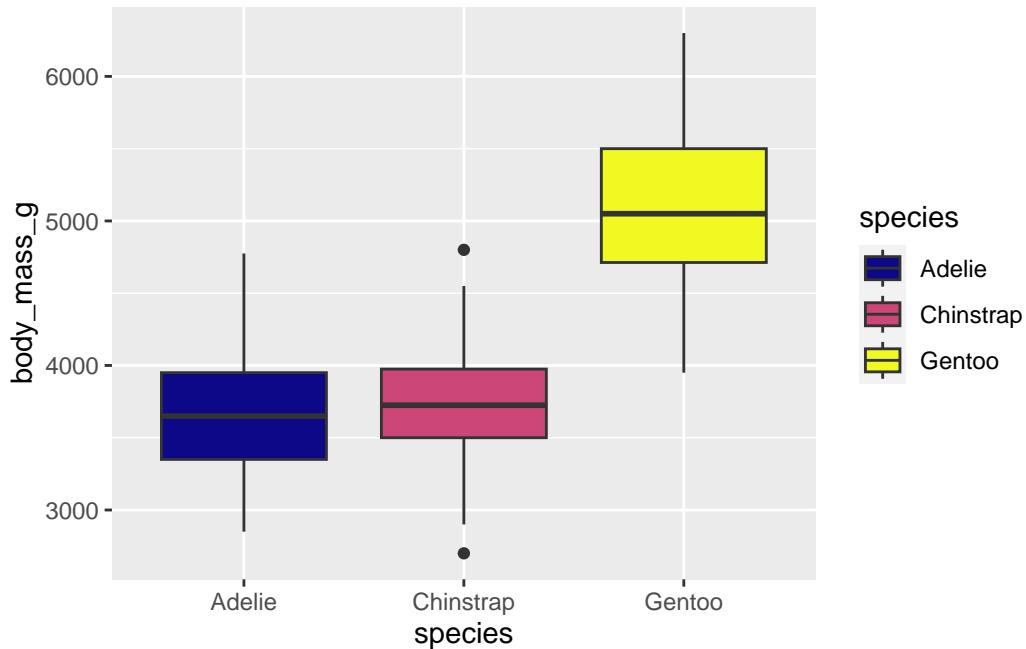
We will now go back to the normal dataset without the artificial outliers.

### 7.4.3. Colour scales

(Fill) colour is a visual element of a plot and its scaling can be adjusted as well.

```
ggplot(peng) +  
  geom_boxplot(aes(x = species, y = body_mass_g, fill = species)) +  
  scale_fill_viridis_d(option = "C")
```

## 7. Introduction to R and the Tidyverse



We use the `scale_*` function to select one of the visually appealing (and robust to colourblindness) viridis colour palettes (<https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html>).

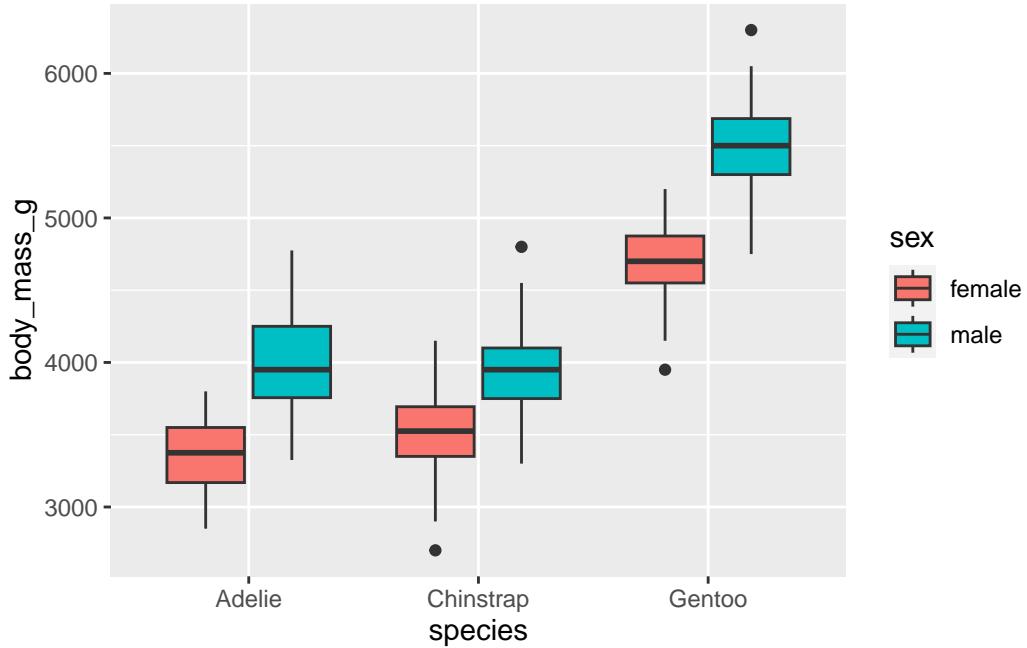
### 7.4.4. More variables! Defining plot matrices via facets

In the previous example we didn't add additional information with the fill colour, as the plot already distinguished by species on the x-axis.

We can instead use colour to encode more information, for example by mapping the variable `sex` to it.

```
ggplot(peng) +  
  geom_boxplot(aes(x = species, y = body_mass_g, fill = sex))
```

## 7. Introduction to R and the Tidyverse

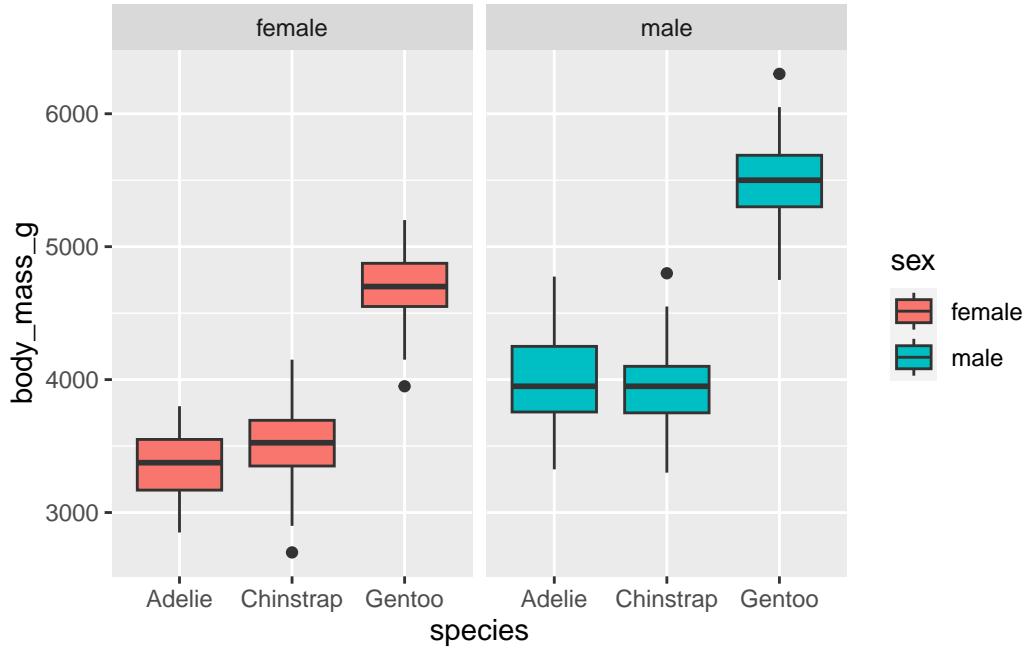


Another way to visualize more variables in one plot is to split the plot by categories into **facets**, so sub-plots per category.

Here we split by sex, which is already mapped to the fill colour.

```
ggplot(peng) +  
  geom_boxplot(aes(x = species, y = body_mass_g, fill = sex)) +  
  facet_wrap(~sex)
```

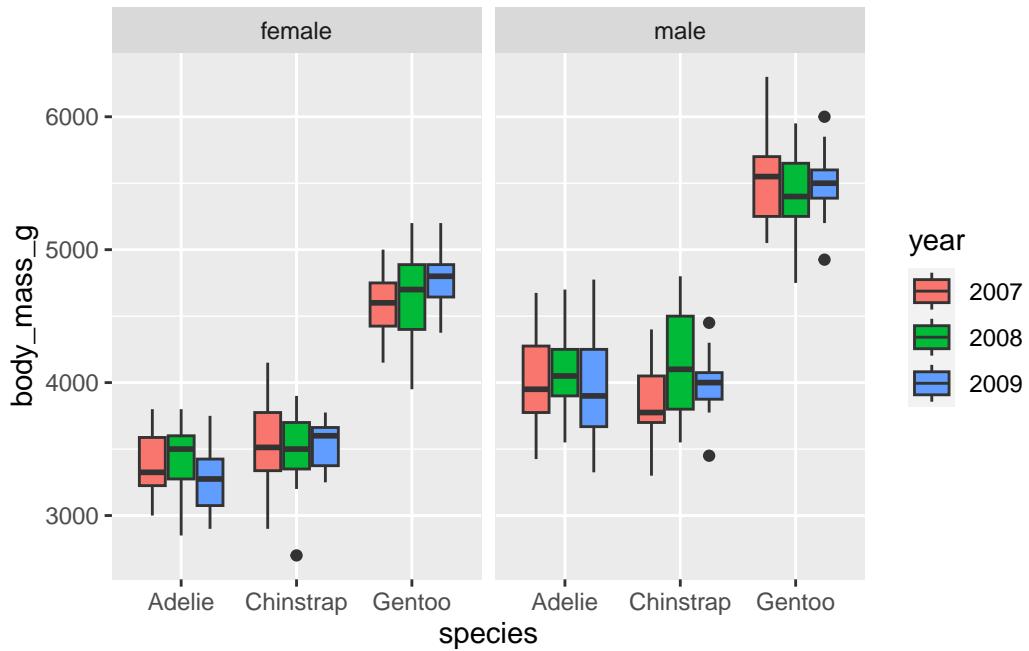
## 7. Introduction to R and the Tidyverse



The fill colour is therefore free again to show yet another variable, for example the year a given penguin was examined.

```
ggplot(peng) +  
  geom_boxplot(aes(x = species, y = body_mass_g, fill = year)) +  
  facet_wrap(~sex)
```

## 7. Introduction to R and the Tidyverse



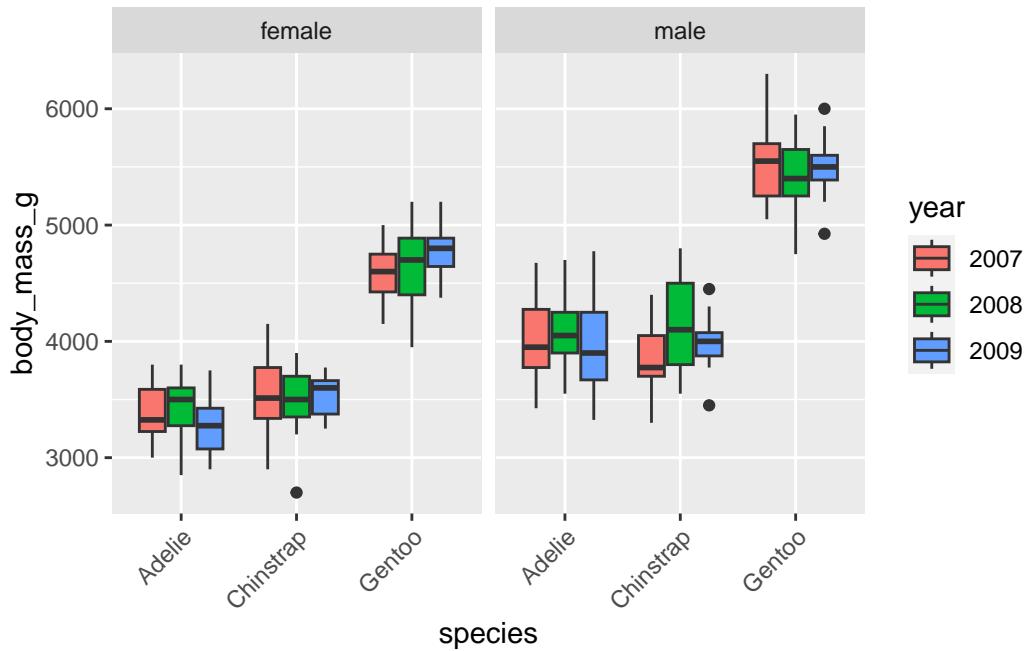
### 7.4.5. Setting purely aesthetic settings with theme

Aesthetic changes can be applied as part of the `theme`, which allows for very detailed configuration (see `?theme`).

Here we rotate the x-axis labels by 45°, which often helps to resolve overplotting.

```
ggplot(peng) +
  geom_boxplot(aes(x = species, y = body_mass_g, fill = year)) +
  facet_wrap(~sex) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```

## 7. Introduction to R and the Tidyverse



### 7.4.6. Ordering elements in a plot with factors

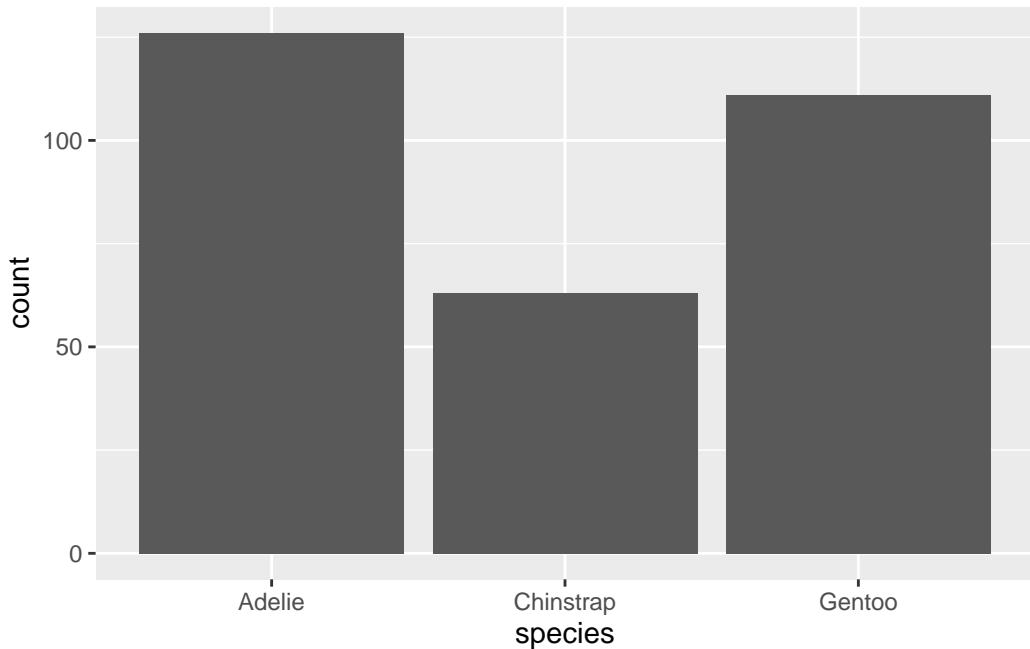
R supports defining ordinal data with **factors**. This can be used to set the order of elements in a plot, e.g. the order of bars in a bar chart.

We do not cover **factors** beyond the following example here, although the tidyverse includes a package (**forcats**) specifically for that purpose.

Elements based on **character** columns are generally ordered alphabetically.

```
ggplot(peng) +  
  geom_bar(aes(x = species)) # bars are alphabetically ordered
```

## 7. Introduction to R and the Tidyverse



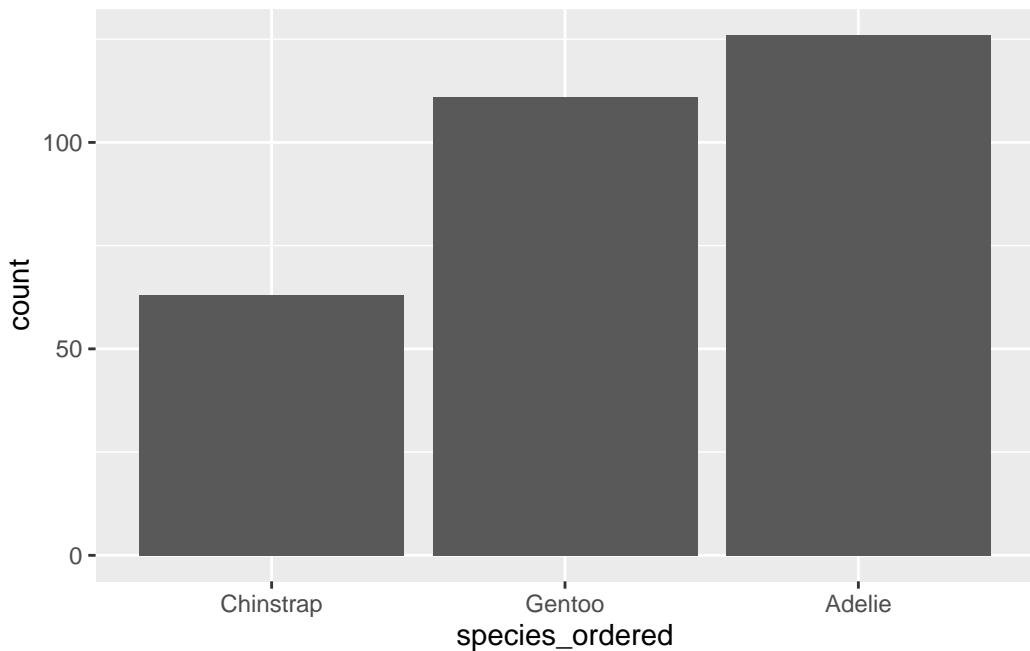
With `forcats::fct_reorder` we can transform an input vector to a `factor`, ordered by a summary statistic (even based on another vector).

```
peng2 <- peng
peng2$species_ordered <- forcats::fct_reorder(
  peng2$species,
  peng2$species, length
)
```

With this change, the plot will be ordered according to the intrinsic order defined for `species_ordered`.

```
ggplot(peng2) +
  geom_bar(aes(x = species_ordered)) # bars are ordered by size
```

## 7. Introduction to R and the Tidyverse



### 7.4.7. Exercise

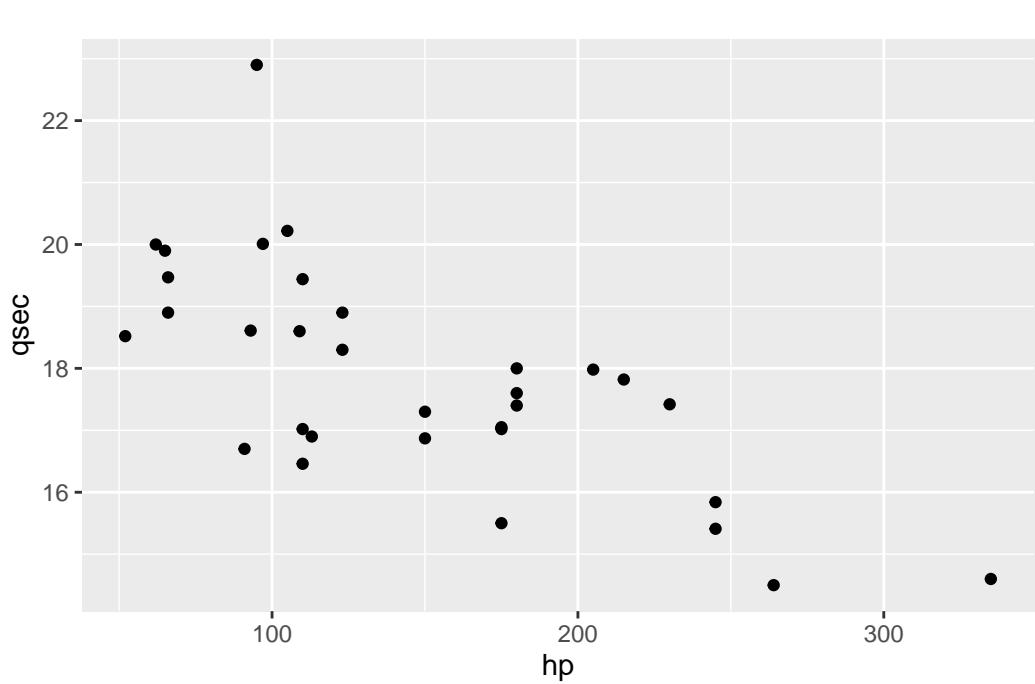
1. Look at the `mtcars` dataset and read up on the meaning of its variables with the help operator `?.`. `mtcars` is a test dataset integrated in R and can be accessed by typing `mtcars` in the console
2. Visualize the relationship between *Gross horsepower* and *1/4 mile time*
3. Integrate the *Number of cylinders* into your plot as an additional variable

#### Possible solutions

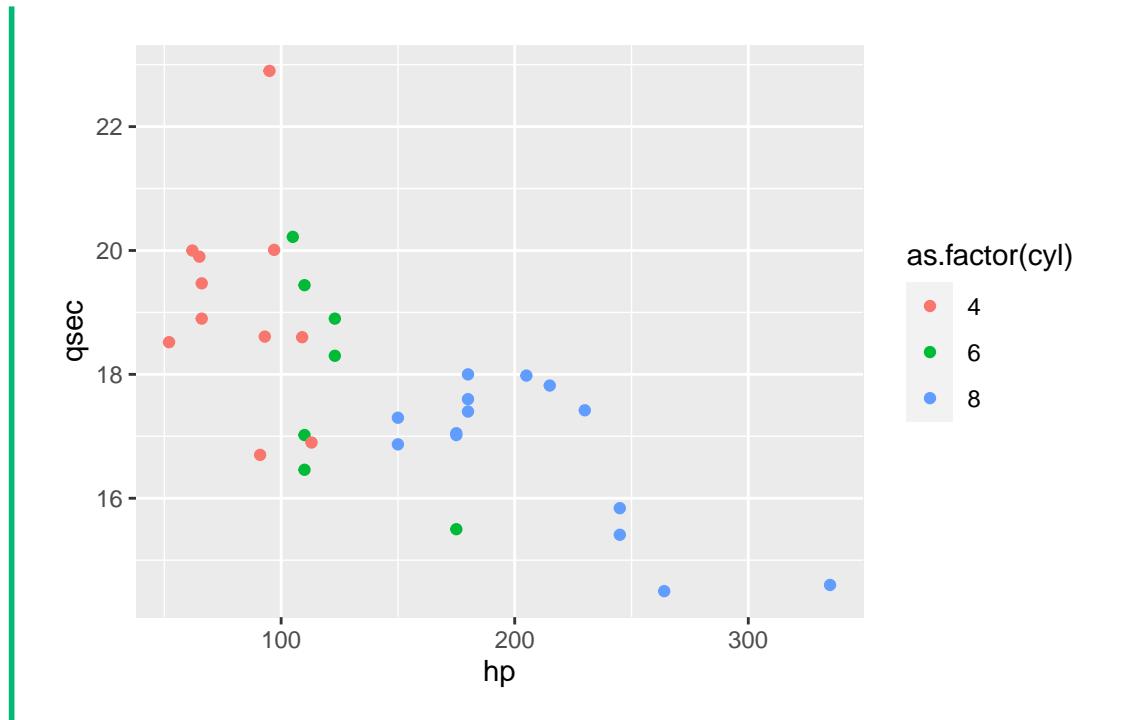
```
?mtcars
```

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

## 7. Introduction to R and the Tidyverse



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



## 7.5. Conditional queries on tibbles

### 7.5.1. Selecting columns and filtering rows with `select` and `filter`

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

```
dplyr::select(peng, id, flipper_length_mm) # reduce tibble to two columns
```

```
# A tibble: 300 x 2
  id    flipper_length_mm
  <int>        <dbl>
1     1            181
2     2            186
3     4            193
4     5            190
5     6            181
# i 295 more rows
```

## 7. Introduction to R and the Tidyverse

```
dplyr::select(peng, -island, -flipper_length_mm) # remove two columns
```

```
# A tibble: 300 x 5
  id species body_mass_g sex     year
  <int> <chr>      <dbl> <chr>   <chr>
1     1 Adelie       3750 male    2007
2     2 Adelie       3800 female  2007
3     4 Adelie       3450 female  2007
4     5 Adelie       3650 male    2007
5     6 Adelie       3625 female  2007
# i 295 more rows
```

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

```
dplyr::filter(peng, year == 2007) # penguins examined in 2007
```

```
# A tibble: 93 x 7
  id species island    flipper_length_mm body_mass_g sex     year
  <int> <chr>   <chr>            <dbl>      <dbl> <chr>   <chr>
1     1 Adelie  Torgersen        181       3750 male    2007
2     2 Adelie  Torgersen        186       3800 female  2007
3     4 Adelie  Torgersen        193       3450 female  2007
4     5 Adelie  Torgersen        190       3650 male    2007
5     6 Adelie  Torgersen        181       3625 female  2007
# i 88 more rows
```

```
dplyr::filter(peng, year == 2007 |
  year == 2009) # penguins examined in 2007 OR 2009
```

```
# A tibble: 198 x 7
  id species island    flipper_length_mm body_mass_g sex     year
  <int> <chr>   <chr>            <dbl>      <dbl> <chr>   <chr>
1     1 Adelie  Torgersen        181       3750 male    2007
2     2 Adelie  Torgersen        186       3800 female  2007
3     4 Adelie  Torgersen        193       3450 female  2007
4     5 Adelie  Torgersen        190       3650 male    2007
5     6 Adelie  Torgersen        181       3625 female  2007
# i 193 more rows
```

## 7. Introduction to R and the Tidyverse

```
dplyr::filter(  
  peng, # an alternative way to express  
  year %in% c(2007, 2009)  
) # OR with the match operator "%in%"  
  
# A tibble: 198 x 7  
  id species island   flipper_length_mm body_mass_g sex  year  
  <int> <chr>   <chr>             <dbl>      <dbl> <chr> <chr>  
1     1 Adelie  Torgersen        181       3750 male  2007  
2     2 Adelie  Torgersen        186       3800 female 2007  
3     4 Adelie  Torgersen        193       3450 female 2007  
4     5 Adelie  Torgersen        190       3650 male  2007  
5     6 Adelie  Torgersen        181       3625 female 2007  
# i 193 more rows  
  
dplyr::filter(peng, species == "Adelie" &  
  body_mass_g >= 4000) # Adelie penguins heavier than 4kg  
  
# A tibble: 29 x 7  
  id species island   flipper_length_mm body_mass_g sex  year  
  <int> <chr>   <chr>             <dbl>      <dbl> <chr> <chr>  
1     7 Adelie  Torgersen        195       4675 male  2007  
2    10 Adelie  Torgersen        198       4400 male  2007  
3    13 Adelie  Torgersen        197       4500 male  2007  
4    31 Adelie  Dream           196       4150 male  2007  
5    39 Adelie  Dream           196       4400 male  2007  
# i 24 more rows
```

### 7.5.2. Chaining functions together with the pipe %>%

A core feature of the tidyverse is the pipe `%>%` in the `magrittr` package. This ‘infix’ operator allows to chain data and operations for concise and clear data analysis syntax.

```
library(magrittr)  
peng %>% dplyr::filter(year == 2007)
```

```
# A tibble: 93 x 7
```

## 7. Introduction to R and the Tidyverse

```
  id species island    flipper_length_mm body_mass_g sex      year
<int> <chr>   <chr>                  <dbl>       <dbl> <chr>   <chr>
1     1 Adelie  Torgersen           181        3750 male    2007
2     2 Adelie  Torgersen           186        3800 female  2007
3     4 Adelie  Torgersen           193        3450 female  2007
4     5 Adelie  Torgersen           190        3650 male    2007
5     6 Adelie  Torgersen           181        3625 female  2007
# i 88 more rows
```

It forwards the LHS (left-hand side) as the first argument of the function appearing on the RHS (right-hand side), which enables sequences of functions (“tidyverse style”).

```
peng %>%
  dplyr::select(id, species, body_mass_g) %>%
  dplyr::filter(species == "Adelie" & body_mass_g >= 4000) %>%
  nrow() # count the resulting rows
```

```
[1] 29
```

`magrittr` also offers some more operators, among which the extraction `%$%` is particularly useful to easily extract individual variables.

```
peng %>%
  dplyr::filter(island == "Biscoe") %$%
  species %>% # extract the species column as a vector
  unique() # get the unique elements of said vector
```

```
[1] "Adelie" "Gentoo"
```

Here we already use the base R summary function `unique`.

### 7.5.3. Summary statistics in base R

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

## 7. Introduction to R and the Tidyverse

```
chinstraps_weights <- peng %>%
  dplyr::filter(species == "Chinstrap") %$%
  body_mass_g

length(chinstraps_weights) # length/size of a vector

[1] 63

unique(chinstraps_weights) # unique elements of a vector

[1] 3500 3900 3650 3525 3725 3950 3250 3750 4150 3700 3800 3775 4050 3300 3450
[16] 4400 3400 2900 4550 3200 4300 3350 4100 3600 3850 4800 2700 4500 3550 3675
[31] 4450 3325 4000

min(chinstraps_weights) # minimum

[1] 2700

max(chinstraps_weights) # maximum

[1] 4800

mean(chinstraps_weights) # mean

[1] 3751.19

median(chinstraps_weights) # median

[1] 3725

var(chinstraps_weights) # variance

[1] 153032.8
```

## 7. Introduction to R and the Tidyverse

```
sd(chinstraps_weights) # standard deviation  
  
[1] 391.1941  
  
quantile(chinstraps_weights, probs = 0.75) # quantiles for the given probabilities  
  
75%  
3975
```

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

### 7.5.4. Group-wise summaries with `group_by` and `summarise`

These summary statistics are particular useful when applied to conditional subsets of a dataset.

`dplyr` allows such summary operations with a combination of the functions `group_by` and `summarise`, where the former tags a `tibble` with categories based on its variables and the latter reduces it to these groups while simultaneously creating new columns.

```
peng %>%  
  dplyr::group_by(species) %>% # group the tibble by the material column  
  dplyr::summarise(  
    min_weight = min(body_mass_g), # new col: min weight for each group  
    median_weight = median(body_mass_g), # new col: median weight for each group  
    max_weight = max(body_mass_g) # new col: max weight for each group  
  )  
  
# A tibble: 3 x 4  
  species   min_weight median_weight max_weight  
  <chr>        <dbl>         <dbl>        <dbl>  
1 Adelie      2850          3650        4775  
2 Chinstrap    2700          3725        4800  
3 Gentoo      3950          5050        6300
```

Grouping can also be applied across multiple columns at once.

## 7. Introduction to R and the Tidyverse

```
peng %>%
  dplyr::group_by(species, year) %>% # group by species and year
  dplyr::summarise(
    n = dplyr::n(), # a new column: number of penguins for each group
    .groups = "drop" # drop the grouping after this summary operation
  )

# A tibble: 9 x 3
  species   year     n
  <chr>     <chr> <int>
1 Adelie    2007    38
2 Adelie    2008    45
3 Adelie    2009    43
4 Chinstrap 2007    23
5 Chinstrap 2008    18
# i 4 more rows
```

If we group by more than one variable, then `summarise` will not entirely remove the group tagging when generating the result dataset. We can force this with `.groups = "drop"` to avoid undesired behaviour with this dataset later on.

### 7.5.5. Sorting and slicing tibbles with `arrange` and `slice`

`dplyr` allows to `arrange` tibbles by one or multiple columns.

```
peng %>% dplyr::arrange(sex) # sort by sex

# A tibble: 300 x 7
  id species island   flipper_length_mm body_mass_g sex      year
  <int> <chr>   <chr>           <dbl>       <dbl> <chr>   <chr>
1     2 Adelie  Torgersen        186       3800 female  2007
2     4 Adelie  Torgersen        193       3450 female  2007
3     6 Adelie  Torgersen        181       3625 female  2007
4    11 Adelie  Torgersen        185       3700 female  2007
5    12 Adelie  Torgersen        195       3450 female  2007
# i 295 more rows
```

## 7. Introduction to R and the Tidyverse

```
peng %>% dplyr::arrange(sex, body_mass_g) # sort by sex and weight
```

```
# A tibble: 300 x 7
  id species island   flipper_length_mm body_mass_g sex   year
  <int> <chr>   <chr>           <dbl>      <dbl> <chr> <chr>
1 304 Chinstrap Dream          192        2700 female 2008
2 53 Adelie    Biscoe         181        2850 female 2008
3 59 Adelie    Biscoe         184        2850 female 2008
4 49 Adelie    Biscoe         187        2900 female 2008
5 111 Adelie   Torgersen     188        2900 female 2009
# i 295 more rows
```

```
peng %>% dplyr::arrange(dplyr::desc(body_mass_g)) # sort descending
```

```
# A tibble: 300 x 7
  id species island   flipper_length_mm body_mass_g sex   year
  <int> <chr>   <chr>           <dbl>      <dbl> <chr> <chr>
1 164 Gentoo  Biscoe          221        6300 male   2007
2 179 Gentoo  Biscoe          230        6050 male   2007
3 260 Gentoo  Biscoe          222        6000 male   2009
4 224 Gentoo  Biscoe          223        5950 male   2008
5 160 Gentoo  Biscoe          213        5850 male   2007
# i 295 more rows
```

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

Here we extract the heaviest individuals per species.

```
peng %>%
  dplyr::group_by(species) %>% # group by species
  dplyr::arrange(dplyr::desc(body_mass_g)) %>% # sort by weight within (!) groups
  dplyr::slice_head(n = 3) %>% # keep the first three penguins per group
  dplyr::ungroup() # remove the still lingering grouping

# A tibble: 9 x 7
  id species island   flipper_length_mm body_mass_g sex   year
  <int> <chr>   <chr>           <dbl>      <dbl> <chr> <chr>
```

## 7. Introduction to R and the Tidyverse

```
1 104 Adelie Biscoe          197      4775 male 2009
2 96 Adelie Biscoe          203      4725 male 2009
3 76 Adelie Torgersen      196      4700 male 2008
4 303 Chinstrap Dream      210      4800 male 2008
5 295 Chinstrap Dream      205      4550 male 2008
# i 4 more rows
```

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

```
peng %>% dplyr::slice_sample(n = 10)
```

```
# A tibble: 10 x 7
  id species island flipper_length_mm body_mass_g sex     year
  <int> <chr>   <chr>           <dbl>       <dbl> <chr>   <chr>
1    47 Adelie  Biscoe            190        3450 female 2008
2   239 Gentoo Biscoe            214        4850 female 2009
3   221 Gentoo Biscoe            209        4600 female 2008
4   104 Adelie  Biscoe            197      4775 male   2009
5   138 Adelie  Dream             190        3725 male   2009
# i 5 more rows
```

### 7.5.6. Exercise

For this exercise we once more go back to the `mtcars` dataset. See `?mtcars` for details

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`)

#### Possible solutions

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

```
[1] 12
```

## 7. Introduction to R and the Tidyverse

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

# A tibble: 3 x 2
  cyl   qsec_mean
  <dbl>     <dbl>
1     4      19.1
2     6      18.0
3     8      16.8

mtcars2 <- tibble::rownames_to_column(mtcars, var = "car")
mtcars2 %>%
  dplyr::group_by(am) %>%
  dplyr::arrange(mpg) %>%
  dplyr::slice_head() %$%
  car

[1] "Cadillac Fleetwood" "Maserati Bora"
```

## 7.6. Transforming and manipulating tibbles

### 7.6.1. Renaming and reordering columns with `rename` and `relocate`

Columns in tibbles can be renamed with `dplyr::rename`.

```
peng %>% dplyr::rename(penguin_name = id) # rename a column

# A tibble: 300 x 7
  penguin_name species island   flipper_length_mm body_mass_g sex   year
  <int> <chr>   <chr>           <dbl>        <dbl> <chr> <chr>
1          1 Adelie  Torgersen       181         3750 male  2007
2          2 Adelie  Torgersen       186         3800 female 2007
3          4 Adelie  Torgersen       193         3450 female 2007
4          5 Adelie  Torgersen       190         3650 male  2007
```

## 7. Introduction to R and the Tidyverse

```
5           6 Adelie Torgersen      181      3625 female 2007
# i 295 more rows
```

And with `dplyr::relocate` they can be reordered.

```
peng %>% dplyr::relocate(year, .before = species) # reorder columns
```

```
# A tibble: 300 x 7
  id year  species island    flipper_length_mm body_mass_g sex
  <int> <chr> <chr>     <dbl>        <dbl> <chr>
1     1 2007  Adelie Torgersen       181      3750 male 
2     2 2007  Adelie Torgersen       186      3800 female
3     4 2007  Adelie Torgersen       193      3450 female
4     5 2007  Adelie Torgersen       190      3650 male 
5     6 2007  Adelie Torgersen       181      3625 female
# i 295 more rows
```

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

A common application of data manipulation is adding new, derived columns, that combine or modify the information in the already available columns. `dplyr` offers this core feature with the `mutate` function.

```
peng %>%
  dplyr::mutate( # add a column that
    kg = body_mass_g / 1000 # manipulates an existing column
  )
```

```
# A tibble: 300 x 8
  id species island    flipper_length_mm body_mass_g sex     year      kg
  <int> <chr> <chr>        <dbl>        <dbl> <chr> <chr> <dbl>
1     1 Adelie Torgersen       181      3750 male   2007    3.75
2     2 Adelie Torgersen       186      3800 female 2007    3.8 
3     4 Adelie Torgersen       193      3450 female 2007    3.45
4     5 Adelie Torgersen       190      3650 male   2007    3.65
5     6 Adelie Torgersen       181      3625 female 2007    3.62
# i 295 more rows
```

## 7. Introduction to R and the Tidyverse

`dplyr::transmute` has the same interface as `dplyr::mutate`, but it removes all columns except for the newly created ones.

```
peng %>%
  dplyr::transmute(
    id = paste("Penguin Nr.", id), # overwrite this column
    flipper_length_mm             # select this column
)
```

```
# A tibble: 300 x 2
  id          flipper_length_mm
  <chr>           <dbl>
1 Penguin Nr. 1     181
2 Penguin Nr. 2     186
3 Penguin Nr. 4     193
4 Penguin Nr. 5     190
5 Penguin Nr. 6     181
# i 295 more rows
```

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

```
peng %>% tibble::add_column(
  flipper_length_cm = .\$flipper_length_mm/10, # not the . representing the LHS of the pipe
  .after = "flipper_length_mm"
)
```

```
# A tibble: 300 x 8
  id species island   flipper_length_mm flipper_length_cm body_mass_g sex
  <int> <chr>   <chr>           <dbl>           <dbl>           <dbl> <chr>
1     1 Adelie  Torgersen       181         18.1        3750 male
2     2 Adelie  Torgersen       186         18.6        3800 female
3     4 Adelie  Torgersen       193         19.3        3450 female
4     5 Adelie  Torgersen       190          19          3650 male
5     6 Adelie  Torgersen       181         18.1        3625 female
# i 295 more rows
# i 1 more variable: year <chr>
```

## 7. Introduction to R and the Tidyverse

`dplyr::mutate` can also be combined with `dplyr::group_by` (instead of `dplyr::summarise`) to add information on a group level. This is relevant, when a value for an individual entity should be put into context of a group-wise summary statistic.

```
peng %>%
  dplyr::group_by(species, sex, year) %>%
  dplyr::mutate(
    mean_weight = mean(body_mass_g, na.rm = T),
    relation_to_mean = body_mass_g / mean_weight
  ) %>%
  dplyr::ungroup() %>%
  dplyr::select(id, species, sex, year, relation_to_mean) %>%
  # mutate does not remove rows, unlike summarise, so we use select
  dplyr::arrange(dplyr::desc(relation_to_mean))
```

```
# A tibble: 300 x 5
  id species   sex   year relation_to_mean
  <int> <chr>     <chr> <chr>        <dbl>
1 104 Adelie male  2009      1.21
2 96 Adelie male  2009      1.20
3 274 Chinstrap female 2007      1.17
4 7 Adelie male  2007      1.17
5 303 Chinstrap male  2008      1.16
# i 295 more rows
```

### 7.6.2. Conditional operations with `ifelse`, `case_when` and `case_match`

`ifelse` allows to implement conditional `mutate` operations, that consider information from other columns.

```
peng %>% dplyr::mutate(
  weight = ifelse(
    test = body_mass_g >= 4200, # is weight below or above mean weight?
    yes = "above mean",
    no = "below mean"
  )
)
```

```
# A tibble: 300 x 8
```

## 7. Introduction to R and the Tidyverse

```

      id species island    flipper_length_mm body_mass_g sex     year   weight
<int> <chr>   <chr>                  <dbl>       <dbl> <chr> <chr> <chr>
1     1 Adelie  Torgersen           181        3750 male  2007 below mean
2     2 Adelie  Torgersen           186        3800 female 2007 below mean
3     4 Adelie  Torgersen           193        3450 female 2007 below mean
4     5 Adelie  Torgersen           190        3650 male  2007 below mean
5     6 Adelie  Torgersen           181        3625 female 2007 below mean
# i 295 more rows

```

`ifelse` gets cumbersome easily. `dplyr::case_when` is more readable and scales much better for this application.

```

peng %>% dplyr::mutate(
  weight = dplyr::case_when(
    body_mass_g >= 4200 ~ "above mean", # the number of conditions is arbitrary
    body_mass_g < 4200 ~ "below mean",
    TRUE            ~ "unknown"      # TRUE catches all remaining cases
  )
)

```

```

# A tibble: 300 x 8
      id species island    flipper_length_mm body_mass_g sex     year   weight
<int> <chr>   <chr>                  <dbl>       <dbl> <chr> <chr> <chr>
1     1 Adelie  Torgersen           181        3750 male  2007 below mean
2     2 Adelie  Torgersen           186        3800 female 2007 below mean
3     4 Adelie  Torgersen           193        3450 female 2007 below mean
4     5 Adelie  Torgersen           190        3650 male  2007 below mean
5     6 Adelie  Torgersen           181        3625 female 2007 below mean
# i 295 more rows

```

`dplyr::case_match` is similar, but unlike `dplyr::case_when` it does not check logical expressions, but matches by value.

```

peng %>%
  dplyr::mutate(
    island_rating = dplyr::case_match(
      island,
      "Torgersen" ~ "My favourite island",
      "Biscoe"    ~ "Overrated tourist trap",
      ...
    )
)

```

## 7. Introduction to R and the Tidyverse

```

"Dream" ~ "Lost my wallet there. 4/10"
)
) %>%
# here we use group_by+summarise only to show the result
dplyr::group_by(island, island_rating) %>%
dplyr::summarise(.groups = "drop")

```

```

# A tibble: 3 x 2
island    island_rating
<chr>      <chr>
1 Biscoe   Overrated tourist trap
2 Dream    Lost my wallet there. 4/10
3 Torgersen My favourite island

```

### 7.6.3. Switching between long and wide data with pivot\_longer and pivot\_wider

To simplify certain analysis or plotting operations data often has to be transformed from a **wide** to a **long** format or vice versa (Figure 7.1).

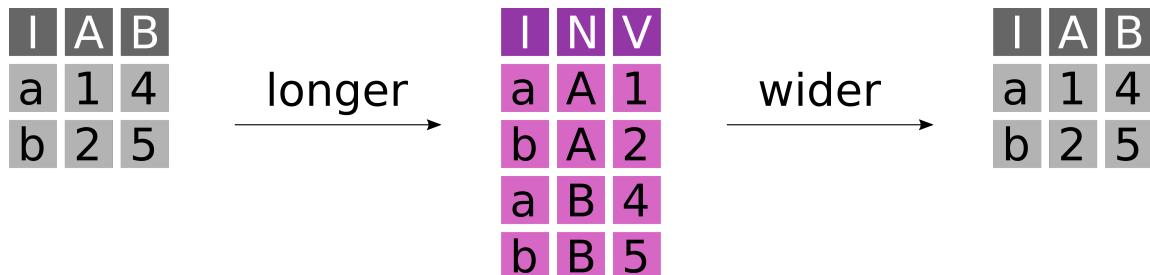


Figure 7.1.: Graphical representation of converting a table from a wide format (first column are categories, e.g. species, first row are also categories e.g., samples, and counts are in cells after first row/column). Conversion from wide to long makes the long-format so that the first column still has categories, but the second column becomes the column-categories from the first row, and third column has values. Converting long format back to wider shows again first row and column as categories and cells as values).

- 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.

## 7. Introduction to R and the Tidyverse

Here is an example of a wide dataset. It features information about the number of cars sold per year per brand at a dealership.

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

# A tibble: 2 x 5
  brand `2014` `2015` `2016` `2017`
  <chr>  <dbl>   <dbl>   <dbl>   <dbl>
1 BMW      20     25     30     45
2 VW       67     40    120     55
```

Wide format data 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.

To transform this dataset to a long format, we can apply `tidyverse::pivot_longer`.

```
carsales_long <- carsales %>% tidyverse::pivot_longer(
  cols = tidyselect::num_range(
    "", 
    range = 2014:2017
  ), # define a 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

# A tibble: 8 x 3
  brand  year sales
  <chr> <int> <dbl>
1 BMW    2014    20
2 BMW    2015    25
3 BMW    2016    30
4 BMW    2017    45
```

## 7. Introduction to R and the Tidyverse

```
5 VW      2014     67  
# i 3 more rows
```

Wide datasets are not always the wrong choice, for example for 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).

So transform data from long to wide, we can use `tidyverse::pivot_wider`

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

```
# A tibble: 2 x 5  
  brand `2014` `2015` `2016` `2017`  
  <chr>   <dbl>   <dbl>   <dbl>   <dbl>  
1 BMW      20      25      30      45  
2 VW       67      40     120      55
```

### 7.6.4. Exercise

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*.

#### Possible solutions

```
mtcars %>%  
  dplyr::relocate(gear, .before = mpg) %>%  
  tibble::as_tibble() # transforming the raw dataset for better printing
```

```
# A tibble: 32 x 11  
  gear   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  carb  
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
```

## 7. Introduction to R and the Tidyverse

```
<dbl> <dbl>
1     4   21      6   160    110   3.9    2.62   16.5     0     1     4
2     4   21      6   160    110   3.9    2.88   17.0     0     1     4
3     4  22.8     4   108    93    3.85   2.32   18.6     1     1     1
4     3  21.4     6   258    110   3.08   3.22   19.4     1     0     1
5     3  18.7     8   360    175   3.15   3.44   17.0     0     0     2
# i 27 more rows

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

# A tibble: 32 x 12
  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb am_v
<dbl> <chr>
1 21     6   160    110   3.9    2.62   16.5     0     1     4     4 manual
2 21     6   160    110   3.9    2.88   17.0     0     1     4     4 manual
3 22.8    4   108    93    3.85   2.32   18.6     1     1     4     1 manual
4 21.4    6   258    110   3.08   3.22   19.4     1     0     3     1 automatic
5 18.7    8   360    175   3.15   3.44   17.0     0     0     3     2 automatic
# i 27 more rows

mtcars2 %>%
  dplyr::group_by(am_v, gear) %>%
  dplyr::tally() %>% # dplyr::tally() is identical to
# dplyr::summarise(n = dplyr::n())
# it counts the number of entities in a group
tidyverse::pivot_wider(
  names_from = am_v,
  values_from = n
)
```

```
# A tibble: 3 x 3
  gear automatic manual
  <dbl>     <int>   <int>
1     3         15     NA
2     4          4      8
3     5         NA      5
```

## 7.7. Combining tibbles with join operations

### 7.7.1. Types of joins

Joins combine two datasets x and y based on overlapping 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.

Types of filtering consist of:

- Semi join: Keep every observation in x that is in y.
- Anti join: Keep every observation in x that is not in y.

The following sections will introduce each join with an example.

To experiment with joins, we need a second dataset with complementary information. This new dataset contains additional variables for a subset of the penguins in our first dataset – both datasets feature 300 penguins, but only with a partial overlap in individuals.

```
bills <- readr::read_csv("penguin_bills.csv")
```

```
# A tibble: 300 x 3
  id bill_length_mm bill_depth_mm
  <dbl>       <dbl>        <dbl>
1     1         39.1        18.7
2     2         39.5        17.4
3     3         40.3        18
```

## 7. Introduction to R and the Tidyverse

```
4      4          36.7        19.3
5      5          39.3        20.6
# i 295 more rows
```

### 7.7.2. Left join with `left_join`

Take observations from x and add fitting information from y (Figure 7.2).

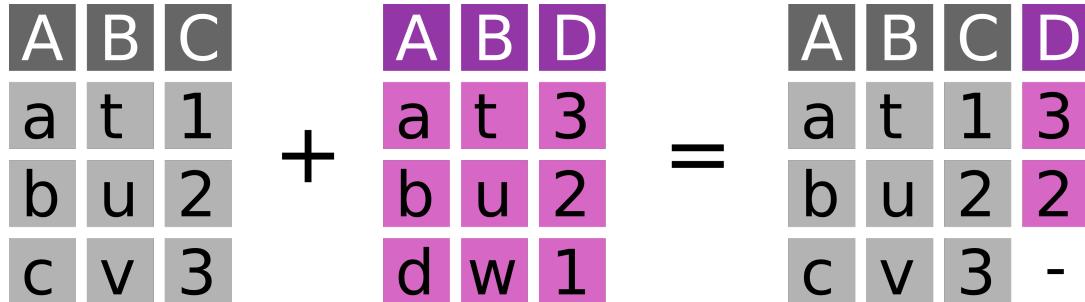


Figure 7.2.: Graphical representation of left join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have columns A B C D. As A and B have a one to one match of values, this remains the same in the joined table. The B column between the two have a different value on the third row, and thus is lost from the second table, retaining row 3 of the first table. Column D (from the second table) has an empty value on row three, as this row was not in row 3 of the second table that column D was derived from.

```
dplyr::left_join(
  x = peng, # 300 observations
  y = bills, # 300 observations
  by = "id" # the key column by which to join
)
```

```
# A tibble: 300 x 9
  id species island flipper_length_mm body_mass_g sex year bill_length_mm
  <dbl> <chr>   <chr>           <dbl>       <dbl> <chr> <chr>       <dbl>
1     1 Adelie  Torger~         181        3750 male  2007     39.1
2     2 Adelie  Torger~         186        3800 fema~ 2007     39.5
3     4 Adelie  Torger~         193        3450 fema~ 2007     36.7
4     5 Adelie  Torger~         190        3650 male  2007     39.3
```

## 7. Introduction to R and the Tidyverse

```
5      6 Adelie Torger~          181          3625 fema~ 2007      38.9
# i 295 more rows
# i 1 more variable: bill_depth_mm <dbl>
```

Left joins are the most common join operation: Add information from y to the main dataset x.

### 7.7.3. Right join with right\_join

Take observations from y and add fitting information from x (Figure 7.3).

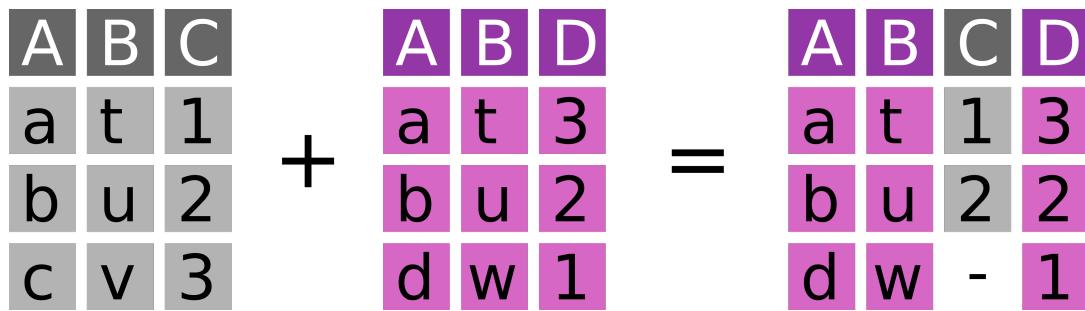


Figure 7.3.: Graphical representation of right join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have columns A B C D. As A and B have a one to one match of values, this remains the same in the joined table. The B column between the two have a different value on the third row, and thus is lost from the first table, retaining row 3 of the second table. Column C (from the first table) has an empty value on row three, as this row was not in row 3 of the first table that column C was derived from.

```
dplyr::right_join(
  x = peng, # 300 observations
  y = bills, # 300 observations
  by = "id"
) %>% # we arrange by id to highlight the missing
  dplyr::arrange(id) # observation in the peng dataset
```

```
# A tibble: 300 x 9
  id species island flipper_length_mm body_mass_g sex   year bill_length_mm
  <dbl> <chr>   <chr>           <dbl>        <dbl> <chr> <chr>       <dbl>
1     1 Adelie  Torger~            181         3750 male  2007       39.1
```

## 7. Introduction to R and the Tidyverse

```

2      2 Adelie Torger~          186      3800 fema~ 2007      39.5
3      3 <NA>    <NA>           NA       NA <NA>    <NA>      40.3
4      4 Adelie Torger~          193      3450 fema~ 2007      36.7
5      5 Adelie Torger~          190      3650 male   2007      39.3
# i 295 more rows
# i 1 more variable: bill_depth_mm <dbl>

```

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

### 7.7.4. Inner join with inner\_join

Join the overlapping observations from x and y (Figure 7.4).

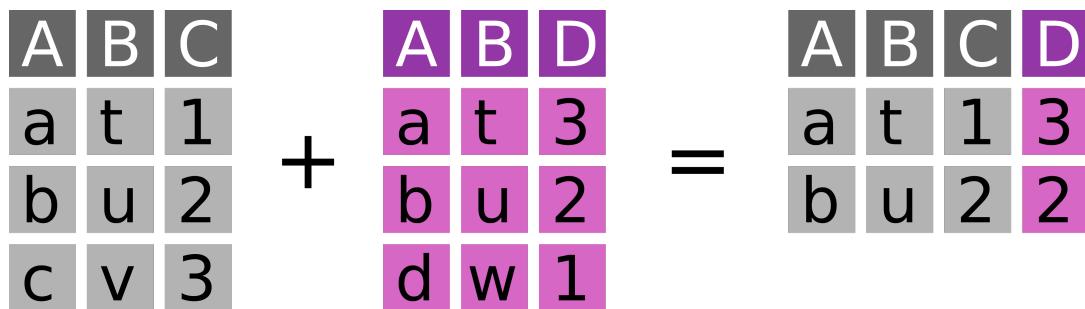


Figure 7.4.: Graphical representation of inner join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have columns A B C D. Only rows from both table that have exact matches on columns A and B are retained. The third rows from both tables that had a different value in column B are lost.

```

dplyr::inner_join(
  x = peng, # 300 observations
  y = bills, # 300 observations
  by = "id"
)

# A tibble: 275 x 9
  id species island flipper_length_mm body_mass_g sex year bill_length_mm
  <dbl> <chr>   <chr>            <dbl>        <dbl> <chr> <chr>        <dbl>
1     1 Adelie  Torger~          181         3750 male  2007      39.1
2     2 Adelie  Torger~          186         3800 fema~ 2007      39.5

```

## 7. Introduction to R and the Tidyverse

```

3     4 Adelie Torger~      193      3450 fema~ 2007    36.7
4     5 Adelie Torger~      190      3650 male   2007    39.3
5     6 Adelie Torger~      181      3625 fema~ 2007    38.9
# i 270 more rows
# i 1 more variable: bill_depth_mm <dbl>

```

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

### 7.7.5. Full join with full\_join

Join all observations from x and y, even if information is missing (Figure 7.5).

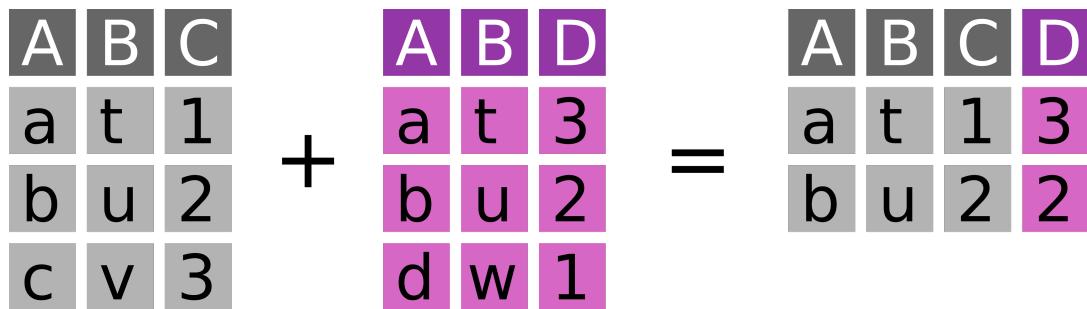


Figure 7.5.: Graphical representation of full join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have columns A B C D. All rows from both tables are retained, even though they do not share the same value on column B on both tables. The missing values for the two third rows (i.e., column C from the second table, and column D from the first table) are filled with an empty cell indicated with -.

```

dplyr::full_join(
  x = peng, # 300 observations
  y = bills, # 300 observations
  by = "id"
) %>% dplyr::arrange(id)

```

```

# A tibble: 325 x 9
  id species island flipper_length_mm body_mass_g sex year bill_length_mm
  <dbl> <chr>   <chr>           <dbl>        <dbl> <chr> <chr>       <dbl>
1     1 Adelie  Torger~          181         3750 male   2007    39.1
2     2 Adelie  Torger~          186         3800 fema~  2007    39.5

```

## 7. Introduction to R and the Tidyverse

```

3     3 <NA>    <NA>          NA          NA <NA>    <NA> 40.3
4     4 Adelie  Torgers~       193        3450 fema~  2007 36.7
5     5 Adelie  Torgers~       190        3650 male   2007 39.3
# i 320 more rows
# i 1 more variable: bill_depth_mm <dbl>

```

Full joins allow to preserve every bit of information.

### 7.7.6. Semi join with semi\_join

Keep every observation in x that is in y (Figure 7.6).

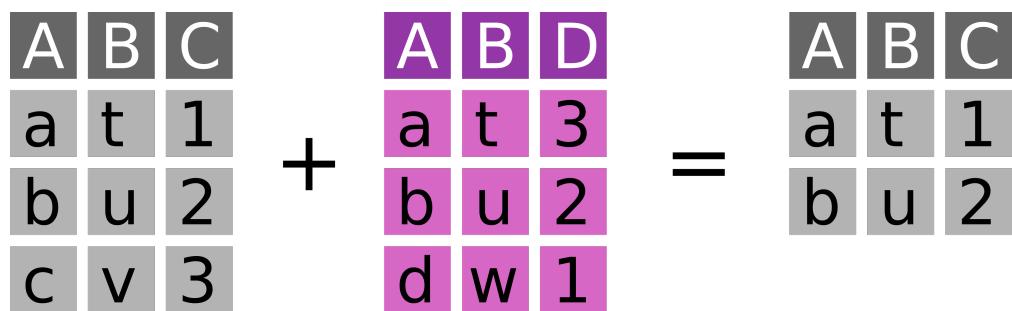


Figure 7.6.: Graphical representation of semi join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have only columns A B C. Only columns A B and C are retained in the joined table. Row three of both tables are not included as the column values in columns A and B do not match.

```

dplyr::semi_join(
  x = peng, # 300 observations
  y = bills, # 300 observations
  by = "id"
)

# A tibble: 275 x 7
  id species island flipper_length_mm body_mass_g sex year
  <int> <chr>   <chr>           <dbl>      <dbl> <chr> <chr>
1     1 Adelie  Torgersen         181        3750 male   2007
2     2 Adelie  Torgersen         186        3800 female 2007
3     4 Adelie  Torgersen         193        3450 female 2007

```

## 7. Introduction to R and the Tidyverse

```
4      5 Adelie  Torgersen      190      3650 male   2007
5      6 Adelie  Torgersen      181      3625 female 2007
# i 270 more rows
```

Semi joins are underused (!) operations to filter datasets.

### 7.7.7. Anti join with anti\_join

Keep every observation in x that is not in y (Figure 7.7).

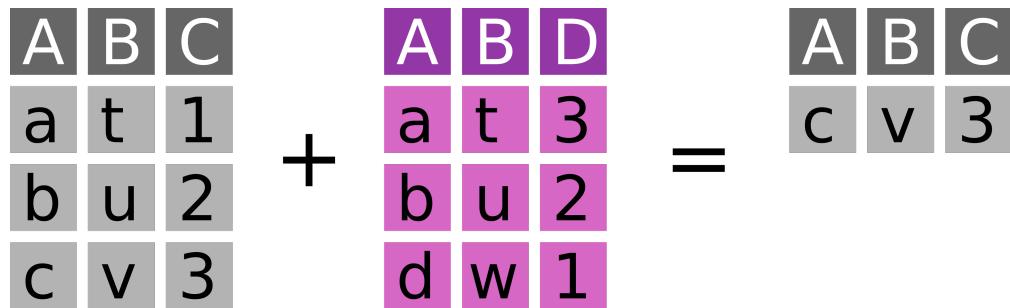


Figure 7.7.: Graphical representation of anti join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have only columns A B C, and of that only row 3 of the first table. Only row three is retained from the first table as this is the only unique row present only in the first table.

```
dplyr::anti_join(
  x = peng, # 300 observations
  y = bills, # 300 observations
  by = "id"
)

# A tibble: 25 x 7
  id species island    flipper_length_mm body_mass_g sex  year
  <int> <chr>   <chr>                <dbl>       <dbl> <chr> <chr>
1    22 Adelie  Biscoe            183        3550 male   2007
2    34 Adelie  Dream             181        3300 female 2007
3    74 Adelie  Torgersen        195        4000 male   2008
4    92 Adelie  Dream             196        4350 male   2008
5    99 Adelie  Biscoe            193        2925 female 2009
# i 20 more rows
```

## 7. Introduction to R and the Tidyverse

Anti joins allow to quickly determine what information is missing in a dataset compared to an other one.

### 7.7.8. Exercise

Consider the following additional dataset with my opinions on cars with a specific number of gears:

```
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 do not have an opinion

#### 💡 Possible solutions

```
dplyr::left_join(mtcars, gear_opinions, by = "gear") %>%  
  tibble::as_tibble()  
  
# A tibble: 32 x 12  
  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb  opinion  
  <dbl> <chr>  
1 21      6   160   110  3.9   2.62  16.5    0     1    4    4 <NA>  
2 21      6   160   110  3.9   2.88  17.0    0     1    4    4 <NA>  
3 22.8    4   108   93   3.85  2.32  18.6    1     1    4    1 <NA>  
4 21.4    6   258   110  3.08  3.22  19.4    1     0    3    1 boring  
5 18.7    8   360   175  3.15  3.44  17.0    0     0    3    2 boring  
# i 27 more rows  
  
dplyr::anti_join(mtcars, gear_opinions, by = "gear") %>%  
  tibble::as_tibble()  
  
# A tibble: 12 x 11  
  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb  
  <dbl>  
1 21      6   160   110  3.9   2.62  16.5    0     1    4    4  
2 21      6   160   110  3.9   2.88  17.0    0     1    4    4
```

## 7. Introduction to R and the Tidyverse

```
3 22.8      4 108      93 3.85  2.32 18.6      1      1      4      1
4 24.4      4 147.     62 3.69  3.19 20        1      0      4      2
5 22.8      4 141.     95 3.92  3.15 22.9      1      0      4      2
# i 7 more rows
```

## 7.8. References

# 8. Introduction to Python and Pandas

## Note

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

## Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate python-pandas
```

## 8.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

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

## Warning

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

## 8.2. Introduction to data manipulation in Python with Pandas and visualisation

Over the last few years, Python has gained popularity thanks to the numerous libraries (packages with pre-written functions) in the field of machine learning, statistical data analysis, and bioinformatics. While a few years ago, it was often necessary to go to R for performing routine data manipulation and analysis tasks, nowadays Python has a vast ecosystem of libraries. Libraries exist for many different file formats that you might encounter in metagenomics, such as fasta, fastq, sam, bam, etc.

This tutorial/walkthrough will provide a short introduction to the most popular libraries for data analysis pandas. This library has functions for reading and manipulating tabular data similar to the `data.frame()` in R together with some basic data plotting codes.

The aim of this walkthrough is to first: get familiar with the Python code syntax and use Jupiter Notebook for executing codes and secondly get a kickstart to utilising the endless possibilities of data analysis in Python that can be applied to your data.

### 8.3. Table of content:

- 8.3 Working in a jupyter environment
- 8.4 Pandas
- 8.5 Reading data with Pandas
- 8.6 Data exploration
- 8.7 Describing a DataFrame
- 8.8 Dealing with missing data
- 8.9 Combining data
- 8.10 Data visualization
- 8.11 Plotnine
- 8.12 [Lecture from 2022]

### 8.4. Working in a jupyter environment

This tutorial run-through is using a Jupyter Notebook for writing & executing Python code and for annotating.

Jupyter notebooks are convenient and have two types of cells: **Markdown** and **Code**. The **markup cell** syntax is very similar to **R markdown**. The markdown cells are used

## 8. Introduction to Python and Pandas

for annotating, which is important for sharing code with collaborators, reproducibility, and documentation.

A few examples are shown below. For a full list of possible syntax click [here](#) for a Jupyter Notebook cheat-sheet.

list of **markdown cell** examples:

### **i** Note

In many cases, there are multiple possible syntaxes which give the same result. We present only one way in this run-through.

Text

- **\*\*bold\*\*** : **bold**
- **\*italics\*** : *italics*

Code

- ‘inline code’ : `inline code`

LaTeX maths

- `$ x = \frac{\pi}{42} $ :`

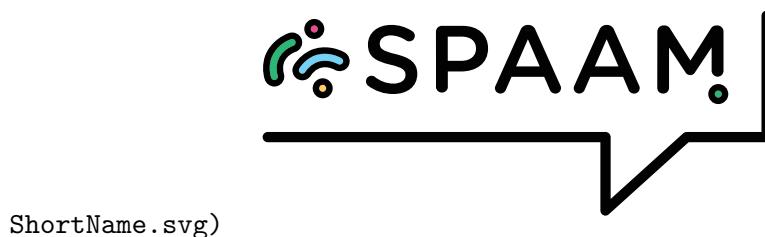
$$x = \frac{\pi}{42}$$

url links

- `[link] (https://www.python.org/)`link

Images

- `![] (https://www.spaam-community.org/assets/media/SPAAM-Logo-Full-Colour_-`



## 8. Introduction to Python and Pandas

The **code cells** can interpret many different coding languages including Python and Bash. The syntax of the code cells is the same as the syntax of the coding languages, in our case python.

Below are some examples of Python **code cells** with some useful basic python functions:

### 💡 Python function

`print()` is a python function for printing lines in the terminal `print() == echo` in bash

```
print("Hello World from Python!")
```

*out - Hello World from Python!*

But it can also, for example, run bash commands by adding a `!` at the start of the line.

```
! echo "Hello World from bash!"
```

*out - Hello World from bash!*

Strings or numbers can be stored as a variable by using the `=` sign

```
i = 0
```

Once a variable is set in one **code cell** they are stored and can be accessed in other downstream **code cells**.

```
print(i)
```

You can also print multiple things together in one `print` statement such as a number and a string:

```
print("The number is", i, "Wow!")
```

*out - The number is 0 Wow!*

## 8.5. Pandas

### 8.5.1. Getting started

Pandas is a Python library used for data manipulation and analysis.

We can import the library like this:

```
import pandas as pd
```



Note

We set “pandas” to the alias “pd” because we are lazy and don’t want to write the full word too many times.

Now, we can print the current version:

```
pd.__version__
```

```
out - '2.0.1'
```

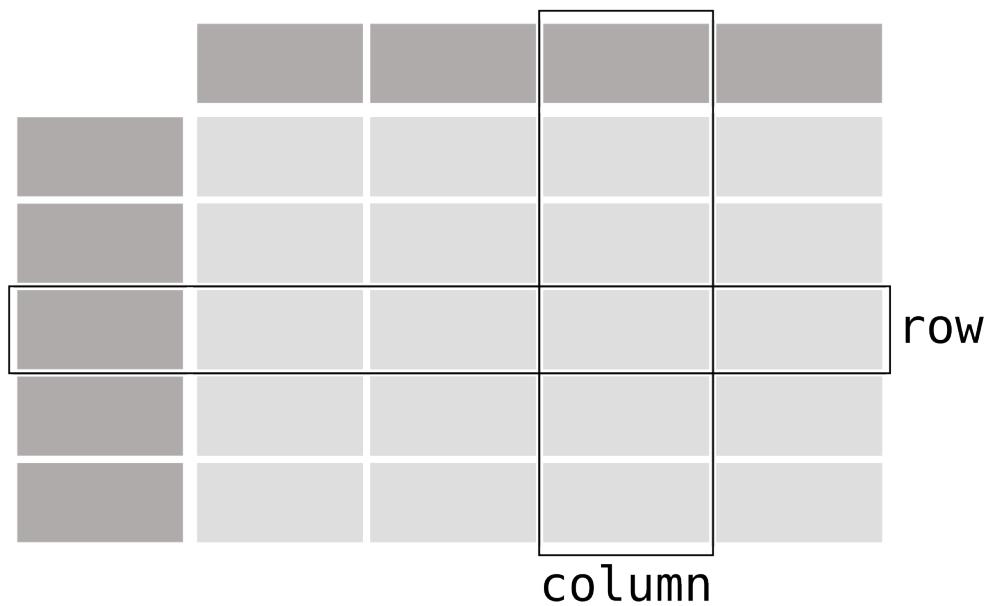
### 8.5.2. Pandas data structures

The primary data structures in Pandas are **Series** and **DataFrame**.

A **DataFrame** is a table with **columns** and **rows**.

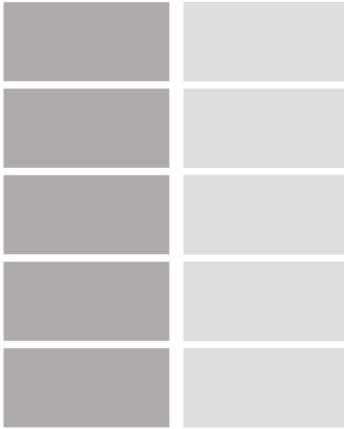
Each **column** has a *column name* and each **row** has an *index*.

# DataFrame



A single row or column (1 dimensional data) is a **Series**.

# Series



**i** Note

For a more in detail pandas getting started tutorial click [here](#)

## 8.6. Reading data with Pandas

Pandas can read in **csv** (comma separated values) files, which are tables in text format.

It's called **csv** because each value is separated from the others via a comma, like this:

```
A,B  
5,6  
8,4
```

*out - | | A | B | |---|---| | 1 | 2 | 3 | | 2 | 3 | 4 |*

Another common tabular separator are **tsv**, where each value is separated by a **tab** \t

```
A\tB  
5\t6  
8\t4
```

## 8. Introduction to Python and Pandas

Our dataset "all\_data.tsv" is tab separated, which Pandas can handle using the `sep` argument.

### Pandas function

`pd.read_csv()` is the pandas function to read in tabular tables. The `sep=` can be specified argument, `sep=`, is the default.

```
df = pd.read_csv("all_data.tsv", sep="\t")
df
```

|            | ID      | Birt    | Education | Inco  | Kid    | TotMon | Winc   | NetFam | NetP   | NetS | NetM   | NetC | NetV   | NetD | NetI   | NetN | NetL   | NetR | NetM   | NetP | NetV   | NetM | Cost   | Z_  | Con-Rev |         |
|------------|---------|---------|-----------|-------|--------|--------|--------|--------|--------|------|--------|------|--------|------|--------|------|--------|------|--------|------|--------|------|--------|-----|---------|---------|
|            | Year    | Sta     | Marital_- | Age   | Gender | Age    | Gender | Age    | Gender | Age  | Gender | Age  | Gender | Age  | Gender | Age  | Gender | Age  | Gender | Age  | Gender | Age  | Gender | Z_- | CostZ_- | Con-Rev |
| 0          | 5524957 | Grad    | Sing      | 1581  | 38.00  | 635    | 88     | 546    | 172    | 88   | 88     | 8    | 10     | 4    | 7      | 0    | 3      | 11   |        |      |        |      |        |     |         |         |
| 1          | 2174954 | Gra     | Sing      | 1663  | 44.01  | 11     | 1      | 6      | 2      | 1    | 6      | 1    | 1      | 2    | 5      | 0    | 3      | 11   |        |      |        |      |        |     |         |         |
| 2          | 4141965 | Grad    | Divor     | 1716  | 30.00  | 426    | 49     | 127    | 111    | 21   | 42     | 8    | 2      | 10   | 4      | 0    | 3      | 11   |        |      |        |      |        |     |         |         |
| 3          | 6182984 | Grad    | Divor     | 1766  | 16.00  | 11     | 4      | 20     | 10     | 3    | 5      | 2    | 0      | 4    | 6      | 0    | 3      | 11   |        |      |        |      |        |     |         |         |
| 4          | 7446967 | Master  | Forge     | 6256  | 30.01  | 520    | 42     | 98     | 0      | 42   | 14     | 6    | 4      | 10   | 6      | 0    | 3      | 11   |        |      |        |      |        |     |         |         |
| ...        | ...     | ...     | ...       | ...   | ...    | ...    | ...    | ...    | ...    | ...  | ...    | ...  | ...    | ...  | ...    | ...  | ...    | ...  | ...    | ...  | ...    | ...  | ...    | ... | ...     |         |
| 1749432977 | Grad    | Divor   | 1666      | 66.0  | 9      | 14     | 18     | 8      | 1      | 12   | 3      | 1    | 3      | 6    | 0      | 3    | 11     |      |        |      |        |      |        |     |         |         |
| 1758372974 | Grad    | Married | 1412      | 11.00 | 3      | 3      | 7      | 6      | 2      | 9    | 1      | 0    | 2      | 7    | 0      | 3    | 11     |      |        |      |        |      |        |     |         |         |
| 1751087067 | Grad    | Married | 1223      | 0.01  | 709    | 43     | 182    | 42     | 118    | 247  | 9      | 3    | 4      | 5    | 0      | 3    | 11     |      |        |      |        |      |        |     |         |         |
| 1752270981 | Grad    | Divor   | 1569      | 0.01  | 908    | 48     | 217    | 32     | 12     | 24   | 2      | 3    | 13     | 6    | 0      | 3    | 11     |      |        |      |        |      |        |     |         |         |
| 1758235956 | Master  | Forge   | 6924      | 5.01  | 428    | 30     | 214    | 80     | 30     | 61   | 6      | 5    | 10     | 3    | 0      | 3    | 11     |      |        |      |        |      |        |     |         |         |

### Tip

When you are unsure what arguments a function can take, it is possible to get a *help documentation* using `help(pd.read_csv)`

## 8.7. Data exploration

The data is from a customer personality analysis of a company trying to better understand how to modify their product catalogue. Here is the link to the original source for more information.

## 8. Introduction to Python and Pandas

### 8.7.1. Columns

The command below prints all the column names.

```
df.columns
```

```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases',
       'NumWebVisitsMonth', 'Complain', 'Z_CostContact', 'Z_Revenue'],
      dtype='object')
```

We can also list their respective data types.

- `int64` are integers
- `float64` are floating point numbers, also called `double` in other languages
- `object` are Python objects, which are strings in this case

```
df.dtypes
```

|                     |         |
|---------------------|---------|
| ID                  | int64   |
| Year_Birth          | int64   |
| Education           | object  |
| Marital_Status      | object  |
| Income              | float64 |
| Kidhome             | int64   |
| Teenhome            | int64   |
| MntWines            | int64   |
| MntFruits           | int64   |
| MntMeatProducts     | int64   |
| MntFishProducts     | int64   |
| MntSweetProducts    | int64   |
| MntGoldProds        | int64   |
| NumWebPurchases     | int64   |
| NumCatalogPurchases | int64   |
| NumStorePurchases   | int64   |
| NumWebVisitsMonth   | int64   |
| Complain            | int64   |
| Z_CostContact       | int64   |

## 8. Introduction to Python and Pandas

```
Z_Revenue           int64  
dtype: object
```

### 8.7.2. Inspecting the DataFrame

What is the size of our DataFrame

```
df.shape
```

(1754, 20)

It has **1754** rows and **20** columns.

Let's look at the first 5 rows:

```
df.head()
```

|   | ID  | Birt | Education | Inco    | Kid       | Marital_ | Year_ | Sta- | Con-Rev- | Z_- | CostZ_- |   |    |    |   |   |   |    |    |
|---|-----|------|-----------|---------|-----------|----------|-------|------|----------|-----|---------|---|----|----|---|---|---|----|----|
| 0 | 552 | 1957 | Grad      | \$singl | 158138.00 | 635      | 88    | 546  | 172      | 88  | 88      | 8 | 10 | 4  | 7 | 0 | 3 | 11 |    |
| 1 | 217 | 1954 | Grad      | \$singl | 16344.01  | 11       | 1     | 6    | 2        | 1   | 6       | 1 | 1  | 1  | 2 | 5 | 0 | 3  | 11 |
| 2 | 414 | 1965 | Grad      | Logi    | 1613.00   | 426      | 49    | 127  | 111      | 21  | 42      | 8 | 2  | 10 | 4 | 0 | 3 | 11 |    |
| 3 | 618 | 2984 | Grad      | Logi    | 26616.00  | 11       | 4     | 20   | 10       | 3   | 5       | 2 | 0  | 4  | 6 | 0 | 3 | 11 |    |
| 4 | 744 | 6967 | Mast      | Forge   | 62513.01  | 520      | 42    | 98   | 0        | 42  | 14      | 6 | 4  | 10 | 6 | 0 | 3 | 11 |    |

What we can see it that, unlike **R**, **Python** and in extension **Pandas** is 0-indexed instead of 1-indexed.

Question: Can you show how to do the same using bash?

```
! head all_data.tsv
```

## 8. Introduction to Python and Pandas

|   | ID      | Year_Birth | Education       | Marital_Status | Kidhome | Teenhome | MntWines | MntFruits | MntMeatProducts | MntFishProducts | MntSweetProducts | MntGoldProds | NumWebPurchases | NumVisitsMonth | Z_Cost | Z_ConRev |   |   |    |
|---|---------|------------|-----------------|----------------|---------|----------|----------|-----------|-----------------|-----------------|------------------|--------------|-----------------|----------------|--------|----------|---|---|----|
| 0 | 5524957 | 1957       | Graduation      | Single         | 158     | 38.00    | 635      | 88        | 546             | 172             | 88               | 88           | 8               | 10             | 4      | 7        | 0 | 3 | 11 |
| 1 | 2174954 | 1954       | Graduation      | Single         | 166     | 44.01    | 11       | 1         | 6               | 2               | 1                | 6            | 1               | 1              | 2      | 5        | 0 | 3 | 11 |
| 2 | 4141965 | 1965       | Graduation      | Together       | 161     | 38.00    | 426      | 49        | 127             | 111             | 21               | 42           | 8               | 2              | 10     | 4        | 0 | 3 | 11 |
| 3 | 6182984 | 1984       | Graduation      | Together       | 166     | 16.00    | 11       | 4         | 20              | 10              | 3                | 5            | 2               | 0              | 4      | 6        | 0 | 3 | 11 |
| 4 | 7446967 | 1967       | Master          | Together       | 125     | 18.01    | 520      | 42        | 98              | 0               | 42               | 14           | 6               | 4              | 10     | 6        | 0 | 3 | 11 |
| 5 | 9651971 | 1971       | Graduation      | Divorced       | 155     | 15.01    | 235      | 65        | 164             | 50              | 49               | 27           | 7               | 3              | 7      | 6        | 0 | 3 | 11 |
| 6 | 1994983 | 1983       | Graduation      | Married        | N       | 0        | 5        | 5         | 6               | 0               | 2                | 1            | 1               | 0              | 2      | 7        | 0 | 3 | 11 |
| 7 | 3871976 | 1976       | Basic Education | Married        | 76      | 0.0      | 6        | 16        | 11              | 11              | 1                | 16           | 2               | 0              | 3      | 8        | 0 | 3 | 11 |
| 8 | 2125959 | 1959       | Graduation      | Divorced       | 160     | 38.00    | 194      | 61        | 480             | 225             | 112              | 30           | 3               | 4              | 8      | 2        | 0 | 3 | 11 |
| 9 | 8180952 | 1952       | Master          | Divorced       | 59      | 14.01    | 233      | 2         | 53              | 3               | 5                | 14           | 6               | 1              | 5      | 6        | 0 | 3 | 11 |

### 8.7.3. Accessing rows and columns

We can access parts of the data in `DataFrames` in different ways.

The first method is sub-setting the rows using the index.

This will take only the second row and all columns, producing a `Series`:

```
df.loc[1, :]
```

|                  |            |
|------------------|------------|
| ID               | 2174       |
| Year_Birth       | 1954       |
| Education        | Graduation |
| Marital_Status   | Single     |
| Income           | 46344.0    |
| Kidhome          | 1          |
| Teenhome         | 1          |
| MntWines         | 11         |
| MntFruits        | 1          |
| MntMeatProducts  | 6          |
| MntFishProducts  | 2          |
| MntSweetProducts | 1          |
| MntGoldProds     | 6          |
| NumWebPurchases  | 1          |

## 8. Introduction to Python and Pandas

```
NumCatalogPurchases          1  
NumStorePurchases           2  
NumWebVisitsMonth          5  
Complain                     0  
Z_CostContact                3  
Z_Revenue                     11  
Name: 1, dtype: object
```

And this will take the second and third row, producing another DataFrame:

```
df.loc[1:2, :]
```

It's important to understand that almost all operations on `DataFrames` are not in-place, meaning that we don't modify the original object and would have to save the results to the same or a new variable to keep the changes.

This, for example will create a new `DataFrame` of only the “Education” and “Marital\_Status” columns.

```
new_df = df.loc[:, ["Education", "Marital_Status"]]
new_df
```

|      | Education  | Marital_Status |
|------|------------|----------------|
| 0    | Graduation | Single         |
| 1    | Graduation | Single         |
| 2    | Graduation | Together       |
| 3    | Graduation | Together       |
| 4    | Master     | Together       |
| ...  | ...        | ...            |
| 1749 | Graduation | Together       |

## 8. Introduction to Python and Pandas

|      | Education  | Marital_Status |
|------|------------|----------------|
| 1750 | Graduation | Married        |
| 1751 | Graduation | Married        |
| 1752 | Graduation | Divorced       |
| 1753 | Master     | Together       |

Selecting only one column by name:

```
df["Year_Birth"]
```

```
0      1957
1      1954
2      1965
3      1984
4      1967
...
1749    1977
1750    1974
1751    1967
1752    1981
1753    1956
```

We can also remove columns from the `DataFrame`.

In this case, we want to remove the columns `Z_CostContact` and `Z_Revenue` and keep those changes.

```
df = df.drop("Z_CostContact", axis=1)
df = df.drop("Z_Revenue", axis=1)
```

### 8.7.4. Conditional subsetting

We can more specifically look at subsets of the data we might be interested in.

This subsetting is a bit weird in the syntax at first but hopefully makes more sense when we go through it step by step.

We can, for example, test each string in the column `Education` if it is equal to `PhD`:

## 8. Introduction to Python and Pandas

```
education_is_grad = (df["Education"] == "Graduation")
education_is_grad
```

```
0      True
1      True
2      True
3      True
4     False
...
1749    True
1750    True
1751    True
1752    True
1753   False
Name: Education, Length: 1754, dtype: bool
```

We can also check for multiple conditions at once:

```
two_at_once = (df["Education"] == "Graduation") & (df["Marital_Status"] == "Single")
two_at_once
```

```
0      True
1      True
2     False
3     False
4     False
...
1749    False
1750    False
1751    False
1752    False
1753    False
Length: 1754, dtype: bool
```

This will create a **Series** of booleans, which can then be used to subset the data to rows where the condition(s) are **True**:

## 8. Introduction to Python and Pandas

```
df[two_at_once]
```

|            |           |           |           |         |               |              |                |           |      |        |     |             |                     |              |                  |                | Z_-      |
|------------|-----------|-----------|-----------|---------|---------------|--------------|----------------|-----------|------|--------|-----|-------------|---------------------|--------------|------------------|----------------|----------|
|            |           |           |           |         |               |              |                |           |      |        |     |             |                     |              |                  |                | CostZ_-  |
|            |           |           |           |         |               |              |                |           |      |        |     |             |                     |              |                  |                | Con-Rev- |
| ID         | BirthYear | Education | Income    | KidHome | TotalWorkWeek | FullTimeWork | Marital_Status | Residence | Race | Gender | Age | HoursWorked | NumCompaniesInMonth | NumDaysUnemp | NumHousingIssues | NumVisitsMonth |          |
| 0          | 5524957   | Graduate  | 158138.00 | 635     | 88            | 546          | 172            | 88        | 88   | 8      | 10  | 4           | 7                   | 0            | 3                | 11             |          |
| 1          | 2174954   | Graduate  | 16344.01  | 11      | 1             | 6            | 2              | 1         | 6    | 1      | 1   | 2           | 5                   | 0            | 3                | 11             |          |
| 18         | 7892969   | Graduate  | 168589.00 | 6       | 4             | 25           | 15             | 12        | 13   | 2      | 1   | 3           | 7                   | 0            | 3                | 11             |          |
| 20         | 5255986   | Graduate  | SingleNaN | 0       | 5             | 1            | 3              | 3         | 263  | 362    | 27  | 0           | 0                   | 1            | 0                | 3              | 11       |
| 33         | 1371976   | Graduate  | 169401.00 | 123     | 164           | 266          | 227            | 30        | 174  | 2      | 4   | 9           | 1                   | 0            | 3                | 11             |          |
| ...        | ...       | ...       | ...       | ...     | ...           | ...          | ...            | ...       | ...  | ...    | ... | ...         | ...                 | ...          | ...              | ...            | ...      |
| 1720096969 | Graduate  | 16731.01  | 266       | 21      | 300           | 65           | 8              | 44        | 8    | 8      | 6   | 6           | 0                   | 3            | 11               |                |          |
| 1723959968 | Graduate  | 165893.01 | 158       | 0       | 23            | 0            | 0              | 18        | 3    | 1      | 5   | 8           | 0                   | 3            | 11               |                |          |
| 1743201962 | Graduate  | 167967.01 | 229       | 7       | 137           | 4            | 0              | 91        | 4    | 2      | 8   | 5           | 0                   | 3            | 11               |                |          |
| 1746004984 | Graduate  | 1612.00   | 24        | 3       | 26            | 7            | 1              | 23        | 3    | 1      | 2   | 9           | 0                   | 3            | 11               |                |          |
| 1748080986 | Graduate  | 16816.00  | 5         | 1       | 6             | 3            | 4              | 3         | 0    | 0      | 3   | 4           | 0                   | 3            | 11               |                |          |

The syntax that seems more complicated and does it in one step without the extra `Series` is this:

```
df[(df["Education"] == "Master") & (df["Marital_Status"] == "Single")]
```

|            |           |           |           |         |               |              |                |           |      |        |     |             |                     |              |                  |                | Z_-      |
|------------|-----------|-----------|-----------|---------|---------------|--------------|----------------|-----------|------|--------|-----|-------------|---------------------|--------------|------------------|----------------|----------|
|            |           |           |           |         |               |              |                |           |      |        |     |             |                     |              |                  |                | CostZ_-  |
|            |           |           |           |         |               |              |                |           |      |        |     |             |                     |              |                  |                | Con-Rev- |
| ID         | BirthYear | Education | Income    | KidHome | TotalWorkWeek | FullTimeWork | Marital_Status | Residence | Race | Gender | Age | HoursWorked | NumCompaniesInMonth | NumDaysUnemp | NumHousingIssues | NumVisitsMonth |          |
| 26         | 1073951   | Master    | 149389.01 | 40      | 0             | 19           | 2              | 1         | 3    | 2      | 0   | 3           | 7                   | 0            | 3                | 11             |          |
| 46         | 6853982   | Master    | 17577.00  | 712     | 26            | 538          | 69             | 13        | 80   | 3      | 6   | 11          | 1                   | 0            | 3                | 11             |          |
| 76         | 1117872   | Master    | 142394.00 | 15      | 2             | 10           | 0              | 1         | 4    | 1      | 0   | 3           | 7                   | 0            | 3                | 11             |          |
| 98         | 6205967   | Master    | 132537.00 | 34      | 3             | 29           | 0              | 4         | 10   | 2      | 1   | 3           | 5                   | 0            | 3                | 11             |          |
| 1108211992 | Master    | 102859.00 | 962       | 61      | 921           | 52           | 61             | 20        | 5    | 4      | 12  | 2           | 0                   | 3            | 11               |                |          |
| ...        | ...       | ...       | ...       | ...     | ...           | ...          | ...            | ...       | ...  | ...    | ... | ...         | ...                 | ...          | ...              | ...            |          |
| 1698520990 | Master    | 101172.00 | 162       | 28      | 818           | 0            | 28             | 56        | 4    | 3      | 7   | 3           | 0                   | 3            | 11               |                |          |
| 1709418983 | Master    | 189616.00 | 671       | 47      | 655           | 145          | 111            | 15        | 7    | 5      | 12  | 2           | 0                   | 3            | 11               |                |          |
| 1712980952 | Master    | 188210.1  | 12        | 0       | 13            | 4            | 2              | 4         | 3    | 0      | 3   | 8           | 0                   | 3            | 11               |                |          |
| 1738366982 | Master    | 17577.00  | 712       | 26      | 538           | 69           | 13             | 80        | 3    | 6      | 11  | 1           | 0                   | 3            | 11               |                |          |

## 8. Introduction to Python and Pandas

## 8.8. Describing a DataFrame

Pandas can easily create overview statistics for all numeric columns:

```
df.describe()
```

You can also directly calculate the relevant statistics on columns you are interested in:

```
df["MntWines"].max()
```

1492

```
df[["Kidhome", "Teenhome"]].mean()
```

```
Kidhome      0.456100  
Teenhome     0.480616  
dtype: float64
```

## 8. Introduction to Python and Pandas

For non-numeric columns, you can get the represented values or their counts:

```
df["Education"].unique()
```

```
array(['Graduation', 'Master', 'Basic', '2n Cycle'], dtype=object)
```

```
df["Marital_Status"].value_counts()
```

```
Marital_Status
Married      672
Together     463
Single       382
Divorced     180
Widow        53
Alone         2
Absurd        2
Name: count, dtype: int64
```

### Task

Subset the DataFrame in two different ways:



#### Tip

Just like with the “PhD” string before, you can subset using integers and `<`, `>`, `<=` and `>=`.

- One where everybody is born before 1970

Solution

```
df_before = df[df["Year_Birth"] < 1970]
```

- One where everybody is born in or after 1970

Solution

```
df_before = df[df["Year_Birth"] >= 1970]
```

- How many people are in the two DataFrames?

Solution

## 8. Introduction to Python and Pandas

```
print("n(before)  =", df_before.shape[0])
print("n(after)   =", df_before.shape[0])
```

```
n(before)  = 804
n(after)   = 950
```

- Do the total number of people sum up to the original DataFrame total?

Solution

```
df_before.shape[0] + df_after.shape[0] == df.shape[0]
```

True

```
print("n(sum)      =", df_before.shape[0] + df_after.shape[0])
print("n(expected) =", df.shape[0])
```

```
n(sum)      = 1754
n(expected) = 1754
```

- How does the mean income of the two groups differ?

Solution

```
print("income(before) =", df_before["Income"].mean())
print("income(after)  =", df_after["Income"].mean())
```

```
income(before) = 55513.38113207547 income(after) = 47490.29255319149
```

**Extra task** - Can you find something else that differs a lot between the two groups?

## 8.9. Dealing with missing data

We can check for missing data for each cell like this:

```
df.isna()
```

---

|   | ID    | Birt  | Educati | Marital_- | Year_- | Sta-  | Cost  | Z_-   | Con-Rev- | Month |
|---|-------|-------|---------|-----------|--------|-------|-------|-------|----------|-------|
| 0 | False | False | False   | False     | False  | False | False | False | False    | False |
| 1 | False | False | False   | False     | False  | False | False | False | False    | False |

## 8. Introduction to Python and Pandas

|     | ID  | Birt  | Edutatio | Incd  | Kid   | Teen  | MntWines | MntFruit | MntMeat | MntFish | MntSweet | MntGold | NumWebPur | NumCatalogPur | NumStorePur | NumWebVisitsMonth | Complain | Z_-   | CostZ_- | Con-Rev- | Month |
|-----|-----|-------|----------|-------|-------|-------|----------|----------|---------|---------|----------|---------|-----------|---------------|-------------|-------------------|----------|-------|---------|----------|-------|
| 2   |     | False | False    | 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    | False | False   |          |       |
| 4   |     | False | False    | False | False | False | False    | False    | False   | False   | False    | False   | False     | False         | False       | False             | False    | False | False   |          |       |
| ... | ... | ...   | ...      | ...   | ...   | ...   | ...      | ...      | ...     | ...     | ...      | ...     | ...       | ...           | ...         | ...               | ...      | ...   | ...     |          |       |
| 174 |     | False | False    | False | False | False | False    | False    | False   | False   | False    | False   | False     | False         | False       | False             | False    | False | False   |          |       |
| 175 |     | False | False    | False | False | False | False    | False    | False   | False   | False    | False   | False     | False         | False       | False             | False    | False | False   |          |       |
| 175 |     | False | False    | False | False | False | False    | False    | False   | False   | False    | False   | False     | False         | False       | False             | False    | False | False   |          |       |
| 175 |     | False | False    | False | False | False | False    | False    | False   | False   | False    | False   | False     | False         | False       | False             | False    | False | False   |          |       |
| 175 |     | False | False    | False | False | False | False    | False    | False   | False   | False    | False   | False     | False         | False       | False             | False    | False | False   |          |       |

By summing over each row, we see how many missing values are in each column.

True is treated as 1 and False as 0.

```
df.isna().sum()
```

```
ID          0
Year_Birth   0
Education    0
Marital_Status 0
Income       19
Kidhome     0
Teenhome    0
MntWines    0
MntFruits   0
MntMeatProducts 0
MntFishProducts 0
MntSweetProducts 0
MntGoldProds 0
NumWebPurchases 0
NumCatalogPur 0
NumStorePur 0
NumWebVisitsMonth 0
Complain    0
dtype: int64
```

## 8. Introduction to Python and Pandas

We don't really know what a missing value means so we are just going to keep them in the data.

However, we could remove them using `df.dropna()`

### 8.9.1. Grouping data

We can group a `DataFrame` using a categorical column (for example `Education` or `Marital_Status`).

This allows us to do perform operations on each group individually.

For example, we could group by `Education` and calculate the mean `Income`:

```
df.groupby(by="Education")["Income"].mean()
```

```
Education
2n Cycle      47633.190000
Basic          20306.259259
Graduation     52720.373656
Master         52917.534247
Name: Income, dtype: float64
```

## 8.10. Combining data

### 8.10.1. Concatenation

One way to combine multiple datasets is through **concatenation**, which either combines all columns or rows of multiple `DataFrames`.

The command to combine two `DataFrames` by appending all rows is `pd.concat([first_dataframe, second_dataframe])`

#### Task

- Read the `tsv` “`phd_data.tsv`” as a new `DataFrame` and name the variable `df2`

Solution

```
df2 = pd.read_csv("phd_data.tsv", sep="\t")
```

## 8. Introduction to Python and Pandas

- Concatenate the “old” DataFrame df and the new df2 and name the concatenated one concat\_df

Solution

```
concat_df = pd.concat([df, df2])
concat_df
```

|            | ID         | BirthYear  | Education | Income    | KidHome | TotKid | MWintM | MWintM | MHSMA | MHSMA | PMSA | MedInc | Age | CapitalGain | CapitalLoss | HrsWorked | CapitalLoss | Z_CostContact | Z_Revenue | Month |
|------------|------------|------------|-----------|-----------|---------|--------|--------|--------|-------|-------|------|--------|-----|-------------|-------------|-----------|-------------|---------------|-----------|-------|
|            | ID         | BirthYear  | Education | Income    | KidHome | TotKid | MWintM | MWintM | MHSMA | MHSMA | PMSA | MedInc | Age | CapitalGain | CapitalLoss | HrsWorked | CapitalLoss | Z_CostContact | Z_Revenue | Month |
| 0          | 5524957    | GradS      | Singl     | 158138.00 | 635     | 88     | 546    | 172    | 88    | 88    | 8    | 10     | 4   | 7           | 0           | 3         | 11          |               |           |       |
| 1          | 2174954    | GradS      | Singl     | 16344.01  | 11      | 1      | 6      | 2      | 1     | 6     | 1    | 1      | 2   | 5           | 0           | 3         | 11          |               |           |       |
| 2          | 4141965    | GradT      | Singl     | 1613.00   | 426     | 49     | 127    | 111    | 21    | 42    | 8    | 2      | 10  | 4           | 0           | 3         | 11          |               |           |       |
| 3          | 6182984    | GradT      | Singl     | 1616.00   | 11      | 4      | 20     | 10     | 3     | 5     | 2    | 0      | 4   | 6           | 0           | 3         | 11          |               |           |       |
| 4          | 7446967    | MasterT    | Toge      | 62513.01  | 520     | 42     | 98     | 0      | 42    | 14    | 6    | 4      | 10  | 6           | 0           | 3         | 11          |               |           |       |
| ...        | ...        | ...        | ...       | ...       | ...     | ...    | ...    | ...    | ...   | ...   | ...  | ...    | ... | ...         | ...         | ...       | ...         | ...           | ...       |       |
| 4811113973 | PhDYOLO    | PhDYOLO    | Singl     | 32.01     | 322     | 3      | 50     | 4      | 3     | 42    | 7    | 1      | 6   | 8           | 0           | 3         | 11          |               |           |       |
| 4829589948 | PhDWidow   | PhDWidow   | Singl     | 32.00     | 332     | 194    | 377    | 149    | 125   | 57    | 4    | 6      | 7   | 1           | 0           | 3         | 11          |               |           |       |
| 4834286970 | PhDSingle  | PhDSingle  | Singl     | 57642.01  | 580     | 6      | 58     | 8      | 0     | 27    | 7    | 6      | 6   | 4           | 0           | 3         | 11          |               |           |       |
| 4844001946 | PhDTogea   | PhDTogea   | Singl     | 61024.01  | 406     | 0      | 30     | 0      | 0     | 8     | 8    | 2      | 5   | 7           | 0           | 3         | 11          |               |           |       |
| 4859405954 | PhDMarried | PhDMarried | Singl     | 52819.01  | 84      | 3      | 61     | 2      | 1     | 21    | 3    | 1      | 4   | 7           | 0           | 3         | 11          |               |           |       |

- Is there anything weird about the new DataFrame and can you fix that?

Solution

We previously removed the columns “Z\_CostContact” and “Z\_Revenue” but they are in the new data again.

We can remove them like before:

```
concat_df = concat_df.drop("Z_CostContact", axis=1)
concat_df = concat_df.drop("Z_Revenue", axis=1)
concat_df
```

|   | ID      | BirthYear | Education | Income    | KidHome | TotKid | MWintM | MWintM | MHSMA | MHSMA | PMSA | MedInc | Age | CapitalGain | CapitalLoss | HrsWorked | CapitalLoss | Z_CostContact | Z_Revenue | Month |
|---|---------|-----------|-----------|-----------|---------|--------|--------|--------|-------|-------|------|--------|-----|-------------|-------------|-----------|-------------|---------------|-----------|-------|
|   | ID      | BirthYear | Education | Income    | KidHome | TotKid | MWintM | MWintM | MHSMA | MHSMA | PMSA | MedInc | Age | CapitalGain | CapitalLoss | HrsWorked | CapitalLoss | Z_CostContact | Z_Revenue | Month |
| 0 | 5524957 | GradS     | Singl     | 158138.00 | 635     | 88     | 546    | 172    | 88    | 88    | 8    | 10     | 4   | 7           | 0           |           |             |               |           |       |
| 1 | 2174954 | GradS     | Singl     | 16344.01  | 11      | 1      | 6      | 2      | 1     | 6     | 1    | 1      | 2   | 5           | 0           |           |             |               |           |       |
| 2 | 4141965 | GradT     | Singl     | 1613.00   | 426     | 49     | 127    | 111    | 21    | 42    | 8    | 2      | 10  | 4           | 0           |           |             |               |           |       |

## 8. Introduction to Python and Pandas

| ID      | Birth   | Education | Income | Kid    |      | Teen   |     | Adult  |     | Married |     | Friends |     | Gardening |     | Religion |     | Sports |     | Parks  |     | Visits |  |
|---------|---------|-----------|--------|--------|------|--------|-----|--------|-----|---------|-----|---------|-----|-----------|-----|----------|-----|--------|-----|--------|-----|--------|--|
|         |         |           |        | Gender | Age  | Gender | Age | Gender | Age | Gender  | Age | Gender  | Age | Gender    | Age | Gender   | Age | Gender | Age | Gender | Age |        |  |
| 3       | 6182984 | Grad      | High   | Male   | 46.0 | 0      | 11  | 4      | 20  | 10      | 3   | 5       | 2   | 0         | 4   | 6        | 0   | 4      | 6   | 0      |     |        |  |
| 4       | 7446967 | Mast      | Toget  | Male   | 25.0 | 1      | 520 | 42     | 98  | 0       | 42  | 14      | 6   | 4         | 10  | 6        | 0   | 6      | 0   | 0      |     |        |  |
| ...     | ...     | ...       | ...    | ...    | ...  | ...    | ... | ...    | ... | ...     | ... | ...     | ... | ...       | ... | ...      | ... | ...    | ... | ...    | ... |        |  |
| 4811113 | 3973    | PhD       | YOLO   | Female | 32.0 | 1      | 322 | 3      | 50  | 4       | 3   | 42      | 7   | 1         | 6   | 8        | 0   | 6      | 8   | 0      |     |        |  |
| 4829589 | 948     | PhD       | Wido   | Male   | 20.0 | 0      | 332 | 194    | 377 | 149     | 125 | 57      | 4   | 6         | 7   | 1        | 0   | 7      | 1   | 0      |     |        |  |
| 4834281 | 970     | PhD       | Singl  | Male   | 76.0 | 1      | 580 | 6      | 58  | 8       | 0   | 27      | 7   | 6         | 6   | 4        | 0   | 6      | 4   | 0      |     |        |  |
| 4844001 | 1946    | PhD       | Toget  | Male   | 40.0 | 1      | 406 | 0      | 30  | 0       | 0   | 8       | 8   | 2         | 5   | 7        | 0   | 5      | 7   | 0      |     |        |  |
| 4859405 | 1954    | PhD       | Marri  | Male   | 28.0 | 1      | 84  | 3      | 61  | 2       | 1   | 21      | 3   | 1         | 4   | 7        | 0   | 4      | 7   | 0      |     |        |  |

- Is there something interesting about the marital status of some people that have a PhD?

## Solution

```
concat_df[concat_df["Education"]=="PhD"]["Marital_Status"].value_counts()
```

```
Marital_Status  
Married        192  
Together       117  
Single         98  
Divorced       52  
Widow          24  
YOLO            2  
Alone           1  
Name: count, dtype: int64
```

There's two people that have "YOLO" as their Marital Status ...

### 8.10.2. Merging

Analysing numbers can be easier than analysing categorial values, like “PhD” and “Master”.

To make our life easier, we might want to have a new column when the Education level is replaced with a number that “ranks” the Education levels by how long it takes.

This information could be stored in a Python Dictionary (Also called Hash Map in other languages), which stores **key** and **value** pairs.

## 8. Introduction to Python and Pandas

We could store the Education information like this:

```
education_dictionary = {  
    "Basic": 1,  
    "2n Cycle": 2,  
    "Graduation": 3,  
    "Master": 4,  
    "PhD": 5  
}
```

We can now convert this Dictionary to a DataFrame:

```
education_df = pd.DataFrame.from_dict(education_dictionary, orient="index")  
education_df
```

|            | 0 |
|------------|---|
| Basic      | 1 |
| 2n Cycle   | 2 |
| Graduation | 3 |
| Master     | 4 |
| PhD        | 5 |

The resulting DataFrame has the Education level as index and the column 0 has the level information.

We can rename the column to “Level”.

```
education_df = education_df.rename(columns={0: "Level"})  
education_df
```

|            | Level |
|------------|-------|
| Basic      | 1     |
| 2n Cycle   | 2     |
| Graduation | 3     |
| Master     | 4     |
| PhD        | 5     |

## 8. Introduction to Python and Pandas

We can now **merge** this new `education_df` with our previous `concat_df`.

The left DataFrame is `concat_df` and we merge on “Education” because that’s where the Education information is.

The right one is `education_df` and the information is in the index.

```
merged_df = pd.merge(left=concat_df, right=education_df, left_on="Education", right_index=True)
```

## 8.11. Data visualization

We can easily create simple graphs using `DataFrame.plot()`.

This uses the package **matplotlib** in the background, which is a very powerful and popular plotting library but is not the most user friendly.

Using this **Pandas** method is very easy and can be a good way to do some initial exploratory plots and later refine them using either pure **matplotlib** or another library.

### 8.11.1. Histogram

We can plot the data from a `DataFrame` like this:

## 8. Introduction to Python and Pandas

`kind` specifies the kind of plot (for example `hist` for histogram, `bar` for bar graph or `scatter` for scatter plot).

We usually specify the columns from which the `x` and `y` components should be taken, but for a histogram we only need to specify one.

```
ax = merged_df.plot(kind="hist", y="Income")
ax.set_xlabel("Income")
ax.set_title("Histogram of income")
```

Text(0.5, 1.0, 'Histogram of income')

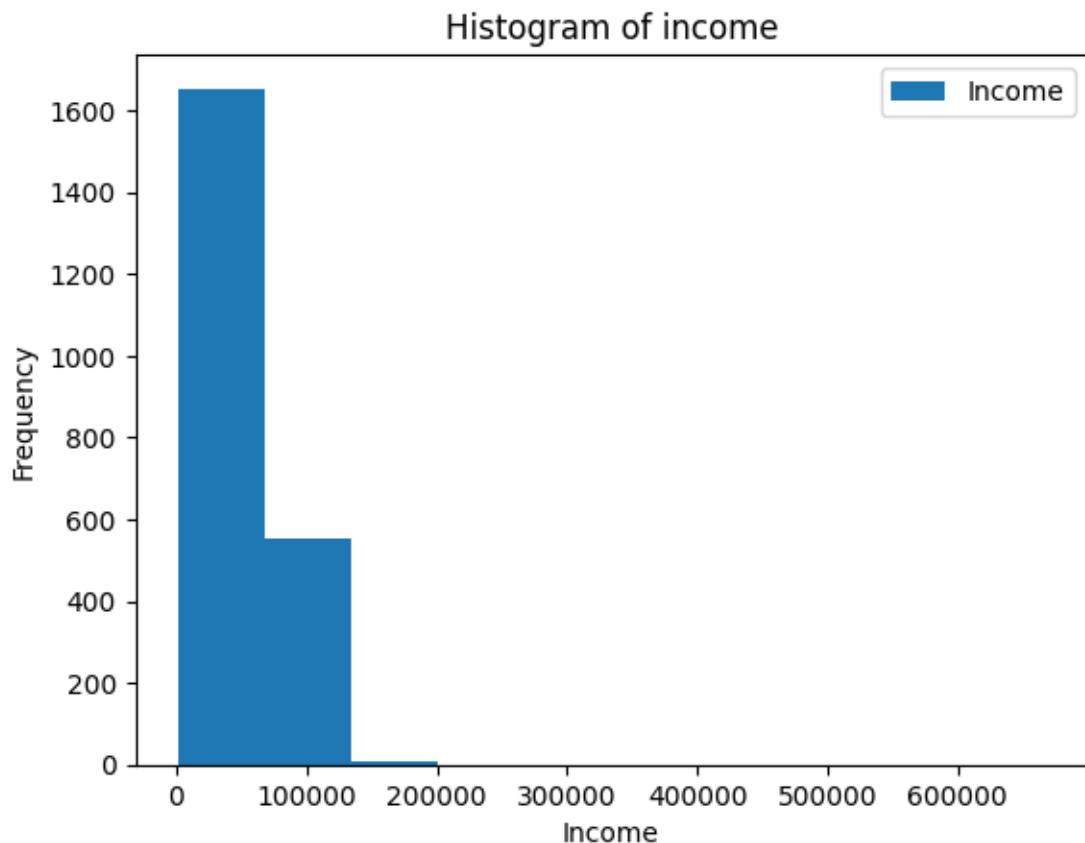


Figure 8.1.: png

### Task

This doesn't look very good because the x-axis extends so much!

## 8. Introduction to Python and Pandas

- Looking at the data, can you figure out what might cause this?  
Solution

When we look at the highest earners, we see that somebody put `666666` as their income.

We can assume that this was put as a joke or is an outlier.

In either way, we can redo the plot with that datapoint removed.

- Can you “fix” the plot?  
Solution

```
ax = merged_df[merged_df["Income"] != 666666].plot(kind="hist",y="Income")
ax.set_xlabel("Income")
ax.set_title("Fixed Histogram of income")
```

Text(0.5, 1.0, ‘Fixed Histogram of income’)

### 8.11.2. Bar plot

Another visualization we could do is a bar plot.

Using the `groupby` and `mean` methods, we can calculate the mean Income like we’ve learned before.

```
grouped_by_education = merged_df.groupby(by="Education")["Income"].mean()
grouped_by_education
```

```
Education
2n Cycle      47633.190000
Basic          20306.259259
Graduation     52720.373656
Master         52917.534247
PhD            56145.313929
Name: Income, dtype: float64
```

Now, this data can be shown:

```
ax = grouped_by_education.plot(kind="bar")
ax.set_ylabel("Mean income")
ax.set_title("Mean income for each education level")
```

Text(0.5, 1.0, ‘Mean income for each education level’)

*8. Introduction to Python and Pandas*

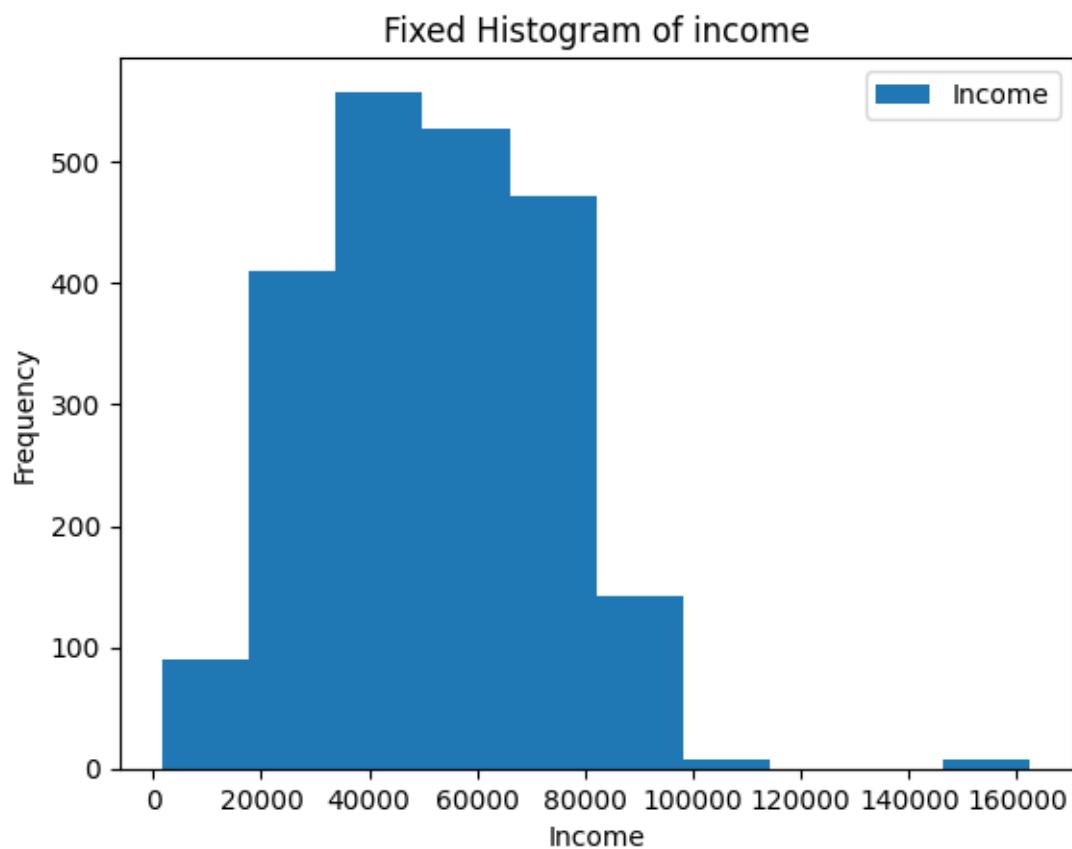


Figure 8.2.: png

*8. Introduction to Python and Pandas*

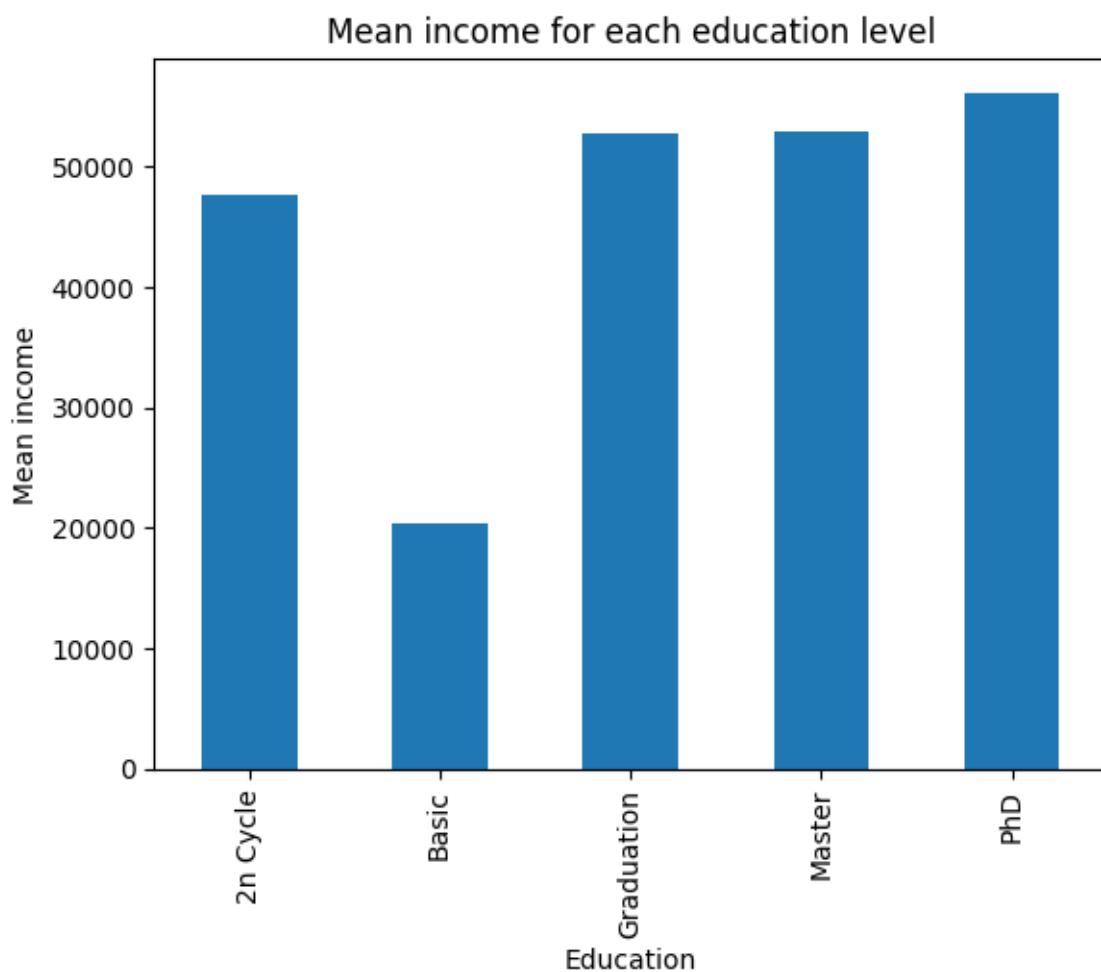


Figure 8.3.: png

## 8. Introduction to Python and Pandas

### 8.11.3. Scatter plot

Another kind of plot is the scatter plot, which needs two columns for the **x** and **y** axis.

```
ax = df.plot(kind="scatter", x="MntWines", y="MntFruits")
ax.set_title("Wine purchases and Fruit purchases")
```

Text(0.5, 1.0, 'Wine purchases and Fruit purchases')

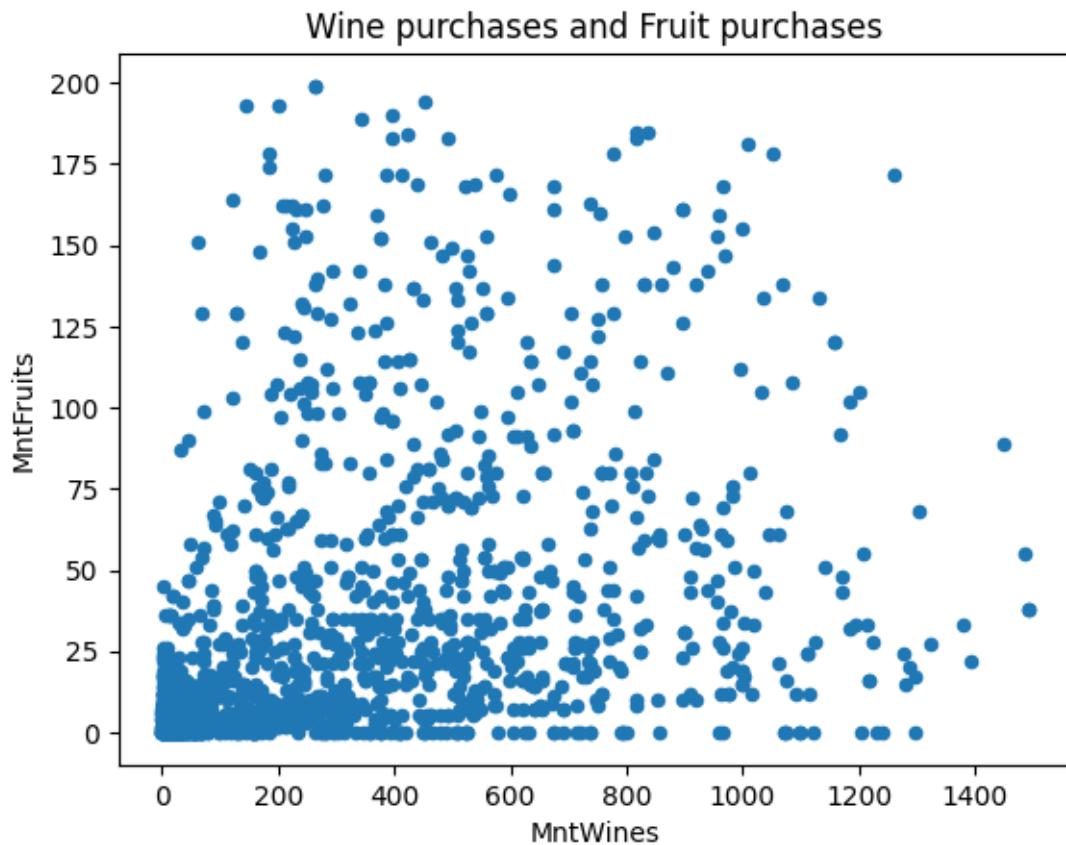


Figure 8.4.: png

You can also specify whether the axes should be on the log scale or not.

```
ax = df.plot(kind="scatter", x="MntWines", y="MntFruits", logy=True, logx=True)
ax.set_title("Wine purchases and Fruit purchases, on log scale")
```

Text(0.5, 1.0, 'Wine purchases and Fruit purchases, on log scale')

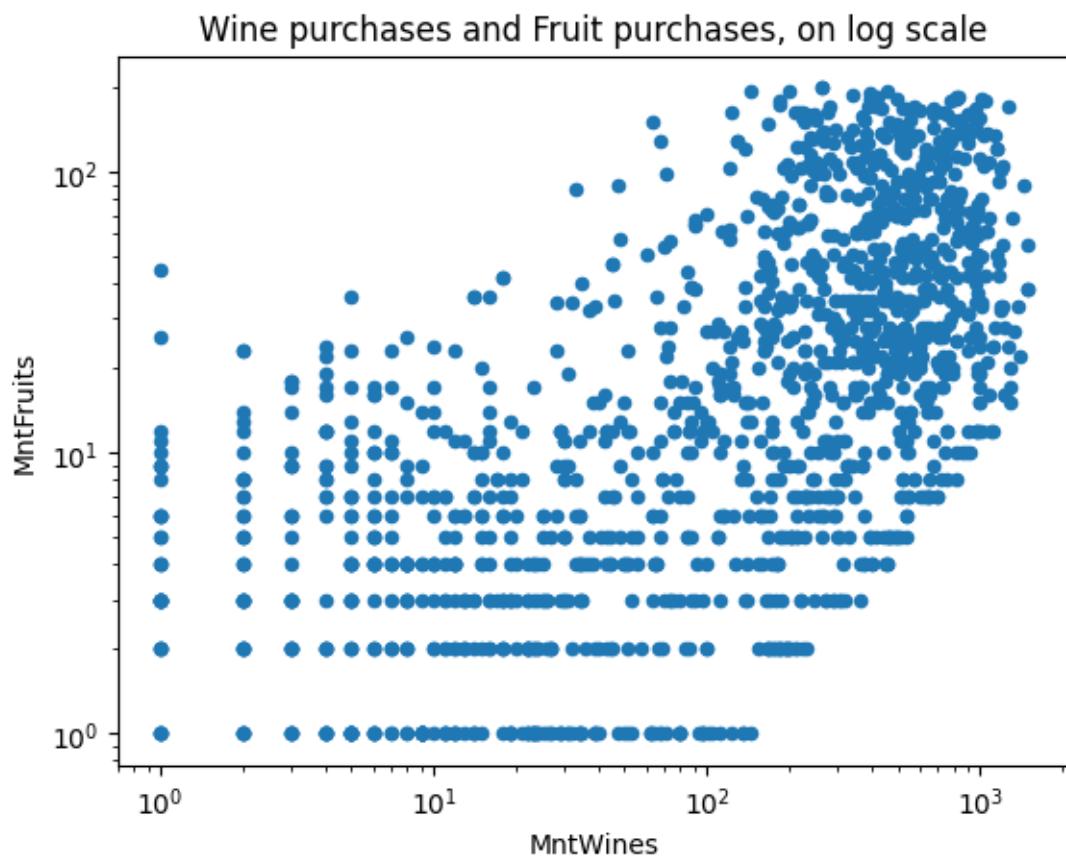


Figure 8.5.: png

## 8.12. Plotnine

Plotnine is the Python clone of ggplot2, which is very powerful and is great if you are already familiar with the ggplot2 syntax!

```
from plotnine import *
```

```
(ggplot(merged_df, aes("Education", "MntWines", fill="Education"))
 + geom_boxplot(alpha=0.8))
```

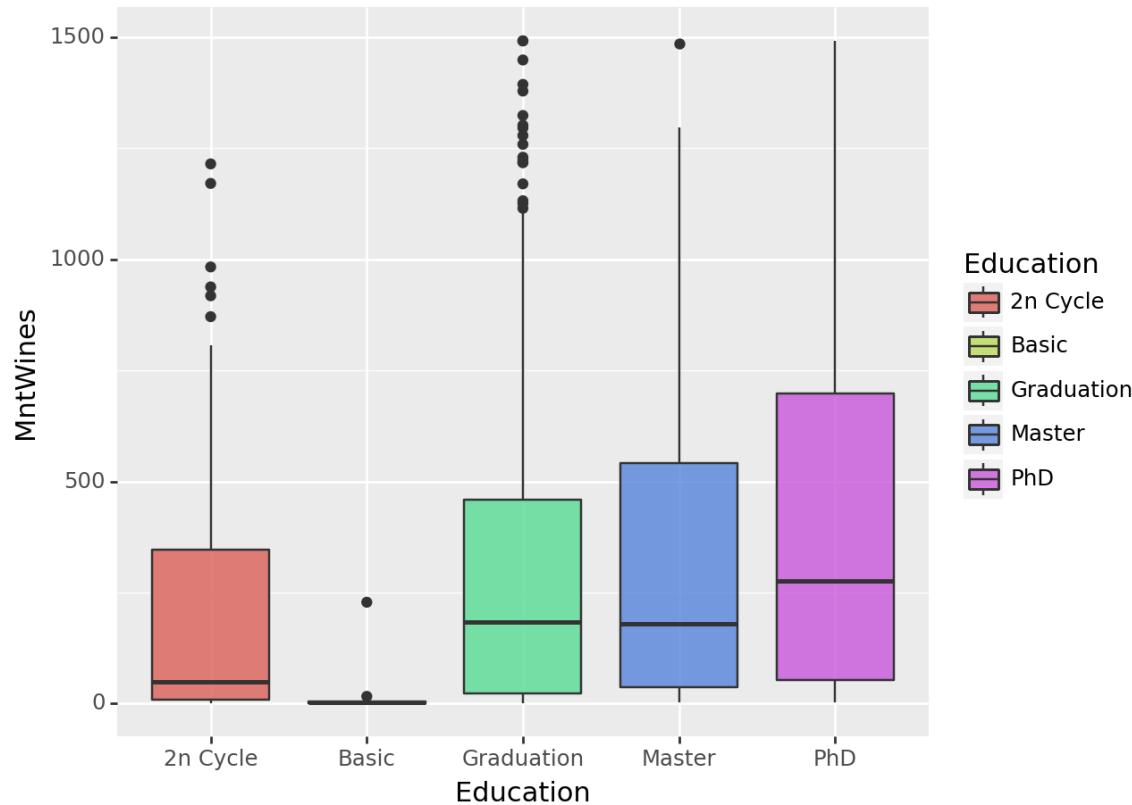


Figure 8.6.: png

```
(ggplot(merged_df[(merged_df["Year_Birth"]>1900) & (merged_df["Income"]!=666666)],
       aes("Year_Birth", "Income", fill="Education"))
```

## 8. Introduction to Python and Pandas

```
+ geom_point(alpha=0.5, stroke=0)
+ facet_wrap("Marital_Status"))
```

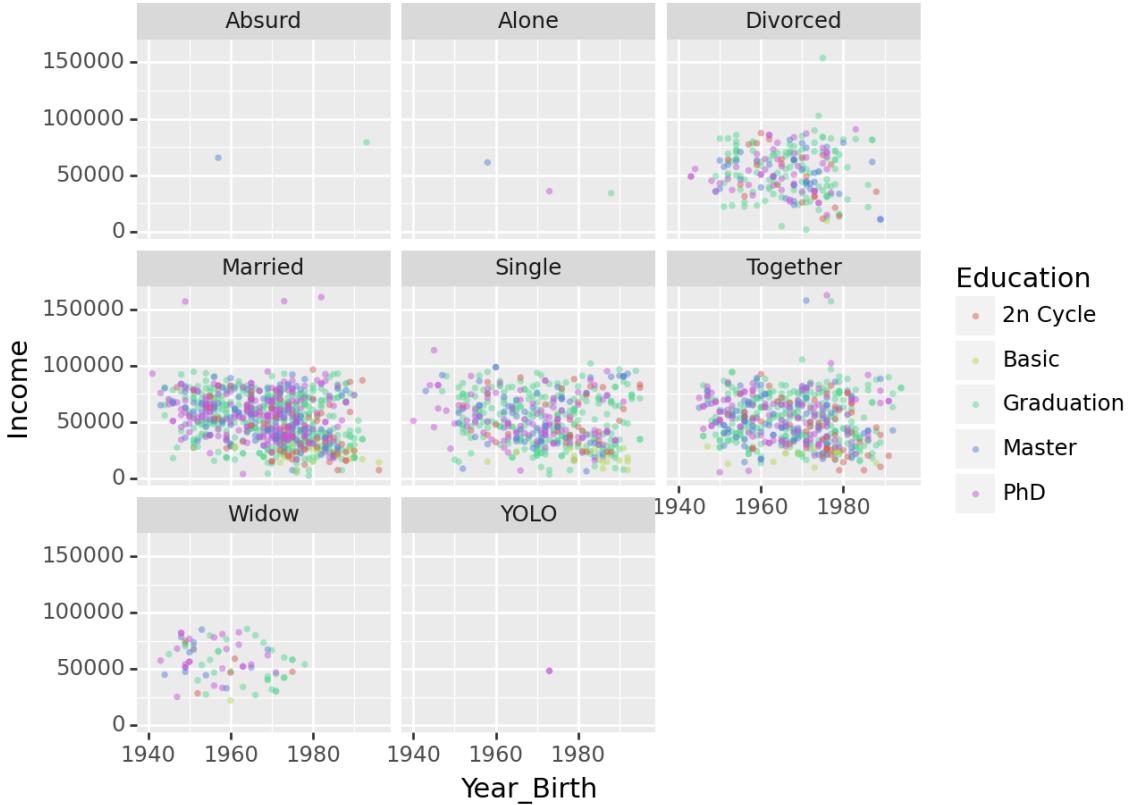


Figure 8.7.: png

### task

Now that you are familiar with python, pandas, and plotting. There are two data.tables from **AncientMetagenomeDir** which contains metadata from metagenomes. You should, by using the code in the tutorial be able to explore the datasets and make some fancy plots.

```
file names:
sample_table_url
library_table_url
```

# 9. Introduction to Git(Hub)

## 9.1. Introduction

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



### Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate git-github
```

## 9.2. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

## 9.3. 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 **Introduction** above).

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

## 9. Introduction to Git(Hub)

```
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_ed25519
```

Next, open your webbrowser and navigate to your github account. Go to settings -> SSH & GPG Keys -> New SSH Key.

Here, give your key a title, and then paste the *public* key into the main text box that you just generated on your local machine - i.e., the *whole* string (starting `ssh-ed` and ending in your email address) when you run the following command:

```
cat ~/.ssh/id_ed25519.pub
```

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>.

## 9.4. The only 6 commands you really need to know

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

**Create a new repository**

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

---

**Owner \***      **Repository name \***

 meganemichel /

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

**Description (optional)**

---

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

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

---

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

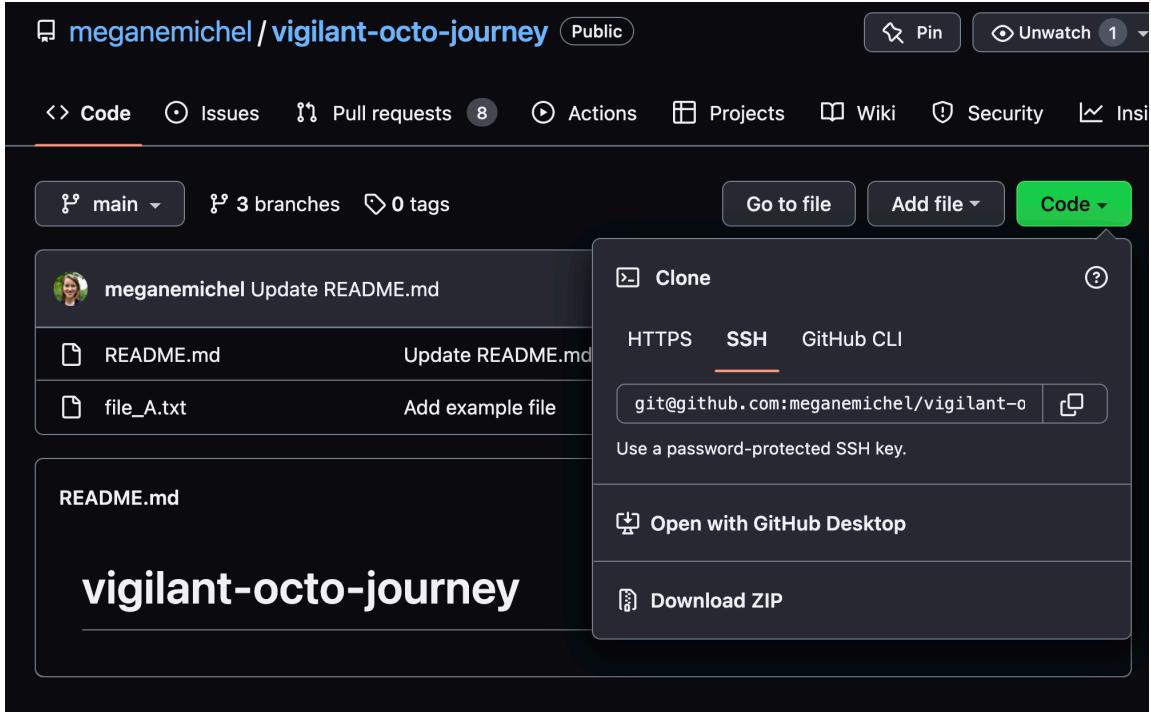
[Add a README file](#)

 Note

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

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

## 9. Introduction to Git(Hub)



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.

## 9. Introduction to Git(Hub)

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

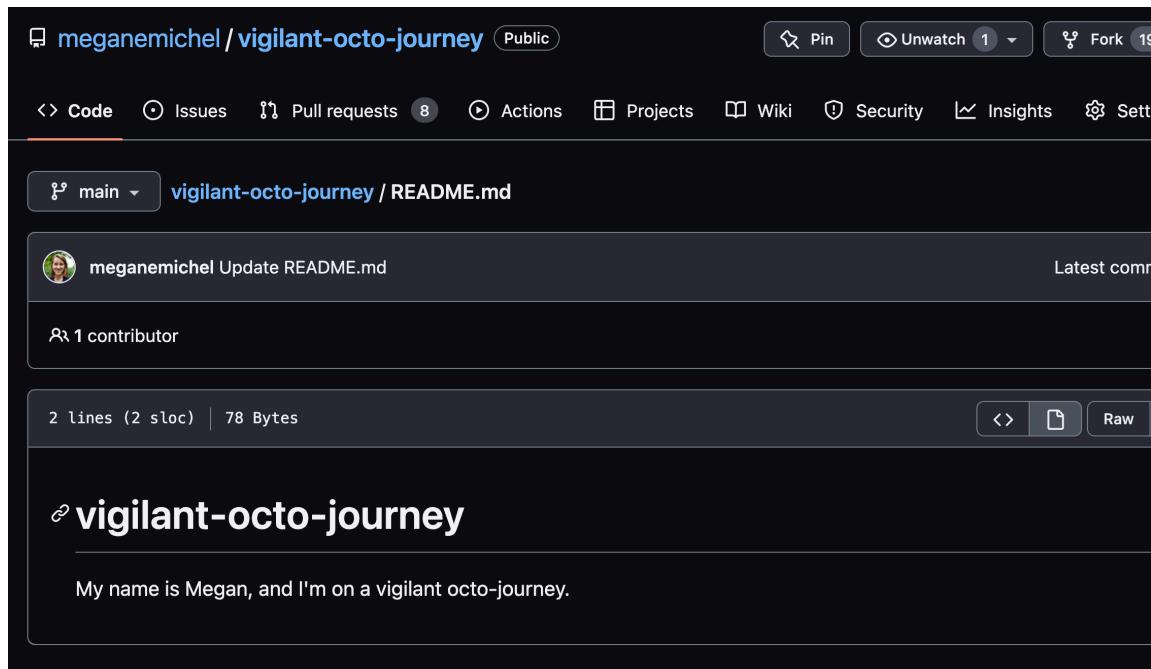
Finally, let's **push** our local commit back to our remote repository.

```
git push
```

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

```
git pull
```

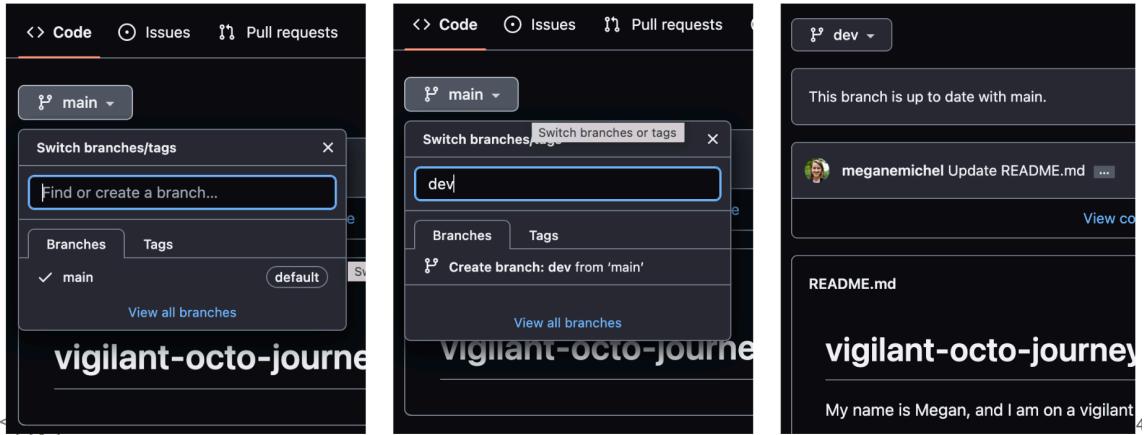
You should see that your repository is already up-to-date, since we have not made new changes to the remote repo. Let's try making a change to the remote repository's README file (as below). Then, back on the command line, pull the repository again.



## 9.5. 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.

## 9. Introduction to Git(Hub)



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!

## 9.6. Pull requests

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

## 9.7. Questions to think about

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

**Part III.**

**Ancient Metagenomics**

## *Taxonomic Profiling*

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

## **Taxonomic Profiling**

*TBC*

## **Functional Profiling**

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

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

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

## ***De novo Assembly***

*De novo* assembly of ancient metagenomic samples enables the recovery of the genetic information of organisms without requiring any prior knowledge about their genomes. Therefore,

## De novo Assembly

this approach is very well suited to study the biological diversity of species that have not been studied well or are simply not known yet.

In this chapter, we will show you how to prepare your sequencing data and subsequently *de novo* assemble them. Furthermore, we will then learn how we can actually evaluate what organisms we might have assembled and whether we obtained enough data to reconstruct a whole metagenome-assembled genome. We will particularly focus on the quality assessment of these reconstructed genomes and how we can ensure that we obtained high-quality genomes.

# 10. Taxonomic Profiling, OTU Tables and Visualisation

## 💡 Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate taxonomic-profiling
```

## 10.1. Introduction

In this chapter, we're going to look at taxonomic profiling, or in other words, how to get the microbial composition of a sample from the DNA sequencing data.

Though there are many algorithms, and even more different tools available to perform taxonomic profiling, the general idea remains the same (Figure 10.1).

After cleaning up the sequencing data, generally saved as FASTQ files, a **taxonomic profiler** is used to compare the sequenced DNA to a reference database of sequences from known organisms, in order to generate a taxonomic profile of all organisms identified in a sample (Figure 10.1)

If you prefer text instead of pictograms, the workflow we're going to cover today is outlined in the figure below, adapted from @sharptonIntroductionAnalysisShotgun2014

Because different organisms can possess the same DNA, especially when looking at shorter sequences, taxonomic profilers need to have a way to resolve the ambiguity in the taxonomic assignation (Figure 10.3).

By leveraging an algorithm known as the Lowest Common Ancestor (LCA), and the taxonomic tree of all known species, ambiguities are going to be resolved by assigning a higher, less precise, taxonomic rank to ambiguous matches(Figure 10.4).

## How do we analyze ancient microbiomes ?

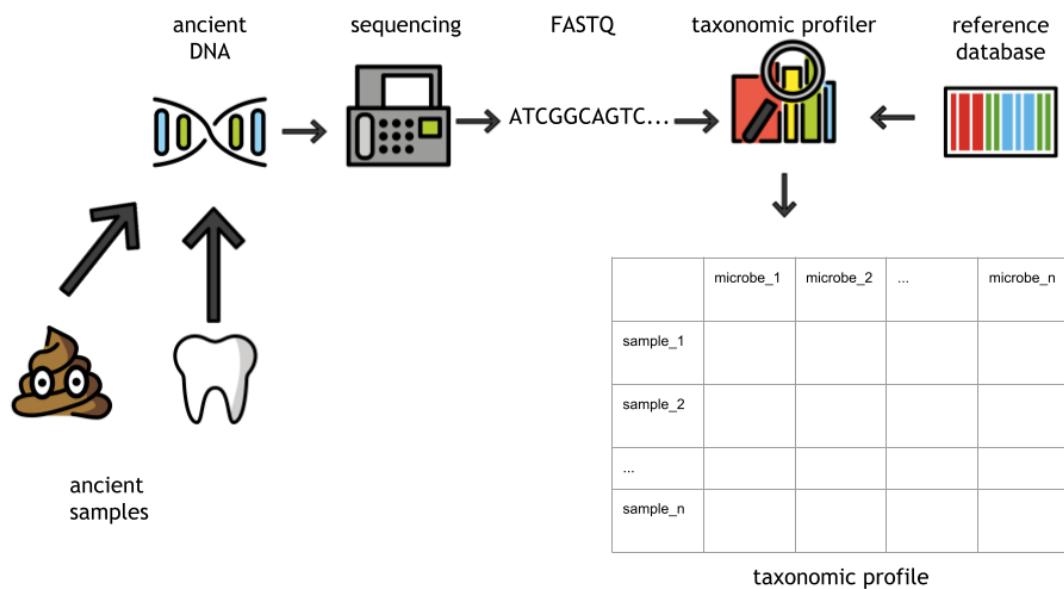


Figure 10.1.: General overview of a taxonomic profiling analysis workflow

## 10. Taxonomic Profiling, OTU Tables and Visualisation

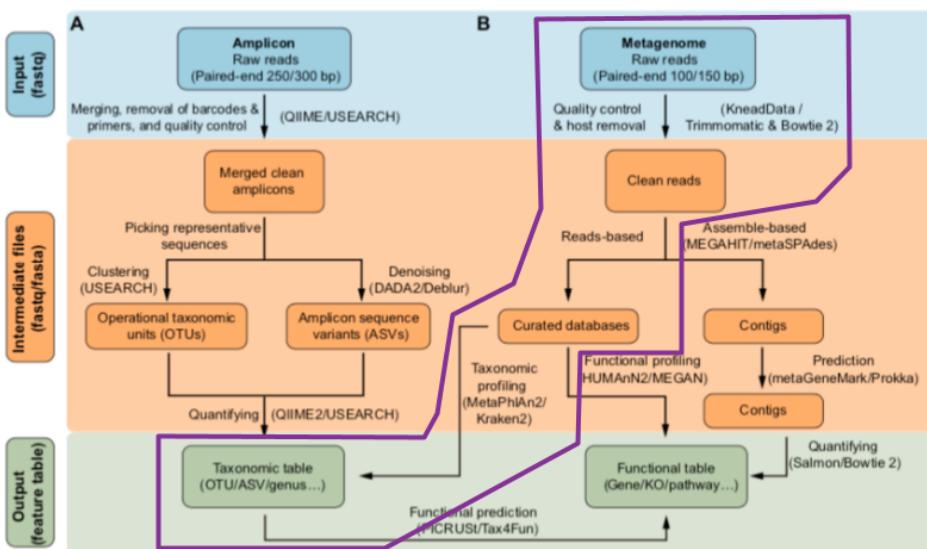


Figure 10.2.: A typical metagenomics analysis workflow, adapted from @sharptonIntroductionAnalysisShotgun2014

## Ambiguity in taxonomic assignation

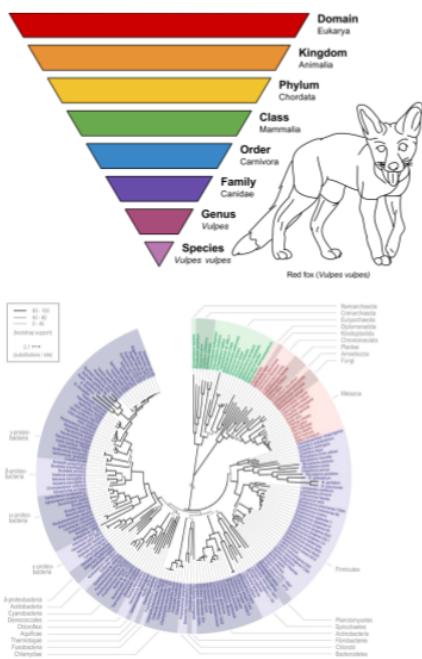
sequence 1 ATGGTCGGGCAGGACGTTGCGAGT

sequence 2 CGAGAAAGGCAGGACGCCACGTAC



Figure 10.3.: Different species can share the same DNA sequence

## Taxonomy and LCA to the rescue



- Species level assignation is not always possible.
- Possibility of hits in different species
- Ambiguities solved by LCA (Lowest Common Ancestor) algorithm.

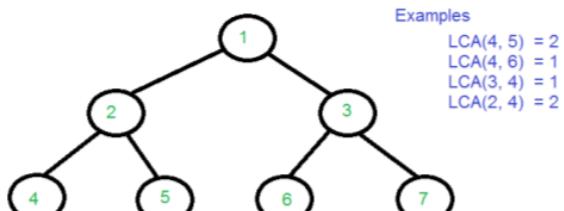


Figure 10.4.: A diagram of the LCA algorithm, a way to resolve these ambiguities

## 10.2. Chapter Overview

Today, we're going to use the following tools:

- fastp [@chenFastpUltrafastAllinone2018] for sequencing data cleaning
- MetaPhlAn [@segata2012metagenomic], for taxonomic profiling
- Krona [@ondovInteractiveMetagenomicVisualization2011] and Pavian [@breitwieser-PavianInteractiveAnalysis2016] for the interactive exploration of the taxonomic profiles
- curatedMetagenomicData [@pasolliAccessibleCuratedMetagenomic2017] for retrieving modern comparison data
- Python, pandas [@rebackPandasdevPandasPandas2022], plotnine , and scikit-bio [@scikit-bioScikitbioBioinformaticsLibrary2022] to perform exploratory data analysis and a bit of microbial ecology

to explore a toy dataset that has already been prepared for you.

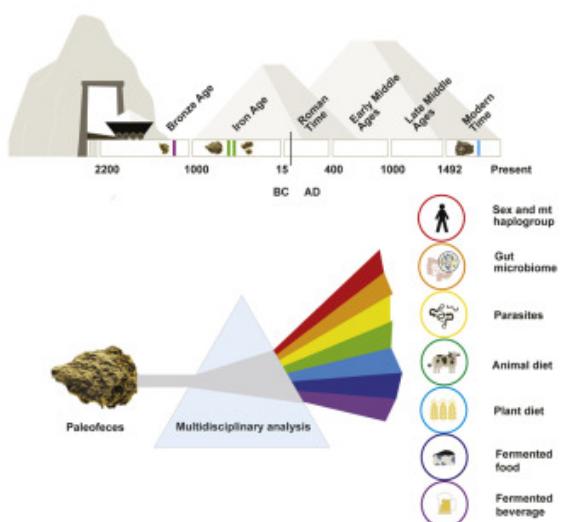
**i** Preamble: what has been done to generate this toy dataset

### 10.2.1. Download and Subsample

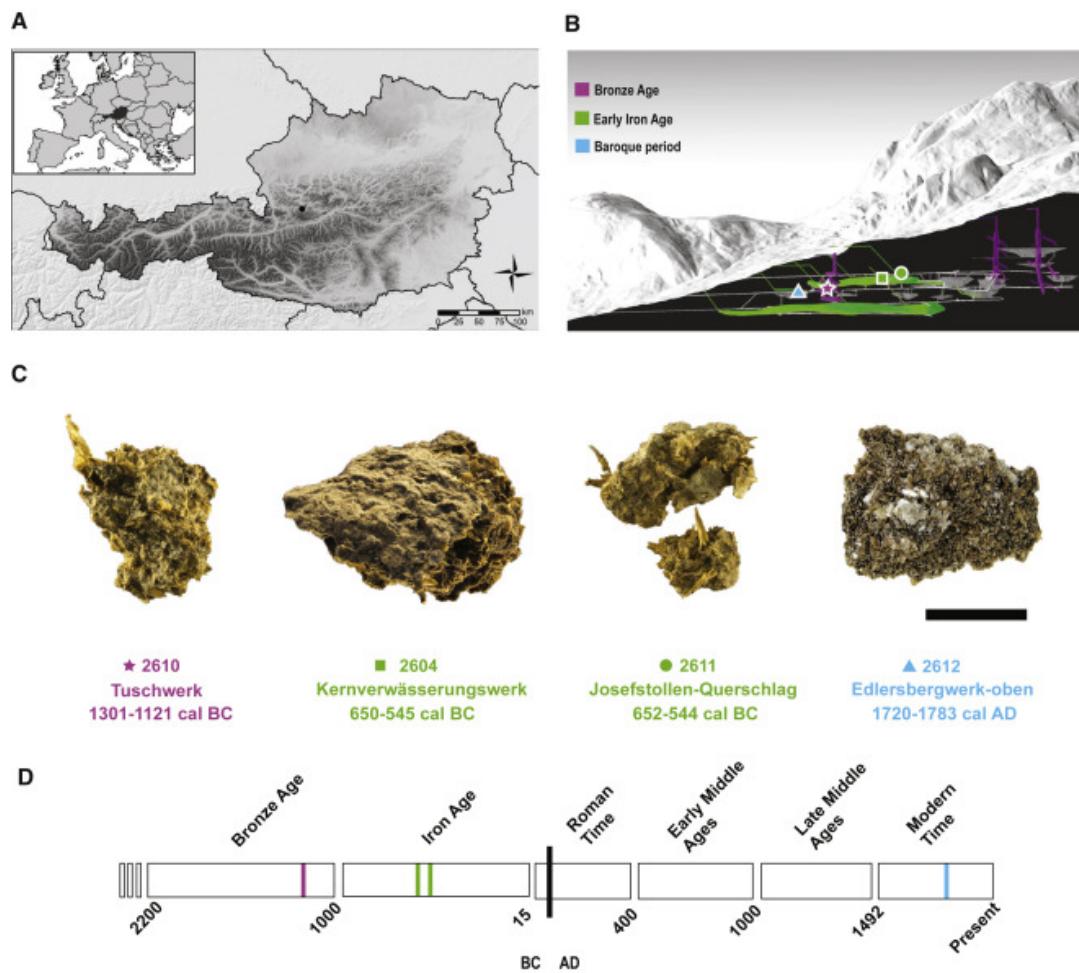
```
import subprocess
import glob
from pathlib import Path
```

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

## 10. Taxonomic Profiling, OTU Tables and Visualisation



## 10. Taxonomic Profiling, OTU Tables and Visualisation



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

This Python code defines a function called `subsample` that takes in a FASTQ file name, an output directory, and a depth value (defaulting to 1000000). The function uses the `seqtk` command-line tool to subsample the input FASTQ file to the desired depth and saves the output to a new file in the specified output directory. The function prints the constructed command string to the console for debugging purposes.

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

This Python code uses a for loop to iterate over all the files in the `../data/raw/` directory that match the pattern `*`, and calls the `subsample` (defined above) function on each file in the directory `../data/subsampled`.

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

 Expand to see command output

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

Finally, we compress all files to gzip format

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

### 10.3. Data pre-processing

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

To do so, I've used the first steps of the nf-core/eager pipeline, more information of which can be found in the Ancient Metagenomic Pipelines chapter.

I've already done some pre-processed the data, and the resulting cleaned files are available in the `data/eager_cleaned/` but the basic eager command to do so is:

```
nextflow run nf-core/eager \
-r 2.4.7 \
-profile <docker/singularity/podman/conda/institute> \
```

```
--input '*_R{1,2}.fastq.gz' \
--fasta 'human_genome.fasta' \
--hostremoval_input_fastq
```

## 10.4. Adapter sequence trimming and low-quality bases trimming

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

To perform both of these tasks, we'll use the program fastp.

The following command gets you the help of fastp (the `--help` option is a common option in command-line tools that displays a list of available options and their descriptions).

```
fastp -h
```

?

Expand to see command output

```
option needs value: --html
usage: fastp [options] ...
options:
  -i, --in1                               read1 input file name (string [=])
  -o, --out1                              read1 output file name (string [=])
  -I, --in2                               read2 input file name (string [=])
  -O, --out2                              read2 output file name (string [=])
  --unpaired1                            for PE input, if read1 passed QC but read2 not, it
                                         Default is to discard it. (string [=])
  --unpaired2                            for PE input, if read2 passed QC but read1 not, it
                                         If --unpaired2 is same as --unpaired1 (default mode)
                                         written to this same file. (string [=])
  --overlapped_out                       for each read pair, output the overlapped region in
                                         base. (string [=])
  --failed_out                           specify the file to store reads that cannot pass t
                                         for paired-end input, merge each pair of reads into
                                         overlapped. The merged reads will be written
```

```

--merged_out
--include_unmerged
-6, --phred64
-z, --compression
--stdin
--stdout
--interleaved_in
--reads_to_process
--dont_overwrite
--fix_mgi_id
-V, --verbose
-A, --disable_adapter_trimming
-a, --adapter_sequence
--adapter_sequence_r2
--adapter_fasta
--detect_adapter_for_pe
-f, --trim_front1
-t, --trim_tail1
-b, --max_len1
-F, --trim_front2
-T, --trim_tail2
-B, --max_len2
-D, --dedup
--dup_calc_accuracy
--dont_eval_duplication
-g, --trim_poly_g

```

to the file given by --merged\_out, the unmerged files specified by --out1 and --out2. The merging in the merging mode, specify the file name to store --stdout to stream the merged output (string [=]) in the merging mode, write the unmerged or unpaired by --merge. Disabled by default.

indicate the input is using phred64 scoring (it'll so the output will still be phred33) compression level for gzip output (1 ~ 9). 1 is fast input from STDIN. If the STDIN is interleaved pair stream passing-filters reads to STDOUT. This option FASTQ output for paired-end output. Disabled by default indicate that <in1> is an interleaved FASTQ which Disabled by default.

specify how many reads/pairs to be processed. Default don't overwrite existing files. Overwriting is allowed if the MGI FASTQ ID format is not compatible with many output verbose log information (i.e. when every 1M adapter trimming is enabled by default. If this option is used, it will trim the adapter for read1. For SE data, if not specified, it will trim the adapter for read1. For PE data, this is used if R1/R2 are found not otherwise, it will trim the adapter for read2 (PE data only). This is used If not specified, it will be the same as <adapter\_sequence>. If specified, it will trim both read1 and read2 by default, the auto-detection for adapter is forced by this option to enable it for PE data.

trimming how many bases in front for read1, default trimming how many bases in tail for read1, default if read1 is longer than max\_len1, then trim read1 long as max\_len1. Default 0 means no limitation (i.e. trimming how many bases in front for read2. If it's not specified, it will trim the adapter for read2. If it's specified, it will trim both read1 and read2 by default, the auto-detection for adapter is forced by this option to enable it for PE data.

trimming how many bases in front for read2. If it's not specified, it will trim the adapter for read2. If it's specified, it will trim both read1 and read2 by default, the auto-detection for adapter is forced by this option to enable it for PE data.

Default 0 means no limitation. If it's not specified, it will enable deduplication to drop the duplicated reads/sequences. accuracy level to calculate duplication (1~6), high for no-dedup mode, and 3 for dedup mode. Default 1 for no-dedup mode, and 3 for dedup mode. --dont\_eval\_duplication don't evaluate duplication rate to save time and use less memory. --trim\_poly\_g force polyG tail trimming, by default trimming is disabled.

```

--poly_g_min_len
-G, --disable_trim_poly_g
-x, --trim_poly_x
--poly_x_min_len
-5, --cut_front

-3, --cut_tail

-r, --cut_right

-W, --cut_window_size
-M, --cut_mean_quality

--cut_front_window_size
--cut_front_mean_quality
--cut_tail_window_size
--cut_tail_mean_quality
--cut_right_window_size
--cut_right_mean_quality
-Q, --disable_quality_filtering
-q, --qualified_quality_phred
-u, --unqualified_percent_limit
-n, --n_base_limit
-e, --average_qual

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

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

--overlap_diff_limit

```

the minimum length to detect polyG in the read tail  
 disable polyG tail trimming, by default trimming is  
 enable polyX trimming in 3' ends.  
 the minimum length to detect polyX in the read tail  
 move a sliding window from front (5') to tail, drop  
 its mean quality < threshold, stop otherwise.  
 move a sliding window from tail (3') to front, drop  
 its mean quality < threshold, stop otherwise.  
 move a sliding window from front to tail, if meet  
 < threshold, drop the bases in the window and the  
 the window size option shared by cut\_front, cut\_ta  
 the mean quality requirement option shared by cut\_  
 Range: 1~36 default: 20 (Q20) (int [=20])  
 the window size option of cut\_front, default to cu  
 the mean quality requirement option for cut\_front,  
 the window size option of cut\_tail, default to cu  
 the mean quality requirement option for cut\_tail,  
 the window size option of cut\_right, default to cu  
 the mean quality requirement option for cut\_right,  
 quality filtering is enabled by default. If this o  
 the quality value that a base is qualified. Defaul  
 how many percents of bases are allowed to be unqua  
 if one read's number of N base is >n\_base\_limit, t  
 if one read's average quality score <avg\_qual, the  
 Default 0 means no requirement (int [=0])  
 length filtering is enabled by default. If this op  
 reads shorter than length\_required will be discarded  
 reads longer than length\_limit will be discarded,  
 enable low complexity filter. The complexity is de  
 that is different from its next base (base[i] != b  
 the threshold for low complexity filter (0~100). D  
 specify a file contains a list of barcodes of inde  
 specify a file contains a list of barcodes of inde  
 the allowed difference of index barcode for index  
 enable base correction in overlapped regions (only  
 the minimum length to detect overlapped region of  
 adapter trimming and correction. 30 by default. (i  
 the maximum number of mismatched bases to detect o  
 This will affect overlap analysis based PE merge,

## 10. Taxonomic Profiling, OTU Tables and Visualisation

```
--overlap_diff_percent_limit
-U, --umi
--umi_loc
--umi_len
--umi_prefix

--umi_skip
-p, --overrepresentation_analysis
-P, --overrepresentation_sampling

-j, --json
-h, --html
-R, --report_title
-w, --thread
-s, --split

-S, --split_by_lines

-d, --split_prefix_digits

--cut_by_quality5
--cut_by_quality3
--cut_by_quality_aggressive
--discard_unmerged
-?, --help
```

the maximum percentage of mismatched bases to determine if two reads are overlapping. This will affect overlap analysis based PE merge, enable unique molecular identifier (UMI) preprocessing. UMI is a unique identifier for each molecule. If the UMI is in read1/read2, its length should be specified. If the UMI is in read1/read2, its length should be specified, an underline will be used to connect the UMI (e.g., prefix=UMI, UMI=AATTCTG, final=UMI\_AATTCTG). No prefix is required if the UMI is in read1/read2, fastp can skip several steps to save time. If enabled, overrepresented sequence analysis will be performed. One in (--overrepresentation\_sampling) reads will be analyzed (1~10000), smaller is slower, default is 1000. The json format report file name (string [=fastp.json]), the html format report file name (string [=fastp.html]), should be quoted with ' or ", default is "fastp report.html". worker thread number, default is 3 (int [=3]). split output by limiting total split file number will be added to output name ( 0001.out fq, 0002.out fq...). split output by limiting lines of each file with the digits for the sequential number padding (1~10000). 0001.xxx, 0 to disable padding (int [=4]). DEPRECATED, use --cut\_front instead. DEPRECATED, use --cut\_tail instead. DEPRECATED, use --cut\_right instead. DEPRECATED, no effect now, see the introduction for print this message.

Here we use fastp to preprocess a pair of FASTQ files. The code specifies the input files, merges the paired-end reads on their overlaps, removes duplicate reads, and generates JSON and HTML reports. The output files are saved in the `../results/fastp/` directory.

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

## 10. Taxonomic Profiling, OTU Tables and Visualisation

💡 Expand to see command output

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

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

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

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

Duplication rate: 0.2479%

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

```
Read pairs merged: 228  
% of original read pairs: 0.0228%  
% in reads after filtering: 0.0114339%
```

JSON report: ../results/fastp/ERR5766177.fastp.json

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

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

## 10.5. Taxonomic profiling with Metaphlan

MetaPhlAn is a computational tool for profiling the composition of microbial communities from metagenomic shotgun sequencing data.

```
metaphlan --help
```

💡 Expand to see command output

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

### DESCRIPTION

MetaPhlAn version 3.1.0 (25 Jul 2022):

METAgenomic PHylogenetic ANalysis for metagenomic taxonomic profiling.

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

#### COMMON COMMANDS

We assume here that MetaPhlAn is installed using the several options available (pip, conda).  
Also BowTie2 should be in the system path with execution and read permissions, and Perl

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

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

\* Profiling a metagenome from raw reads:

```
$ metaphlan metagenome.fastq --input_type fastq -o profiled_metagenome.txt
```

\* You can take advantage of multiple CPUs and save the intermediate BowTie2 output for reusing MetaPhlAn extremely quickly:

```
$ metaphlan metagenome.fastq --bowtie2out metagenome.bowtie2.bz2 --nproc 5 --input_type fastq
```

\* If you already mapped your metagenome against the marker DB (using a previous MetaPhlAn run) you can obtain the results in few seconds by using the previously saved --bowtie2out file specifying the input (--input\_type bowtie2out):

```
$ metaphlan metagenome.bowtie2.bz2 --nproc 5 --input_type bowtie2out -o profiled_metagenome.txt
```

\* bowtie2out files generated with MetaPhlAn versions below 3 are not compatible.

Starting from MetaPhlAn 3.0, the BowTie2 ouput now includes the size of the profiled metagenome.

If you want to re-run MetaPhlAn using these file you should provide the metagenome size:

```
$ metaphlan metagenome.bowtie2.bz2 --nproc 5 --input_type bowtie2out --nreads 520000 -o profiled_metagenome.txt
```

\* You can also provide an externally BowTie2-mapped SAM if you specify this format with --input\_type. Two steps: first apply BowTie2 and then feed MetaPhlAn with the obtained SAM.

```
$ bowtie2 --sam-no-hd --sam-no-sq --no-unal --very-sensitive -S metagenome.sam -x \${mpa_dir}/metaphlan_databases/mpa_v30_CHOCOPhla_201901 -U metagenome.fastq
```

```
$ metaphlan metagenome.sam --input_type sam -o profiled_metagenome.txt
```

## 10. Taxonomic Profiling, OTU Tables and Visualisation

- \* We can also natively handle paired-end metagenomes, and, more generally, metagenomes spanning multiple files (but you need to specify the --bowtie2out parameter):  
\$ metaphlan metagenome\_1.fastq,metagenome\_2.fastq --bowtie2out metagenome.bowtie2.bz2 --n

---

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

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

- \* The following command will output the abundance of each marker with a RPK (reads per kilobase per cell) value higher than 0.0. (we are assuming that metagenome\_outfmt.bz2 has been generated before as shown above).  
\$ metaphlan -t marker\_ab\_table metagenome\_outfmt.bz2 --input\_type bowtie2out -o marker\_ab\_table.out  
The obtained RPK can be optionally normalized by the total number of reads in the metafile to guarantee fair comparisons of abundances across samples. The number of reads in the sample needs to be passed with the '--nreads' argument
- \* The list of markers present in the sample can be obtained with '-t marker\_pres\_table'  
\$ metaphlan -t marker\_pres\_table metagenome\_outfmt.bz2 --input\_type bowtie2out -o marker\_pres\_table.out  
The --pres\_th argument (default 1.0) set the minimum RPK value to consider a marker present.
- \* The list '-t clade\_profiles' analysis type reports the same information of '-t marker\_pres\_table' but the markers are reported on a clade-by-clade basis.  
\$ metaphlan -t clade\_profiles metagenome\_outfmt.bz2 --input\_type bowtie2out -o marker\_abundance.out
- \* Finally, to obtain all markers present for a specific clade and all its subclades, the '-t clade\_specific\_strain\_tracker' should be used. For example, the following command is reporting the presence/absence of the markers for the *B. fragilis* species and its subclades. The optional argument --min\_ab specifies the minimum clade abundance for reporting the markers.  
\$ metaphlan -t clade\_specific\_strain\_tracker --clade s\_\_Bacteroides\_fragilis metagenome\_outfmt.bz2 --min\_ab 1

```
bowtie2out -o marker_abundance_table.txt
```

---

positional arguments:

|             |  |
|-------------|--|
| INPUT_FILE  | the input file can be:<br>* a fastq file containing metagenomic reads<br>OR<br>* a BowTie2 produced SAM file.<br>OR<br>* an intermediary mapping file of the metagenome generated by a p<br>If the input file is missing, the script assumes that the input i<br>input, or named pipes.<br>IMPORTANT: the type of input needs to be specified with --input_t |
| OUTPUT_FILE | the tab-separated output file of the predicted taxon relative abu<br>[stdout if not present]   |

Required arguments:

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

Mapping arguments:

|                                   |  |
|-----------------------------------|--|
| --force                           | Force profiling of the input file by removing the bowtie2out file  |
| --bowtie2db METAPHPLAN_BOWTIE2_DB | Folder containing the MetaPhlAn database. You can specify the loc<br>DEFAULT_DB_FOLDER variable in the shell.<br>[default /Users/maxime/mambaforge/envs/summer_school_microbiome/l   |
| -x INDEX, --index INDEX           | Specify the id of the database version to use. If "latest", MetaP<br>If an index name is provided, MetaPhlAn will try to use it, if av<br>If the database files are not found on the local MetaPhlAn instal<br>will be automatically downloaded [default latest] |
| --bt2_ps BowTie2 presets          | Presets options for BowTie2 (applied only when a FASTA file is pr<br>The choices enabled in MetaPhlAn are:<br>* sensitive<br>* very-sensitive<br>* sensitive-local   |

## 10. Taxonomic Profiling, OTU Tables and Visualisation

```
* very-sensitive-local  
[default very-sensitive]  
--bowtie2_exe BOWTIE2_EXE  
Full path and name of the BowTie2 executable. This option allows M  
executable even when it is not in the system PATH or the system P  
--bowtie2_build BOWTIE2_BUILD  
Full path to the bowtie2-build command to use, deafult assumes th  
--bowtie2out FILE_NAME  
The file for saving the output of BowTie2  
--min_mapq_val MIN_MAPQ_VAL  
Minimum mapping quality value (MAPQ) [default 5]  
--no_map  
Avoid storing the --bowtie2out map file  
--tmp_dir  
The folder used to store temporary files [default is the OS depen  
  
Post-mapping arguments:  
--tax_lev TAXONOMIC_LEVEL  
The taxonomic level for the relative abundance output:  
'a' : all taxonomic levels  
'k' : kingdoms  
'p' : phyla only  
'c' : classes only  
'o' : orders only  
'f' : families only  
'g' : genera only  
's' : species only  
[default 'a']  
--min_cu_len  
minimum total nucleotide length for the markers in a clade for  
estimating the abundance without considering sub-clade abundances  
[default 2000]  
--min_alignment_len  
The sam records for aligned reads with the longest subalignment  
length smaller than this threshold will be discarded.  
[default None]  
--add_viruses  
Allow the profiling of viral organisms  
--ignore_eukaryotes  
Do not profile eukaryotic organisms  
--ignore_bacteria  
Do not profile bacterial organisms  
--ignore_archaea  
Do not profile archeal organisms  
--stat_q  
Quantile value for the robust average  
[default 0.2]  
--perc_nonzero  
Percentage of markers with a non zero relative abundance for misi
```

## 10. Taxonomic Profiling, OTU Tables and Visualisation

```
--ignore_markers IGNORE_MARKERS
    File containing a list of markers to ignore.

--avoid_disqm
    Deactivate the procedure of disambiguating the quasi-markers based
    on marker abundance pattern found in the sample. It is generally recommended
    to keep the disambiguation procedure in order to minimize false positives.

--stat
    Statistical approach for converting marker abundances into clade
    averages:
        'avg_g' : clade global (i.e. normalizing all markers together) and
        'avg_l' : average of length-normalized marker counts
        'tavg_g' : truncated clade global average at --stat_q quantile
        'tavg_l' : truncated average of length-normalized marker counts
        'wavg_g' : winsorized clade global average (at --stat_q)
        'wavg_l' : winsorized average of length-normalized marker counts
        'med' : median of length-normalized marker counts
    [default tavg_g]

Additional analysis types and arguments:
-t ANALYSIS_TYPE      Type of analysis to perform:
    * rel_ab: profiling a metagenomes in terms of relative abundance
    * rel_ab_w_read_stats: profiling a metagenomes in terms of relative
                           abundance taking into account the number of reads coming from each clade
    * reads_map: mapping from reads to clades (only reads hitting a clade)
    * clade_profiles: normalized marker counts for clades with at least one read
    * marker_ab_table: normalized marker counts (only when > 0.0 and < 1.0)
    * marker_counts: non-normalized marker counts [use with extreme caution]
    * marker_pres_table: list of markers present in the sample (threshold)
    * clade_specific_strain_tracker: list of markers present for a specific clade
    [default 'rel_ab']

--nreads NUMBER_OF_READS
    The total number of reads in the original metagenome. It is used
    when -t marker_table is specified for normalizing the length-normalized
    marker counts with the metagenome size as well. No normalization applied if --nreads
    is not specified

--pres_th PRESENCE_THRESHOLD
    Threshold for calling a marker present by the -t marker_pres_table

--clade
    The clade for clade_specific_strain_tracker analysis

--min_ab
    The minimum percentage abundance for the clade in the clade_specific_strain_tracker analysis

Output arguments:
-o output file, --output_file output file
```

## 10. Taxonomic Profiling, OTU Tables and Visualisation

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

The following command uses MetaPhlAn to profile the taxonomic composition of the ERR5766177 metagenomic sample. The input file is specified as a merged FASTQ file, and the output is saved as a text file containing the taxonomic profile. The `--bowtie2out` option is used to specify the output file for the Bowtie2 alignment, and the `-nproc` option is used to specify the number of CPUs to use for the analysis.

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

The main results files that we're interested in is located at `../results/metaphlan/ERR5766177.metaphlan_profile.txt`

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

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

💡 Expand to see command output

```
#mpa_v30_CHOCOPhlaN_201901
#/home/maxime_borry/.conda/envs/maxime/envs/summer_school_microbiome/bin/metaphlan ../res
--input_type fastq --bowtie2out ../results/metaphlan/ERR5766177.bt2.out --nproc 8
#SampleID Metaphlan_Analysis
#clade_name NCBI_tax_id relative_abundance additional_species
k__Bacteria 2 82.23198
k__Archaea 2157 17.76802
k__Bacteria|p__Firmicutes 2|1239 33.47957
k__Bacteria|p__Bacteroidetes 2|976 28.4209
k__Bacteria|p__Actinobacteria 2|201174 20.33151
k__Archaea|p__Euryarchaeota 2157|28890 17.76802
```

## 10.6. Visualizing the taxonomic profile

### 10.6.1. Visualizing metaphlan taxonomic profile with Pavian

Pavian is a web-based tool for interactive visualization and analysis of metagenomics data. It provides a user-friendly interface for exploring taxonomic and functional profiles of microbial communities, and allows users to generate interactive plots and tables that can be customized and shared (Figure 10.5).

You can open Pavian in your browser by visiting [fibreitwieser.shinyapps.io/pavian](http://fibreitwieser.shinyapps.io/pavian).

💡 Expand for instructions for running yourself

There are different ways to run it:

- If you have docker installed

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

Then open your browser and visit localhost:5000

- If you are familiar with R

## 10. Taxonomic Profiling, OTU Tables and Visualisation

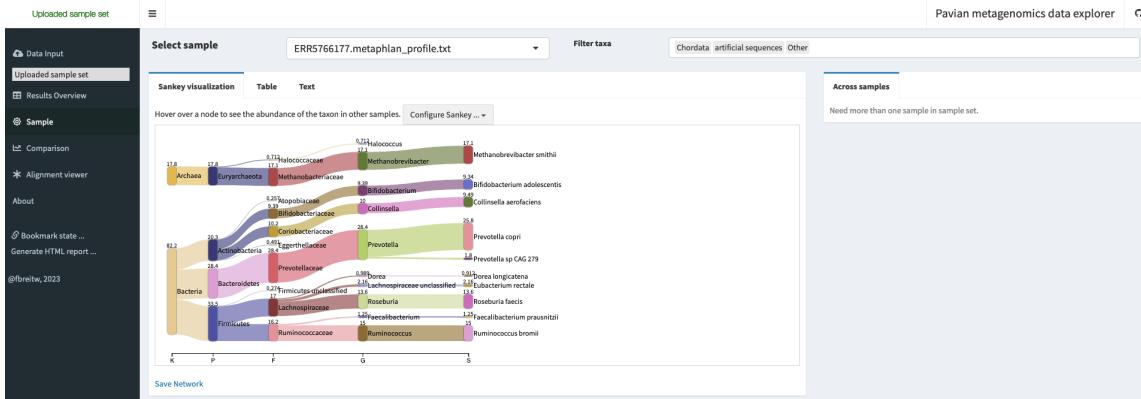


Figure 10.5.: Screenshot of the pavian metagenomics visualisation interface, with menus on the left, a select sample and filter taxa search bar at the top, and a Sankey visualisation of the example metagenome sample

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

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

Then open your browser and visit localhost:5000

### 10.6.2. Visualizing metaphlan taxonomic profile with Krona

Krona is a software tool for interactive visualization of hierarchical data, such as taxonomic profiles generated by metagenomics tools like MetaPhlAn. Krona allows users to explore the taxonomic composition of microbial communities in a hierarchical manner, from the highest taxonomic level down to the species level.

The `metaphlan2krona.py` script is used to convert the MetaPhlAn taxonomic profile output to a format that can be visualized by Krona. The output of the script is a text file that contains the taxonomic profile in a hierarchical format that can be read by Krona. The `ktImportText` command is then used to generate an interactive HTML file that displays the taxonomic profile in a hierarchical manner using Krona.

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

💡 Expand to see command output

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

## 10.7. Getting modern comparative reference data

In order to compare our sample with modern reference samples, I used the curatedMetagenomics package, which provides both curated metadata, and pre-computed metaphlan taxonomic profiles for published modern human samples.

The full R code to get these data is available in `curatedMetagenomics/get_sources.Rmd`.

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

```
# Load required packages
library(curatedMetagenomicData)
library(tidyverse)

# Filter samples based on specific criteria
sampleMetadata %>%
  filter(body_site == "stool" & antibiotics_current_use == "no" & disease == "healthy") %>%
  group_by(non_westernized) %>%
  sample_n(100) %>%
  ungroup() -> selected_samples

# Extract relative abundance data for selected samples
selected_samples %>%
  returnSamples("relative_abundance") -> rel_ab

# Split relative abundance data by taxonomic rank and write to CSV files
data_ranks <- splitByRanks(rel_ab)

for (r in names(data_ranks)) {
  # Print taxonomic rank and output file name
  print(r)
  output_file <- paste0("../data/curated_metagenomics/modern_sources_", tolower(r), ".csv")
  print(output_file)
```

## 10. Taxonomic Profiling, OTU Tables and Visualisation

```
# Write relative abundance data to CSV file
assay_rank <- as.data.frame(assay(data_ranks[[r]]))
write.csv(assay_rank, output_file)
}
```

- The resulting pre-computed metaphlan taxonomic profiles (split by taxonomic ranks) are available in `data/curated_metagenomics`
- The associated metadata is available at `data/metadata/curated_metagenomics_modern_sources.csv`

### 10.8. Loading the ancient sample taxonomic profile

This is the moment where we will use the Pandas library Python to perform some data manipulation.

We will also use the Taxopy library to work with taxonomic information.

In python we need to import necessary libraries, i.e. pandas and taxopy, and a couple of other utility libraries.

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

And we then create an instance of the taxopy taxonomy database. This will take a few seconds/minutes, as it needs to download the entire NCBI taxonomy before storing in a local database.

```
taxdb = taxopy.TaxDb()
```

Let's read the metaphlan profile table with pandas (a python package with a similar concept to tidyverse dplyr, tidyr packa). It's a tab separated file, so we need to specify the delimiter as \t, and skip the comment lines of the files that start with #.

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

## 10. Taxonomic Profiling, OTU Tables and Visualisation

To look at the head of a dataframe (Table 10.1) with pandas

```
ancient_data.head()
```

Table 10.1.: Top few lines of a MetaPhlAn taxonomic profile

|   | clade_name                | NCBI_tax_id | relative_abundance | additional_species |
|---|---------------------------|-------------|--------------------|--------------------|
| 0 | k__Bacteria               | 2           | 82.23198           | NaN                |
| 1 | k__Archaea                | 2157        | 17.76802           | NaN                |
| 2 | k__Bacteria p__Firmicutes | 2 1239      | 33.47957           | NaN                |
| 3 | k__Bacteria p__-          | 2 976       | 28.42090           | NaN                |
|   | Bacteroidetes             |             |                    |                    |
| 4 | k__Bacteria p__-          | 2 201174    | 20.33151           | NaN                |
|   | Actinobacteria            |             |                    |                    |

We can also specify more rows by randomly picking 10 rows to display (Table 10.2).

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

Table 10.2.: Ten randomly selected lines of a MetaPhlAn taxonomic profile

|    | clade_name               | NCBI_tax_id                              | relative_abundance | additional_species      |
|----|--------------------------|--|--------------------|-------------------------|
| 1  | k__Archaea               | 2157                                     | 17.76802           | NaN                     |
| 46 | k__Bacteria p__-         | 2 976 200643 171549 171257 3384165 179   | Bacteria p__-      | Bacteroidetes c__-      |
|    | Bacteroidetes c__-       |  |                    | Bacteroidetes c__-      |
|    | <i>Bacteroidia/o...</i>  |  |                    | <i>Bacteroidia/o...</i> |
| 55 | k__Bacteria p__-         | 2 1239 186801 186802 186803 189330 88431 |                    |                         |
|    | Firmicutes c__-          |  |                    |                         |
|    | Clostridia o__Clo...     |  |                    |                         |
| 18 | k__Archaea p__-          | 2157 28890 183963 22350.71177            |                    | NaN                     |
|    | Euryarchaeota c__-       |  |                    |                         |
|    | <i>Halobacteria/o...</i> |  |                    |                         |
| 36 | k__Bacteria p__-         | 2 201174 1760 85004 31953 35778          |                    | NaN                     |
|    | Actinobacteria c__-      |  |                    |                         |
|    | <i>Actinobacteri...</i>  |  |                    |                         |

|    | clade_name  | NCBI_tax_id                             | relative_-abundance                              | additional_species |
|----|---|---|--|--------------------|
| 65 | k__Bacteria p__-Actinobacteria c__-Actinobacteri... | 2 201174 1760 85004 3195851678 216816   | Bacteria p__-Actinobacteria c__-Actinobacteri... |                    |
| 37 | k__Bacteria p__-Firmicutes c__-Clostridia o__Clo... | 2 1239 186801 186802 185863 25          |  | NaN                |
| 38 | k__Bacteria p__-Firmicutes c__-Clostridia o__Clo... | 2 1239 186801 186802 5411000 21685      | N  | NaN                |
| 26 | k__Bacteria p__-Actinobacteria c__-Actinobacteri... | 2 201174 1760 85004 319539377           |  | NaN                |
| 48 | k__Bacteria p__-Firmicutes c__-Clostridia o__Clo... | 2 1239 186801 186802 5411000 3263 40518 | Bacteria p__-Firmicutes c__-Clostridia o__Clo... |                    |

Because for this analysis, we're only going to look at the relative abundance, we'll only use this column, and the Taxonomic ID (TAXID) information, so we can `drop` (get rid of) the unnecessary columns.

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

### ! Important

Always investigate your data at first !

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

700.00007

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

## *10. Taxonomic Profiling, OTU Tables and Visualisation*

💡 Unfold to see what's happening

Let's proceed further and try to understand what's happening (Table 10.3).

```
ancient_data.head()
```

Table 10.3.: A two column table of TAXIDs and the organisms corresponding relative abundance

|   | TAXID    | relative_abundance |
|---|----------|--------------------|
| 0 | 2        | 82.23198           |
| 1 | 2157     | 17.76802           |
| 2 | 2 1239   | 33.47957           |
| 3 | 2 976    | 28.42090           |
| 4 | 2 201174 | 20.33151           |

## 10. Taxonomic Profiling, OTU Tables and Visualisation

To make sense of the TAXID, we will use taxopy to get all the taxonomic related informations such as (Table 10.4):

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

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

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

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

ancient_data.head()
```

Table 10.4.: A three column table of TAXIDs and the organisms corresponding relative abundance, and the attached taxonomic path associated with the TAXID

|   | TAXID    | relative_-<br>abundance | taxopy   |
|---|----------|-------------------------|--|
| 0 | 2        | 82.23198                | s__Bacteria  |
| 1 | 2157     | 17.76802                | s__Archaea   |
| 2 | 2 1239   | 33.47957                | s__Bacteria;c__Terrabacteria<br>group;p__Firmicutes  |
| 3 | 2 976    | 28.42090                | s__Bacteria;c__FCB<br>group;p__Bacteroidetes         |
| 4 | 2 201174 | 20.33151                | s__Bacteria;c__Terrabacteria<br>group;p__Actinoba... |

10. Taxonomic Profiling, OTU Tables and Visualisation

```
ancient_data = ancient_data.assign(  
    rank = ancient_data.taxopy.apply(lambda x: x.rank),  
    name = ancient_data.taxopy.apply(lambda x: x.name),  
    lineage = ancient_data.taxopy.apply(lambda x: x.name_lineage),  
)  
  
ancient_data
```

## 10. Taxonomic Profiling, OTU Tables and Visualisation

Table 10.5.: A five column table of TAXIDs and the organisms corresponding relative abundance, and the attached taxonomic path associated with the TAXID, but also the rank and name of the particular taxonomic ID

|     | TAXID                                      | relative_abundance | taxony  | rank         | name                                 | lineage   |
|-----|--|--------------------|---|--------------|--------------------------------------|---|
| 0   | 2  | 82.23198           | s__Bacteria   | superkingdom | [Bacteria, cellular organisms, root] |   |
| 1   | 2157                                       | 17.76802           | s__Archaea  | superkingdom | [Archaea, cellular organisms, root]  |   |
| 2   | 2 1239                                     | 33.47957           | s__-Bacteria;c__-Terrabacteria;group;p__-Firmicutes | phylum       | Firmicutes                           | [Firmicutes, Terrabacteria group, Bacteria, ce...]  |
| 3   | 2 976                                      | 28.42090           | s__-Bacteria;c__-FCB;group;p__-Bacteroidetes        | phylum       | Bacteroidetes                        | [Bacteroidetes, Bac-teroidetes/Chlorobi group, ...] |
| 4   | 2 201174                                   | 20.33151           | s__-Bacteria;c__-Terrabacteria;group;p__-Actinob... | phylum       | Actinobacteria                       | [Actinobacteria, Terrabacteria group, Bacteria...]  |
| ... | ...  | ...                | ...   | ...          | ...                                  | ...   |
| 62  | 2 1239 186801 186802 186803 572511 33039   |                    | Bacteria;c__-Terrabacteria;group;p__-Firmicut...    | species      | Ruminococcus                         | [Ruminococcus torques, Mediter-raneibacter, L...]   |
| 63  | 2 201174 84998 84999 84997 1472762 1232426 |                    | Bacteria;c__-Terrabacteria;group;p__-Actinoba...    | species      | Collinsella                          | [[Collinsella]] massiliensis, Enorma, Coriobact...  |

10. Taxonomic Profiling, OTU Tables and Visualisation

|    |  |  |                              |  |
|----|--|--|------------------------------|--|
| 64 | 2 1239 186801 186800  <del>186803</del>  189330 39486  | Bacteria;c__-_group;p__-_Firmicut...               | speciesDorea formicigenerans | [Dorea formicigenerans, Dorea, Lachnospiraceae...]         |
| 65 | 2 201174 1760 85004051953  <del>1678</del>  216816     | Bacteria;c__-_Terrabacteria group;p__-_Actinoba... | speciesBifidobacter longum   | [Bifidobacterium longum, Bifidobacterium, Bifi...]         |
| 66 | 2 1239 186801 186800  <del>1541000</del>  1263 1262959 | Bacteria;c__-_Terrabacteria group;p__-_Firmicut... | speciesRuminococcus sp.      | [Ruminococcus sp. CAG:488, CAG:488 environmental sampl...] |

## *10. Taxonomic Profiling, OTU Tables and Visualisation*

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

Which of the entries are at the `species` rank level?

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

Table 10.6.: A five column table of TAXIDs and the organisms corresponding relative abundance, and the attached taxonomic path associated with the TAXID, but also the rank and name of the particular taxonomic ID, filtered to only species

|    | TAXID                                      | relative_abundance | taxony  | rank name                  | lineage  |
|----|--|--------------------|---|----------------------------|--|
| 46 | 2 976 200643 171529 1735442 838 165179;c_- |                    | specie<br>_FCB<br>group;p__Bac-<br>teroidetes;c__-<br>B...  | Prevotella copri           | [Prevotella copri,<br>Prevotella,<br>Prevotellaceae,...]   |
| 47 | 2157 28890 183925 21058 2659 2172 2173;p_- |                    | specie<br>_Eur-<br>yarchaeota;c__-<br>Methanomada<br>gro... | Methanobrevibacter smithii | [Methanobrevibacter smithii,<br>Methanobre-<br>vibacte...] |
| 48 | 2 1239 186801 186802 58100 12B3 10518;c_-  |                    | specie<br>_Terrabacteria<br>group;p__-<br>Firmicut...       | Ruminococcus bromii        | [Ruminococcus bromii,<br>Ruminococcus,<br>Oscillospi...]   |
| 49 | 2 1239 186801 186802 796803 84B 261302;c_- |                    | specie<br>_Terrabacteria<br>group;p__-<br>Firmicut...       | Roseburia faecis           | [Roseburia faecis,<br>Roseburia, Lach-<br>nospiraceae,...] |
| 50 | 2 201174 84998 849998 6507 102B 674126;c_- |                    | specie<br>_Terrabacteria<br>group;p__-<br>Actinoba...       | Collinsella aerofaciens    | [Collinsella aerofaciens,<br>Collinsella,<br>Corioba...]   |

## *10. Taxonomic Profiling, OTU Tables and Visualisation*

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

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

10. Taxonomic Profiling, OTU Tables and Visualisation

Table 10.7.: Reconstruction of the first two column taxonomic profile but with species-level organism names rather than TAXIDs

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

## 10. Taxonomic Profiling, OTU Tables and Visualisation

```
ancient_phylums = ancient_data.query("rank == 'phylum'")  
  
ancient_phylums = ancient_phylums[['relative_abundance', 'name']].set_index('name').rename  
  
ancient_phylums
```

Table 10.8.: Reconstruction of the first two column taxonomic profile but with phylum-level organism names rather than TAXIDs

| name           | ERR5766177 |
|----------------|------------|
| Firmicutes     | 33.47957   |
| Bacteroidetes  | 28.42090   |
| Actinobacteria | 20.33151   |
| Euryarchaeota  | 17.76802   |

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

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

💡 Expand to see command output

```
rank  
class          99.72648  
family         83.49854  
genus          97.56524  
no_rank        19.48331  
order          99.72648  
phylum         100.00000  
species        100.00002  
superkingdom   100.00000  
Name: relative_abundance, dtype: float64
```

Seems better, right ?

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

## 10.9. Bringing together ancient and modern samples

Now let's load our modern reference samples

First at the phylum level (Table 10.9)

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

Table 10.9.: Taxonomic profiles at phylum level of multiple modern samples

|              | PNP_-  | A46_-                             | A94_-                              | A48_- | A09_- |
|--------------|--|-----------------------------------|------------------------------------|-------|-------|
| de028ad4-    | Val-   |                                   |                                    |       |       |
| 7ae6-        | i-   |                                   |                                    |       |       |
| 11e9-        | PNRla-   | PNP_-                             |                                    |       |       |
| a106-        | Maintion_-   | Main_-                            | HDEGARSDID5122977BZ_01_KHGDIKH01_- |       |       |
| 68b599263384 | G80273   | SAMSAANTEAS012301100142B9F8753FE7 | 4                                  | 9     | 1FE1  |
| Bact         | 00000015.48239610.030.83087620378059.610746.39694971649670.24.4206841.68.904.8162870.810.31544     |                                   |                                    |       |       |
| Firm         | 95.242360.47635324619.270794928266H465342.230976.6889226239197.716.871.685.387.894.72113           |                                   |                                    |       |       |
| Prot         | 0.99500i770985649.77.273133970200728.650001.8101616.572301.559.585.7323.926.9463.54.1355.820177    |                                   |                                    |       |       |
| Actin        | 0.2580010.2763518712.62.073201576617630.35138199513.018940.4699639.4709.35.4636.077.147.26894      |                                   |                                    |       |       |
| Verr         | 0.0000000.00780000.99225.051000000000.009943.29.795.0.050.100.0000000.000.000.0000000000.000.00000 |                                   |                                    |       |       |

Then at the species level

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

As usual, we always check if our data has been loaded correctly (Table 10.10)

```
modern_species.head()
```

## 10. Taxonomic Profiling, OTU Tables and Visualisation

Table 10.10.: Taxonomic profiles at species level of multiple modern samples

### 10.9.1. Time to merge !

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

For that, we'll use the pandas `merge` function which will merge the two tables together, using the index as the merge key.

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

## 10. Taxonomic Profiling, OTU Tables and Visualisation

Finally, let's load the metadata, which contains the information about the modern samples (Table 10.11).

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

metadata.head()
```

Table 10.11.: Taxonomic profiles at species level of multiple modern samples

|      | study_id           | sample_id | subject_id | antibiotics | cur- | con- | age    | cat-    | infant       | hla    | or- | birth | ho | started | breastfeeding | twins | brink | hatchl | -formula | -  | first | nu- | ra- | day | ALT | GFR |
|------|--------------------|-----------|------------|-------------|------|------|--------|---------|--------------|--------|-----|-------|----|---------|---------------|-------|-------|--------|----------|----|-------|-----|-----|-----|-----|-----|
| 0    | Shaoy028a6C01528   | alo       | cont       | hol         | hol  | 4.0  | newbor | N       | NN           | NN     | NN  | NN    | NN | NN      | NN            | NN    | NN    | NN     | NN       | NN | NN    | NN  | NN  | NN  | NN  | NN  |
| 2019 | 7ae6-              | ba        |            | 11e9-       |      |      | a106-  |         | 68b59976a384 |        |     |       |    |         |               |       |       |        |          |    |       |     |     |     |     |     |
| 1    | ZeevFNP_-PNPste    | alo       | cont       | hol         | hol  | Na   | Na     | adult.  | Na           | NN     | NN  | NN    | NN | NN      | NN            | NN    | NN    | NN     | NN       | NN | NN    | NN  | NN  | NN  | NN  |     |
| 2015 | Main_-             |           | 283        | 283         |      |      |        |         |              |        |     |       |    |         |               |       |       |        |          |    |       |     |     |     |     |     |
| 2    | ZeevFNP_-PNPste    | alo       | cont       | hol         | hol  | Na   | Na     | adult.  | Na           | NN     | NN  | NN    | NN | NN      | NN            | NN    | NN    | NN     | NN       | NN | NN    | NN  | NN  | NN  | NN  |     |
| 2015 | Vali-              | Val-      | da-        | i-          |      |      | tion_- | da-     | 55           | tion_- | 55  |       |    |         |               |       |       |        |          |    |       |     |     |     |     |     |
| 3    | VataG8nT275T014806 | alo       | cont       | hol         | hol  | Na   | Na     | child.. | Na           | NN     | NN  | NN    | NN | NN      | NN            | NN    | NN    | NN     | NN       | NN | NN    | NN  | NN  | NN  | NN  |     |
| 2016 |                    |           |            |             |      |      |        |         |              |        |     |       |    |         |               |       |       |        |          |    |       |     |     |     |     |     |
| 4    | ZeevFNP_-PNPste    | alo       | cont       | hol         | hol  | Na   | Na     | adult.  | Na           | NN     | NN  | NN    | NN | NN      | NN            | NN    | NN    | NN     | NN       | NN | NN    | NN  | NN  | NN  | NN  |     |
| 2015 | Main_-             | 363       | 363        |             |      |      |        |         |              |        |     |       |    |         |               |       |       |        |          |    |       |     |     |     |     |     |

## 10.10. Comparing ancient and modern samples

### 10.10.1. Taxonomic composition

One common plot in microbiome papers is a stacked barplot, often at the phylum or family level.

First, we'll do some renaming, to make the value of the metadata variables a bit easier to understand (Table 10.12)

```
group_info = pd.concat(
    [
        (
            metadata['non_westernized']
            .map({'no': 'westernized', 'yes': 'non_westernized'}) # for the non_westernized in the
            .to_frame(name='group').set_index(metadata['sample_id']) # rename the column to group
            .reset_index()
        ),
        (
            pd.Series({'sample_id': 'ERR5766177', 'group': 'ancient'}).to_frame().transpose()
        )
    ],
    axis=0, ignore_index=True
)
group_info
```

 Expand to see command output

```
/var/folders/1c/l1qb09f15jddsh65f6xv1n_r0000gp/T/ipykernel_40830/27419655.py:2:
FutureWarning: The frame.append method is deprecated and will be removed from pandas
Use pandas.concat instead.
metadata['non_westernized']
```

Table 10.12.: Table of samples and their group

|   | sample_id                            | group       |
|---|--------------------------------------|-------------|
| 0 | de028ad4-7ae6-11e9-a106-68b59976a384 | westernized |
| 1 | PNP_Main_283                         | westernized |
| 2 | PNP_Validation_55                    | westernized |

## 10. Taxonomic Profiling, OTU Tables and Visualisation

|     | sample_id    | group           |
|-----|--------------|-----------------|
| 3   | G80275       | westernized     |
| 4   | PNP_Main_363 | westernized     |
| ... | ...          | ...             |
| 196 | A48_01_1FE   | non_westernized |
| 197 | KHG_1        | non_westernized |
| 198 | TZ_81781     | non_westernized |
| 199 | A09_01_1FE   | non_westernized |
| 200 | ERR5766177   | ancient         |

We need transform our data in tidy format to plot with plotnine, a python clone of ggplot.

Table 10.13.: Table of the raw multi-sample taxonomic table

|                | Candidatus     | Melan-      | abac-    | te-          | sample_-  |                 |              |               |                |              |             |                 |        |
|----------------|----------------|-------------|----------|--------------|-----------|-----------------|--------------|---------------|----------------|--------------|-------------|-----------------|--------|
|                | Actinobacteria | Alveolates  | Bivalvia | Chlorophytes | Crustacea | Dermatofagoides | Fusobacteria | Rhipidophytes | Proteobacteria | Syphilitic   | Tenericutes | Verrucomicrobia | group  |
| 1965.47050     | 0.0            | 14.808280.0 | 0.0      | 0.0          | 0.0       | 0.0             | ...          | 0.0           | 0.00000        | 3.646599061  | 0.0         | 0.000008        | _non_- |
|                |                |             |          |              |           |                 |              |               |                |              | 01_-        | west-           |        |
|                |                |             |          |              |           |                 |              |               |                |              | 1FE         | ern-            |        |
|                |                |             |          |              |           |                 |              |               |                |              |             | ized            |        |
| 19736.701450.0 | 10.1089080.0   | 0.0         | 0.0      | 0.0          | 0.0       | ...             | 0.0          | 0.00000       | 17.601600000   | 0.0          | 0.00000     | HG              | _non_- |
|                |                |             |          |              |           |                 |              |               |                |              | 1           | west-           |        |
|                |                |             |          |              |           |                 |              |               |                |              |             | ern-            |        |
|                |                |             |          |              |           |                 |              |               |                |              |             | ized            |        |
| 1981.16026     | 0.0            | 57.100310.0 | 0.0      | 0.0          | 0.0       | 0.0             | ...          | 0.0           | 0.00000        | 0.305870067  | 0.0         | 0.00000         | _non_- |
|                |                |             |          |              |           |                 |              |               |                |              | 81781       | west-           |        |
|                |                |             |          |              |           |                 |              |               |                |              |             | ern-            |        |
|                |                |             |          |              |           |                 |              |               |                |              |             | ized            |        |
| 1997.40894     | 0.0            | 11.605940.0 | 0.0      | 0.0          | 0.0       | 0.0             | ...          | 0.0           | 0.00000        | 56.201000000 | 0.0         | 0.00000         | _non_- |
|                |                |             |          |              |           |                 |              |               |                |              | 01_-        | west-           |        |
|                |                |             |          |              |           |                 |              |               |                |              | 1FE         | ern-            |        |
|                |                |             |          |              |           |                 |              |               |                |              |             | ized            |        |

Now, we need transform this (Table 10.13) in the tidy format, with the `melt` function.

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

Finally, we only want to keep the mean relative abundance for each phylum. To do so, we will compute the mean relative abundance, for each phylum, for each group (`ancient`, `westernized`, and `non_westernized`).

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

We then verify that the sum of the mean relative abundance is still ~100%, as an extra sanity check.

```
tidy_phylums.groupby('group')['relative_abundance'].sum()
```

```
group
ancient           100.000000
non_westernized   99.710255
westernized       99.905089
Name: relative_abundance, dtype: float64
```

## 10.11. Let's make some plots

We first import plotnine

```
from plotnine import *
```

And then run plotnine to a barplot of the mean abundance per group (Figure 10.6).

```
ggplot(tidy_phylums, aes(x='group', y='relative_abundance', fill='Phylum')) \
+ geom_bar(position='stack', stat='identity') \
+ ylab('mean abundance') \
+ xlab("") \
+ theme_classic()
```

## 10.12. Ecological diversity

### 10.12.1. Alpha diversity

Alpha diversity is the measure of diversity within each sample. It is used to estimate how many species are present in a sample, and how diverse they are.

We'll use the python library scikit-bio to compute it, and the plotnine library (a python port of ggplot2 to visualize the results).

```
import skbio
```

Let's compute the species richness, the Shannon, and Simpson index of diversity index (Table 10.14)

## 10. Taxonomic Profiling, OTU Tables and Visualisation

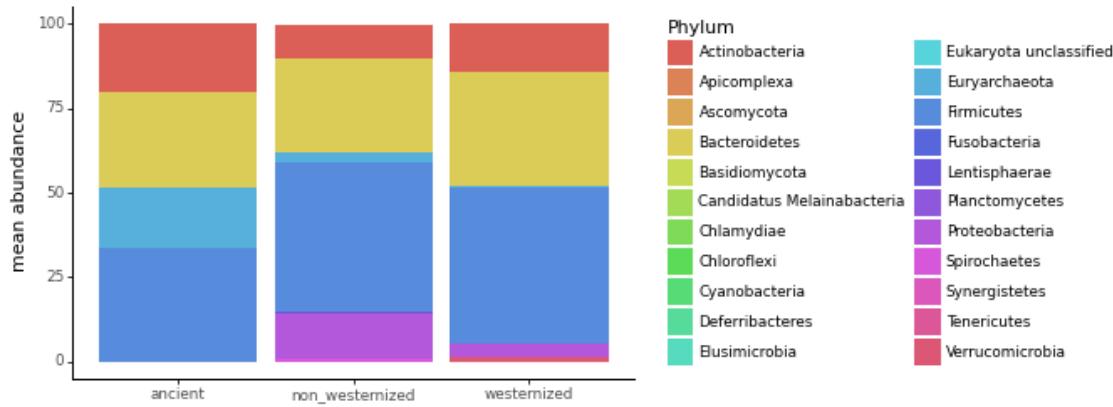


Figure 10.6.: Stacked bar chart of ancient, non-westernised, and westernised sample groups on the X axis columns, and mean abundance percentage on the Y-axis. The legend and stacks of the bar represent different phyla each with a different colour

```
shannon = skbio.diversity.alpha_diversity(metric='shannon', counts=all_species.transpose(),
simpson = skbio.diversity.alpha_diversity(metric='simpson', counts=all_species.transpose(),
richness = (all_species != 0).astype(int).sum(axis=0)
alpha_diversity = (shannon.to_frame(name='shannon')
                   .merge(simpson.to_frame(name='simpson'), left_index=True, right_index=True)
                   .merge(richness.to_frame(name='richness'), left_index=True, right_index=True))
alpha_diversity
```

Table 10.14.: Table of the shannon, simpson, and richness alpha diversity indicies for a subset of samples

|                                      | shannon  | simpson  | richness |
|--------------------------------------|----------|----------|----------|
| ERR5766177                           | 3.032945 | 0.844769 | 21       |
| de028ad4-7ae6-11e9-a106-68b59976a384 | 0.798112 | 0.251280 | 11       |
| PNP_Main_283                         | 5.092878 | 0.954159 | 118      |
| PNP_Validation_55                    | 3.670162 | 0.812438 | 72       |
| G80275                               | 3.831358 | 0.876712 | 66       |
| ...                                  | ...      | ...      | ...      |
| KHG_9                                | 3.884285 | 0.861683 | 87       |
| A48_01_1FE                           | 4.377755 | 0.930024 | 53       |
| KHG_1                                | 3.733834 | 0.875335 | 108      |

## 10. Taxonomic Profiling, OTU Tables and Visualisation

|            | shannon  | simpson  | richness |
|------------|----------|----------|----------|
| TZ_81781   | 2.881856 | 0.719491 | 44       |
| A09_01_1FE | 2.982322 | 0.719962 | 75       |

Let's load the group information from the metadata. To do so, we merge the alpha diversity dataframe that we compute previously, with the metadata dataframe, using the `sample_id` as a merge key. Finally, we do a bit of renaming to re-encode `yes/no` as `non_westernized/westernized`.

```
alpha_diversity = (
    alpha_diversity
    .merge(metadata[['sample_id', 'non_westernized']], left_index=True, right_on='sample_id')
    .set_index('sample_id')
    .rename(columns={'non_westernized':'group'})
)
alpha_diversity['group'] = alpha_diversity['group'].replace({'yes':'non_westernized','no':'westernized'})
```

Table 10.15.: Table of the shannon, simpson, and richness alpha diversity indicies for a subset of samples but with the group metadata

|                                      | shannon  | simpson  | richness | group             |
|--------------------------------------|----------|----------|----------|-------------------|
| sample_id                            |          |          |          |                   |
| ERR5766177                           | 3.032945 | 0.844769 | 21       | ERR5766177        |
| de028ad4-7ae6-11e9-a106-68b59976a384 | 0.798112 | 0.251280 | 11       | westernized       |
| PNP_Main_283                         | 5.092878 | 0.954159 | 118      | westernized       |
| PNP_Validation_55                    | 3.670162 | 0.812438 | 72       | westernized       |
| G80275                               | 3.831358 | 0.876712 | 66       | westernized       |
| ...                                  | ...      | ...      | ...      | ...               |
| KHG_9                                | 3.884285 | 0.861683 | 87       | non_- westernized |
| A48_01_1FE                           | 4.377755 | 0.930024 | 53       | non_- westernized |
| KHG_1                                | 3.733834 | 0.875335 | 108      | non_- westernized |

## 10. Taxonomic Profiling, OTU Tables and Visualisation

|            | shannon  | simpson  | richness | group            |
|------------|----------|----------|----------|------------------|
| TZ_81781   | 2.881856 | 0.719491 | 44       | non_-westernized |
| A09_01_1FE | 2.982322 | 0.719962 | 75       | non_-westernized |

And as always, we need it in tidy format (Table 10.17) for plotnine.

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

```
alpha_diversity
```

Table 10.16.: Table of the shannon, simpson, and richness alpha diversity indices for a subset of samples but with the group metadata but in long-form tidy format

|                                      | group            | diversity_index | alpha diversity |
|--------------------------------------|------------------|-----------------|-----------------|
| sample_id                            |                  |                 |                 |
| ERR5766177                           | ERR5766177       | shannon         | 3.032945        |
| de028ad4-7ae6-11e9-a106-68b59976a384 | westernized      | shannon         | 0.798112        |
| PNP_Main_283                         | westernized      | shannon         | 5.092878        |
| PNP_Validation_55                    | westernized      | shannon         | 3.670162        |
| G80275                               | westernized      | shannon         | 3.831358        |
| ...                                  | ...              | ...             | ...             |
| KHG_9                                | non_-westernized | richness        | 87.000000       |
| A48_01_1FE                           | non_-westernized | richness        | 53.000000       |
| KHG_1                                | non_-westernized | richness        | 108.000000      |
| TZ_81781                             | non_-westernized | richness        | 44.000000       |
| A09_01_1FE                           | non_-westernized | richness        | 75.000000       |

We now make a violin plot to compare the alpha diversity for each group, faceted by the type of alpha diversity index (Figure 10.7).

## 10. Taxonomic Profiling, OTU Tables and Visualisation

```

g = ggplot(alpha_diversity, aes(x='group', y='alpha diversity', color='group'))
g += geom_violin()
g += geom_jitter()
g += theme_classic()
g += facet_wrap(~diversity_index, scales = 'free')
g += theme(axis_text_x=element_text(rotation=45, hjust=1))
g += scale_color_manual({'ERR5766177':'#DB5F57','westernized':'#5F57DB','non_westernized': '#5F57DB'})
g += theme(subplots_adjust={'wspace': 0.15})
g

```

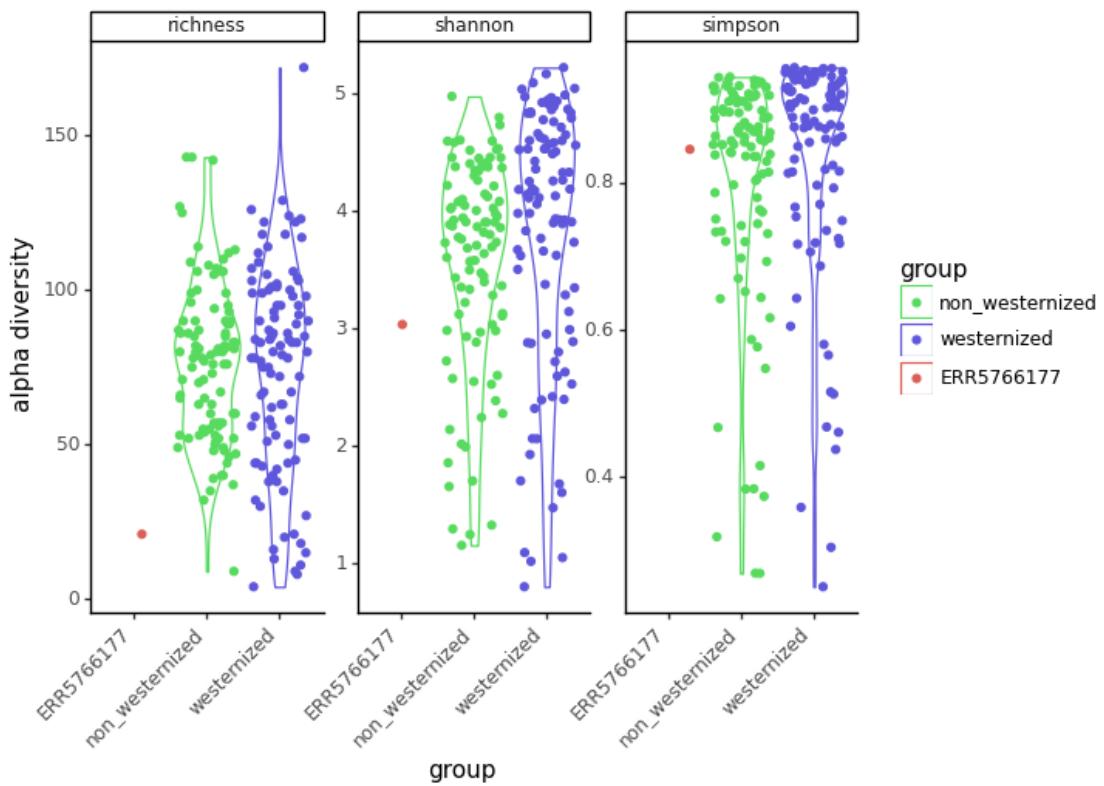


Figure 10.7.: Three groups of violin plots of an ancient sample, westernised samples and non-westernised samples (x-axis) of the alpha diversity (y-axis) calculated for richness, shannon and simpson alpha indicies

**i** Note

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

### 10.12.2. Beta diversity

The Beta diversity is the measure of diversity between a pair of samples. It is used to compare the diversity between samples and see how they relate.

We will compute the beta diversity using the bray-curtis dissimilarity

```
beta_diversity = skbio.diversity.beta_diversity(metric='braycurtis', counts=all_species.tra
```

We get a distance matrix

```
print(beta_diversity)
```

?

Expand to see command output

```
201x201 distance matrix
IDs:
'ERR5766177', 'de028ad4-7ae6-11e9-a106-68b59976a384', 'PNP_Main_283', ...
Data:
[[0.          1.          0.81508134 ... 0.85716612 0.69790092 0.8303726 ]
 [1.          0.          0.99988327 ... 0.99853413 0.994116  0.99877258]
 [0.81508134 0.99988327 0.          ... 0.82311942 0.87202543 0.91363156]
 ...
 [0.85716612 0.99853413 0.82311942 ... 0.          0.84253376 0.76616679]
 [0.69790092 0.994116  0.87202543 ... 0.84253376 0.          0.82409272]
 [0.8303726  0.99877258 0.91363156 ... 0.76616679 0.82409272 0.        ]]
```

To visualize this distance matrix in a lower dimensional space, we'll use a PCoA, which is a method very similar to a PCA, but taking a distance matrix as input.

```
pcoa = skbio.stats.ordination.pcoa(beta_diversity)
```

## 10. Taxonomic Profiling, OTU Tables and Visualisation

💡 Expand to see command output

```
/Users/maxime/mambaforge/envs/summer_school_microbiome/lib/python3.9/site-packages/sklearn/pca/_base.py:103: UserWarning: The result contains negative eigenvalues. Please compare their magnitude with the magnitude of the positive ones. If the negative ones are smaller, it's probably safe to ignore them, but if they are larger than the positive ones, it might indicate that the data is not well-suited for PCA.
  warnings.warn("The result contains negative eigenvalues. Please compare their magnitude with the magnitude of the positive ones. If the negative ones are smaller, it's probably safe to ignore them, but if they are larger than the positive ones, it might indicate that the data is not well-suited for PCA.", UserWarning)
See the Notes section for more details. The smallest eigenvalue is -0.253348427457239
```

pcoa.samples

Table 10.17.: Table principal coordinates (columns) for each of the samples (rows)

|                                     | PC1                            | PC2           | PC3 | PC4                         | PC5 | PC6  | PC7                        | PC8 | PC9                      | PC10          | PC11 | PC12 | PC13 | PC14 | PC15 | PC16 | PC17 | PC18 | PC19 | PC20 |
|-------------------------------------|--------------------------------|---------------|-----|-----------------------------|-----|--|----------------------------|-----|--------------------------|---------------|------|------|------|------|------|------|------|------|------|------|
| ERR576617769010.107422327205104876- |                                |               |     |                             |     |  |                            |     |                          | 0.097450210.0 | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
|                                     | 0.039778                       |               |     |                             |     |  |                            |     |                          | 0.2563321069  |      |      |      |      |      |      |      |      |      |      |
| de028ad4-                           | 0.1452240.12706269754-         |               |     |                             |     |  |                            |     | 0.0367281294...          | 0.0           | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
| 7ae6-                               | 0.0993550.191676               |               |     |                             |     | 0.1322097382                               |                            |     | 0.056686                 |               |      |      |      |      |      |      |      |      |      |      |
| 11e9-                               |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| a106-                               |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| 68b59976a384                        |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| PNP_- - -                           | 0.116002700596611053321026477- |               |     |                             |     |  |                            |     |                          | ... 0.0       | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
| Main_-                              | 0.21401087466                  |               |     |                             |     |  |                            |     | 0.0064618592             |               |      |      |      |      |      |      |      |      |      |      |
| 283                                 |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| PNP_-                               | 0.244827- -                    |               |     | 0.030759751487150.034730... |     |  |                            |     | 0.0                      | 0.0           | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
| Vali-                               | 0.17393601972836               |               |     |                             |     |  | 0.1356410.009395           |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| da-                                 |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| tion_-                              |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| 55                                  |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| G80275                              | - - - -                        | 0.0885385970- | -   |                             |     |  | 0.015710.0                 | 0.0 | 0.0                      | 0.0           | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
|                                     | 0.261358014543745932           |               |     |                             |     | 0.00520287392016                           |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| ...                                 | ...                            | ...           | ... | ...                         | ... | ...  | ...                        | ... | ...                      | ...           | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  | ...  |      |
| KHG_-                               | 0.2960570.01394326490.019663-  |               |     |                             |     |  |                            |     | 0.0700850.0              | 0.0           | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
| 9                                   | 0.150300                       |               |     |                             |     |  |                            |     | 0.1476920.06312304573514 |               |      |      |      |      |      |      |      |      |      |      |
| A48_-                               | 0.11062309754231-              | -             |     |                             |     |  | 0.0281690.04190285970.0    | 0.0 | 0.0                      | 0.0           | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |      |
| 01_-                                |                                |               |     |                             |     |  | 0.18590018510234200.044530 |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| 1FE                                 |                                |               |     |                             |     |  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| KHG_- -                             | 0.167885990176842-             | -             | -   | -                           |     |  |                            |     | 0.150000.0               | 0.0           | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
| 1                                   | 0.100009                       |               |     |                             |     | 0.40558201006401970742252                  |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |
| TZ_-                                | 0.405716-                      | -             | -   | -                           |     | 0.0682610.198152...                        |                            |     | 0.0                      | 0.0           | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |      |
| 81781                               |                                |               |     |                             |     | 0.1392975002697058261192710.0188210.012792 |                            |     |                          |               |      |      |      |      |      |      |      |      |      |      |

## 10. Taxonomic Profiling, OTU Tables and Visualisation

|       | PC1   | PC2   | PC3   | PC4   | PC5  | PC6   | PC7   | PC8   | PC9   | PC10 | PC11 | PC12 | PC13 | PC14 | PC15 | PC16 | PC17 | PC18 | PC19 | PC20 |
|-------|-------|-------|-------|-------|------|-------|-------|-------|-------|------|------|------|------|------|------|------|------|------|------|------|
| A09_- | 0.089 | 0.107 | 0.106 | 0.096 | 0.29 | -     | -     | -     | 0.0   | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  | 0.0  |      |
| 01_-  |       | 0.125 | 0.023 | 0.067 | 0.93 | 0.000 | 0.012 | 0.021 | 0.084 | 0.36 |      |      |      |      |      |      |      |      |      |      |
| 1FE   |       |       |       |       |      |       |       |       |       |      |      |      |      |      |      |      |      |      |      |      |

Let's look at the variance explained by the first axes by using a scree plot (Figure 10.8).

```
var_explained = pcoa.proportion_explained[:9].to_frame(name='variance explained').reset_index()

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

In this scree plot, we're looking for the “elbow”, where there is a drop in the slope. Here, it seems that most of the variance is captured by the 3 first principal components

```
pcoa_embed = pcoa.samples[['PC1', 'PC2', 'PC3']].rename_axis('sample').reset_index()

pcoa_embed = (
    pcoa_embed
    .merge(metadata[['sample_id', 'non_westernized']], left_on='sample', right_on='sample_id')
    .drop('sample_id', axis=1)
    .rename(columns={'non_westernized': 'group'})
)

pcoa_embed['group'] = pcoa_embed['group'].replace({'yes': 'non_westernized', 'no': 'westernized'})
```

Let's first look at these components with 2D plots (Figure 10.9, Figure 10.10)

```
ggplot(pcoa_embed, aes(x='PC1', y='PC2', color='group')) \
+ geom_point() \
+ theme_classic() \
+ scale_color_manual({'ERR5766177': '#DB5F57', 'westernized': '#5F57DB', 'non_westernized': '#57DB5F'})

ggplot(pcoa_embed, aes(x='PC1', y='PC3', color='group')) +
geom_point() +
theme_classic() +
scale_color_manual({'ERR5766177': '#DB5F57', 'westernized': '#5F57DB', 'non_westernized': '#57DB5F'})
```

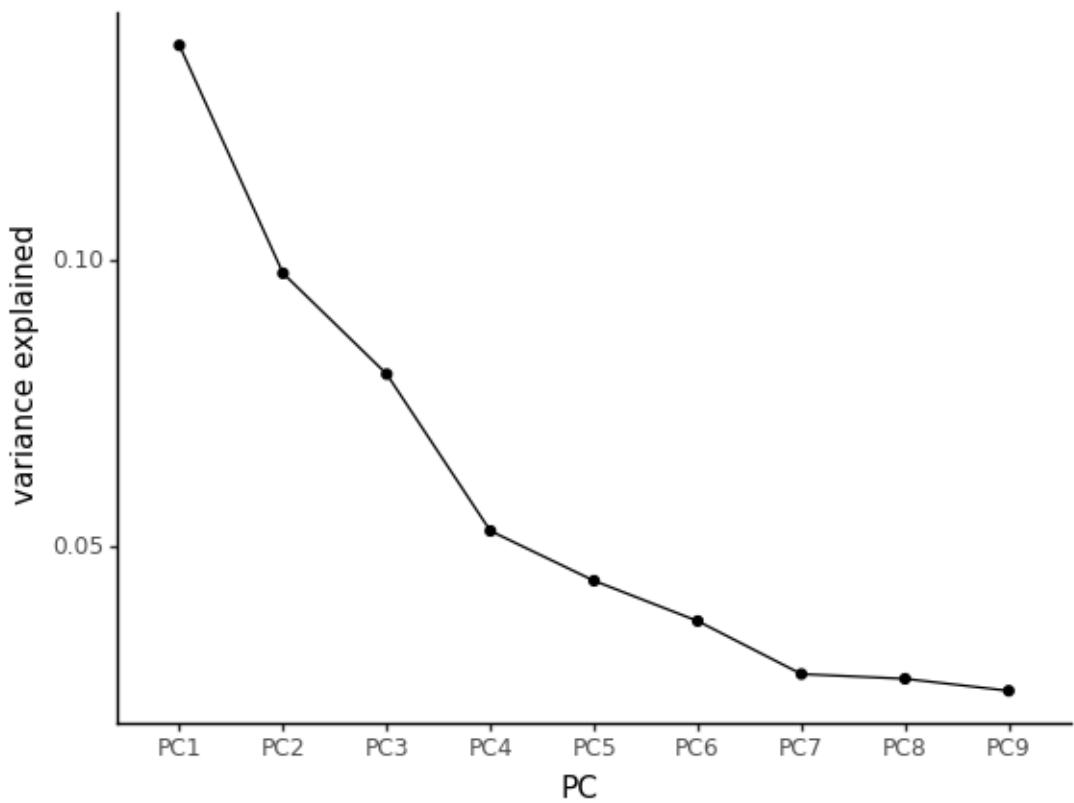


Figure 10.8.: Scree plot describing the variance explained (Y-axis), for each Principal Component (X-axis), with a curved line from PC1 having highest variance to lowest on PC9.

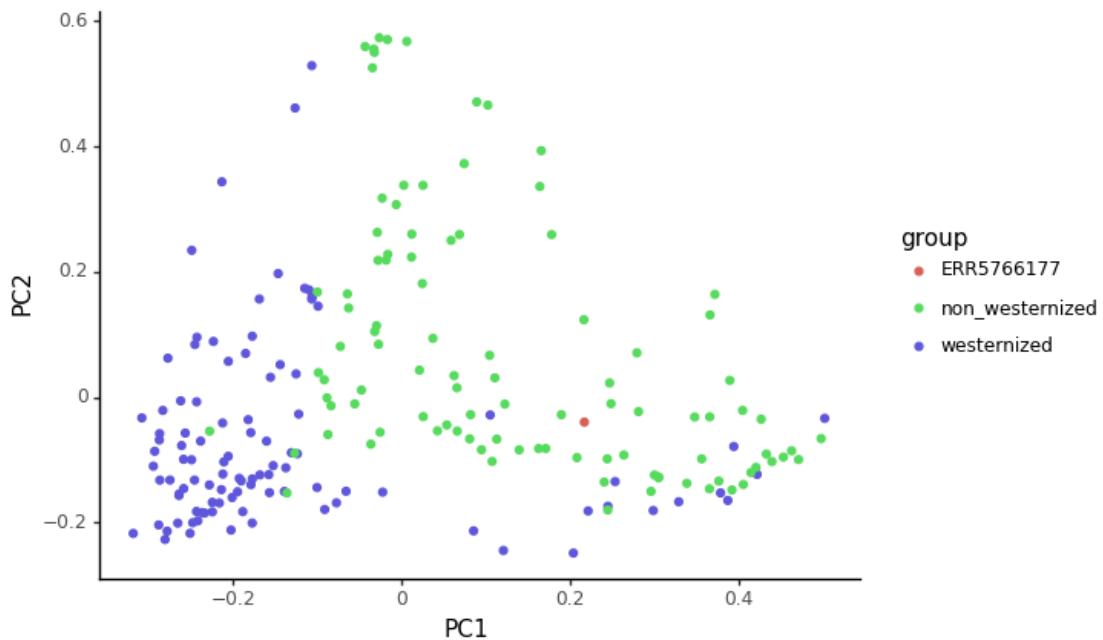


Figure 10.9.: Principal Coordinate Analysis plot of PC1 (X-axis) and PC2 (Y-axis), with three groups of points in the scatter plot - blue circles of westernised data points in the bottom left, overlapping with green circles of non-westernised datapoints in the top right, and the single ancient sample as a red circle falling in between the two on the right of the overlap

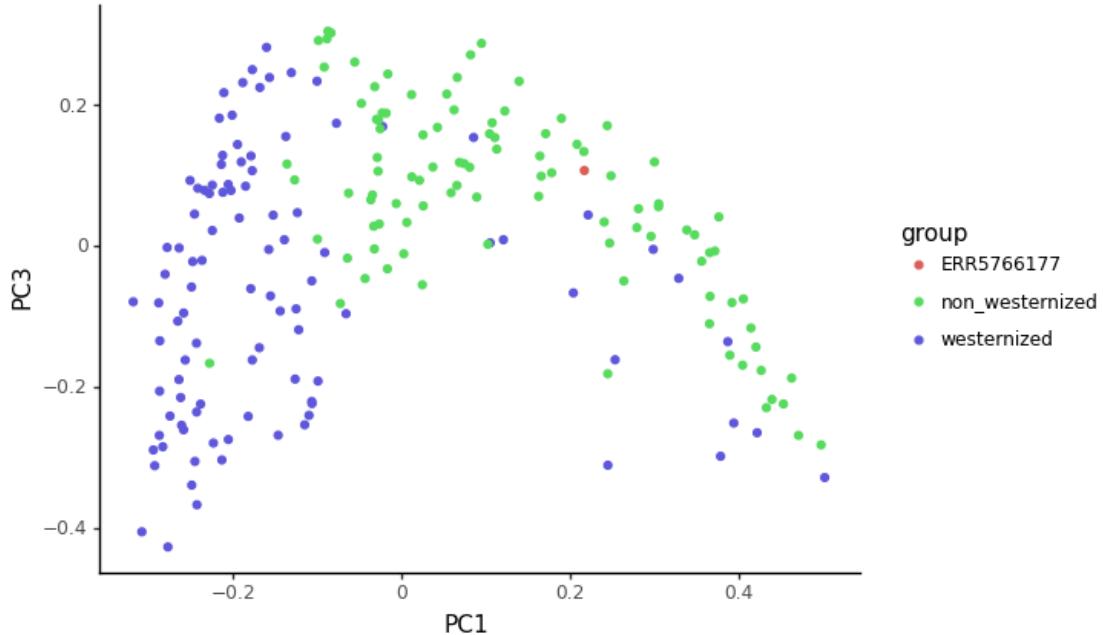


Figure 10.10.: Principal Coordinate Analysis plot of PC1 (X-axis) and PC3 (Y-axis), a similar overlap between westernised/non-westernised individuals and position of the ancient sample as in the PC1-PC2 PCoA, however this time in a horseshoe shape from bottom left for the westernised data points, curving up to the top of PC3 at a peak, and then falling again at the top of PC1

💡 Expand to see additional visualisations

You can also plot the data above as a with a 3d plot if you were to run the following command

```
import plotly.express as px

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

fig.show()
```

ℹ Note

**Pause and think:** How do you think this embedding represents how our sample relates to modern reference samples ?

Finally, we can also visualize this distance matrix using a clustered heatmap, where pairs of sample with a small beta diversity are clustered together (Figure 10.11).

```
import seaborn as sns
import scipy.spatial as sp, scipy.cluster.hierarchy as hc
```

We set the color in seaborn to match the color palette we've used so far.

```
pcoa_embed['colour'] = pcoa_embed['group'].map({'ERR5766177': '#DB5F57', 'westernized': '#5F57DB', 'non'

linkage = hc.linkage(sp.distance.squareform(beta_diversity.to_data_frame()), method='average')

sns.clustermap(
    beta_diversity.to_data_frame(),
    row_linkage=linkage,
    col_linkage=linkage,
    row_colors = pcoa_embed['colour'].to_list()
)
```

## 10.13. References

## 10. Taxonomic Profiling, OTU Tables and Visualisation

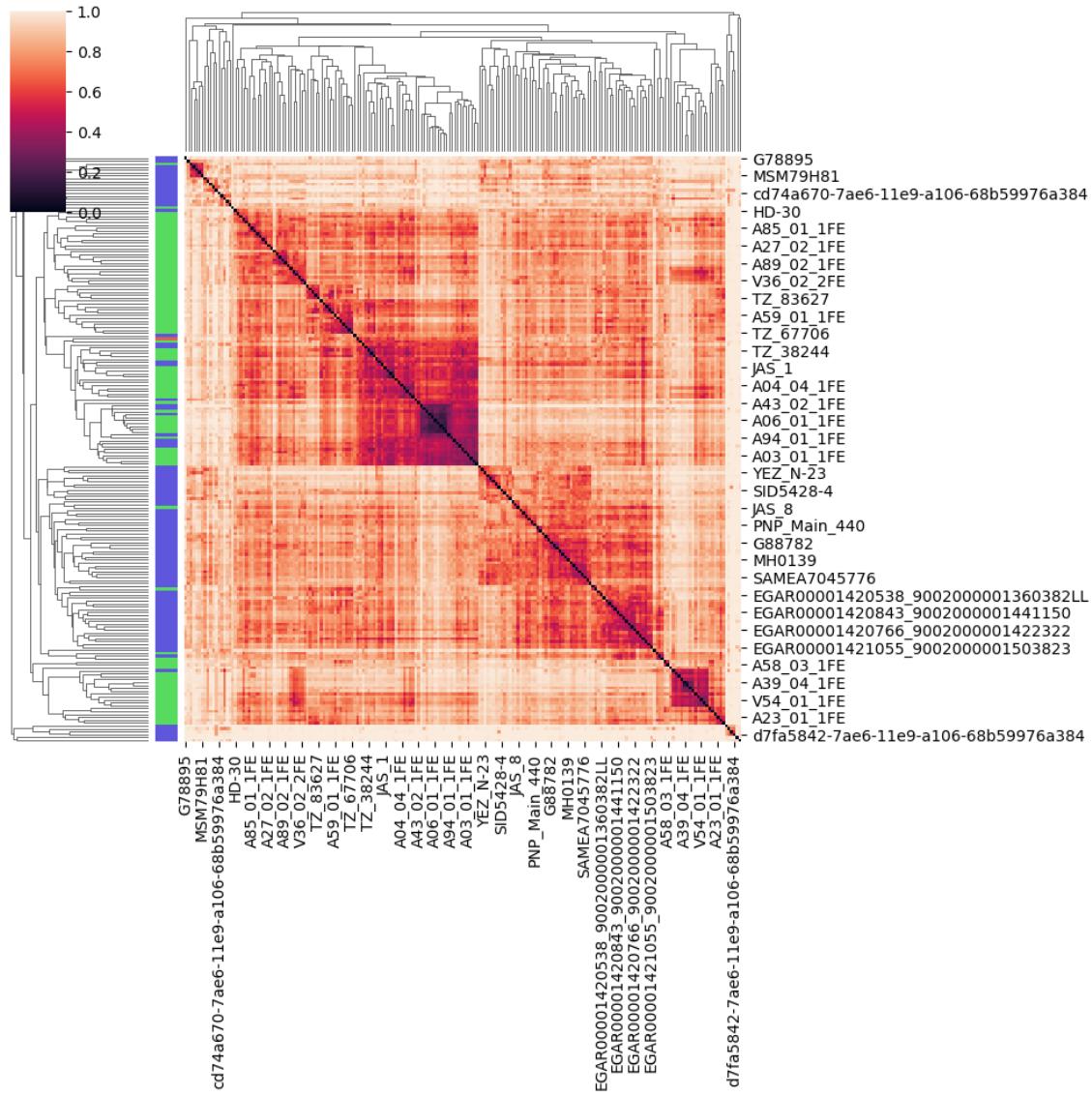


Figure 10.11.: Sample-by-sample clustered heatmap, with tree representation of the clustering on the left and top of the heatmap

# 11. Functional Profiling



Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate functional-profiling
```

## 11.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

## 11.2. Preparation

The data and conda environment `.yaml` file for this practical session can be downloaded from here: <https://doi.org/10.5281/zenodo.6983188>. See instructions on page.

Change into the session directory

```
cd /<path>/<to>/functional-genomics/
```

Load the conda environment.

```
conda activate phylogenomics-functional
```

Open R Studio from within the conda environment, and we can load the required libraries for this walkthrough.

```
library(mixOmics) ## For PCA generation

## Utility packages (pretty stuff)
library(knitr)
library(data.table)
library(tidyverse)
library(gplots)
library(ggrepel)
library(viridis)
library(patchwork)
```

### 11.3. HUMAnN3 Pathways

First, we need to run HUMMAn3 to align reads against gene databases and convert to gene family names counts.

#### ⚠ Warning

We will not run HUMANn3 here as it requires very large databases and takes a long time to run, so we will give you the commands you normally would run but we provide with you pre-made results files before you (see below).

```
## DO NOT RUN!

# run humann3
humann3 --input file.fastq --output output --threads <threads>

# join all output tables (can do for both gene and pathways)
humann_join_tables -i output/ -o genefamilies_joined.tsv --file_name unmapped_genefamilies

# normalize the output (here by tss - total sum scaling, can do for both gene and pathway)
humann_renorm_table --input genefamilies_joined.tsv --output genefamilies_joined_cpm.tsv

# regroup the table to combine gene families (standardise gene family IDs across taxa)
humann_regroup_table --input genefamilies_joined_cpm.tsv --output genefamilies_joined_cpm

# give the gene families names
humann_rename_table --input genefamilies_joined_cpm_ur90rxn.tsv --output genefamilies_joi
```

## 11.4. humann3 tables

First lets load a pre-made pathway abundance file

```
## load the species and genus tables generated with humann3
humann3_path_full <- fread("./pathabundance_joined_cpm.tsv")
humann3_path_full <- as_tibble(humann3_path_full)

# clean the file names
humann3_path_full <- rename(humann3_path_full, Pathway = `# Pathway`)
colnames(humann3_path_full) <- gsub(".unmapped_Abundance","", colnames(humann3_path_full))
colnames(humann3_path_full) <- gsub(".SG1","", colnames(humann3_path_full))

# remove unmapped and ungrouped reads
humann3_path <- humann3_path_full %>% filter(!str_detect(Pathway, "UNMAPPED|UNINTEGRATED"))
```

Then lets load associated sample metadata to help make it easier for comparative analysis and make actual informative inferences.

The data being used in this session, is from Velsko et al. 2022 (PNAS Nexus), where we tried to find associations between dental pathologies and taxonomic and genome content. We had a large skeletal collection from a single site in the Netherlands, with a lot of osteological metadata. The study aimed to see if there were any links between the oral microbiome and groups of dental pathologies.

```
# load the metadata file
full_metadata <- fread("full_combined_metadata.tsv")

## Example of metadata
tibble(full_metadata %>%
      filter(Site_code == "MID") %>%
      select(Site, Time_period, Library_ID, Sequencing_instrument, Pipenotch, Max_Perio_Score)
```

First step: we can pre-define various functions for generate PCAs we will use downstream - you don't have to worry about these too much they are just custom functions to quickly plot PCAs from a `mixOmics` PCA output object with `ggplot`, but we leave the code here for if you're curious.

## 11. Functional Profiling

```
# plot PCA with colored dots and the title including the # of species or genera
plot_pca <- function(df, pc1, pc2, color_group, shape_group, ncomps) {
  metadata_group_colors <- get(paste(color_group, "_colors", sep = ""))
  metadata_group_shapes <- get(paste(shape_group, "_shapes", sep = ""))

  pca.list <- mixOmics::pca(df, ncomp = ncomps, logratio = 'CLR')

  ## Pull out loadings
  exp_var <- paste0(round(pca.list$explained_variance * 100, 2), "%")
  df_X <- pca.list$variates$X %>%
    as.data.frame() %>%
    rownames_to_column("Library_ID") %>%
    inner_join(full_metadata, by = "Library_ID")

  color_group = df_X[[color_group]]
  shape_group = df_X[[shape_group]]

  ## Selecting which PCs to plot
  if (pc1 == 'PC1') {
    pc1 <- df_X$PC1
    exp_var_pc1 <- exp_var[1]
    xaxis <- c("PC1")
  } else if (pc1 == 'PC2') {
    pc1 <- df_X$PC2
    exp_var_pc1 <- exp_var[2]
    xaxis <- c("PC2")
  } else if (pc1 == 'PC3') {
    pc1 <- df_X$PC3
    exp_var_pc1 <- exp_var[3]
    xaxis <- c("PC3")
  }

  if (pc2 == 'PC1') {
    pc2 <- df_X$PC1
    exp_var_pc2 <- exp_var[1]
    yaxis <- c("PC1")
  } else if (pc2 == 'PC2') {
    pc2 <- df_X$PC2
    exp_var_pc2 <- exp_var[2]
    yaxis <- c("PC2")
```

## 11. Functional Profiling

```
    } else if (pc2 == 'PC3') {
      pc2 <- df_X$PC3
      exp_var_pc2 <- exp_var[3]
      yaxis <- c("PC3")
    }

    ## Generate figure
    pca_plot <- ggplot(df_X, aes(pc1, pc2)) +
      geom_point(aes(fill = color_group, shape = shape_group), size = 4.5, stroke = 0.3) +
      scale_fill_manual(values = metadata_group_colors) +
      scale_shape_manual(values = metadata_group_shapes) +
      # stat_ellipse() +
      xlab(paste(xaxis, " - ", exp_var_pc1)) +
      ylab(paste(yaxis, " - ", exp_var_pc2)) +
      theme_minimal(base_size = 16) +
      theme(text = element_text(size=16)) +
      theme(legend.title = element_blank(),
            legend.key.size = unit(2,"mm"),
            legend.text = element_text(size = 6)) +
      theme(legend.position = "top")

    return(pca_plot)
  }

# for continuous data
plot_pca_cont <- function(df, pc1, pc2, color_group, shape_group, ncomps, title_text) {

  pca.list <- mixOmics::pca(df, ncomp = ncomps, logratio = 'CLR')

  exp_var <- paste0(round(pca.list$explained_variance * 100, 2), "%")
  df_X <- pca.list$variates$X %>%
    as.data.frame() %>%
    rownames_to_column("Library_ID") %>%
    inner_join(full_metadata, by = "Library_ID")

  color_group = df_X[[color_group]]
  shape_group = df_X[[shape_group]]

  if (pc1 == 'PC1') {
    pc1 <- df_X$PC1
```

## 11. Functional Profiling

```
exp_var_pc1 <- exp_var[1]
xaxis <- c("PC1")
} else if (pc1 == 'PC2') {
  pc1 <- df_X$PC2
  exp_var_pc1 <- exp_var[2]
  xaxis <- c("PC2")
} else if (pc1 == 'PC3') {
  pc1 <- df_X$PC3
  exp_var_pc1 <- exp_var[3]
  xaxis <- c("PC3")
}

if (pc2 == 'PC1') {
  pc2 <- df_X$PC1
  exp_var_pc2 <- exp_var[1]
  yaxis <- c("PC1")
} else if (pc2 == 'PC2') {
  pc2 <- df_X$PC2
  exp_var_pc2 <- exp_var[2]
  yaxis <- c("PC2")
} else if (pc2 == 'PC3') {
  pc2 <- df_X$PC3
  exp_var_pc2 <- exp_var[3]
  yaxis <- c("PC3")
}

pca_plot <- ggplot(df_X, aes(pc1, pc2, fill = color_group, shape = shape_group)) +
  geom_point(size = 5, color = "black") +
  scale_fill_viridis_c(option = "C") +
  scale_shape_manual(values = c(24,21)) +
  # stat_ellipse() +
  xlab(paste(xaxis, " - ", exp_var_pc1)) +
  ylab(paste(yaxis, " - ", exp_var_pc2)) +
  theme_minimal(base_size = 16) +
  theme(text = element_text(size=16)) +
  theme(legend.title = element_blank(),
        legend.key.size = unit(2,"mm"),
        legend.text = element_text(size = 6)) +
  theme(legend.position = "top") +
  ggtitle(title_text) + theme(plot.title = element_text(size = 10))
```

## 11. Functional Profiling

```
    return(pca_plot)
}

plot_pca.bi <- function(df, pc1, pc2, metadata_group, columntitle) {
  metadata_group_colors <- get(paste(metadata_group, "_colors", sep = ""))
  metadata_group_shapes <- get(paste(metadata_group, "_shapes", sep = ""))

  arrow_pc <- enquo(columntitle)

  exp_var <- paste0(round(df$explained_variance * 100, 2), "%") # explained variance for

  # select only the PCs from the PCA and add metadata
  df_X <- df$variates$X %>%
    as.data.frame() %>%
    rownames_to_column("Library_ID") %>%
    inner_join(full_metadata, by = "Library_ID")

  metadata_group = df_X[[metadata_group]]

  corr_lam <- df$sdev[c("PC1", "PC2", "PC3")] * sqrt(nrow(df_X))

  df_X <- df_X %>%
    mutate(PC1 = PC1 / corr_lam[1],
           PC2 = PC2 / corr_lam[2],
           PC3 = PC3 / corr_lam[3])

  # select the correct PC column and explained variance for PC1
  if (pc1 == 'PC1') {
    Pc1 <- df_X$PC1
    exp_var_pc1 <- exp_var[1]
    xaxis <- c("PC1")
  } else if (pc1 == 'PC2') {
    Pc1 <- df_X$PC2
    exp_var_pc1 <- exp_var[2]
    xaxis <- c("PC2")
  } else if (pc1 == 'PC3') {
    Pc1 <- df_X$PC3
    exp_var_pc1 <- exp_var[3]
    xaxis <- c("PC3")
  }
}
```

## 11. Functional Profiling

```
# select the correct PC column and explained variance for PC2
if (pc2 == 'PC1') {
  Pc2 <- df_X$PC1
  exp_var_pc2 <- exp_var[1]
  yaxis <- c("PC1")
} else if (pc2 == 'PC2') {
  Pc2 <- df_X$PC2
  exp_var_pc2 <- exp_var[2]
  yaxis <- c("PC2")
} else if (pc2 == 'PC3') {
  Pc2 <- df_X$PC3
  exp_var_pc2 <- exp_var[3]
  yaxis <- c("PC3")
}

# Identify the 10 pathways that have highest positive and negative loadings in the selected PC
pws_10 <- df$loadings$X %>%
  as.data.frame(.) %>%
  rownames_to_column(var = "Pathway") %>%
  separate(Pathway, into = "Pathway", sep = ":", extra = "drop") %>%
  top_n(10, !!arrow_pc)

neg_10 <- df$loadings$X %>%
  as.data.frame(.) %>%
  rownames_to_column(var = "Pathway") %>%
  separate(Pathway, into = "Pathway", sep = ":", extra = "drop") %>%
  top_n(-10, !!arrow_pc)

pca_plot_bi <- ggplot(df_X, aes(x = Pc1, y = Pc2)) +
  geom_point(size = 3.5, aes(shape = metadata_group, fill = metadata_group)) +
  geom_segment(data = pws_10,
               aes(xend = get(paste(pc1)), yend = get(paste(pc2))),
               x = 0, y = 0, colour = "black",
               size = 0.5,
               arrow = arrow(length = unit(0.03, "npc"))) +
  geom_label_repel(data = pws_10,
                   aes(x = get(paste(pc1)), y = get(paste(pc2)), label = Pathway),
                   size = 2.5, colour = "grey20", label.padding = 0.2, force = 5, max.overlap = 5) +
  geom_segment(data = neg_10,
```

## 11. Functional Profiling

```
        aes(xend = get(paste(pc1)), yend = get(paste(pc2))),
        x = 0, y = 0, colour = "grey50",
        size = 0.5,
        arrow = arrow(length = unit(0.03, "npc")))) +
  geom_label_repel(data = neg_10,
    aes(x = get(paste(pc1)), y = get(paste(pc2)), label = Pathway),
    size = 2.5, colour = "grey20", label.padding = 0.2, max.overlaps = 12) +
  labs(x = paste(xaxis, " - ", exp_var_pc1),
    y = paste(yaxis, " - ", exp_var_pc2)) +
  scale_fill_manual(values = metadata_group_colors) +
  scale_shape_manual(values = metadata_group_shapes) +
  theme_minimal() + theme(text = element_text(size = 16)) +
  theme(text = element_text(size=16)) +
  theme(legend.position = "top")

  return(pca_plot_bi)
}
```

As we are dealing with aDNA, and we often have bad samples, its sometimes interesting to see differences between well/badly preserved samples at all stages of analysis.

Therefore we may generate results for all samples. However for actual analysis where we want to intepret biological differences, should exclude outliers (in this case highly contaminated samples - as identified by the `decontam` package - see Velsko et al. 2022 \_ PNAS Nexus for more details).

We can make a list the outliers from the previous authentication analyses.

```
outliers_mpa3 <- c("EXB059.A2101", "EXB059.A2501", "EXB015.A3301", "EXB034.A2701",
                     "EXB059.A2201", "EXB059.A2301", "EXB059.A2401", "LIB058.A0103", "LIB058.A0101",
                     poor_samples_mpa3 <- c("CS28", "CS38", "CSN", "ELR003.A0101", "ELR010.A0101",
                     "KT09calc", "MID024.A0101", "MID063.A0101", "MID092.A0101")

outliersF <- str_c(outliers_mpa3, collapse = "|")
```

## 11.5. Sample Clustering with PCA

### 11.5.1. Pathway abundance analyses

Once we've removed outlier samples, our first simple question is - what is the functional relationships of the groups?

Can we already see distinctive patterns between the different groups in our dataset?

To do this lets clean up the data a bit (cleaning names, removing samples with no metadata etc.), normalise (via a ‘centered-log-ratio’ transform ), and run a PCA.

Once we've done this we should always check our PCA's Scree plot first.

```
humann3_path_11 <- humann3_path %>%
  filter(!str_detect(Pathway, "\\|")) %>%
  # no full_metadata, remove these
  select(-c("MID025.A0101", "MID033.A0101", "MID052.A0101", "MID056.A0101",
           "MID065.A0101", "MID068.A0101", "MID076.A0101", "MID078.A0101")) %>%
  # remove poorly preserved samples
  select(-c("MID024.A0101", "MID063.A0101", "MID092.A0101")) %>%
  select(-matches("EXB|LIB")) %>%
  # inner_join(., humann3_path.decontam_noblocks_presence_more_30, by = "Pathway") %>%
  gather("Library_ID", "Counts", 2:ncol(.)) %>%
  mutate(Counts = Counts + 1) %>%
  spread(Pathway, Counts) %>%
  column_to_rownames("Library_ID")

# prepare to run a PCA
# check the number of components to retain by tuning the PCA
mixOmics::tune.pca(humann3_path_11, logratio = 'CLR')

humann3_all_otu.pca <- mixOmics::pca(humann3_path_11, ncomp = 3, logratio = 'CLR')
humann3_all_pca_values <- humann3_all_otu.pca$variates$X %>%
  as.data.frame() %>%
  rownames_to_column("Library_ID") %>%
  inner_join(., full_metadata, by = "Library_ID")
```

We can see the first couple of PCs in the scree plot account for a good chunk of the variation of our dataset, so lets visualise the PCA itself.

## 11. Functional Profiling

We visualise the PCA with one of our custom functions defined above, and colour by the Pipe notch metadata column.

```
# pipenotch colors/shapes
Pipenotch_colors = c("#311068", "#C83E73")
Pipenotch_shapes = c(16,17)

# by minimum number of pipenotches
pipenotch <- plot_pca_cont(humann3_path_11, "PC1", "PC2", "Min_no_Pipe_notches", "Pipenotch",
pipenotch
```

We can see there is a slight separation between the groups, but how do we find out which pathways are maybe driving this pattern?

For this we can generate a PCA bi-plot which show what loadings are driving the spread of the samples.

```
Pipenotch_colors = c("#311068", "#C83E73")
Pipenotch_shapes = c(24,21)

biplot <- plot_pca_bi(humann3_all_otu.pca, "PC1", "PC2", "Pipenotch", PC1)
biplot
```

From the biplot we can see which pathways are differentiating along PC1.

We can pull these IDs out to find out what pathways there are from the biplot object itself.

```
# make a table of the pathways to save, to use again later in another R notebook
humann3_pathway_biplot_list <- biplot$plot_env$pws_10 %>% arrange(desc(PC1)) %>% select(Pathway)
humann3_pathway_biplot_list <- humann3_pathway_biplot_list %>%
  bind_rows(biplot$plot_env$neg_10 %>% arrange(desc(PC1)) %>% select(Pathway, PC1, PC2)%>%
```

### 11.5.2. Species contributions to pathways

However, this ID numbers aren't very informative to us. At this point we have to do a bit of literature review/database scraping to pull the human-readable names/descriptions of the IDs - which we have already done for you.

We can load these back into our environment

## 11. Functional Profiling

```
# PC biplot loading top 10s
humann3_pathway_biplot_list <- fread("./humann3_pathway_biplot_list.tsv")
humann3_pathway_biplot_list <- humann3_pathway_biplot_list %>%
  rename(Pathway = pathway) %>%
  mutate(Path = sapply(Pathway, function(f) {
    unlist(str_split(f, ":"))[1]
  })) %>%
  select(Pathway, Path, everything()) %>%
  # remove 3 of the 4 ubiquinol pathways w/identical loadings
  filter(!str_detect(Pathway, "5856|5857|6708"))

tibble(humann3_pathway_biplot_list)
```

We now have the pathway ID, and a pathway description for each of the loadings of the PCA.

### 11.5.2.0.1. PC1

While we have the pathways, we don't *who* contributed these.

For this, we can join our pathway table back onto the original output from HUMANn3 we loaded at the beginning, which includes the taxa information.

```
# list the 10 orthologs with strongest loading in PC1 + values
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%
  filter(Direction == "PC1+") %>%
  pull(Path) %>%
  str_c(., collapse = "|") # need this format for filtering in the next step

# select only those 10 pathways from the list, and split the column with names into 3 (Pathway, Genus, Species)
humann3_path_pc1pws <- humann3_path %>%
  filter(str_detect(Pathway, humann3_path_biplot_pc)) %>%
  filter(str_detect(Pathway, "\\|")) %>%
  gather("SampleID", "CPM", 2:ncol(.)) %>%
  mutate(Pathway = str_replace_all(Pathway, "\\.s__", "|s__")) %>%
  separate(., Pathway, into = c("Pathway", "Genus", "Species"), sep = "\\|") %>%
  mutate(Species = replace_na(Species, "unclassified"),
        Genus = str_replace_all(Genus, "g__", ""),
        Species = str_replace_all(Species, "s__", "")) %>%
```

## 11. Functional Profiling

```
inner_join(., humann3_pathway_biplot_list %>%
  select(Pathway, Path) %>%
  distinct(.), by = "Pathway") %>%
inner_join(., humann3_pathway_biplot_list %>%
  filter(Direction == "PC1+") %>%
  select(Path), by = "Path") %>%
# select(-Pathway) %>%
select(Path, everything()) %>%
arrange(Path)

tibble(humann3_path_pc1pws)
```

We can now see who contributed which pathway, and also the abundance information (CPM)!

Given many taxa may contribute to the same pathway, we may want to see which taxa are more ‘dominantly’ contributing to this.

For this we can calculate of all copies of a given pathway what fraction comes from which taxa (you can imagine this like ‘depth’ coverage in genomic analysis), based on the percentage of the total copies per million for that pathway.

```
# calculate the % for each pathway contributed by each genus
humann3_path_pc1pws_stats <- humann3_path_pc1pws %>%
  group_by(Path, Genus) %>%
  summarize(Sum = sum(CPM)) %>%
  mutate(Percent = Sum/sum(Sum)*100) %>%
  ungroup()

# create the list of 10 orthologs again, but don't collapse the list as above
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%
  filter(Direction == "PC1+") %>%
  arrange(Path) %>%
  pull(Path)

# calculate the total % of all genera that contribute < X% to each ortholog
humann3_path_pc1pws_stats_extra <- lapply(humann3_path_biplot_pc, function(eclass) {
  high_percent <- humann3_path_pc1pws_stats %>%
    filter(Path == eclass) %>%
    filter(Percent < 5) %>%
    summarise(Remaining = sum(Percent)) %>%
```

## 11. Functional Profiling

```

    mutate(Path = eclass,
          Genus = "Other")
  }) %>%
  bind_rows(.)

# add this additional % to the main table
humann3_path_pcbi_bar_df <- humann3_path_pc1pws_stats_extra %>%
  rename(Percent = Remaining) %>%
  bind_rows(., humann3_path_pc1pws_stats %>%
              select(-Sum)) %>%
  select(Path, Genus, Percent) %>%
  mutate(Direction = "PC1+") %>%
  distinct()

```

And we can visualize the contributors to the top 10 pathways driving the main variation along PC1 (with the assumption these maybe the most biological significant, and to reduce the numbers of pathways we have to research).

For the loadings falling in the positive direction of the PC1:

```

# plot the values in a bar chart
paths_sp_pc1 <- humann3_path_pcbi_bar_df %>%
  # filter(Direction == "PC1+", Genus != "Other") %>% # removing Other plots all species/un
  filter(Percent >= 5 | (Percent <= 5 & Genus == "Other")) %>% # filter out the genera with
  # filter(Percent >= 5) %>% # filter out the genera with % < 5, but keep Other < 5
  mutate(
    Genus = fct_relevel(Genus, "Other", "unclassified", "Aggregatibacter", "Capnocytophaga", "C
                           "Eikenella", "Haemophilus", "Kingella", "Lautropia", "Neisseria", "Ottow
    Path = fct_relevel(Path, humann3_pathway_biplot_list %>%
                            filter(Direction == "PC1+")) %>%
                            pull(Path))) %>%
  ggplot(., aes(x=Path, y=Percent, fill = Genus)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  scale_fill_manual(values = c("#OD0887FF", "#969696", "#5D01A6FF", "#7E03A8FF",
                               "#9C179EFF", "#B52F8CFF", "#CC4678FF", "#DE5F65FF",
                               "#ED7953FF", "#F89441FF", "#FDB32FFF", "#FBD424FF", "#F0F921FF
  theme(text = element_text(size=18),
        axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)) +
  ylab("Percent") +

```

## 11. Functional Profiling

```
ggtile("Metacyc pathways - PC1 positive") +  
  theme(title = element_text(size=10))  
  
# viridis_pal(option = "B")(13)  
paths_sp_pc1
```



Warning

PWY-5345 has no species assignment to that pathway.

And the negative loadings:

```
# list the 10 orthologs with strongest loading in PC1 + values  
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%  
  filter(Direction == "PC1-") %>%  
  arrange(Path) %>%  
  pull(Path) %>%  
  str_c(., collapse = "|") # need this format for filtering in the next step  
  
# select only those 10 orthologs from the list, and split the column with names into 3 (Ortho  
humann3_path_pc1neg <- humann3_path %>%  
  filter(str_detect(Pathway, humann3_path_biplot_pc)) %>%  
  filter(str_detect(Pathway, "\\|")) %>%  
  gather("SampleID", "CPM", 2:ncol(.)) %>%  
  mutate(Pathway = str_replace_all(Pathway, "\\s_ ", "|s_ ")) %>%  
  separate(., Pathway, into = c("Pathway", "Genus", "Species"), sep = "\\|") %>%  
  mutate(Species = replace_na(Species, "unclassified"),  
         Genus = str_replace_all(Genus, "g_ ", ""),  
         Species = str_replace_all(Species, "s_ ", "")) %>%  
  inner_join(., humann3_pathway_biplot_list %>%  
             select(Pathway, Path) %>%  
             distinct(.), by = "Pathway") %>%  
  inner_join(., humann3_pathway_biplot_list %>%  
             filter(Direction == "PC1-") %>%  
             select(Path), by = "Path") %>%  
  select(-Pathway) %>%  
  select(Path, everything()) %>%  
  arrange(Path)  
  
# calculate the % for each ortholog contributed by each genus
```

## 11. Functional Profiling

```
humann3_path_pc1neg_stats <- humann3_path_pc1neg %>%
  group_by(Path, Genus) %>%
  summarize(Sum = sum(CPM)) %>%
  mutate(Percent = Sum/sum(Sum)*100) %>%
  ungroup()

# create the list of 10 orthologs again, but don't collapse the list as above
humann3_path_biplot_pc <- humann3_pathway_biplot_list %>%
  filter(Direction == "PC1-") %>%
  arrange(Path) %>%
  pull(Path)

# calculate the total % of all genera that contribute < X% to each ortholog
humann3_path_pc1neg_stats_extra <- lapply(humann3_path_biplot_pc, function(eclass) {
  high_percent <- humann3_path_pc1neg_stats %>%
    filter(Path == eclass) %>%
    filter(Percent < 5) %>%
    summarise(Remaining = sum(Percent)) %>%
    mutate(Path = eclass,
          Genus = "Other")
}) %>%
bind_rows()

# add this additional % to the main table
humann3_path_pcbi_bar_df <- humann3_path_pcbi_bar_df %>%
  bind_rows(humann3_path_pc1neg_stats_extra %>%
    rename(Percent = Remaining) %>%
    bind_rows(., humann3_path_pc1neg_stats %>%
      select(-Sum)) %>%
    select(Path, Genus, Percent) %>%
    mutate(Direction = "PC1-")) %>%
  distinct()

# plot the values in a bar chart
paths_sp_pc2 <- humann3_path_pcbi_bar_df %>%
  filter(Direction == "PC1-") %>% # removing Other plots all species/unassigned - no need to
  filter(Percent >= 5 | (Percent <= 5 & Genus == "Other")) %>% # filter out the genera with
  mutate(Genus = fct_relevel(Genus, "Other", "unclassified", "Desulfobulbus", "Desulfomicrobium",
    Path = fct_relevel(Path, humann3_pathway_biplot_list %>%
      filter(Direction == "PC1-")) %>%
```

## 11. Functional Profiling

```
pull(Path))) %>%
ggplot(., aes(x=Path, y=Percent, fill = Genus)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  scale_fill_manual(values = c("#0D0887FF", "#969696", "#B52F8CFF", "#ED7953FF", "#FCFFA4FF"))
  theme(text = element_text(size=18),
        axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)) +
# facet_wrap(~pathrtholog, nrow=2) +
  ylab("Percent") +
  ggtitle("Metacyc pathways - PC1 negative") +
  theme(title = element_text(size=12))

paths_sp_pc2
```

### 11.5.3. Final Visualisation

Finally, we can stick the biplot and the taxon contribution plots together!

```
h3biplots <- biplot + paths_sp_pc2 + paths_sp_pc1 +
  plot_layout(widths = c(2, 1, 1))

ggsave("./h3_paths_biplots.pdf", plot = h3biplots,
       device = "pdf", scale = 1, width = 20, height = 9.25, units = c("in"), dpi = 300)

system(paste0('firefox "h3_paths_biplots.pdf"'))
```

This allows us to evaluate all the information together.

From this point onwards, we would have to do manual research/literature reviews into each of the pathways, see if they make ‘sense’ to the sample type and associated groups of samples, and evaluate whether they are interesting or not..

# 12. Introduction to *de novo* Genome Assembly

## 💡 Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate denovo-assembly
```

## 12.0.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

## 12.1. Introduction

First of all, what is a **metagenomic** sample? Metagenomic sample is a sample that consists of DNA from more than one source. The number and the type of sources might vary widely between different samples. Typical sources for ancient remains are e.g. the host organism and the microbial species. The important part is that we generally do not know the origin of a DNA molecule prior to analysing the sequencing data generated from the DNA library of this sample. In the example presented in (Figure 12.1), our metagenomic sample has DNA from three different sources, here coloured in blue, red, and yellow.

How can we determine the sources of the DNA that we have in our metagenomic sample? There are three main options whose *pros* and *cons* are summarised in (Table 12.1).

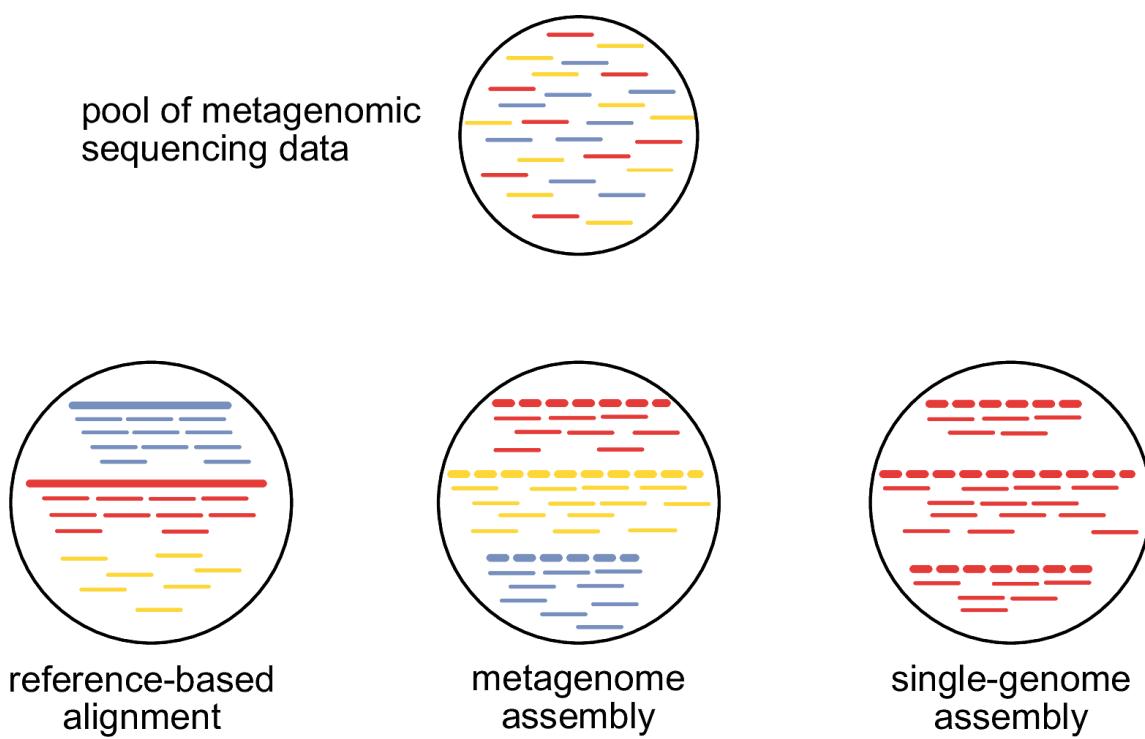


Figure 12.1.: Overview of the ways how to analyse metagenomic sequencing data.

## 12. Introduction to de novo Genome Assembly

Table 12.1.: Pros and cons of the major methods for determining the sources of metagenomic DNA.

| method                    | pros  | cons   |
|---------------------------|---|--|
| reference-based alignment | highly sensitive, applicable to aDNA samples        | requires all sources to be represented in database |
| single-genome assembly    | high-quality genomes from cultivated bacteria       | not available for ancient DNA samples              |
| metagenome assembly       | able to recover unknown diversity present in sample | highly dependent on preservation of ancient DNA    |

Until recently, the only option for ancient DNA samples was to take the short-read sequencing data and align them against some known reference genomes. However, this approach is heavily relying on whether all sources of our samples are represented in the available reference genomes. If a source is missing in the reference database - in our toy example, this is the case for the yellow source (Figure 12.1) -, then we won't be able to detect it using this reference database.

While there is a potential workaround for modern metagenomic samples, *single-genome assembly* relies on being able to cultivate a microbial species to obtain an isolate. This is unfeasible for ancient metagenomic samples because there are no more viable microbial cells available that could be cultivated.

Around 2015, a technical revolution started when the first programs, e.g. MEGAHIT [@LiMegahit2015] and metaSPAdes [@Nurk2017], were published that could successfully perform *de novo* assembly from metagenomic data. Since then, tens of thousands metagenomic samples have been assembled and it was revealed that even well studied environments, such as the human gut microbiome, have a lot of additional microbial diversity that has not been observed previously via culturing and isolation [@Almeida2021].

The technical advancement of being able to perform *de novo* assembly on metagenomic samples led to an explosion of studies that analysed samples that were considered almost impossible to study beforehand. For researchers that are exposed to ancient DNA, the imminent question arises: can we apply the same methods to ancient DNA data? In this practical course, we will walk through all required steps that are necessary to successfully perform *de novo* assembly from ancient DNA metagenomic sequencing data and show you what you can do once you have obtained the data.

## 12.2. Practical course

### 12.2.1. Sample overview

For this practical course, I selected a palaeofaeces sample from the study by @Maixner2021, who generated deep metagenomic sequencing data for four palaeofaeces samples that were excavated from an Austrian salt mine in Hallstatt and were associated with the Celtic Iron Age. We will focus on the youngest sample, **2612**, which was dated to be just a few hundred years old (Figure 12.2).

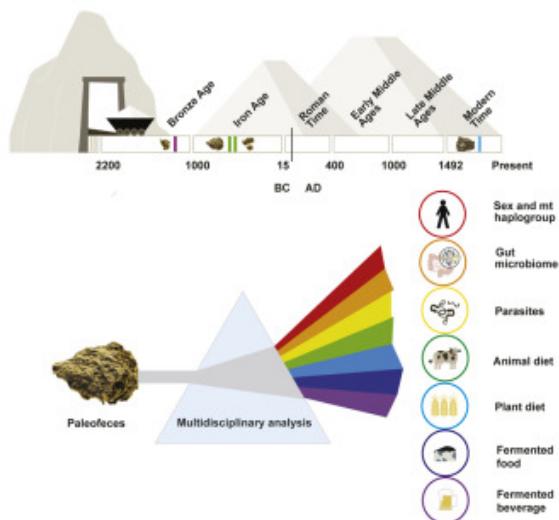


Figure 12.2.: The graphical abstract of Maixner et al. (2021).

However, because the sample was very deeply sequenced for more than 250 million paired-end reads and we do not want to wait for days for the analysis to finish, we will not use all data but just a sub-sample of them.

You can access the sub-sampled data by changing to the folder

```
ln -s /vol/volume/denovo-assembly/2612_R1.fastq.gz 2612_R1.fastq.gz
ln -s /vol/volume/denovo-assembly/2612_R2.fastq.gz 2612_R2.fastq.gz
```

on the cluster. If you would like to repeat the practical on your own computational infrastructure, you will find information on how to access the files in the following boxes with

## 12. Introduction to de novo Genome Assembly

the title “Self-guided: data preparation”:

### **i** Self-guided: data preparation

For everyone, who runs this practical on their own infrastructure, you can download the sequencing data from here:

```
wget https://share.eva.mpg.de/index.php/s/CtLq2R9iqEcAFyg/download/2612_R1.fastq.gz
wget https://share.eva.mpg.de/index.php/s/mc5JrpDWdL4rC24/download/2612_R2.fastq.gz
```

### **?** Question

**How many sequences are in each FastQ file?**

Hint: You can run either `seqtk size <FastQ file>` or `bioawk -c fastx 'END{print NR}' <FastQ file>` to find this out.

### **i** Answer

There are about 3.25 million paired-end sequences in these files.

### 12.2.2. Preparing the sequencing data for *de novo* assembly

Before running the actual assembly, we need to pre-process our sequencing data. Typical pre-processing steps include the trimming of adapter sequences and barcodes from the sequencing data and the removal of host or contaminant sequences, such as the bacteriophage PhiX, which is commonly sequenced as a quality control.

Many assembly pipelines, such as nf-core/mag, have these steps automatically included, however, these steps can be performed prior to it, too. For this practical course, I have performed these steps for us and we could directly continue with the *de novo* assembly.

### **!** Important

The characteristic of ancient DNA samples that pre-determines the success of the *de novo* assembly is the **distribution of the DNA molecule length**. Determine this distribution prior to running the *de novo* assembly to be able to predict the results of the *de novo* assembly.

However, since the average length of the insert size of sequencing data (Figure 12.3) is highly correlated with the success of the assembly, we want to first evaluate it. For this we

## 12. Introduction to de novo Genome Assembly

can make use of the program fastp [Chen2018].

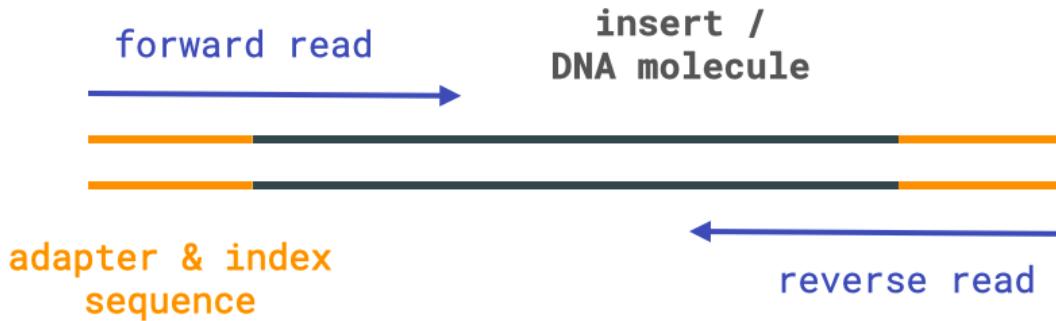


Figure 12.3.: Scheme of a read pair of Illumina sequencing data.

Fastp will try to overlap the two mates of a read pair and, if this is possible, return the length of the merged sequence, which is identical to insert size or the DNA molecule length. If the two mates cannot be overlapped, we will not be able to know the exact DNA molecule length but only know that it is longer than 290 bp (each read has a length of 150 bp and FastP requires a 11 bp overlap between the reads).

The final histogram of the insert sizes that is returned by FastP can tell us how well preserved the DNA of an ancient sample is (Figure 12.4). The more the distribution is skewed to the right, i.e. the longer the DNA molecules are, the more likely we are to obtain long contiguous DNA sequences from the *de novo* assembly. A distribution that is skewed to the left means that the DNA molecules are more highly degraded and this lowers our chances for obtaining long continuous sequences.

To infer the distribution of the DNA molecules, we can run the command

```
fastp --in1 2612_R1.fastq.gz \
      --in2 2612_R2.fastq.gz \
      --stdout --merge -A -G -Q -L --json /dev/null --html overlaps.html \
> /dev/null
```

12. Introduction to de novo Genome Assembly

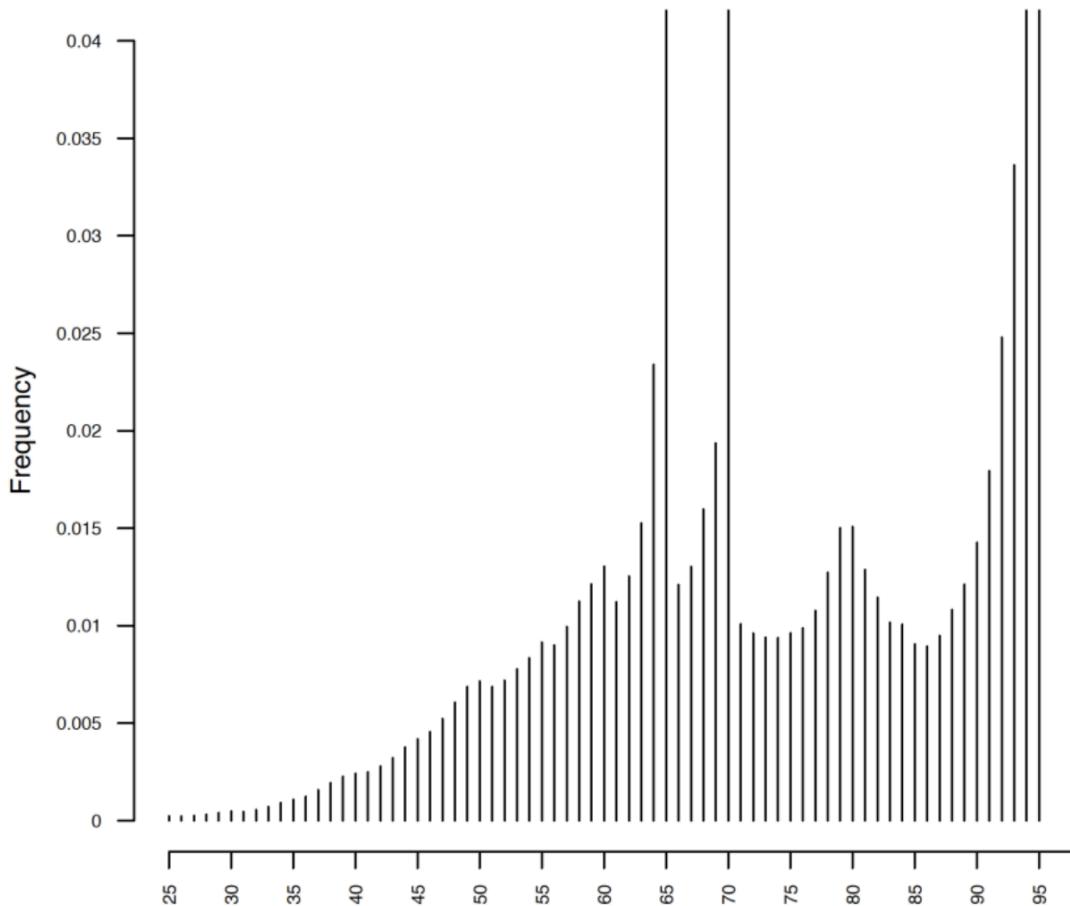


Figure 12.4.: Example of a DNA molecule length distribution of a well-preserved ancient DNA sample. This histogram belongs to a 700,000-year-old horse that was preserved in permafrost, as reported in @Orlando2013, Fig. S4.

?

### Question

Infer the distribution of the DNA molecule length of the sequencing data. Is this sample well-preserved?

Hint: You can easily inspect the distribution by opening the HTML report `overlaps.html`.

i

### Answer

Here is the histogram of the insert sizes determined by fastp (Figure 12.5). By default, fastp will only keep reads that are longer than 30 bp and requires an overlap between the read mates of 30 bp. The maximum read length is 150 bp, therefore, the histogram only spreads from 31 to 271 bp in total.

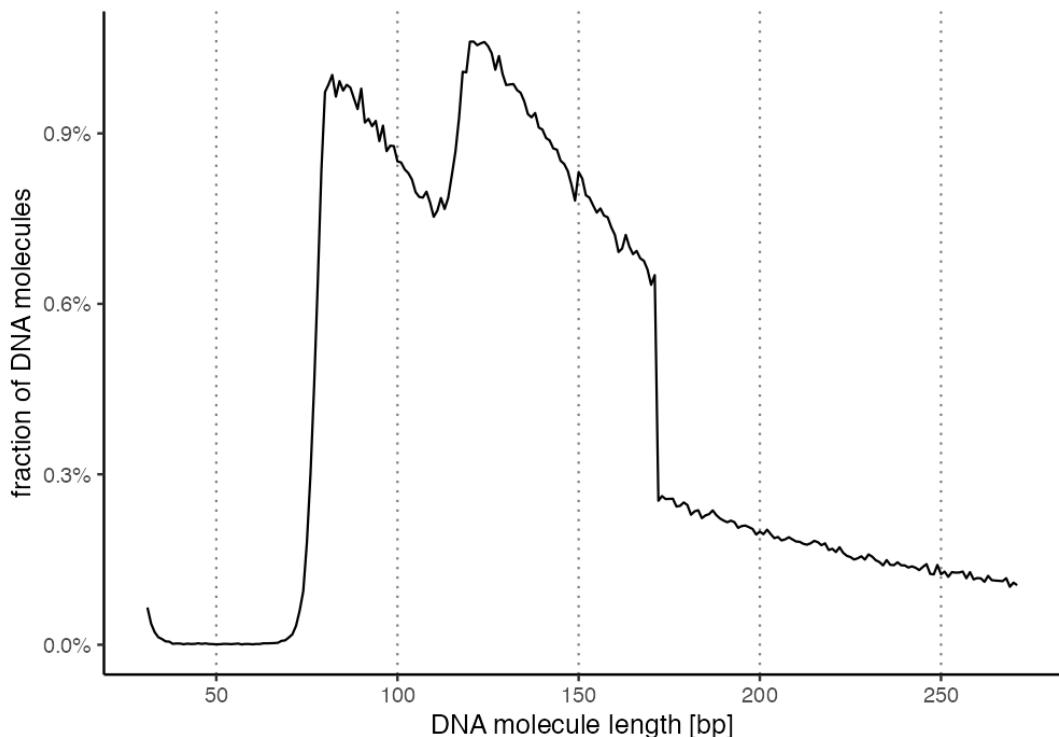


Figure 12.5.: Histogram of the insert sizes determined by fastp

The sequencing data for the sample **2612** were generated across eight different sequencing runs, which differed in their nominal length. Some sequencing runs were 2x

## 12. Introduction to de novo Genome Assembly

100 bp, while others were 2x 150 bp. This is the reason why we observe two peaks just short of 100 and 150 bp. The difference to the nominal length is caused by the quality trimming of the data.

Overall, we have almost no short DNA molecules ( $< 50$  bp) but most DNA molecules are longer than 80 bp. Additionally, there were  $> 200,000$  read pairs that could not be overlapped. Therefore, we can conclude that the sample **2612** is moderately degraded ancient DNA sample and has many long DNA molecules.

### 12.2.3. *De novo* assembly

Now, we will actual perform the *de novo* assembly on the sequencing data. For this, we will use the program MEGAHIT [@LiMegahit2015], a *de Bruijn*-graph assembler.

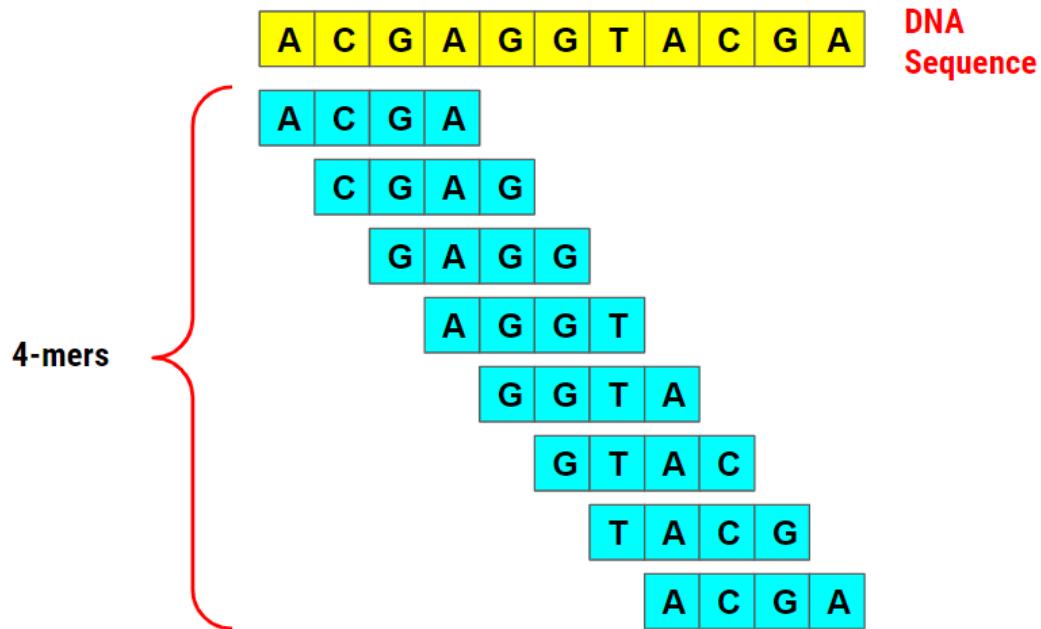


Figure 12.6.: Overview of inferring  $k$ -mers from a DNA sequence. Credit: <https://medium.com/swlh/bioinformatics-1-k-mer-counting-8c1283a07e29>

*De Bruijn* graph assemblers are together with overlap assemblers the predominant group of assemblers. They use  $k$ -mers (see Figure 12.6 for an example of 4-mers) extracted from the short-read sequencing data to build the graph. For this, they connect each  $k$ -mer to

## 12. Introduction to de novo Genome Assembly

adjacent  $k$ -mer using a directional edge. By walking along the edges and visiting each  $k$ -mer or node once, longer continuous sequences are reconstructed. This is a very rough explanation and I would advise you to watch this excerpt of a lecture by Rob Edwards from San Diego State University and a Coursera lecture by Ben Langmead from Johns Hopkins University, if you would like to learn more about it.

All *de Bruijn* graph assemblers work in a similar way so the question is why do we use MEGAHIT and not other programs, such as metaSPAdes?

### Pros and cons of MEGAHIT

#### Pros:

- low-memory footprint: can be run on computational infrastructure that does not have large memory resources
- can assembly both single-end and paired-end sequencing data
- the assembly algorithm can cope with the presence of high amounts of ancient DNA damage

#### Cons:

- lower assembly quality on modern comparative data, particularly a higher rate of mis-assemblies (CAMI II challenge)

In the Warinner group, we realised after some tests that MEGAHIT has a clear advantage when ancient DNA damage is present at higher quantities. While it produced a higher number of mis-assemblies compared to metaSPAdes when being evaluated on simulated modern metagenomic data [Critical Assessment of Metagenome Interpretation II challenge, @Meyer2022], it produces more long contigs when ancient DNA damage is present.

To *de novo* assemble the short-read sequencing data of the sample **2612** using MEGAHIT, we can run the command

```
megahit -1 2612_R1.fastq.gz \
        -2 2612_R2.fastq.gz \
        -t 14 --min-contig-len 500 \
        --out-dir megahit
```

This will use the paired-end sequencing data as input and return all contigs that are at least 500 bp long.

 Caution

While MEGAHIT is able to use merged sequencing data, it is advised to use the unmerged paired-end data as input. In tests using simulated data I have observed that MEGAHIT performed slightly better when using the unmerged data and it likely has something to do with its internal algorithm to infer insert sizes from the paired-end data.

While we are waiting for MEGAHIT to finish, here is a question:

 Question

**Which  $k$ -mer lengths did MEGAHIT select for the *de novo* assembly?**

 Answer

Based on the maximum read length, MEGAHIT decided to use the  $k$ -mer lengths of 21, 29, 39, 59, 79, 99, 119, and 141.

Now, as MEGAHIT has finished, we want to evaluate the quality of the assembly results. MEGAHIT has written the contiguous sequences (contigs) into a single FastA file stored in the folder `megahit`. We will process this FastA file with a small script, `calN50`, which will count the number of contigs and give us an overview of their length distribution.

To download the script and run it, we can execute the following commands:

```
wget https://raw.githubusercontent.com/lh3/calN50/master/calN50.js
k8 ./calN50.js megahit/final.contigs.fa
```

 Question

**How many contigs were assembled? What is the sum of the lengths of all contigs? What is the maximum, the median, and the minimum contig length?**

Hint: The maximum contig length is indicated by the label “N0”, the median by the label “N50”, and the minimum by the label “N100”?

**i** Answer

MEGAHIT assembled 3,606 contigs and their lengths sum up to 11.4 Mb. The maximum contig length was 448 kb, the median length was 15.6 kb, and the minimum length was 500 bp.

There is a final caveat when assembling ancient metagenomic data with MEGAHIT: while it is able to assemble sequencing data with a high percentage of C-to-T substitutions, it tends to introduce these changes into the contig sequences, too.

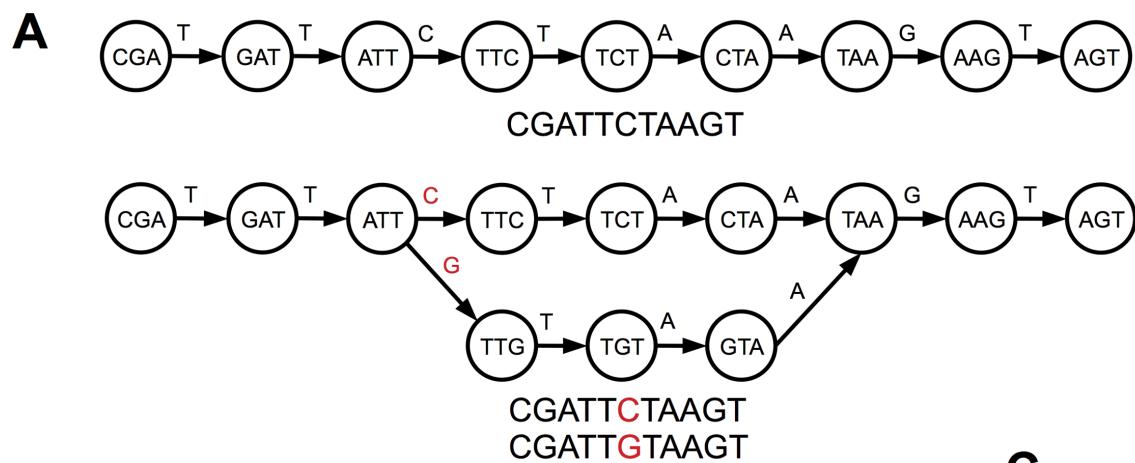


Figure 12.7.: *De Bruijn* graph with a bubble caused by a second allele. Adapted from [Leggett2013, Figure 1a]

These C-to-T substitutions are similar to biological single-nucleotide polymorphisms in the sequencing data. Both lead the introduction of bubbles in the *de Bruijn* graph when two alleles are present in the k-mer sequences (Figure 12.7) and the assembler decides during its pruning steps which allele to keep in the contig sequence.

While it does not really matter which allele is kept for biological polymorphisms, it does matter for technical artefacts that are introduced by the presence of ancient DNA damage. In our group we realised that the gene sequences that were annotated on the contigs of MEGAHIT tended to have a higher number of nonsense mutations compared to the known variants of the genes. After manual inspection, we observed that many of these mutations appeared because MEGAHIT chose the damage-derived T allele over the C allele or the damage-derived A allele over a G allele [see Klapper2023, Figure S1].

To overcome this issue, my colleagues Maxime Borry, James Fellows Yates and I developed a strategy to replace these damage-derived alleles in the contig sequences. This approach consists of aligning the short-read data against the contigs, performing genotyping along

## 12. Introduction to de novo Genome Assembly

them, and finally replacing all alleles for which we have strong support for an allele that is different from the one selected by MEGAHIT.

We standardised this approach and added it to the Nextflow pipeline nf-core/mag [@Krakau2022]. It can simply be activated by providing the parameter `--ancient_dna`.

### Caution

While MEGAHIT is able to assemble ancient metagenomic sequencing data with high amounts of ancient DNA damage, it tends to introduce damage-derived T and A alleles into the contig sequences instead of the true C and G alleles. This can lead to a higher number of nonsense mutations in coding sequences. We strongly advise you to correct such mutations, e.g. by using the ancient DNA workflow of the Nextflow pipeline nf-core/mag.

### 12.2.4. Aligning the short-read data against the contigs

After the assembly, the next detrimental step that is required for many subsequent analyses is the alignment of the short-read sequencing data back to assembled contigs.

Analyses that require these alignment information are for example:

- the correction of the contig sequences to remove damage-derived alleles
- the non-reference binning of contigs into MAGs for inferring the coverage along the contigs
- the quantification of the presence of ancient DNA damage

Aligning the short-read data to the contigs requires multiple steps:

1. Build an index from the contigs for the alignment program BowTie2
2. Align the short-read data against the contigs using this index with BowTie2
3. Calculate the mismatches and deletions of the short-read data against the contig sequences
4. Sort the aligned reads by the contig name and the coordinate they were aligned to
5. Index the resulting alignment file for faster access

To execute the alignment step we can run the following commands:

```
mkdir alignment
bowtie2-build -f megahit/final.contigs.fa alignment/2612
bowtie2 -p 14 --very-sensitive -N 1 -x alignment/2612 \
    -1 2612_R1.fastq.gz -2 2612_R2.fastq.gz | \
```

## 12. Introduction to de novo Genome Assembly

```
samtools view -Sb - | \
samtools calmd -u /dev/stdin megahit/final.contigs.fa | \
samtools sort -o alignment/2612.sorted.calmd.bam - \
samtools index alignment/2612.sorted.calmd.bam
```

However, these steps are rather time-consuming, even when we just have so little sequencing data as we do for our course example. The alignment is rather slow because we allow a single mismatch in the seeds that are used by the aligner BowTie2 to quickly determine the position of a read along the contig sequences (parameter `-N 1`). This is necessary because otherwise we might not be able to align reads with ancient DNA damage present on them. Secondly, the larger the resulting alignment file is the longer it takes to sort it by coordinate.

To save us some time and continue with the more interesting analyses, I prepared the resulting files for us. For this, I also corrected damage-derived alleles in the contig sequences. You can access these files on the cluster by running the following commands:

```
mkdir alignment
ln -s /vol/volume/denovo-assembly/2612.sorted.calmd.bam alignment/
ln -s /vol/volume/denovo-assembly/2612.sorted.calmd.bam.bai alignment/
ln -s /vol/volume/denovo-assembly/2612.fa alignment/
```

### Self-guided: data preparation

For everyone, who runs this practical on their own infrastructure, you can download the files:

```
mkdir alignment
wget -O alignment/2612.sorted.calmd.bam \
      https://share.eva.mpg.de/index.php/s/bDKgFLj9GpRFdPg/download/2612.sorted.calmd.bam
wget -O alignment/2612.sorted.calmd.bam.bai \
      https://share.eva.mpg.de/index.php/s/HWqg6fJj6ZEEBAL/download/2612.sorted.calmd.bam.bai
wget -O alignment/2612.fa \
      https://share.eva.mpg.de/index.php/s/z6ZAai42RPribX5/download/final.contigs.fa
```

### 12.2.5. Reconstructing metagenome-assembled genomes

There are typically two major approaches on how to study biological diversity of samples using the results obtained from the *de novo* assembly. The first one is to reconstruct metagenome-assembled genomes (MAGs) and to study the species diversity.

## 12. Introduction to de novo Genome Assembly

“Metagenome-assembled genome” is a convoluted term that means that we reconstructed a genome from metagenomic data via *de novo* assembly. While these reconstructed genomes, particularly the high-quality ones, are most likely a good representation of the genomes of the organisms present in the sample, the scientific community refrains from calling them a species or a strain. The reason is that for calling a genome a species or a strain additional analyses would be necessary, of which many would include the cultivation of the organism. For many samples, this is not feasible and therefore the community stuck to the term MAG instead.

The most commonly applied method to obtain MAGs is the so-called “non-reference binning”. Non-reference binning means that we do not try to identify contigs by aligning them against known reference genomes, but only use the characteristics of the contigs themselves to cluster them (Figure 12.8).

The two most commonly used characteristics are:

- the tetra-nucleotide frequency: the frequency of all 4-mers (e.g. AAAA, AAAC, AAAG etc.) in the contig sequence
- the coverage along the contig

The idea here is that two contigs that are derived from the same bacterial genome will likely have a similar nucleotide composition and coverage.

This approach works very well when the contigs are longer and they strongly differ in the nucleotide composition and the coverage from each other. However, if this is not the case, e.g. there is more than one strain of the same species in the sample, these approaches will likely not be able to assign some contigs to clusters: these contigs remain unbinned. In case there is a high number of unbinned contigs, one can also employ the more sensitive reference-based binning strategies but we will not cover this in this practical course.

There are a number of different binning tools out there that can be used for this task. Since this number is constantly growing, there have been attempts to standardise the test data sets that these tools are run on so that their performances can be easily compared. The most well-known attempt is the Critical Assessment of Metagenome Interpretation [Critical Assessment of Metagenome Interpretation II challenge, @Meyer2022], which released the latest comparison of commonly used tools for assembly, binning, and bin refinement in 2022. I recommend that you first check the performance of a new tool against the CAMI datasets before testing it to see whether it is worth using it.

Three commonly used binners are:

- metaBAT2 [@Kang2019, more than 1,300 citations]
- MaxBin2 [@WuMaxbin2016, more than 1,200 citations]
- CONCOCT [@Alneberg2014, more than 1,000 citation]

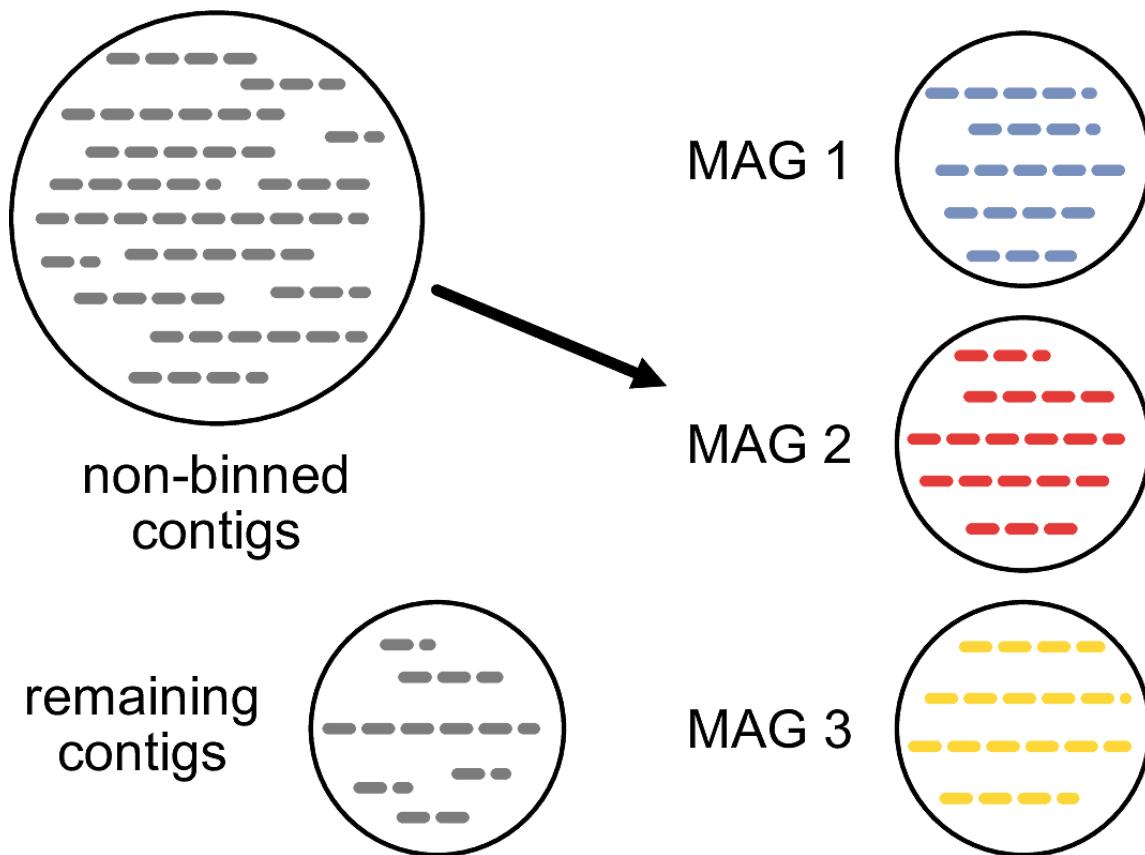


Figure 12.8.: Scheme of binning *de novo* assembled contigs into metagenome-assembled genomes. During the binning contigs are grouped into clusters based on their characteristics, such as tetra-nucleotide frequency and the coverage along the contigs. Clusters of contigs that fulfil a minimum of quality criteria are then considered as metagenome-assembled genomes. However, depending on the sample, a number of contigs will remain unbinned.

## 12. Introduction to de novo Genome Assembly

Each of these three binners employs a slightly different strategy. While metaBAT2 simply uses the two previously mentioned metrics, the tetra-nucleotide frequency and the coverage along the contigs, MaxBin2 additionally uses single-copy marker genes to infer the taxonomic origin of contigs. In contrast, CONCOCT also just uses the two aforementioned metrics but first performs a Principal Component Analysis (PCA) on these metrics and uses the PCA results for the clustering.

The easiest way to run all these three programs is the program metaWRAP [@Uritskiy2018]. metaWRAP is in fact a pipeline that allows you to assemble your contigs, bin them, and subsequently refine the resulting MAGs. However, the pipeline is not very well written and does not contain any strategies to deal with ancient metagenomic sequencing data. Therefore, I prefer to use different pipelines, such as nf-core/mag for the assembly, and only use metaWRAP for binning and bin refinement.

To skip the first steps of metaWRAP and start straight with the binning, we need to create the folder structure and files that metaWRAP expects:

```
mkdir -p metawrap/INITIAL_BINNING/2612/work_files  
ln -s $PWD/alignment/2612.sorted.calmd.bam \  
    metawrap/INITIAL_BINNING/2612/work_files/2612.bam  
mkdir -p metawrap/faux_reads  
echo "@" > metawrap/faux_reads/2612_1.fastq  
echo "@" > metawrap/faux_reads/2612_2.fastq
```

Now, we can start to run the binning. In this practical course, we will focus on metaBAT2 and MaxBin2. To bin the contigs with these binners, we execute:

```
conda activate metawrap-env  
metawrap binning -o metawrap/INITIAL_BINNING/2612 \  
    -t 14 \  
    -a alignment/2612.fa \  
    --metabat2 --maxbin2 --universal \  
    metawrap/faux_reads/2612_1.fastq metawrap/faux_reads/2612_2.fastq  
conda deactivate
```

### 🔥 Caution

MetaWRAP is not a well-written Python software and has not been updated for more than three years. It still relies on the deprecated Python v2.7. This is in conflict with many other tools and therefore it requires its own conda environment, `metawrap-env`. Do not forget to deactivate this environment afterwards again!

## 12. Introduction to de novo Genome Assembly

MetaWRAP will run metaBAT2 and MaxBin2 for us and store their respective output into sub-folders in the folder `metawrap/INITIAL_BINNING/2612`.

### 💡 Question

**How many bins did metaBAT2 and MaxBin2 reconstruct, respectively?  
Is there a difference in the genome sizes of these reconstructed bins?**

Hint: You can use the previously introduced script `k8 ./calN50.js` to analyse the genome size of the individual bins.

### 💡 Answer

metaBAT2 reconstructed seven bins, while MaxBin2 reconstructed only five bins. When comparing the genome sizes of these bins, we can see that despite having reconstructed fewer bins, MaxBin2's bins have on average larger genome size and all of them are at least 1.5 Mb. In contrast, five out of seven metaBAT2 bins are shorter than 1.5 Mb.

While we could have run these two binning softwares manually ourselves, there is another reason why we should use metaWRAP: it has a powerful bin refinement algorithm.

As we just observed, binning tools come to different results when performing non-reference binning on the same contigs. So how do we know which binning tool delivered the better or even correct results?

A standard approach is to identify single-copy marker genes that are specific for certain taxonomic lineages, e.g. to all members of the family *Prevotellaceae* or to all members of the kingdom archaea. If we find lineage-specific marker genes from more than one lineage in our bin, something likely went wrong. While in certain cases horizontal gene transfer could explain such a finding, it is much more common that a binning tool clustered two contigs from two different taxons.

During its bin refinement, metaWRAP first combines the results of all binning tools in all combinations. So it would merge the results of metaBAT2 and MaxBin2, metaBAT2 and CONCOCT, MaxBin2 and CONCOCT, and all three together. Afterwards, it evaluates the presence of lineage-specific marker genes on the contigs of the bins for every combination and the individual binning tools themselves. In the case that it would find marker genes of more than one lineage in a bin, it would split the bin into two. After having evaluated everything, metaWRAP selects the refined bins that have the highest quality score across the whole set of bins.

Using this approach, the authors of metaWRAP could show that they can outperform

## 12. Introduction to de novo Genome Assembly

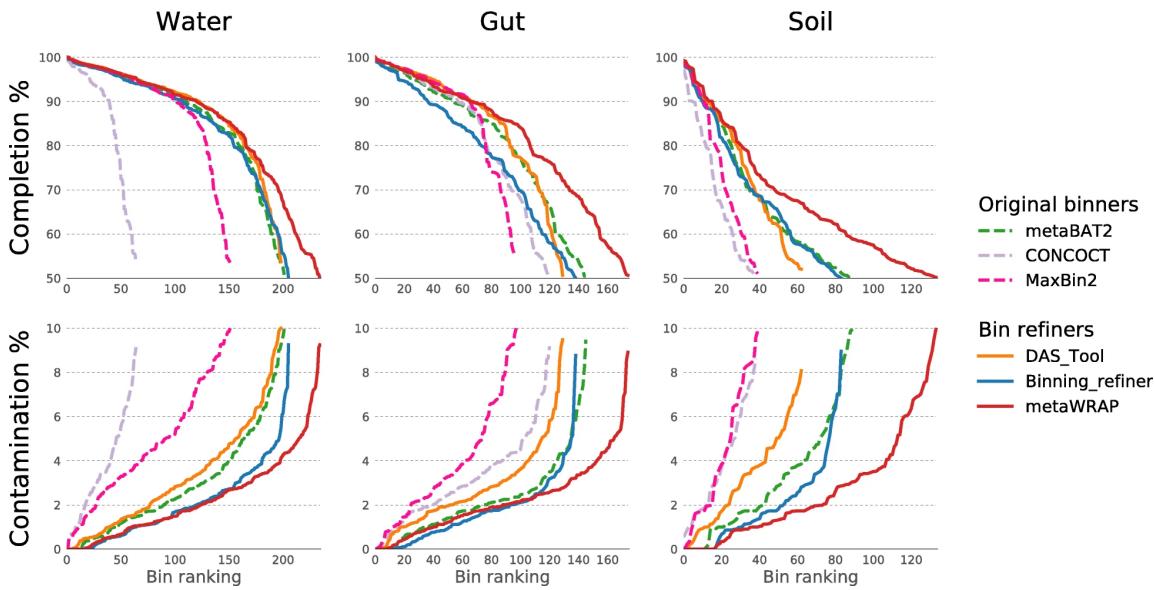


Figure 12.9.: Performance of metaWRAP’s bin refinement algorithm compared to other tools. Adapted from @Uritskiy2018, Fig. 4

the individual binning tools and other bin refinement algorithms both regarding the **completeness** and **contamination** that was estimated for the MAGs (Figure 12.9).

### Caution

Running metaWRAP’s bin refinement module requires about 72 GB of memory because it has to load a larger reference database containing the lineage-specific marker genes of checkM.

If your computer infrastructure cannot provide so much memory, I have prepared the results of metaWRAP’s bin refinement algorithm, `metawrap_50_10_bins.stats`, which can be found in the folder `/vol/volume/denovo-assembly`.

To apply metaWRAP’s bin refinement to the bins that we obtained from metaBAT2 and MaxBin2, we first need to install the software checkM [@Parks2015] that will provide the lineage-specific marker gene catalogue:

```
mkdir checkM
wget -O checkM/checkm_data_2015_01_16.tar.gz \
    https://data.ace.uq.edu.au/public/CheckM_databases/checkm_data_2015_01_16.tar.gz
tar xvf checkM/checkm_data_2015_01_16.tar.gz -C checkM
```

## 12. Introduction to de novo Genome Assembly

```
echo checkM | checkm data setRoot checkM
```

Afterwards, we can execute metaWRAP's bin refinement module:

```
mkdir -p metawrap/BIN_REFINEMENT/2612
metawrap bin_refinement -o metawrap/BIN_REFINEMENT/2612 \
    -t 14 \
    -c 50 \
    -x 10 \
    -A metawrap/INITIAL_BINNING/2612/maxbin2_bins \
    -B metawrap/INITIAL_BINNING/2612/metabat2_bins
```

The latter step will produce a summary file, `metawrap_50_10_bins.stats`, that lists all retained bins and some key characteristics, such as the genome size, the completeness estimate, and the contamination estimate. The latter two can be used to assign a quality score according to the Minimum Information for MAG (MIMAG; see info box).

### The Minimum Information for MAG (MIMAG)

The two most common metrics to evaluate the quality of MAGs are:

- the **completeness**: how many of the expected lineage-specific single-copy marker genes were present in the MAG?
- the **contamination**: how many of the expected lineage-specific single-copy marker genes were present more than once in the MAG?

These metric is usually calculated using the marker-gene catalogue of checkM [@Parks2015], also if there are other estimates from other tools such as BUSCO [@Manni2021], GUNC [@Orakov2021] or checkM2 [@Chklovski2022].

Depending on the estimates on completeness and contamination plus the presence of RNA genes, MAGs are assigned to the quality category following the Minimum Information for MAG criteria [@Bowers2017] You can find the overview here.

As these two steps will run rather long and need a large amount of memory and disk space, I have provided the results of metaWRAP's bin refinement. You can find the file here: `/vol/volume/denovo-assembly/metawrap_50_10_bins.stats`. Be aware that these results are based on the bin refinement of the results of three binning tools and include CONCOCT.

 Question

**How many bins were retained after the refinement with metaWRAP? How many high-quality and medium-quality MAGs did the refinement yield following the MIMAG criteria?**

Hint: You can more easily visualise tables on the terminal using the Python program `visidata`. After installing it with `pip install visidata`, you can open a table using `vd -f tsv /vol/volume/denovo-assembly/metawrap_50_10_-bins.stats`. Next to separating the columns nicely, it allows you to perform a lot of operations like sorting conveniently. Check the cheat sheet here.

 Answer

In total, metaWRAP retained five bins, similarly to MaxBin2. Of these five bins, the bins `bin.3` and `bin.4` had completeness and contamination estimates that would qualify them for being high-quality MAGs. However, we would need to check the presence of rRNA and tRNA genes. The other three bins are medium-quality bins because their completeness estimate was < 90%.

### 12.2.6. Taxonomic assignment of contigs

What should we do when we simply want to know to which taxon a certain contig most likely belongs to?

Reconstructing metagenome-assembled genomes requires multiple steps and might not even provide the answer in the case that the contig of interest is not binned into a MAG. Instead, it is sufficient to perform a sequence search against a reference database.

There are plenty of tools available for this task, such as:

- BLAST/DIAMOND
- Kraken2
- Centrifuge
- MMSeqs2

For each tool, we can either use pre-computed reference databases or compute our own one. The two taxonomic classification systems that are most commonly used are:

- NCBI Taxonomy
- GTDB

## 12. Introduction to de novo Genome Assembly

As for any task that involves the alignment of sequences against a reference database, the chosen reference database should fit the sequences you are searching for. If your reference database does not capture the diversity of your samples, you will not be able to assign a subset of the contigs. There is also a trade-off between a large reference database that contains all sequences and its memory requirement. @Wright2023 elaborated on this quite extensively when comparing Kraken2 against MetaPhlAn.

While all of these tools can do the job, I typically prefer to use the program MMSeqs2 [@Steinegger2017] because it comes along with a very fast algorithm based on aminoacid sequence alignment and implements a lowest common ancestor (LCA) algorithm (Figure 12.10). Recently, they implemented a *taxonomy* workflow [@Mirdita2021] that allows to efficiently assign contigs to taxons. Luckily, it comes with multiple pre-computed reference databases, such as the GTDB v207 reference database [@Parks2020], and therefore it is even more accessible for users.

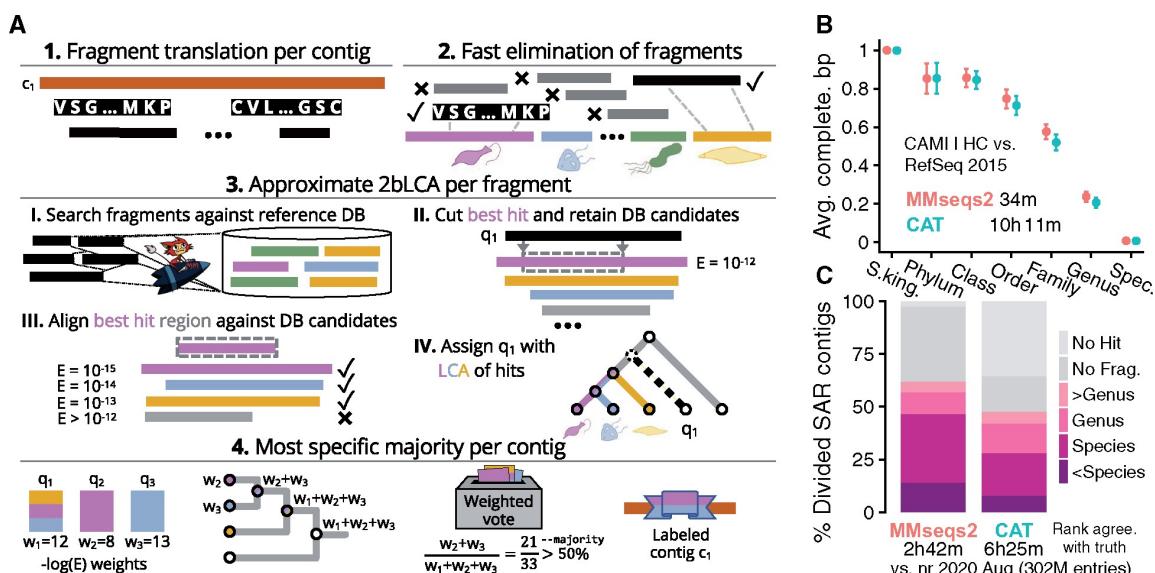


Figure 12.10.: Scheme of the *taxonomy* workflow implemented into MMSeqs2. Adapted from @Mirdita2021, Fig. 1.

### Caution

The latest version of the pre-computed GTDB reference database (r207) requires about 105 GB of harddisk space and 700 GB of memory for running.

As our computer infrastructure does not provide so much memory for every user, I pre-computed the results. You can find the results `2612.mmseqs2_gtdb.tsv` in the

## 12. Introduction to de novo Genome Assembly

folder `/vol/volume/denovo-assembly`.

An alternative for users with a less powerful infrastructure is the program KrakenUniq.

Before running MMSeqs2's *taxonomy* workflow against the GTDB reference database, we need to install it.

```
mkdir -p refdbs/mmseqs2/gtdb
mmseqs databases GTDB \
    refdbs/mmseqs2/gtdb /tmp --threads 14
```

Subsequently, we can align all the contigs of the sample 2612 against the GTDB r207 with MMSeqs2:

```
mkdir mmseqs2
mmseqs createdb alignment/2612.fa mmseqs2/2612.contigs
mmseqs taxonomy mmseqs2/2612.contigs \
    refdbs/mmseqs2/gtdb/mmseqs2_gtdb \
    mmseqs2/2612.mmseqs2_gtdb \
    /tmp \
    -a --tax-lineage 1 \
    --lca-ranks kingdom,phylum,class,order,family,genus,species \
    --orf-filter 1 \
    --remove-tmp-files 1 \
    --threads 14
mmseqs createtsv mmseqs2/2612.contigs \
    mmseqs2/2612.mmseqs2_gtdb \
```

### 💡 Question

**What is the proportion of contigs that could be assigned to the different taxonomic ranks, such as species or genus? What are the dominant taxa?**

Hint: You can access this information easily by opening the file using visidata: `vd 2612.mmseqs2_gtdb.tsv`

### 💡 Answer

From the 3,606 contigs, MMSeqs2's *taxonomy* workflow assigned 3,523 contigs to any taxonomy. For the rest, there was not enough information and they were discarded. From the 3,523 assigned contigs, 2,013 were assigned to the rank "species", while

1,137 could only be assigned to the rank “genus”.

The most contigs were assigned the archaeal species *Halococcus morrhuae* (n=386), followed by the bacterial species *Olsenella E sp003150175* (n=298) and *Collinsella sp900768795* (n=186).

### 12.2.7. Taxonomic assignment of MAGs

MMSeqs2’s *taxonomy* workflow is very useful to classify all contigs taxonomically. However, how would we determine which species we reconstructed by binning our contigs?

The simplest approach would be that we could summarise MMSeqs2’s taxonomic assignments of all contigs of a MAG and then determine which lineage is the most frequent one. Although this would work in general, there is another approach that is more sophisticated: GTDB toolkit [GTDBTK, @Chaumeil2020].

GTDBTK performs three steps to assign a MAG to a taxonomic lineage:

1. **Lineage identification** based on single-copy marker genes using Hidden Markov Models (HMMs)
2. **Multi-sequence alignment** of the identified marker genes
3. Placement of the MAG genome into a **fixed reference tree** at class level

The last step is particularly clever. Based on the known diversity of a lineage present in the GTDB, it will construct a reference tree with all known taxa of this lineage. Afterwards, the tree structure is fixed and an algorithm attempts to create a new branch in the tree for placing the unknown MAG based on both the tree structure and the multi-sequence alignment.

In most cases, both the simple approach of taking the majority of the MMSeqs2’s *taxonomy* results and the GTDBTK approach lead to very similar results. However, GTDBTK performs better when determining whether a new MAG potentially belongs to a new genus or even a new family.

To infer which taxa our five reconstructed MAGs represent, we can run the GTDBTK.

#### Caution

The latest version of the database used by GTDBTK (r207) requires about 70 GB of harddisk space and 80 GB of memory for running.

As our computer infrastructure does not provide so much memory for every user, I pre-computed the results. You can find the results `2612.gtdbtk_archaea.tsv` and

## 12. Introduction to de novo Genome Assembly

2612.gtdbtk\_bacteria.tsv in the folder /vol/volume/denovo-assembly.

First, we need to install the GTDB database:

```
mkdir -p refdbs/gtdbtk  
wget -O refdbs/gtdbtk/gtdbtk_v2_data.tar.gz \  
      https://data.gtdb.ecogenomic.org/releases/latest/auxillary_files/gtdbtk_v2_data.tar.gz  
tar xvf refdbs/gtdbtk/gtdbtk_v2_data.tar.gz -C refdbs/gtdbtk
```

Afterwards, we can run GTDBTK's *classify* workflow:

```
mkdir gtdbtk  
GTDBTK_DATA_PATH="$PWD/refdbs/gtdbtk/gtdbtk_r207_v2" \  
gtdbtk classify_wf --cpu 14 --extension fa \  
  --genome_dir metawrap/BIN_REFINEMENT/2612/metawrap_50_10_bins \  
  --out_dir gtdbtk
```

### 💡 Question

Do the classifications obtained by GTDBTK match the classifications that were assigned to the contigs using MMSeqs2? Would you expect these taxa given the archaeological context of the samples?

Hint: You can access the classification results of GTDBTK easily by opening the file using visidata: vd 2612.gtdbtk\_archaea.tsv and vd 2612.gtdbtk\_bacteria.tsv

### 💡 Answer

The five MAGs reconstructed from the sample 2612 were assigned to the taxa:

- bin.1: *Agathobacter rectalis*
- bin.2: *Halococcus morrhuae*
- bin.3: *Methanobrevibacter smithii*
- bin.4: *Ruminococcus bromii*
- bin.5: *Bifidobacterium longum*

All of these species were among the most frequent lineages that were identified by MMSeqs2's *taxonomy* workflow highlighting the large overlap between the methods. We would expect all five species to be present in our sample. All MAGs but bin.2 were assigned to human gut microbiome commensal species that are typically found in healthy humans. The MAG bin.2 was assigned to a halophilic archaeal species,

which is typically found in salt mines.

### 12.2.8. Evaluating the amount of ancient DNA damage

One of the common questions that remain at this point of our analysis is whether the contigs that we assembled show evidence for the presence of ancient DNA damage. If yes, we could argue that these microbes are indeed ancient, particularly when their DNA fragment length distribution is rather short, too.

MAGs typically consist of either tens or hundreds of contigs. For this case, many of the commonly used tools for quantifying the amount of ancient DNA damage, such as damageprofiler [@Neukamm2021] or mapDamage2 [@Jonsson2013], are not very well suited because they would require us to manually check each of their generated “smiley plots”, which visualise the amount of C-to-T substitutions at the end of reads.

Instead, we will use the program pyDamage [@Borry2021] that was written with the particular use-case of metagenome-assembled contigs in mind. Although pyDamage can visualise the amount of C-to-T substitutions at the 5' end of reads, it goes a step further and fits two models upon the substitution frequency (Figure 12.11). The null hypothesis is that the observed distribution of C-to-T substitutions at the 5' end of reads reflects a flat line, i.e. a case when no ancient DNA damage is present. The alternative model assumes that the distribution resembles an exponential decay, i.e. a case when ancient DNA damage is present. By comparing the fit of these two models to the observed data for each contig, pyDamage can quickly identify contigs that are likely of ancient origin without requiring the user to inspect the plots visually.

We can run pyDamage directly on the alignment data in BAM format:

```
pydamage analyze -w 30 -p 14 alignment/2612.sorted.calmd.bam
```

#### 💡 Question

Evaluate the pyDamage results with respect to the amount of C-to-T substitutions observed on the contigs! How many contigs does pyDamage consider to show evidence for ancient DNA damage? How much power (prediction accuracy) does it have for this decision? Which MAGs are strongly “ancient” or “modern”?

Hint: You can access pyDamage’s results easily by opening the file using visidata: vd pydamage\_results/pydamage\_results.tsv

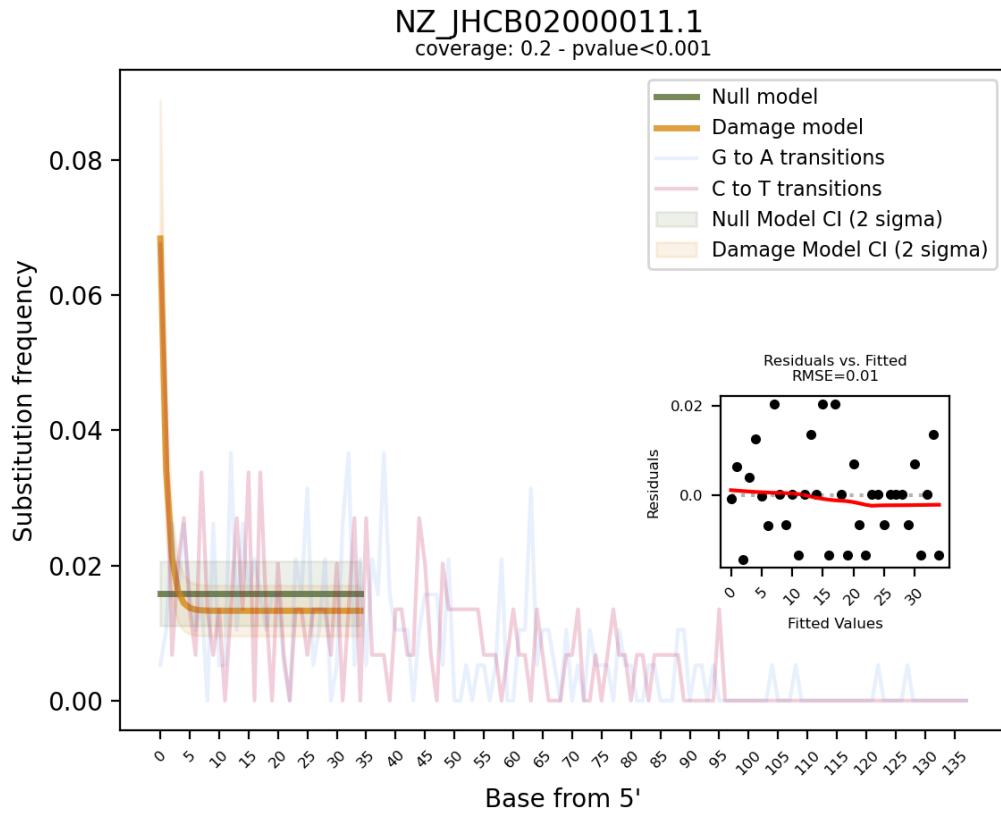


Figure 12.11.: Overview of the model comparison performed by pyDamage. The green line represents the null model, i.e. the absence of ancient DNA damage, while the orange line represents the alternative model, i.e. the presence of ancient DNA damage.

**i** Answer

From the 3,606 contigs, pyDamage inferred a q-value, i.e. a p-value corrected for multiple testing, of < 0.5 to 26 contigs. This is partially due to a high fraction of the contigs with no sufficient information to discriminate between the models (predicted accuracy of < 0.5). However, the majority of the contigs with a prediction accuracy of > 0.5 still had q-values of 0.05 and higher. This suggests that overall the sample did not show large evidence of ancient DNA damage.

This reflects also on the MAGs. Although four of the five MAGs were human gut microbiome taxa, they did not show strong evidence of ancient DNA damage. This suggests that the sample is too young and is well preserved.

### 12.2.9. Annotating genomes for function

The second approach on how to study biological diversity of samples using the assembly results is to compare the reconstructed genes and their functions with each other.

While it is very interesting to reconstruct the metagenome-assembled genomes from contigs and speculate about the evolution of a certain taxon, this does not always help when studying the ecology of a microbiome. Studying the ecology of a microbiome means to understand which taxa are present in an environment, what type of community do they form, what kind of natural products do they produce etc.?

With the lack of any data from culture isolates, it is rather difficult to discriminate from which species a reconstructed gene sequence is coming from, particularly when the contigs are short. Many microbial species exchange genes among each other via horizontal gene transfer which leads to multiple copies of a gene to be present in our metagenome and increases the level of difficulty further.

Because of this, many researchers tend to annotate all genes of a MAGs and compare the presence and absence of these genes across other genomes that were taxonomically assigned to the same taxon. This analysis, called pan-genome analysis, allows us to estimate the diversity of a microbial species with respect to their repertoire of protein-coding genes.

One of the most commonly used annotation tools for MAGs is Prokka [@Seemann2014], although it has recently been challenged by Bakta [@Schwengers2021]. The latter provides the same functionality as Prokka but incorporates more up-to-date reference databases for the annotation. Therefore, the scientific community is slowly shifting to Bakta.

Next to returning information on the protein-coding genes, Prokka also returns the annotation of RNA genes (tRNAs and rRNAs), which will help us to evaluate the quality of MAGs regarding the MIMAG criteria.

## 12. Introduction to de novo Genome Assembly

For this practical course, we will use Prokka and we will focus on annotating the MAG `bin.3.fa` that we reconstructed from the sample 2612.

```
prokka --outdir prokka \
    --prefix 2612_003 \
    --compliant --metagenome --cpus 14 \
    metawrap_50_10_bins/bin.3.fa
```

### 💡 Question

**Prokka has annotated our MAG. What type of files does Prokka return?**

**How many genes/tRNAs/rRNAs were detected?**

Hint: Check the file `prokka/2612_003.txt` for the number of annotated elements.

### 💡 Answer

Prokka returns the following files:

- `.faa`: the amino acid sequences of all identified coding sequences
- `.ffn`: the nucleotide sequences of all identified coding sequences
- `.fna`: all contigs in nucleotide sequence format renamed following Prokka's naming scheme
- `.gbk`: all annotations and sequences in GenBank format
- `.gff`: all annotations and sequences in GFF format
- `.tsv`: the tabular overview of all annotations
- `.txt`: the short summary of the annotation results

Prokka found 1,797 coding sequences, 32 tRNAs, but no rRNAs. Finding no rRNAs is a common issue when trying to assemble MAGs without long-read sequencing data and is not just characteristic for ancient DNA samples. However, this means that we cannot technically call this MAG a high-quality MAG due to the lack of the rRNA genes.

### 12.2.10. Summary

In this practical course you have gone through all the important steps that are necessary for *de novo* assembling ancient metagenomic sequencing data to obtain contiguous DNA sequences with little error. Furthermore, you have learned how to cluster these sequences into bins without using any references and how to refine them based on lineage-specific

## 12. Introduction to de novo Genome Assembly

marker genes. For these refined bins, you have evaluated their quality regarding common standards set by the scientific community and assigned the MAGs to its most likely taxon. Finally, we learned how to infer the presence of ancient DNA damage and annotate them for RNA genes and protein-coding sequences.

### 🔥 Cautionary note - sequencing depth

Be aware of the sequencing depth when you assemble your sample. This sample used in this practical course was not obtained by randomly subsampling but I subsampled the sample so that we are able to reconstruct MAGs.

The original sample had almost 200 million reads, however, I subsampled it to less than 5 million reads. You usually need a lot of sequencing data for *de novo* assembly and definitely more data than for reference-alignment based profiling. However, it also heavily depends on the complexity of the sample. So the best advice is: **just give it a try!**

## 13. Authentication and Decontamination



Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

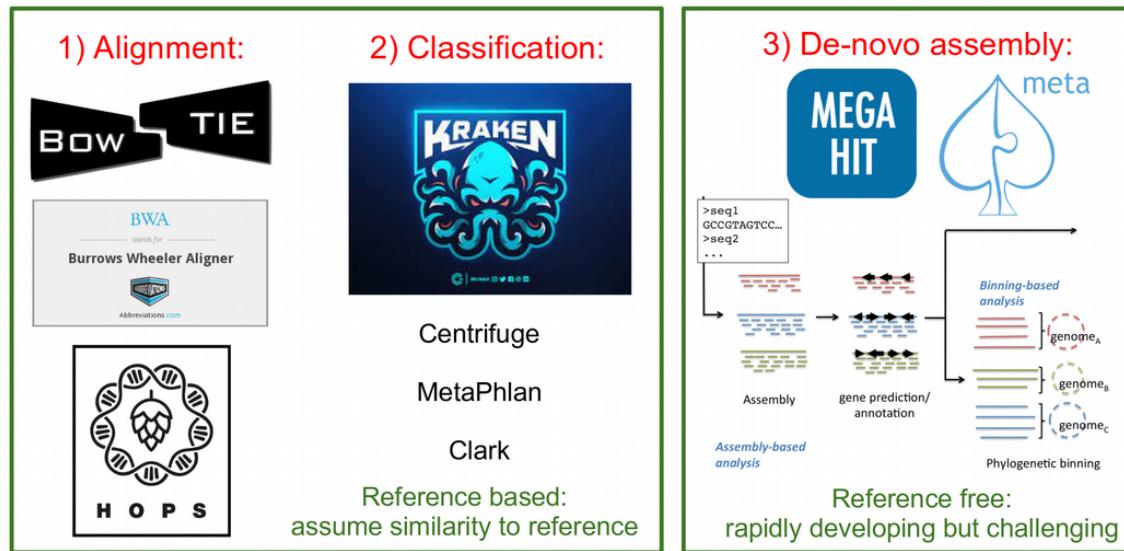
```
conda activate authentication-decontamination
```

# 14. Introduction

In ancient metagenomics we typically try to answer two questions: “Who is there?” and “How ancient?”, meaning we would like to detect an organism and investigate whether this organism is ancient. There are three typical ways to identify the presence of an organism in a metagenomic sample:

- alignment of DNA fragments to a reference genome (Bowtie, BWA, Malt etc.)
- taxonomic (kmer-based) classification of DNA fragments (Kraken, MetaPhlan, Centrifuge etc.)
- *de-novo* genome assembly (Megahit, metaSPAdes etc.)

The first two are reference-based, i.e. they assume a similarity of a query ancient DNA fragment to a modern reference genome in a database. This is a strong assumption, which might not be true for very old or very diverged ancient organisms. This is the case when the reference-free *de-novo* assembly approach becomes powerful. However, *de-novo* assembly has its own computational challenges for low-coverage ancient metagenomic samples that typically contain very short DNA fragments.



## 14. Introduction

While all the three types of metagenomic analysis are suitable for exploring composition of metagenomic samples, they do not directly validate the findings or provide information about ancient or endogenous status of the detected organism. It can happen that the detected organism

1. was mis-identified (the DNA belongs to another organism than initially thought),
2. has a modern origin (for example, lab or sequencing contaminant)
3. is of exogenous origin (for example, an ancient microbe that entered the host *post-mortem*).

Therefore, additional analysis is needed to follow-up each hit and demonstrate its ancient origin. Below, we describe a few steps that can help ancient metagenomic researchers to verify their findings and put them into biological context.

In this chapter, we will cover:

- how to recognize that a detected organism was mis-identified based on breadth / evenness of coverage
- how to validate findings by breadth of coverage filters via k-mer based taxonomic classification with KrakenUniq
- how to validate findings using alignments and assess mapping quality, edit distance and evenness of coverage profile
- how to detect modern contaminants via deamination profile, DNA fragmentation and post-mortem damage (PMD) scores
- how negative (blank) controls can help disentangle ancient organisms from modern contaminants
- how microbial source tracking can facilitate separating endogenous and exogenous microbial communities

The chapter has the following outline:

- Introduction
- Simulated ancient metagenomic data
- Genomic hit confirmation (how we see a true-positive hit)
  - Modern validation criteria
    - \* evenness and breadth of coverage
    - \* alignment quality (edit distance, mapq)
    - \* affinity to reference (percent identity, multi-allelic SNPs)
  - Ancient-specific validation
    - \* deamination profile (PMD scores)
    - \* DNA fragmentation

#### *14. Introduction*

- Microbiome contamination correction
  - Decontamination via negative controls (blanks)
  - Similarity to expected microbiome source (microbial source tracking)

## 15. Simulated ancient metagenomic data

You can begin the tutorial by changing into

```
cd /<path>/<to>/authentication-decontamination
```

In this chapter, we will use 10 simulated with gargammel ancient metagenomic samples from @Pochon2022-hj.

The screenshot shows the bioRxiv website interface. At the top left is the CSHL logo and the text "Cold Spring Harbor Laboratory". Next to it is the bioRxiv logo with the tagline "THE PREPRINT SERVER FOR BIOLOGY". To the right are links for "HOME", "SUBMIT", "FAQ", "BLOG", "ALERTS / RSS", "ABOUT", and "CHANNELS". Below these are search fields labeled "Search" and "Advanced Search". On the left, there's a "New Results" section. In the center, the preprint details are shown: title "aMeta: an accurate and memory-efficient ancient Metagenomic profiling workflow", authors (Zoé Pochon, Nora Bergfeldt, Emrah Kirdök, Mário Vicente, Thijessen Naidoo, Tom van der Valk, N. Ezgi Altınışık, Maja Krzewińska, Love Dalen, Anders Götherström, Claudio Mirabello, Per Unneberg, Nikolay Oskolkov), and DOI "doi: https://doi.org/10.1101/2022.10.03.510579". To the right, the preprint was posted on "October 05, 2022". There are download options: "Download PDF", "Print/Save Options", and "Data/Code". There are also links for "Email", "Share", and "Citation Tools". Navigation arrows for "Previous" and "Next" are also present.

Figure 15.1.: Screenshot of preprint of aMeta by Pochon et al. 2022

## 15. Simulated ancient metagenomic data

### Self guided: data preparation

The raw simulated data can be accessed via <https://doi.org/10.17044/scilifelab.21261405>

To download the simulated ancient metagenomic data please use the following command lines:

```
mkdir ameta/ && cd ameta/
wget https://figshare.scilifelab.se/ndownloader/articles/21261405/versions/1 \
&& unzip 1 && rm 1
```

The DNA reads were simulated with damage, sequencing errors and Illumina adapters, therefore one will have to trim the adapters first:

```
for i in $(ls *.fastq.gz)
do
sample_name=$(basename $i .fastq.gz)
cutadapt -a AGATCGGAAGAG --minimum-length 30 -o ${sample_name}.trimmed.fastq.gz ${sample_name}
done
```

Now, after the basic data pre-processing has been done, we can proceed with validation, authentication and decontamination analyses.

In here you will see a range of directories, each representing different parts of this tutorial. One set of trimmed ‘simulated’ reads from @Pochon2022-hj in `rawdata/`.

# 16. Genomic hit confirmation

Once an organism has been detected in a sample (via alignment, classification or *de-novo* assembly), one needs to take a closer look at multiple quality metrics in order to reliably confirm that the organism is not a false-positive detection and is of ancient origin. The methods used for this purpose can be divided into modern validation and ancient-specific validation criteria. Below, we will cover both of them.

## 16.1. Modern validation criteria

The modern validation methods aim at confirming organism presence regardless of its ancient status. The main approaches include evenness / breadth of coverage computation, assessing alignment quality, and monitoring affinity of the DNA reads to the reference genome of the potential host.

### 16.1.1. Depth vs breadth and evenness of coverage

Concluding organism presence by relying solely on the numbers of assigned sequenced reads (aka depth of coverage metric) turns out to be not optimal and too permissive, which may result in a large amount of false-positive discoveries. For example, when using alignment to a reference genome, the mapped reads may demonstrate non-uniform coverage as visualized in the Integrative Genomics Viewer (IGV) below.

In this case, DNA reads originating from another microbe were (mis-)aligned to *Yersina pestis* reference genome. It can be observed that a large number of the reads align only to a few conserved genomic loci. Therefore, even if many thousands of DNA reads are capable of aligning to the reference genome, the overall uneven alignment pattern suggests no presence of *Yersina pestis* in the metagenomic sample. Thus, not only the number of assigned reads (proportional to depth of coverage metric) but also the **breadth and evenness of coverage** metrics become of particular importance for verification of metagenomic findings, i.e. hits with DNA reads uniformly aligned across the reference genome are more likely to be true-positive detections (Figure 16.2).

## 16. Genomic hit confirmation

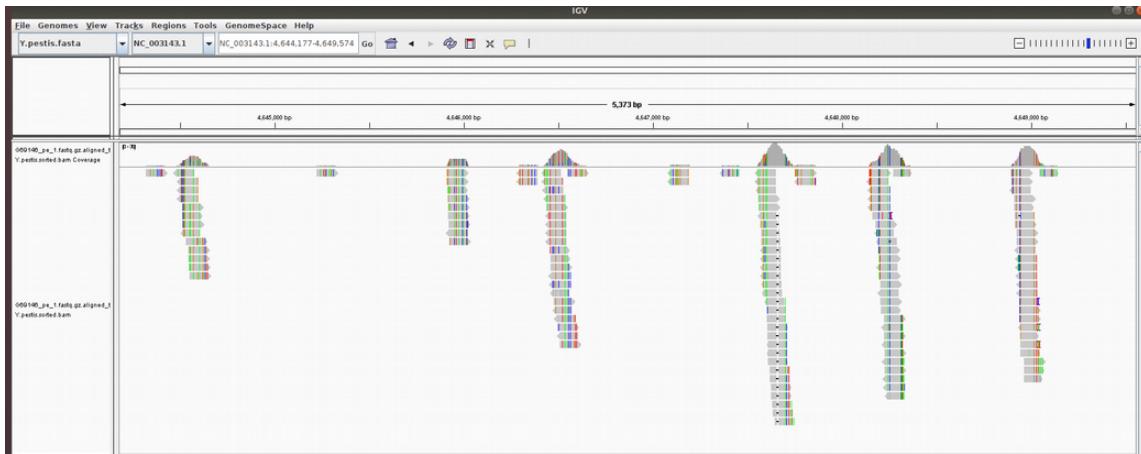


Figure 16.1.: Screenshot of the IGV software. The reference genome bar is shown at the top, and in the main panel tall isolated ‘towers’ of reads with lots of coloured bases representing read ‘stacking’ i.e., un-uniform distribution of reads across the whole genome (as expected of the correct reference genome), but accumulation of all reads in the same isolated places on the reference genome, with the many variants on the reads suggesting they are from different species and aligning to conserved regions.

In the next sections, we will show how to practically compute the breadth and evenness of coverage via KrakenUniq and Samtools.

### 16.1.2. Breadth of coverage via KrakenUniq

Here we are going to demonstrate that one can assess breadth of coverage information already at the taxonomic profiling step. Although taxonomic classifiers do not perform alignment, some of them, such as KrakenUniq and Kraken2 provide a way to infer breadth of coverage in addition to the number of assigned reads to a taxon. This allows for immediate filtering out a lot of false positive hits. Since Kraken-family classifiers are typically faster and less memory-demanding, i.e. can work with very large reference databases, compared to genome aligners, they provide a robust and fairly unbiased initial taxonomic profiling, which can still later be followed-up with proper alignment and computing evenness of coverage as described above.

16. Genomic hit confirmation

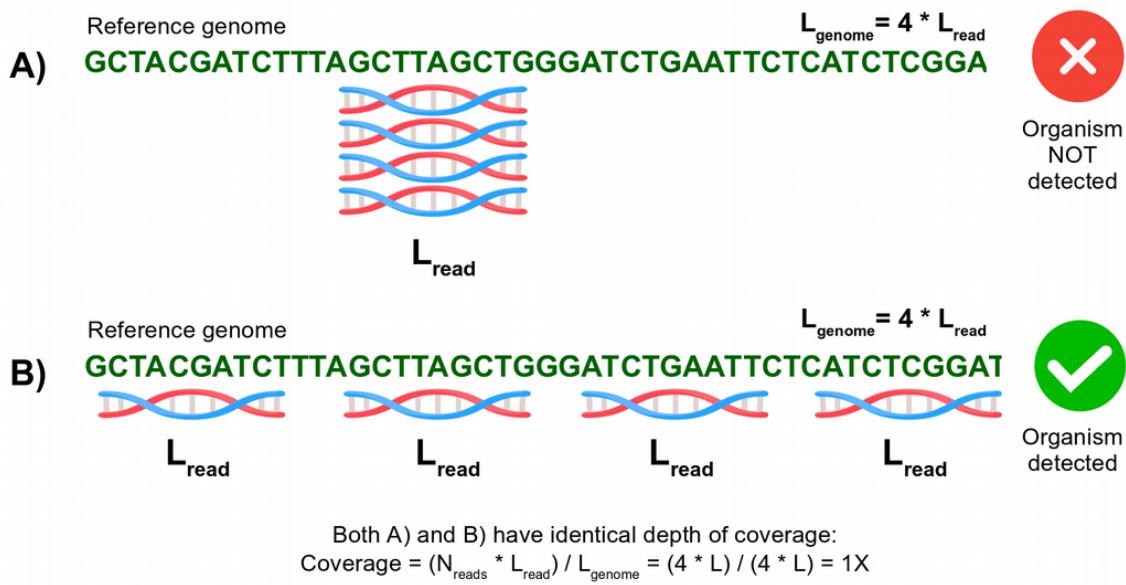


Figure 16.2.: Schematic diagram of the differences between a) read stacking (all reads aligned at one position fo the genome), indicating you've not correctly identified the organism, vs b) reads distributed across all the genome. A formula at the bottom of the image shows how both A and B have the same depth of coverage even though they have very different actual patterns on the genome.

**i** Self guided: data preparation

This step will require large amounts of memory and CPUs!, so if running yourself please note this step is better suited for a server, HPC cluster, or the cloud rather than on a laptop!

To profile the data with KrakenUniq one needs a database, a pre-built complete microbial NCBI RefSeq database can be accessed via <https://doi.org/10.17044/scilifelab.21299541>.

Please use the following command line to download the database:

```
cd krakenuniq/ ## If you've left it...

wget https://figshare.scilifelab.se/ndownloader/articles/21299541/versions/1 \
&& unzip 1 && rm 1
```

The following example command is how you would execute KrakenUniq.

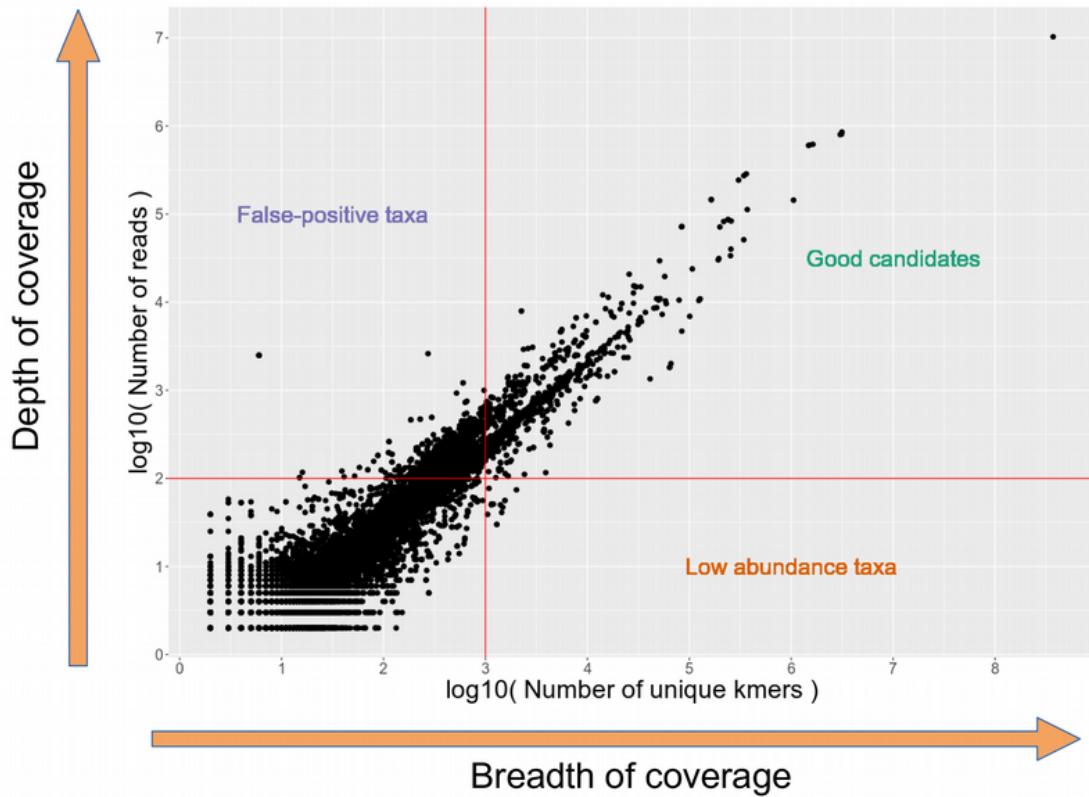
```
for i in $(ls *.trimmed.fastq.gz)
do
krakenuniq --db KRAKENUNIQ_DB --fastq-input $i --threads 20 \
--classified-out ${i}.classified_sequences.krakenuniq \
--unclassified-out ${i}.unclassified_sequences.krakenuniq \
--output ${i}.sequences.krakenuniq --report-file ${i}.krakenuniq.output
done
```

Taxonomic k-mer-based classification of the ancient metagenomic reads can be done via KrakenUniq. However as this requires a very large database file, the results from running KrakenUniq on the 10 simulated genomes can be found in.

```
cd krakenuniq/
```

KrakenUniq by default delivers a proxy metric for breadth of coverage called the **number of unique kmers** (in the 4th column of its output table) assigned to a taxon. KrakenUniq output can be easily filtered with respect to both depth and breadth of coverage, which substantially reduces the number of false-positive hits.

## 16. Genomic hit confirmation



We can filter the KrakenUniq output with respect to both depth (*taxReads*) and breadth (*kmers*) of coverage with the following custom Python script, which selects only species with at least 200 assigned reads and 1000 unique k-mers. After the filtering, we can see a *Yersinia pestis* hit in the *sample 10* that possess the filtering thresholds with respect to both depth and breadth of coverage.

```
for i in $(ls *.krakenuniq.output)
do
./scripts/filter_krakenuniq.py $i 1000 200 ../scripts/pathogensfound.very_inclusive.tab
done
```

## 16. Genomic hit confirmation

| %       | reads | taxReads | kmers  | dup  | cov       | taxID  | rank    | taxName                          |  |  |
|---------|-------|----------|--------|------|-----------|--------|---------|----------------------------------|--|--|
| 1.523   | 11855 | 11553    | 164882 | 1.05 | 0.03772   | 632    | species | <i>Yersinia pestis</i>           |  |  |
| 1.047   | 8151  | 7310     | 267081 | 1.06 | 0.03794   | 28450  | species | <i>Burkholderia pseudomallei</i> |  |  |
| 0.4386  | 3413  | 2560     | 49800  | 1    | 0.004508  | 28901  | species | <i>Salmonella enterica</i>       |  |  |
| 0.4294  | 3342  | 749      | 36158  | 1.03 | 0.003238  | 305    | species | <i>Ralstonia solanacearum</i>    |  |  |
| 0.2475  | 1926  | 1763     | 26882  | 1    | 0.01061   | 1314   | species | <i>Streptococcus pyogenes</i>    |  |  |
| 0.08545 | 665   | 294      | 5727   | 1.05 | 0.0004944 | 587753 | species | <i>Pseudomonas chlororaphis</i>  |  |  |

We can also easily produce a KrakenUniq taxonomic abundance table *krakenuniq\_abundance\_matrix.txt* using the custom R script below, which takes as argument the contents of the **krakenuniq/** folder containing the KrakenUniq output files.

```
Rscript ./scripts/krakenuniq_abundance_matrix.R . krakenuniq_abundance_matrix/ 1000 200
```

From the *krakenuniq\_abundance\_matrix.txt* table inside the resulting directory, it becomes clear that *Yersinia pestis* seems to be present in a few other samples in addition to sample 10.

|                                  | sample1 | sample2 | sample3 | sample4 | sample5 | sample6 | sample7 | sample8 | sample9 | sample10 |
|----------------------------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|----------|
| <i>Ralstonia solanacearum</i>    | 3628    | 3751    | 619     | 1804    | 1384    | 1608    | 1375    | 0       | 1112    | 749      |
| <i>Mycobacterium avium</i>       | 8236    | 8546    | 273     | 265     | 3221    | 4808    | 7750    | 6382    | 0       | 0        |
| <i>Burkholderia pseudomallei</i> | 7095    | 0       | 0       | 0       | 13082   | 0       | 4885    | 1456    | 0       | 7310     |
| <i>Salmonella enterica</i>       | 4356    | 4471    | 4205    | 3588    | 0       | 13854   | 0       | 0       | 1959    | 2560     |
| <i>Pseudomonas chlororaphis</i>  | 296     | 1024    | 0       | 977     | 374     | 677     | 276     | 0       | 0       | 294      |
| <i>Neisseria meningitidis</i>    | 465     | 502     | 0       | 0       | 7341    | 0       | 0       | 3268    | 5643    | 0        |
| <i>Yersinia pestis</i>           | 0       | 0       | 0       | 7174    | 957     | 0       | 11485   | 6461    | 0       | 11553    |

While KrakenUniq delivers information about breadth of coverage by default, you can also get this information from Kraken2.

For this one has to use a special flag *-report-minimizer-data* when running Kraken2 in order to get the breadth of coverage proxy which is called the **number of distinct minimizers** for the case of Kraken2. Below, we provide an example Kraken2 command line containing the distinct minimizer flag:

```
# Example command - do not run!
DBNAME=Kraken2_DB_directory
KRAKEN_INPUT=sample.fastq.gz
KRAKEN_OUTPUT=Kraken2_output_directory
kraken2 --db $DBNAME --fastq-input $KRAKEN_INPUT --threads 20 \
--classified-out $KRAKEN_OUTPUT/classified_sequences.kraken2 \
--unclassified-out $KRAKEN_OUTPUT/unclassified_sequences.kraken2 \
--output $KRAKEN_OUTPUT/sequences.kraken2 \
--report $KRAKEN_OUTPUT/kraken2.output \
--use-names --report-minimizer-data
```

Then the filtering of Kraken2 output with respect to breadth and depth of coverage can be done by analogy with filtering KrakenUniq output table. In case of *de-novo* assembly, the original DNA reads are typically aligned back to the assembled contigs, and the evenness / breadth of coverage can be computed from these alignments.

### 16.1.3. Evenness of coverage via Samtools

Now, after we have detected an interesting *Y. pestis* hit, we would like to follow it up, and compute multiple quality metrics (including proper breadth and evenness of coverage) from alignments (Bowtie2 aligner will be used in our case) of the DNA reads to the *Y. pestis* reference genome. Below, we download *Yersinia pestis* reference genome from NCBI, build its Bowtie2 index, and align trimmed reads against *Yersinia pestis* reference genome with Bowtie2. Do not forget to sort and index the alignments as it will be important for computing the evenness of coverage. It is also recommended to remove multi-mapping reads, i.e. the ones that have MAPQ = 0, at least for Bowtie and BWA aligners that are commonly used in ancient metagenomics. Samtools with *-q* flag can be used to extract reads with MAPQ > = 1.

 Self guided: data preparation

```
cd /<path/<to>/authentication-decontamination/bowtie2

## Download reference genome
NCBI=https://ftp.ncbi.nlm.nih.gov; ID=GCF_000222975.1_ASM22297v1
wget $NCBI/genomes/all/GCF/000/222/975/${ID}/${ID}_genomic.fna.gz
```

## 16. Genomic hit confirmation

```
cd /<path/<to>/authentication-decontamination/bowtie2

## Prepare reference genome and build Bowtie2 index
gunzip ${ID}_genomic.fna.gz; echo NC_017168.1 > region.bed
seqtk subseq ${ID}_genomic.fna region.bed > NC_017168.1.fasta
bowtie2-build --large-index NC_017168.1.fasta NC_017168.1.fasta --threads 10

## Run alignment of raw reads against FASTQ
bowtie2 --large-index -x NC_017168.1.fasta --end-to-end --threads 10 \
--very-sensitive -U ../rawdata/sample10.trimmed.fastq.gz | samtools view -bS -h -q 1 \
-@ 20 - > Y.pestis_sample10.bam

## Sort and index BAM files for rapid access in downstream commands
samtools sort Y.pestis_sample10.bam -@ 10 > Y.pestis_sample10.sorted.bam
samtools index Y.pestis_sample10.sorted.bam
```

Next, the breadth / evenness of coverage can be computed from the BAM-alignments via *samtools depth* as follows:

```
samtools depth -a Y.pestis_sample10.sorted.bam > Y.pestis_sample10.sorted.boc
```

and visualized using for example the following R code snippet (alternatively aDNA-BAMPlotter can be used):

Load R by typing R into your terminal. Note the following may take a minute or so to run.

```
# Read output of samtools depth commands
df <- read.delim("Y.pestis_sample10.sorted.boc", header = FALSE, sep = "\t")
names(df) <- c("Ref", "Pos", "N_reads")

# Split reference genome in tiles, compute breadth of coverage for each tile
N_tiles <- 500
step <- (max(df$Pos) - min(df$Pos)) / N_tiles
tiles <- c(0:N_tiles) * step; boc <- vector()
for(i in 1:length(tiles))
{
  df_loc <- df[df$Pos >= tiles[i] & df$Pos < tiles[i+1], ]
  boc <- append(boc, rep(sum(df_loc$N_reads > 0) / length(df_loc$N_reads),
  dim(df_loc)[1]))
```

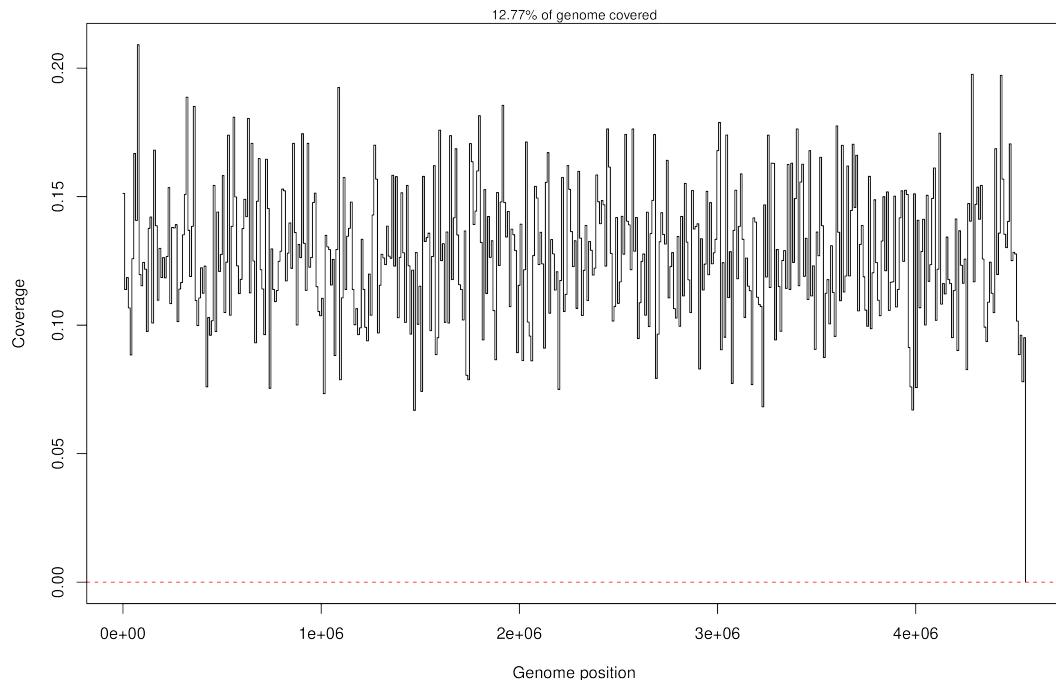
## 16. Genomic hit confirmation

```
}

boc[is.na(boc)]<-0; df$boc <- boc
plot(df$boc ~ df$Pos, type = "s", xlab = "Genome position", ylab = "Coverage")
abline(h = 0, col = "red", lty = 2)
mtext(paste0(round((sum(df$N_reads > 0) / length(df$N_reads)) * 100, 2),
"% of genome covered"), cex = 0.8)
```

Once finished examining the plot you can quit R

```
## Press 'n' when asked if you want to save your workspace image.
quit()
```



In the R script above, we simply split the reference genome into  $N_{\text{tiles}}$  tiles and compute the breadth of coverage (number of reference nucleotides covered by at least one read normalized by the total length) locally in each tile. By visualizing how the local breadth of coverage changes from tile to tile, we can monitor the distribution of the reads across the

## 16. Genomic hit confirmation

reference genome. In the evenness of coverage figure above, the reads seem to cover all parts of the reference genome uniformly, which is a good evidence of true-positive detection, even though the total mean breadth of coverage is low due to the low total number of reads.

### 16.1.4. Alignment quality

In addition to evenness and breadth of coverage, it is very informative to monitor how well the metagenomic reads map to a reference genome. Here one can control for **mapping quality** (MAPQ field in the BAM-alignments) and the number of mismatches for each read, i.e. **edit distance**.

Mapping quality (MAPQ) can be extracted from the 5th column of BAM-alignments using Samtools and *cut* command in bash.

```
 samtools view Y.pestis_sample10.sorted.bam | cut -f5 > mapq.txt
```

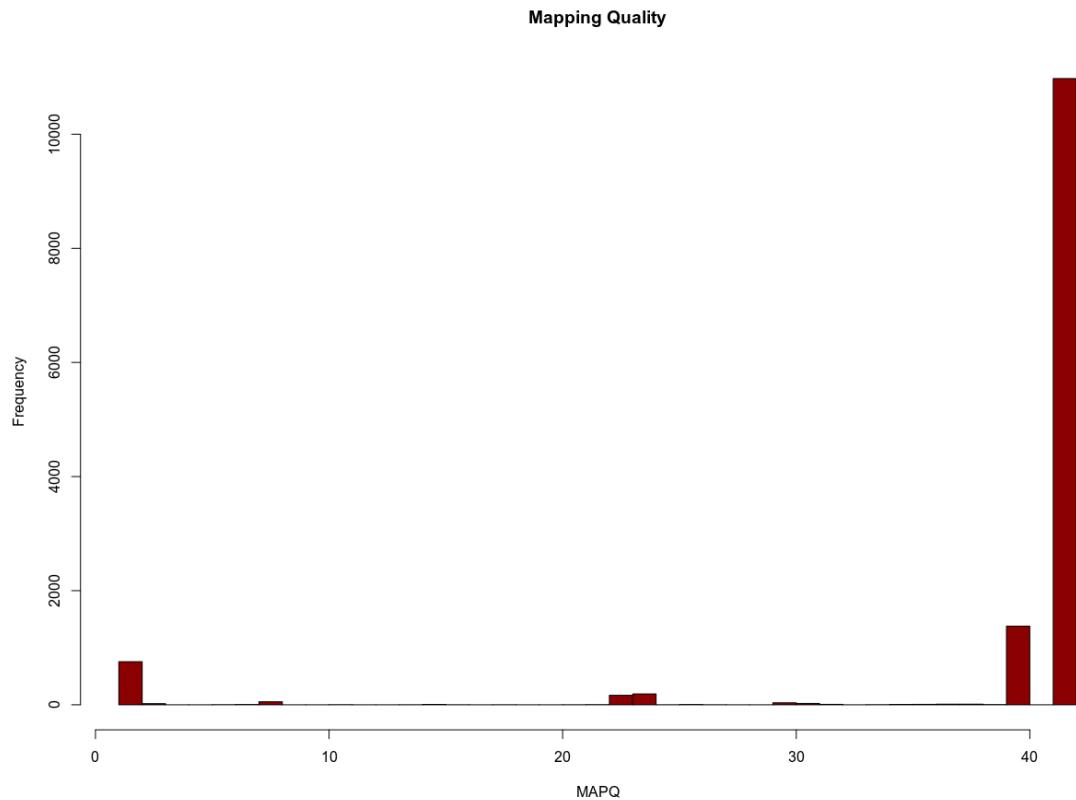
Then the 5th column of the filtered BAM-alignment can be visualized via a simple histogram in R as below for two random metagenomic samples.

```
 hist(as.numeric(readLines("mapq.txt")), col = "darkred", breaks = 100)
```

Once finished examining the plot you can quit R

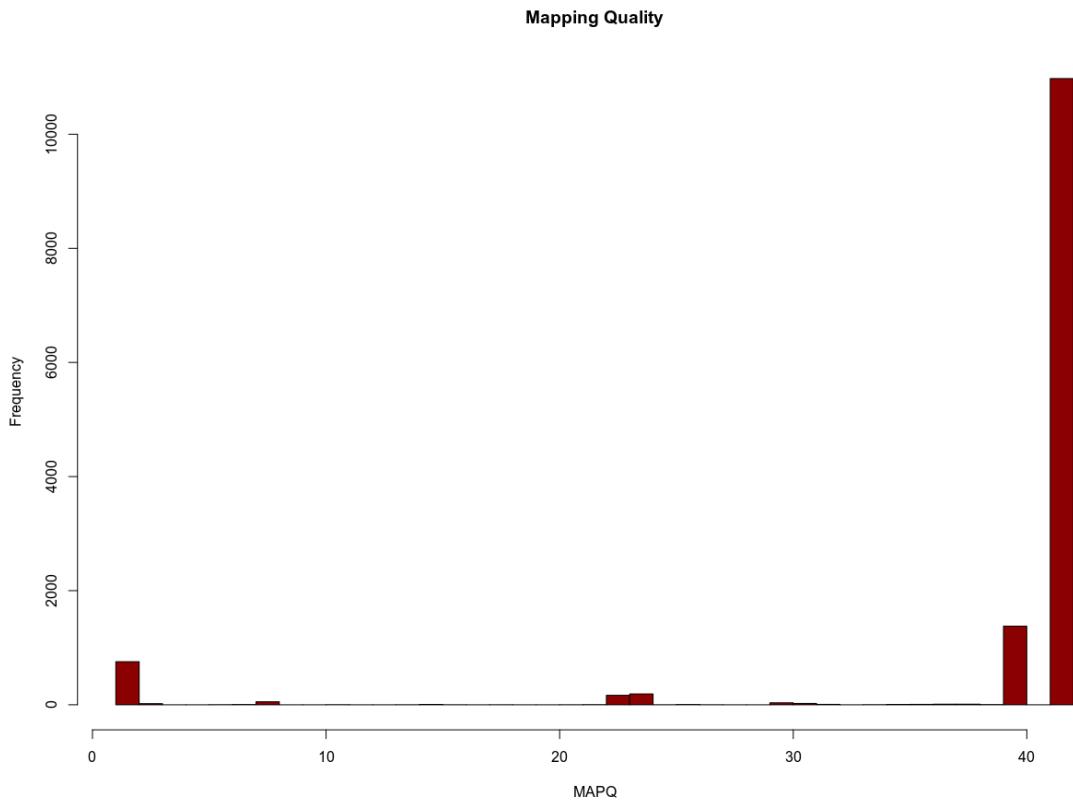
```
## Press 'n' when asked if you want to save your workspace image.  
quit()
```

## 16. Genomic hit confirmation



Note that MAPQ scores are computed slightly differently for Bowtie and BWA, so they are not directly comparable, however, for both  $\text{MAPQ} \sim 10-30$ , as in the histograms below, indicates good affinity of the DNA reads to the reference genome. here we provide some examples of how typical MAPQ histograms for Bowtie2 and BWA alignments can look like:

## 16. Genomic hit confirmation

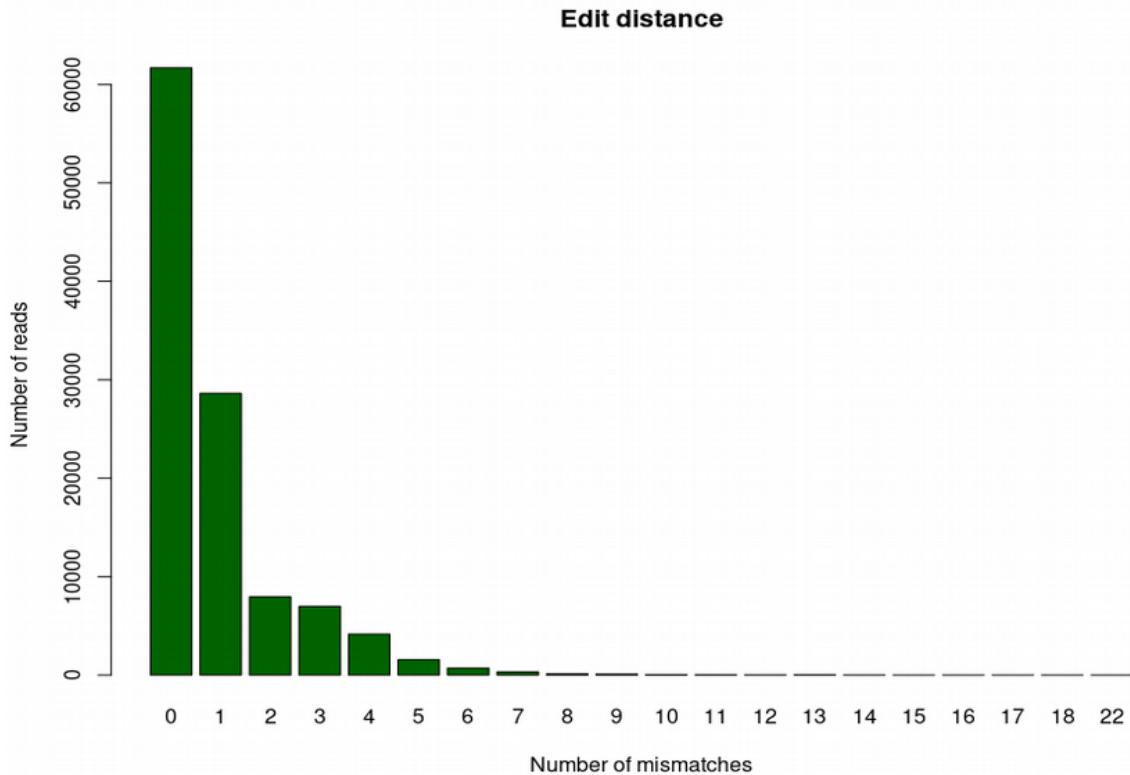


Edit distance can be extracted by gathering information in the NM-tag inside BAM-alignments, which reports the number of mismatches for each aligned read. This can be done either in bash / awk, or using handy functions from *Rsamtools* R package:

```
library("Rsamtools")
param <- ScanBamParam(tag = "NM")
bam <- scanBam("Y.pestis_sample10.sorted.bam", param = param)
barplot(table(bam[[1]]$tag$NM), ylab="Number of reads", xlab="Number of mismatches")
```

Once finished examining the plot you can quit R

```
## Press 'n' when asked if you want to save your workspace image.
quit()
```



In the barplot above we can see that the majority of reads align either without or with very few mismatches, which is an evidence of high affinity of the aligned reads with respect to the reference genome. For a true-positive finding, the edit distance barplot typically has a decreasing profile. However, for a very degraded DNA, it can have a mode around 1 or 2, which can also be reasonable. A false-positive hit would have a mode of the edit distance barplot shifted toward higher numbers of mismatches.

### 16.1.5. Affinity to reference

Very related to edit distance is another alignment validation metric which is called **percent identity**. It represents a barplot demonstrating the numbers of reads that are 100% identical to the reference genome (i.e. map without a single mismatch), 99% identical, 98% identical etc. Misaligned reads originating from another related organism have typically most reads with percent identity of 93-96%. In the figure below, the panels (c-e) demonstrate different percent identity distributions. In panel c, most reads show a high similarity to the reference, which indicates a correct assignment of the reads to the reference. In panel d, most reads are highly dissimilar to the reference, which suggests that they originate from

## 16. Genomic hit confirmation

different related species. In some cases, as in panel e, a mixture of correctly assigned and misassigned reads can be observed.



Another important way to detect reads that cross-map between related species is **haplody** or checking the amount of **multi-allelic SNPs**. Because bacteria are haploid organisms, only one allele is expected for each genomic position. Only a small number of multi-allelic sites are expected, which can result from a few mis-assigned or incorrectly aligned reads. In the figure above, panels (f-i) demonstrate histograms of SNP allele frequency distributions. Panel f demonstrates the situation when we have only a few multi-allelic sites originating from a misaligned reads. This is a preferable case scenario corresponding to correct assignment of the reads to the reference. Please also check the corresponding “Good alignments” IGV visualization to the right in the figure above.

In contrast, a large number of multi-allelic sites indicates that the assigned reads originate from more than one species or strain, which can result in symmetric allele frequency distributions (e.g., if two species or strains are present in equal abundance) (panel g) or asymmetric distributions (e.g., if two species or strains are present in unequal abundance) (panel h). A large number of mis-assigned reads from closely related species can result in a large number of multi-allelic sites with low frequencies of the derived allele (panel i). The situations (g-i) correspond to incorrect assignment of the reads to the reference. Please also check the corresponding “Bad alignments” IGV visualization to the right in the figure above.

## 16.2. Ancient-specific validation criteria

In contrast to modern genomic hit validation criteria, the ancient-specific validation methods concentrate on DNA degradation and damage pattern as ultimate signs of ancient DNA.

## 16. Genomic hit confirmation

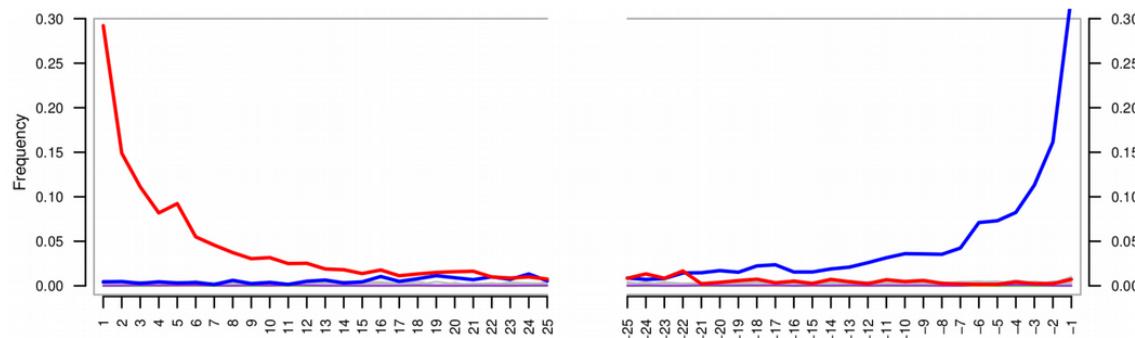
Below, we will discuss deamination profile, read length distribution and post mortem damage (PMD) scores metrics that provide good confirmation of ancient origin of the detected organism.

### 16.2.1. Ancient status

Checking evenness of coverage and alignment quality can help us to make sure that the organism we are thinking about is really present in the metagenomic sample. However, we still need to address the question “How ancient?”. For this purpose we need to compute **deamination profile** and **read length distribution** of the aligned reads in order to prove that they demonstrate damage pattern and are sufficiently fragmented, which would be a good evidence of ancient origin of the detected organisms.

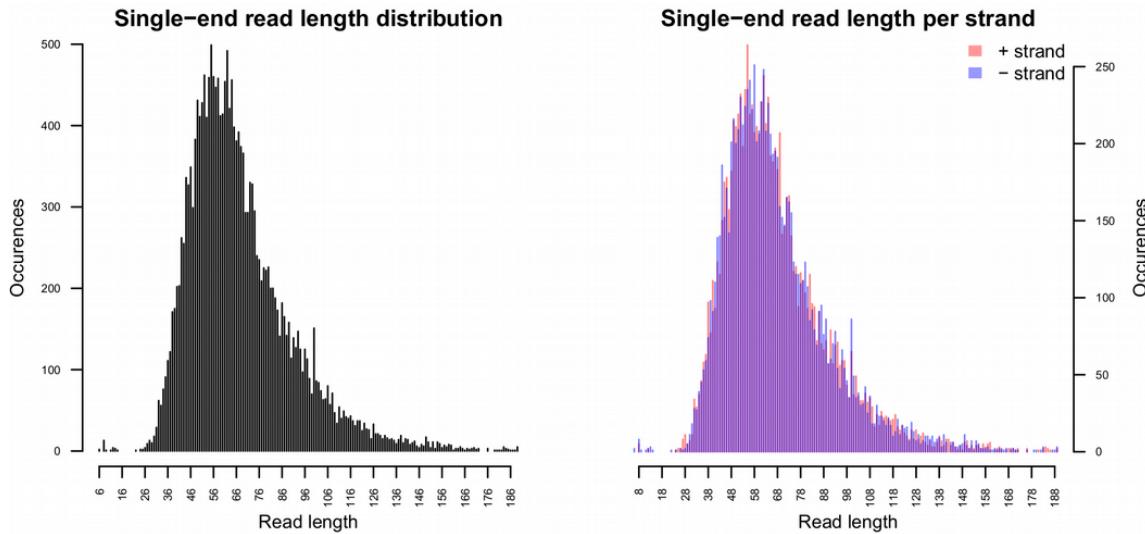
Deamination profile of a damaged DNA demonstrate an enrichment of C / T polymorphisms at the ends of the reads compared to all other single nucleotide substitutions. There are several tools for computing deamination profile, but perhaps the most popular is mapDamage. The tool can be run using the following command line, still in the `authentication-decontamination/bowtie2/` directory:

```
mapDamage -i Y.pestis_sample10.sorted.bam -r NC_017168.1.fasta -d mapDamage_results/ --merge
```



maDamage delivers a bunch of useful statistics, among other read length distribution can be checked. A typical mode of DNA reads should be within a range 30-70 base-pairs in order to be a good evidence of DNA fragmentation. Reads longer than 100 base-pairs are more likely to originate from modern contamination.

## 16. Genomic hit confirmation



Another useful tool that can be applied to assess how DNA is damaged is PMDtools which is a maximum-likelihood probabilistic model that calculates an ancient score, **PMD score**, for each read. The ability of PMDtools to infer ancient status with a single read resolution is quite unique and different from mapDamage that can only assess deamination based on a number of reads. PMD scores can be computed using the following command line, please note that Python2 is needed for this purpose.

```
samtools view -h Y.pestis_sample10.bam | pmdtools --printDS > PMDscores.txt
```

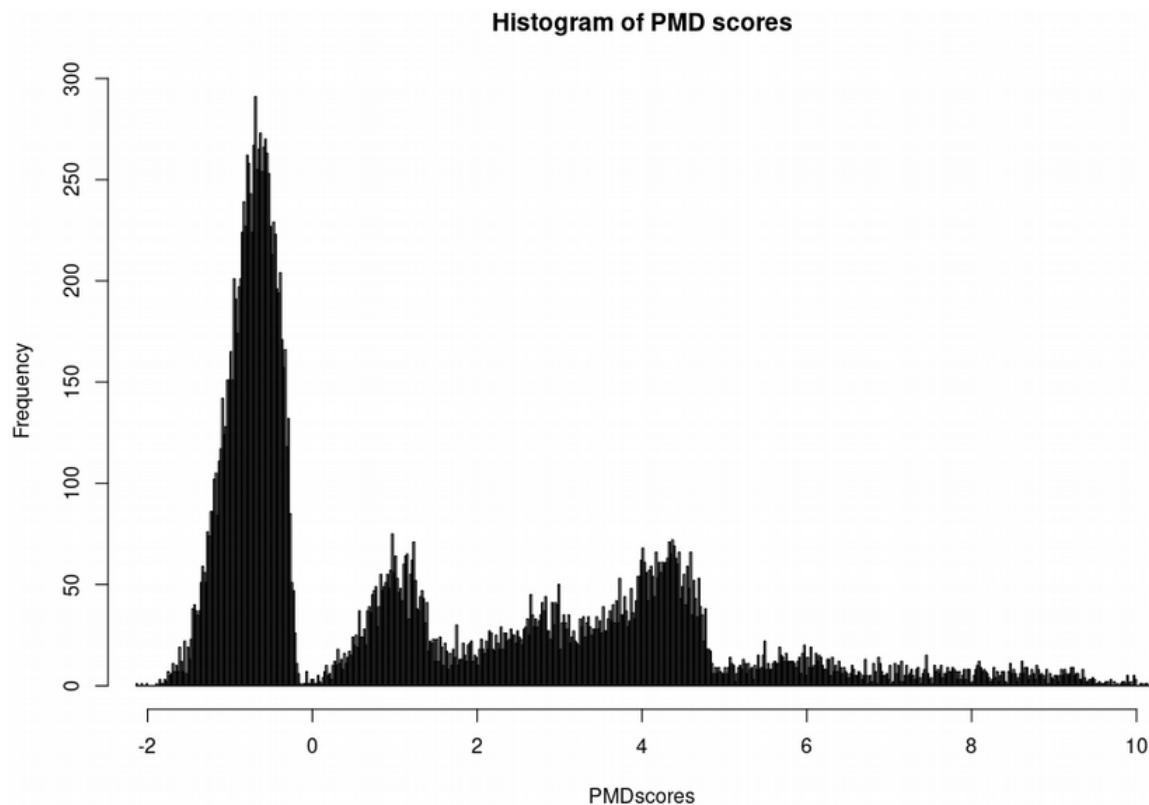
The distribution of PMD scores can be visualized via a histogram in R as follows:

```
pmd_scores <- read.delim("PMDscores.txt", header = FALSE, sep = "\t")
hist(pmd_scores$V4, breaks = 1000, xlab = "PMDscores")
```

Once finished examining the plot you can quit R

```
## Press 'n' when asked if you want to save your workspace image.
quit()
```

## 16. Genomic hit confirmation

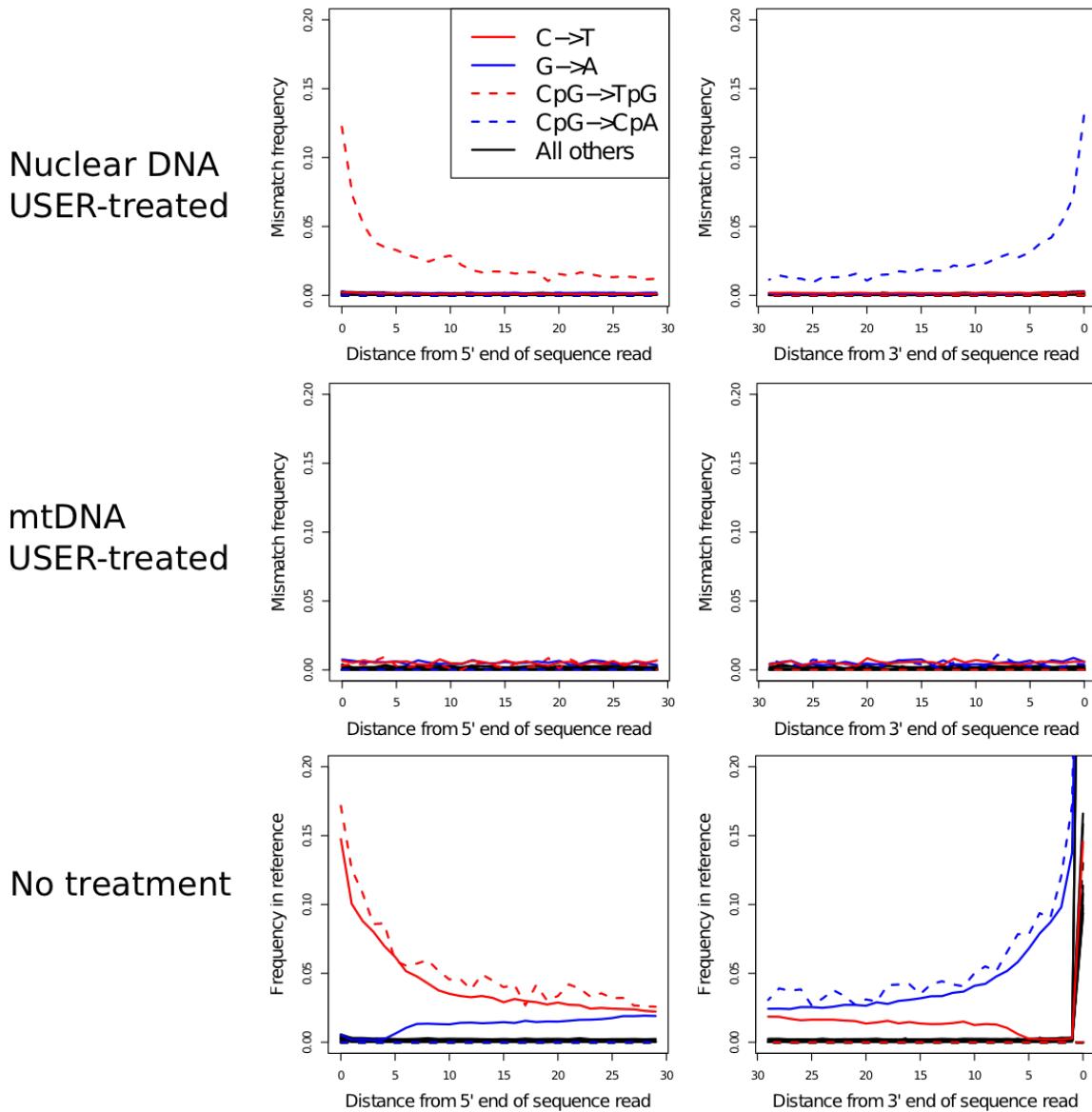


Typically, reads with PMD scores greater than 3 are considered to be reliably ancient, i.e. damaged, and can be extracted for taking a closer look. Therefore PMDtools is great for separating ancient reads from modern contaminant reads.

As mapDamage, PMDtools can also compute deamination profile. However, the advantage of PMDtools that it can compute deamination profile for UDG / USER treated samples (with the flag *-CpG*). For this purpose, PMDtools uses only CpG sites which escape the treatment, so deamination is not gone completely and there is a chance to authenticate treated samples. Computing deamination pattern with PMDtools can be achieved with the following command line (please note that the scripts *pmdtools.0.60.py* and *plotPMD.v2.R* can be downloaded from the github repository here <https://github.com/pontussk/PMDtools>):

```
 samtools view Y.pestis_sample10.bam | pmdtools --platypus > PMDtemp.txt  
 R CMD BATCH plotPMD
```

## 16. Genomic hit confirmation



When performing ancient status analysis on **de-novo** assembled contigs, it can be computationally challenging and time consuming to run mapDamage or PMDtools on all of them as there can be hundreds of thousands contigs. In addition, the outputs from mapDamage and PMDtools lacking a clear numeric quantity or a statistical test that could provide an “ancient vs. non-ancient” decision for each **de-novo** assembled contig. To address these limitations, pyDamage tool was recently developed. PyDamage evaluates the amount of aDNA damage and tests the hypothesis whether a model assuming presence of aDNA damage better explains the data than a null model.

## 16. Genomic hit confirmation



### PyDamage: automated ancient damage identification and estimation for contigs in ancient DNA *de novo* assembly

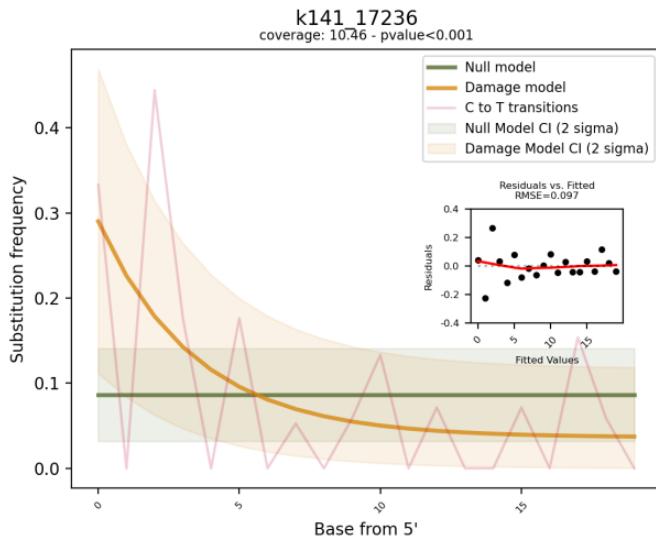
Maxime Berry<sup>1</sup>, Alexander Huber<sup>2</sup>, Adam B. Rohrach<sup>3,4</sup> and Christian Warinner<sup>1,5,6</sup>

<sup>1</sup> Max Planck Institute for the Science of Human History, Department of Archaeogenetics, Jena, Germany  
<sup>2</sup> Faculty of Biological Sciences, Friedrich-Schiller University Jena, Jena, Germany  
<sup>3</sup> Population Genetics Group, Max Planck Institute for the Science of Human History, Department of Archaeogenetics, Jena, Germany  
<sup>4</sup> ARC Centre of Excellence for Mathematical and Statistical Frontiers, The University of Adelaide, Adelaide, Australia  
<sup>5</sup> Department of Anthropology, Harvard University, Cambridge, MA, United States of America

#### ABSTRACT

DNA *de novo* assembly can be used to reconstruct longer stretches of DNA (contigs), including genes and even genomes, from short DNA sequencing reads. Applying this technique to metagenomic datasets from archaeological remains, such as paleofaeces and dental calculus, we can investigate patterns of ancient DNA damage that may be absent or underrepresented in modern microbial gene catalogues. However, compared to modern samples, ancient samples are often burdened with environmental contamination, resulting in metagenomic datasets that represent mixtures of ancient and modern DNA. The ability to rapidly and reliably establish the authenticity and integrity of ancient samples is crucial for ancient DNA studies, and the ability to distinguish between ancient and modern DNA is particularly important for ancient microbiome studies. Characteristic patterns of ancient DNA damage, namely DNA fragmentation and cytosine deamination (observed as C-to-T transitions) are typically used to authenticate ancient samples and sequences. These existing tools for investigating and filtering ancient DNA damage are limited in the range of tasks they lead to, high data loss and lower quality when used in combination with *de novo* assemblies, or require manual inspection, which is impractical for ancient assemblies that typically contain tens to hundreds of thousands of contigs. To address these challenges, we designed PyDamage, a robust, automated approach for aDNA damage detection and analysis of *de novo* assembled aDNA. PyDamage uses a likelihood ratio based approach to discriminate between truly ancient contigs and contigs originating from modern contamination. We test PyDamage on both on simulated aDNA data and archaeological paleofaeces, and we demonstrate its ability to reliably and automatically identify contigs bearing DNA damage characteristic of aDNA. Coupled with aDNA *de novo* assembly, PyDamage opens up new doors to explore functional diversity in ancient metagenomic datasets.

Submitted: 29 March 2021  
 Accepted: 1 July 2021  
 Published: 27 July 2021  
 Corresponding authors:  
 Maxime Berry, berry@shh.mpg.de  
 Christian Warinner, warinner@fas.harvard.edu  
 Academic editor:  
 Michaela Schubert  
 Additional Information and  
 Declarations can be found on  
 page 16  
 DOI: 10.7717/peerj.11845  
 © Copyright  
 2021 Berry et al.



pyDamage can be run on a sorted BAM-alignments of the microbial reads to the **de-novo** assembled contigs using the following command line:

```
## Do not run!
pydamage analyze -w 30 -p 14 filtered.sorted.bam
```

# **17. Microbiome contamination correction**

Modern contamination can severely bias ancient metagenomic analysis. Also, ancient contamination, i.e. entered *post-mortem*, can potentially lead to false biological interpretations. Therefore, a lot of efforts in the ancient metagenomics field are directed on establishing methodology for identification of contaminants. Among them, the use of negative (blank) control samples is perhaps the most reliable and straightforward method. Additionally, one often performs microbial source tracking for predicting environment (including contamination environment) of origin for ancient metagenomic samples.

## **17.1. Decontamination**

Modern contamination is one of the major problems in ancient metagenomics analysis. Large fractions of modern bacterial, animal or human DNA in metagenomic samples can lead to false biological and historical conclusions. A lot of scientific literature is dedicated to this topic, and comprehensive tables and sources of potential contamination (e.g. animal and bacterial DNA present in PCR reagents) are available.

## 17. Microbiome contamination correction

The image shows two scientific articles side-by-side. On the left is a screenshot of the BMC Biology website. At the top, it says 'BMC Part of Springer Nature' and 'BMC Biology'. Below that is a navigation bar with links for 'Home', 'About', 'Articles', and 'Submission Guidelines'. The main content area features a purple micrograph of bacteria. Below the image, it says 'Research article | Open Access | Published: 12 November 2014'. The title of the article is 'Reagent and laboratory contamination can critically impact sequence-based microbiome analyses'. Below the title, it lists the authors: Susannah J Salter, Michael J Cox, Elena M Turek, Szymon T Calus, William O Cookson, Miriam F Moffatt, Paul Turner, Julian Parkhill, Nicholas J Loman & Alan W Walker. It also mentions 'BMC Biology 12, Article number: 87 (2014) | Cite this article' and '84k Accesses | 1303 Citations | 341 Altmetric | Metrics'. A section titled 'Abstract' follows, followed by 'Background' and 'Results'. The 'Results' section discusses the ubiquity of contaminating DNA in common DNA extraction kits and its impact on sequencing results. On the right is a screenshot of the Journal of Archaeological Science article. At the top, it says 'Journal of Archaeological Science' and 'Volume 34, Issue 9, September 2007, Pages 1361-1366'. Below that is a small image of a tree. The title of the article is 'Animal DNA in PCR reagents plagues ancient DNA research'. It lists the authors: Jennifer A. Leonard, Orin Shanks, Michael Hofreiter, Eva Kreuz, Larry Hodges, Walt Ream, Robert K. Wayne, and Robert C. Fleischer. It includes a 'Show more' link, 'Add to Mendeley' button, 'Share' button, 'Cite' button, and the URL 'https://doi.org/10.1016/j.jas.2006.10.023'. A 'Get rights and content' link is also present. A section titled 'Abstract' follows, followed by 'Keywords' which include 'Sus scrofa', 'Bos taurus', 'Gallus gallus', and 'Deoxynucleoside triphosphates'. Navigation arrows for 'Previous article in issue' and 'Next article in issue' are also visible.

## 17. Microbiome contamination correction

**Table 1 List of contaminant genera detected in sequenced negative 'blank' controls**

From: [Reagent and laboratory contamination can critically impact sequence-based microbiome analyses](#)

| Phylum              | List of constituent contaminant genera   |
|---------------------|--|
| Proteobacteria      | Alpha-proteobacteria:<br><i>Afipia, Aquabacterium<sup>a</sup>, Asticcacaulis, Aurantimonas, Beijerinckia, Bosea, Bradyrhizobium<sup>d</sup>, Brevundimonas<sup>c</sup>, Caulobacter, Craurococcus, Devosia, Hoeftea<sup>b</sup>, Mesorhizobium, Methylobacterium<sup>a</sup>, Novosphingiobium, Ochrobactrum, Paracoccus, Pedomicrobium, Phyllobacterium<sup>a</sup>, Rhizobium<sup>c-d</sup>, Roseomonas, Sphingomonas<sup>c-d,e</sup>, Sphingopyxis</i>  |
|                     | Beta-proteobacteria:<br><i>Acidovorax<sup>b</sup>, Azoarcus<sup>b</sup>, Azospira, Burkholderia<sup>d</sup>, Comamonas<sup>c</sup>, Cupriavidius<sup>c</sup>, Curvibacter, Delftia<sup>b</sup>, Duganella<sup>b</sup>, Herbaspirillum<sup>a,c</sup>, Janthinobacterium<sup>b</sup>, Kingella, Leptothrix<sup>b</sup>, Limnobacter<sup>b</sup>, Massilia<sup>b</sup>, Methylophilus, Methyloversatilis<sup>b</sup>, Oxalobacter, Pelomonas, Polaromonas<sup>b</sup>, Ralstonia<sup>c,d,e</sup>, Schlegelella, Sulfuritalea, Undibacterium<sup>b</sup>, Variovorax</i> |
|                     | Gamma-proteobacteria:<br><i>Acinetobacter<sup>a,d,c</sup>, Enhydrobacter, Enterobacter, Escherichia<sup>a,c,d,e</sup>, Nevskaia<sup>b</sup>, Pseudomonas<sup>b,d,e</sup>, Pseudoxanthomonas, Psychrobacter, Stenotrophomonas<sup>a,b,c,d,e</sup>, Xanthomonas<sup>b</sup></i>  |
| Actinobacteria      | <i>Aeromicrobium, Arthrobacter, Beutenbergia, Breibacterium, Corynebacterium, Curtobacterium, Dietzia, Geodermatophilus, Janibacter, Kocuria, Microbacterium, Micrococcus, Microlunatus, Patulibacter, Propionibacterium<sup>a</sup>, Rhodococcus, Tsukamurella</i>  |
| Firmicutes          | <i>Abiotrophia, Bacillus<sup>b</sup>, Brevibacillus, Brochotrichia, Facklamia, Paenibacillus, Streptococcus</i>  |
| Bacteroidetes       | <i>Chryseobacterium, Dyadobacter, Flavobacterium<sup>b</sup>, Hydrotalea, Niastella, Olivibacter, Pedobacter, Wautersiella</i>   |
| Deinococcus-Thermus | <i>Deinococcus</i>   |
| Acidobacteria       | Predominantly unclassified Acidobacteria Gp2 organisms   |

The listed genera were all detected in sequenced negative controls that were processed alongside human-derived samples in our laboratories (WTSI, ICL and UB) over a period of four years. A variety of DNA extraction and PCR kits were used over this period, although DNA was primarily extracted using the FastDNA SPIN Kit for Soil. Genus names followed by a superscript letter indicate those that have also been independently reported as contaminants previously. <sup>a</sup>also reported by Tanner *et al.* [12]; <sup>b</sup>also reported by Grahn *et al.* [14]; <sup>c</sup>also reported by Barton *et al.* [17]; <sup>d</sup>also reported by Laurence *et al.* [18]; <sup>e</sup>also detected as contaminants of multiple displacement amplification kits (information provided by Paul Scott, Wellcome Trust Sanger Institute). ICL, Imperial College London; UB, University of Birmingham; WTSI, Wellcome Trust Sanger Institute.

It is typically assumed that an organism found on a blank has a lower confidence to be endogenous to the studied metagenomic sample, and sometimes it is even excluded from the downstream analysis as an unreliable hit. Despite there are attempts to automate filtering out modern contaminants (we will discuss them below), decontamination process still remains to be a tedious manual work where each candidate should be carefully investigated from different contexts in order to prove its ancient and endogenous origin.

If negative control samples (blanks) are available, contaminating organisms can be detected by comparing their abundances in the negative controls with true samples. In this case, contaminant organisms stand out by their high prevalence in both types of samples if one simply plots mean across samples abundance of each detected organism in true samples and negative controls against each other as in the figure below.

First move back into the KrakenUniq folder and load R.

```
cd /<path>/<to>/authentication-decontamination/krakenuniq
R
```

Then in here we can compare between true samples and negative controls

## 17. Microbiome contamination correction

```
samples<-read.delim("krakenuniq_abundance_matrix/krakenuniq_abundance_matrix.txt",header=TRUE)
row.names = 1, check.names = FALSE, sep = "\t")
## The blanks abundance matrix has already been made for you
controls<-read.delim("blank_krakenuniq_abundance_matrix.txt",header=TRUE,
row.names = 1, check.names = FALSE, sep = "\t")

df <- merge(samples, controls, all = TRUE, by = "row.names")
rownames(df)<-df$Row.names; df$Row.names <- NULL; df[is.na(df)] <- 0

true_sample <- subset(df,select=colnames(df)[!grepl("control",colnames(df))])
negative_control <- subset(df,select=colnames(df)[grepl("control",colnames(df))])

plot(log10(rowMeans(true_sample)+1) ~ log10(rowMeans(negative_control)+1),
xlab = "Log10 ( Negative controls )", ylab = "Log10 ( True samples )",
main = "Organism abundance in true samples vs. negative controls",
pch = 19, col = "blue")

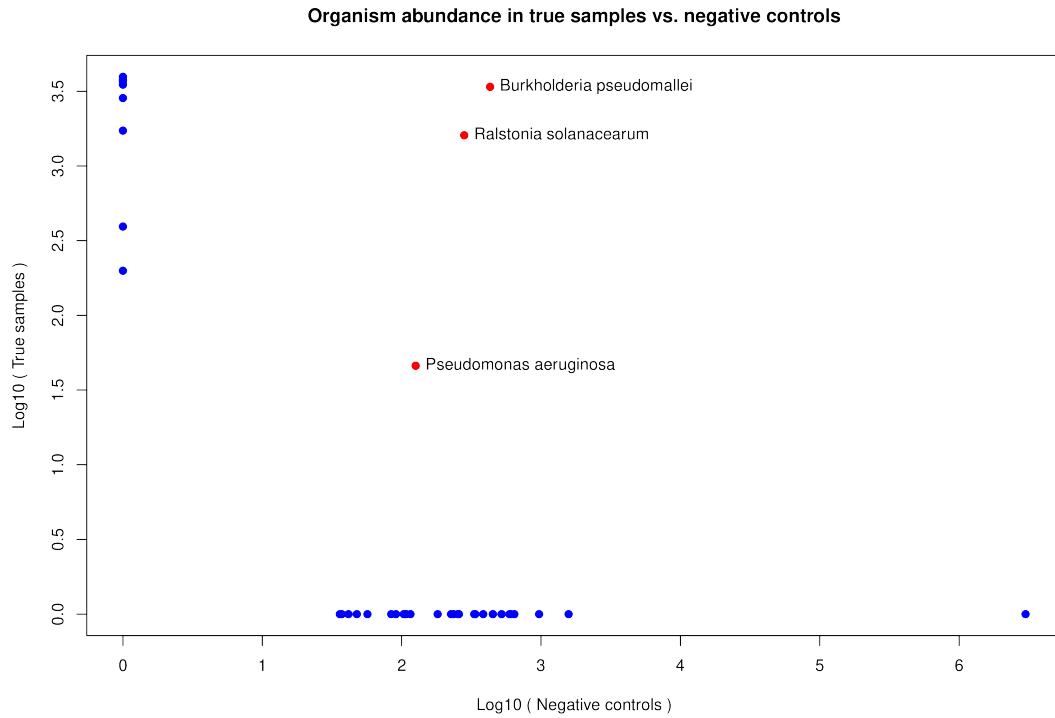
points(log10(rowMeans(true_sample)+1)[(log10(rowMeans(true_sample)+1) > 1) &
(log10(rowMeans(negative_control)+1)>1)] ~
log10(rowMeans(negative_control)+1)[(log10(rowMeans(true_sample)+1) > 1) &
(log10(rowMeans(negative_control)+1)>1)], pch = 19, col = "red")

text(log10(rowMeans(true_sample)+1)[(log10(rowMeans(true_sample)+1) > 1) &
(log10(rowMeans(negative_control)+1) > 1)] ~
log10(rowMeans(negative_control)+1)[(log10(rowMeans(true_sample)+1) > 1) &
(log10(rowMeans(negative_control)+1) >1)],
labels = rownames(true_sample)[(log10(rowMeans(true_sample)+1) > 1) &
(log10(rowMeans(negative_control)+1) > 1)], pos = 4)
```

Once finished examining the plot you can quit R

```
## Press 'n' when asked if you want to save your workspace image.
quit()
```

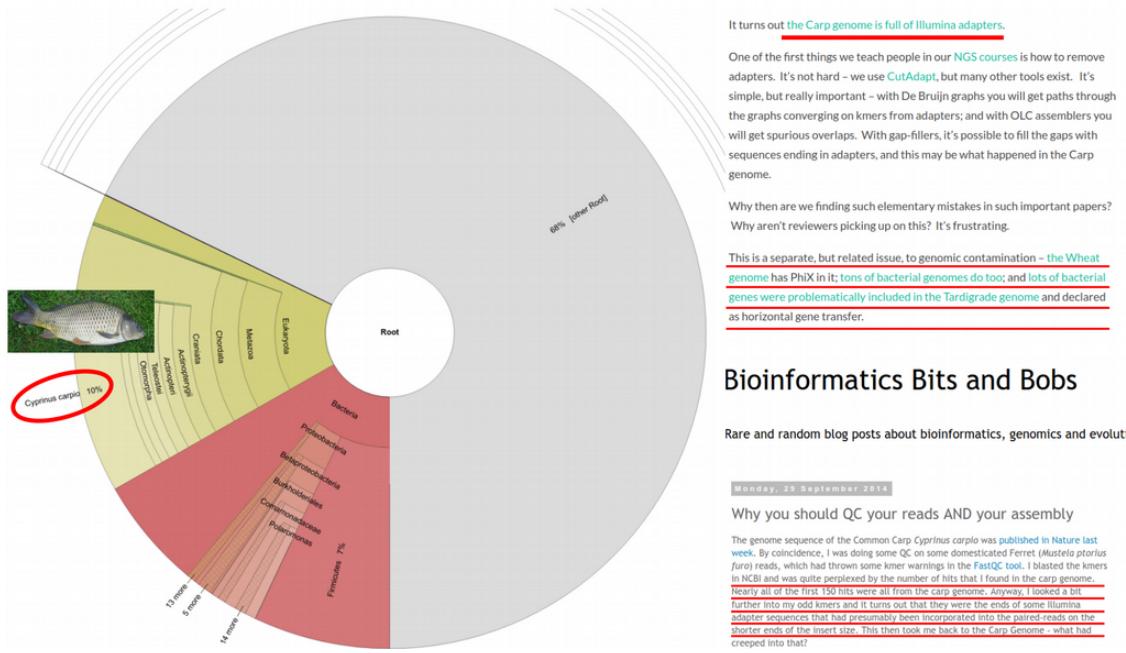
## 17. Microbiome contamination correction



In the figure above, one point indicates an organism detected in a group of metagenomic samples. The points highlighted by red have high abundance in negative control samples, and therefore they are likely contaminants.

In addition to PCR reagents and lab contaminants, reference databases can also be contaminated by various, often microbial, organisms. A typical example that when screening environmental or sedimentary ancient DNA samples, a fish *Cyprinus carpio* can pop up if adapter trimming procedure was not successful for some reason.

## 17. Microbiome contamination correction



It was noticed that the *Cyprinus carpio* reference genome available at NCBI contains large fraction of Illumina sequencing adapters. Therefore, appearance of this organism in your analysis may falsely lead your conclusion toward potential lake or river present in the excavation site.

Let us now discuss a few available computational approaches to decontaminate metagenomic samples. One of them is decontam R package that offers a simple statistical test for whether a detected organism is likely contaminant. This approach is useful when DNA quantitation data recording the concentration of DNA in each sample (e.g. PicoGreen fluorescent intensity measures) is available. The idea of the *decontam* is that contaminant DNA is expected to be present in approximately equal and low concentrations across samples, while sample DNA concentrations can vary widely. As a result, the expected frequency of contaminant DNA varies inversely with total sample DNA concentration (red line in the figure below), while the expected frequency of non-contaminant DNA does not (blue line).

It turns out the Carp genome is full of Illumina adapters.

One of the first things we teach people in our NGS courses is how to remove adapters. It's not hard - we use [CutAdapt](#), but many other tools exist. It's simple, but really important - with De Bruijn graphs you will get paths through the graphs converging on kmers from adapters; and with OLC assemblers you will get spurious overlaps. With gap-filters, it's possible to fill the gaps with sequences ending in adapters, and this may be what happened in the Carp genome.

Why then are we finding such elementary mistakes in such important papers? Why aren't reviewers picking up on this? It's frustrating.

This is a separate, but related issue, to genomic contamination - the Wheat genome has PhiX in it; tons of bacterial genomes do too; and lots of bacterial genes were problematically included in the Tardigrade genome and declared as horizontal gene transfer.

### Bioinformatics Bits and Bobs

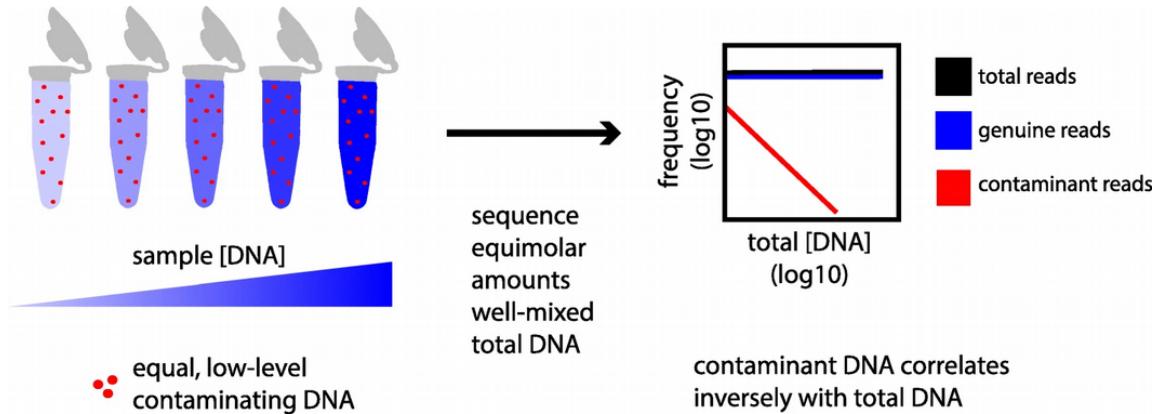
Rare and random blog posts about bioinformatics, genomics and evolution.

Monday, 29 September 2014

Why you should QC your reads AND your assembly

The genome sequence of the Common Carp *Cyprinus carpio* was published in Nature last week. By coincidence I was doing some QC on some domesticated Ferret (*Mustela putorius furo*) which had been sequenced by the Wellcome Trust Sanger Institute. I was looking at odd kmers in NCBI and was quite perplexed by the number of hits that I found in the carp genome. Nearly all of the first 150 hits were all from the carp genome. Anyway, I looked a bit further into my odd kmers and it turns out that they were the ends of some Illumina adapter sequences that had presumably been incorporated into the paired reads on the shorter ends of the insert size. This then took me back to the Carp Genome - what had crepted into that?

## 17. Microbiome contamination correction



Another popular tool for detecting contaminating microorganisms is Recentrifuge. It works as a classifier that is trained to recognize contaminant microbial organisms. In case of Recentrifuge, one has to use blanks or other negative controls and provide microbial names and abundances on the blanks in order to train Recentrifuge to recognize endogenous vs. contaminant sources.

If one wants to assess the degree of contamination for each sample, there is a handy tool cuperdec, which is an R package that allows a quick comparison of microbial profiles in a query metagenomic sample against a database. The idea of *cuperdec* is to rank organisms in each sample by their abundance and then using an “expanding window” approach to compute their enrichment in a reference database that contains a comprehensive list of microbial organisms which are specific to a tissue / environment in question. The tool produces so-called *Cumulative Percent Decay* curves that aim to represent the level of endogenous content of microbiome samples, such as ancient dental calculus, to help to identify samples with low levels of preservation that should be discarded for downstream analysis.

First change into the cuperdec directory, and load R

```
cd /<path>/<to>/authentication-decontamination/cuperdec
```

Then we can load the files, generate the decay curves, set a preservation threshold cut-off, and plot the result.

```
library("cuperdec"); library("magrittr"); library("dplyr")

# Load database (in this case an internal 'test' oral database from the package)
data(cuperdec_database_ex)
database <- load_database(cuperdec_database_ex, target = "oral") %>% print()
```

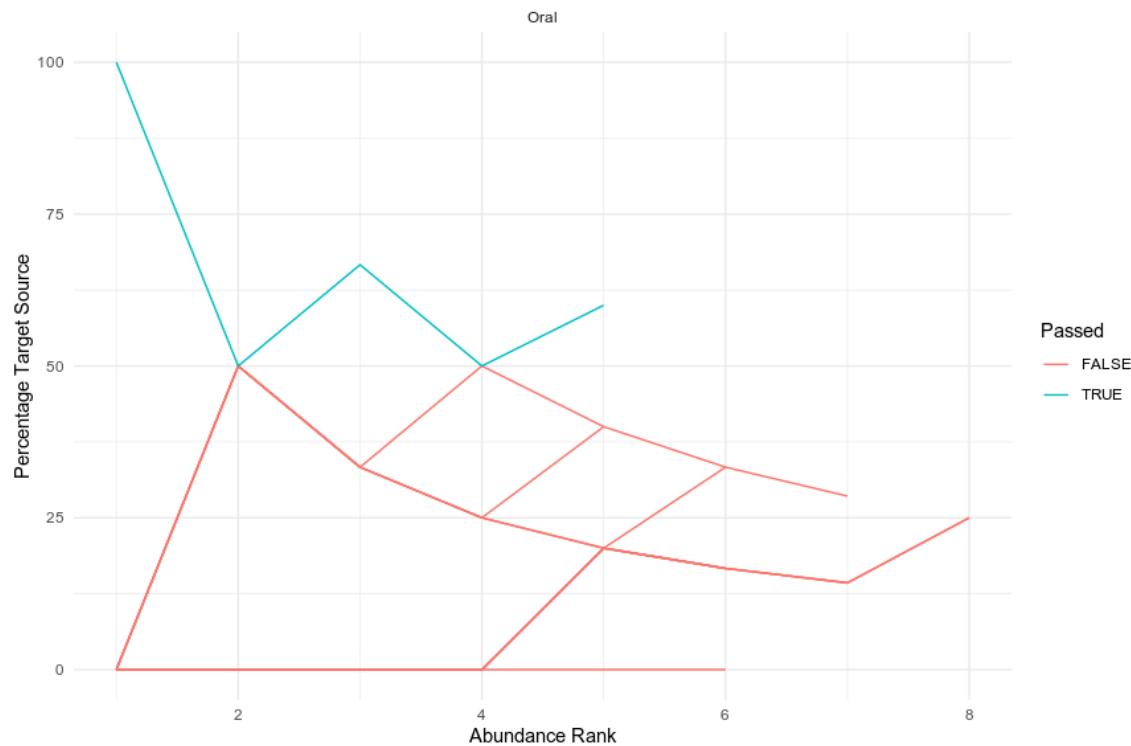
## 17. Microbiome contamination correction

```
# Load abundance matrix and metadata
raw_table <- read.delim("../krakenuniq/krakenuniq_abundance_matrix/krakenuniq_abundance_matrix")
taxatable <- load_taxa_table(raw_table) %>% print()
metadata <- as_tibble(data.frame(Sample = unique(taxatable$Sample),
Sample_Source = "Oral"))

# Compute cumulative percent decay curves, filter and plot results
curves <- calculate_curve(taxatable, database = database) %>% print()
filter_result <- simple_filter(curves, percent_threshold = 50) %>% print()
plot_cuperdec(curves, metadata = metadata, filter_result)
```

Once finished examining the plot you can quit R

```
## Press 'n' when asked if you want to save your workspace image.
quit()
```



In the figure above, one curve represents one sample, and the red curves have a very high

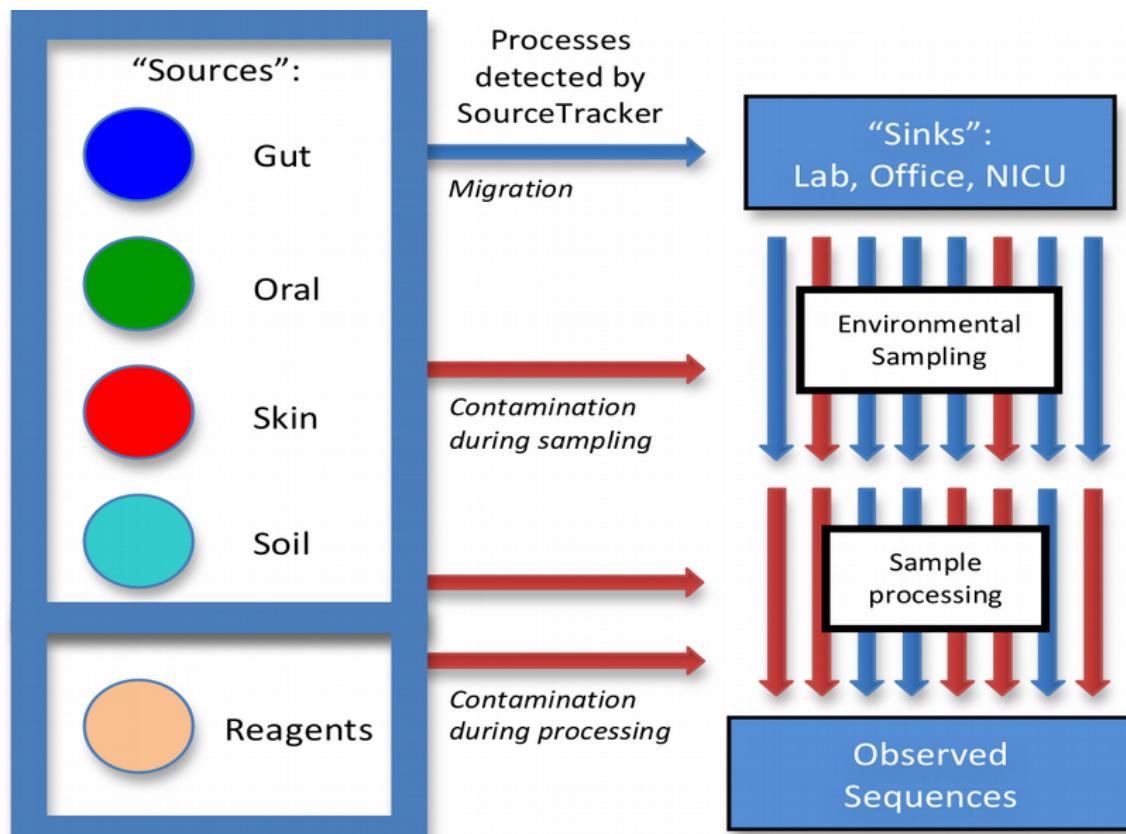
## 17. Microbiome contamination correction

amount of contamination and very low amount of endogenous DNA. These samples might be considered to be dropped from the downstream analysis.

### 17.2. Microbial source tracking

For the case of ancient microbiome profiling, in addition to traditional inspection of the list of detected organisms and comparing it with the ones detected on blanks, we can use tools that make a prediction on what environment the detected organisms most likely come from.

The most popular and widely used tool is called **SourceTracker**. SourceTracker is a Bayesian version of the Gaussian Mixture Model (GMM) clustering algorithm that is trained on a user-supplied reference data called **Sources**, i.e. different classes such as Soil or Human Oral or Human Gut microbial communities etc., and then it can estimate proportion / contribution of each of these sources the users actual samples called **Sinks**.



## 17. Microbiome contamination correction

Originally, SourceTracker was developed for 16S data, i.e. using only 16S ribosomal RNA genes, but it can be easily trained using also shotgun metagenomics data, which was demonstrated in its metagenomic extension called mSourceTracker and its faster and more scalable version FEAST. The input data for SourceTracker are metadata, i.e. each sample has to have “source” or “sink” annotation as well as environmental label (e.g. Oral, Gut, Soil etc.), and microbial abundances (OTU abundances) quantified in some way, for example through QIIME pipeline, MetaPhlAn or Kraken. The SourceTracker R script can be downloaded from <https://github.com/danknights/sourcetracker>.

### Self guided: data preparation

In addition to the input files (already prepared for you), you will also need to clone the Sourcetracker 1 R code repository (unfortunately it's not available as a package)

```
cd /<path>/<to>/authentication-decontamination/sourcetracker/  
git clone https://github.com/danknights/sourcetracker.git
```

### Patching sourcetracker for newer R versions

Unfortunately Sourcetracker's original R code was done a very long time ago, and has resulted in some incompatibilities with more recent versions of R.

If using a more recent version of R (such as in this chapter's conda environment), you will need to perform the following ‘patching’ of the code, to prevent an error. Run the following command to comment out the offending lines (538, 539), where R's behaviour in later versions of a fundamental validation function has changed.

```
sed -i '538,539s/^#/' sourcetracker/src/SourceTracker.r
```

Sourcetracker expects two input data frames: metadata with at least sample name, environment and source / sink labels, and abundance matrix. Note that source and sink metadata and abundances have to be merged together prior to using SourceTracker. Here we are going to use data from the Human Microbiome Project (HMP) as sources, and we are going to merge the HMP data with the sink samples into single OTU table and meta-data table.

```
cd /<path>/<to>/authentication-decontamination/sourcetracker/  
  
## Load R  
R
```

## 17. Microbiome contamination correction

Then in R we can load the HMP source files, and our ‘sink samples’ from the KrakenUniq matrix we made earlier, and clean them up to make them compatible for Sourcetracker.

```
## Load files
otus_hmp <- read.delim("otus_hmp.txt", header = TRUE, row.names = 1, sep = "\t")
meta_hmp <- read.delim("meta_hmp.txt", header = TRUE, row.names = 1, sep = "\t")
otus_sink<-read.delim("../krakenuniq/krakenuniq_abundance_matrix/krakenuniq_abundance_matrix.txt", header = TRUE, row.names = 1, sep = "\t")

## Join source and sink tables into one
otus <- merge(otus_hmp, otus_sink, all = TRUE, by = "row.names")

## Put OTU IDs as row names and replace NAs with 0
rownames(otus) <- otus$Row.names; otus$Row.names <- NULL; otus[is.na(otus)] <- 0

## Set column and row names for meta
meta_sink <- data.frame(ID = colnames(otus_sink), Env = "Unknown", SourceSink = "sink")
rownames(meta_sink) <- meta_sink$ID; meta_sink$ID<-NULL
metadata <- rbind(metadata, meta_sink)

## Further value cleanups and sample filtering to remove not useful HMP samples
otus <- as.data.frame(t(as.matrix(otus)))
otus[otus > 0] <- 1; otus <- otus[rowSums(otus)!=0,]
metadata<-metadata[as.character(metadata$Env)!="Vaginal",]; envs <- metadata$Env
common.sample.ids <- intersect(rownames(metadata), rownames(otus))
otus <- otus[common.sample.ids,]; metadata <- metadata[common.sample.ids,]
```

Next, training SourceTracker on source samples and running predictions on sink samples can be done using following commands:

```
# Train SourceTracker on sources (HMP) and run predictions on sinks
source('sourcetracker/src/SourceTracker.r')
train.ix <- which(metadata$SourceSink=='source')
test.ix <- which(metadata$SourceSink=='sink')
st <- sourcetracker(otus[train.ix,], envs[train.ix])
results <- predict(st, otus[test.ix,], alpha1 = 0.001, alpha2 = 0.001)
```

Finally, we can plot SourceTracker environment inference in the form of barcharts as follows:

## 17. Microbiome contamination correction

```
# Sort SourceTracker proportions for plotting
props <- results$proportions
props <- props[order(-props[, "Oral"]), ]
results$proportions <- props

# Prepare SourceTracker output for plotting
name <- rep(rownames(results$proportions), each = 4)
value <- as.numeric(t(results$proportions))
labels <- c("Gut", "Oral", "Skin", "Unknown"); condition<-rep(labels, length(test.ix))
data <- data.frame(name, condition, value)

# Plot SourceTracker inference as a barplot
library("ggplot2")
ggplot(data, aes(fill=condition, y=value, x = reorder(name, seq(1:length(name))))) +
  geom_bar(position = "fill", stat = "identity") +
  theme(axis.text.x = element_text(angle = 90, size=5, hjust=1, vjust=0.5)) +
  xlab("Sample") + ylab("Fraction")
```

Once finished examining the plot you can quit R

```
## Press 'n' when asked if you want to save your workspace image.
quit()
```

In the figure above (Figure 17.1) the SourceTracker was trained on Human Microbiome Project (HMP) data, and was capable of predicting the fractions of oral, gut, skin or other microbial composition on the query sink samples. In a similar way, environmental soil or marine microbes can be used as Sources. In this way, environmental percentage of contamination can be detected per sample.

A drawback of SourceTracker, mSourceTracker and FEAST is that they require a microbial abundance table after a taxonomic classification with e.g. QIIME or Kraken has been performed. Such taxonomic classification can be biased since it is computed against a reference database with known taxonomic annotation. In contrast, a novel microbial source tracking tool decOM aims at moving away from database-dependent methods and using unsupervised approaches exploiting read-level sequence composition.

## 17. Microbiome contamination correction

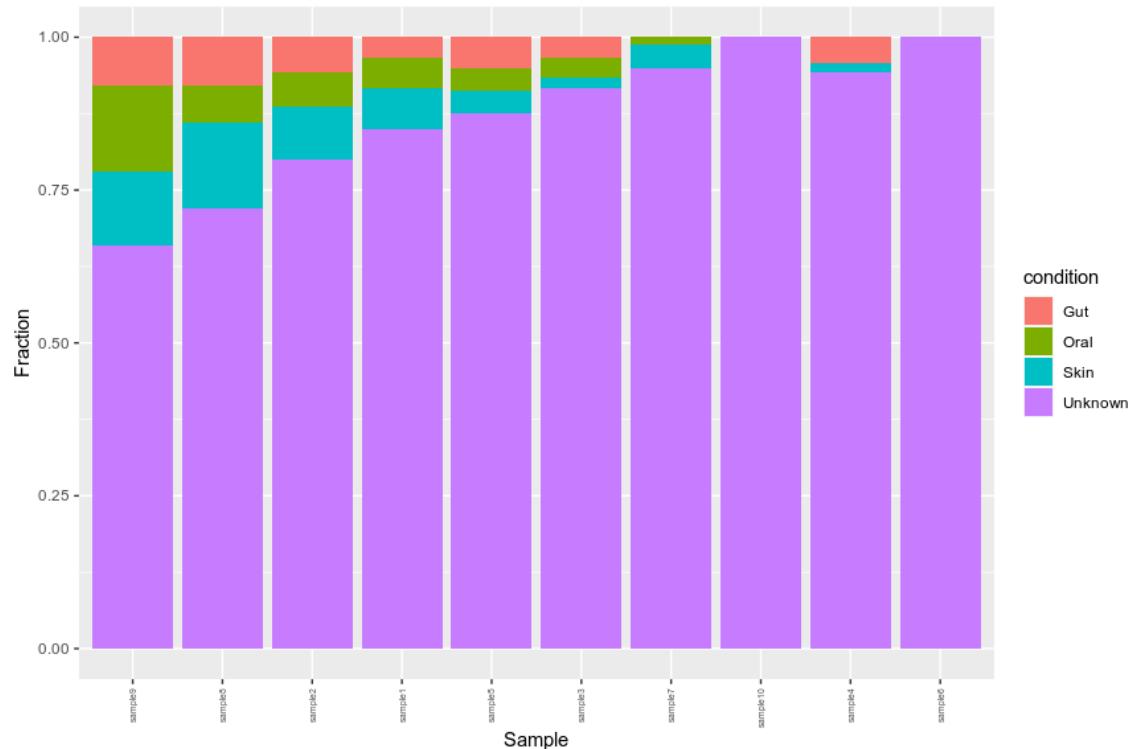
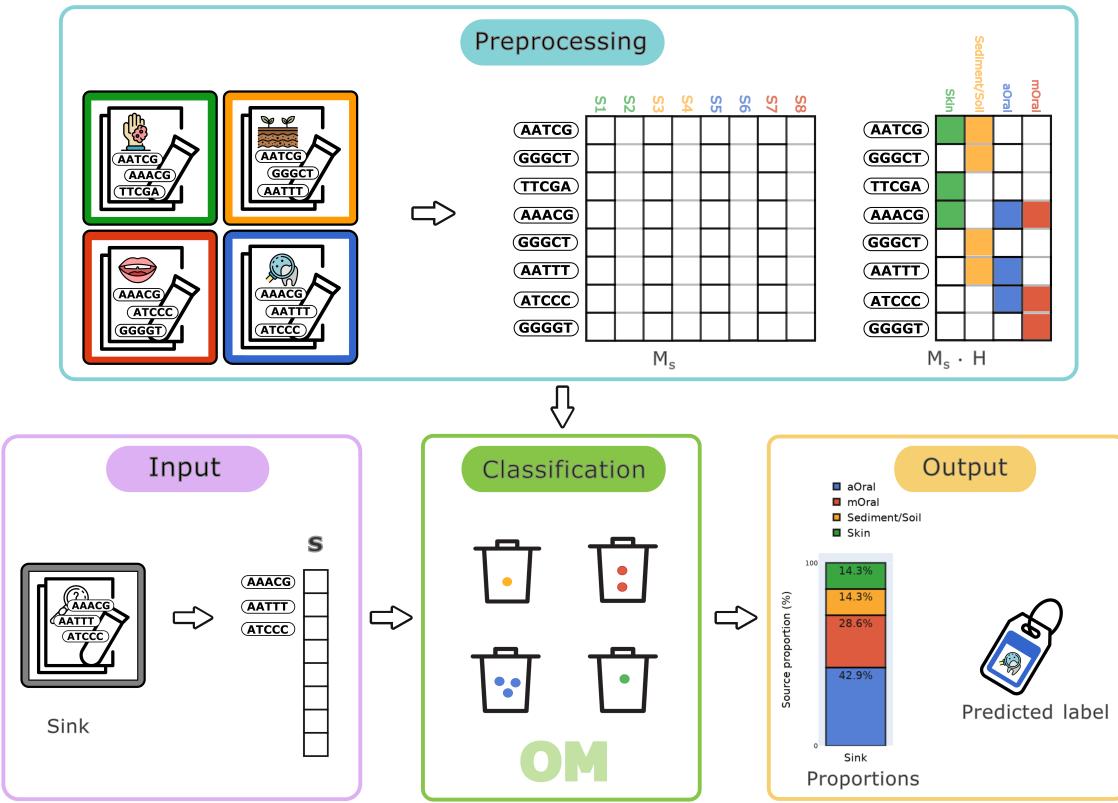


Figure 17.1.: Stacked barchart of each sample (sinks) with the estimated fraction of each source ‘sample group’ that each sink is made up of. Note that the X-axis sorting of samples may change between each person, thus your plot may not match exactly the same as the one here.

## 17. Microbiome contamination correction



*decOM* uses kmtricks to compute a matrix of k-mer counts on raw reads (FASTQ files) from source samples, and then uses the source k-mer abundance matrix for looking up k-mer composition of sink samples. This allows *decOM* to calculate microbial contributions / fractions from the sources. For example, for estimating contributions from ancient Oral (aOral), modern Oral (mOral), Skin and Sediment / Soil environments one can use an already computed source matrix from here <https://github.com/CamilaDuitama/decOM/> and provide it as a *-p\_sources* parameter.

### i Self guided: data preparation

If doing a self-guided tutorial, you will need to download a rather large pre-built database.

## 17. Microbiome contamination correction

```
cd /<path>/<to>/authentication-decontamination/decom
git clone https://github.com/CamilaDuitama/decOM.git
cd decOM
conda env create -n decOM --file environment.yml
conda deactivate
conda activate decOM

export PATH=/absolute/path/to/decOM:${PATH}

## In the directory
## Note that decOM does not like dots in names so make symlinks
for i in ./rawdata/*trimmed.fastq.gz; do
    ln -s "$i" "${i/.trimmed/_trimmed}"
done

# Prepare input fof-files that have a key - value format
for i in {1..10}; do
    echo "sample${i}_trimmed : sample${i}_trimmed.fastq.gz" > sample${i}_trimmed.fof
    echo sample${i}_trimmed >> FASTQ_NAMES_LIST.txt;
done

# Download pre-built kmer-matrix of sources (aOral, mOral, Sediment/Soil, Skin)
wget https://zenodo.org/record/6513520/files/decOM_sources.tar.gz
tar -xf decOM_sources.tar.gz
```

The following example command is how you would execute deCOM. However as this requires a very large database file and thus large memory requirements, therefore we have already made available for you the output files from this step.

```
# DO NOT RUN!: Run decOM predictions
decOM -p_sources decOM_sources/ -p_sinks FASTQ_NAMES_LIST.txt -p_keys decOM/FASTQ -mem 900G
```

You can find the relevant output files in the following directory (the output of deCOM with our simulated data from the command above was 29GB worth of files!).

```
cd /<path>/<to>/authentication-decontamination/decom
```

In the command line used to generate, the `-p_sinks` parameter provides a list of sink samples, for example `SRR13355807`.

## 17. Microbiome contamination correction

The sink fastq-files are placed together with *keys* fof-files containing the mapping between fastq file names and locations of the fastq-files, for example `SRR13355807 : SRR13355807.fastq.gz`. The contributions from the sources to the sink samples, which are recorded in the `dec0M_output.csv` output file (which is in the authentication-decontamination/decom directory already for you), can then be processed and plotted as follows:

Load R

R

Then we can make a plot

```
df<-read.csv("dec0M_output.csv", check.names=FALSE)

result <- subset(df, select = c("Sink", "Sediment/Soil", "Skin", "a0ral",
"m0ral", "Unknown"))
rownames(result) <- result$Sink; result$Sink <- NULL
result <- result / rowSums(result)
result<-result[order(-result$a0ral),]

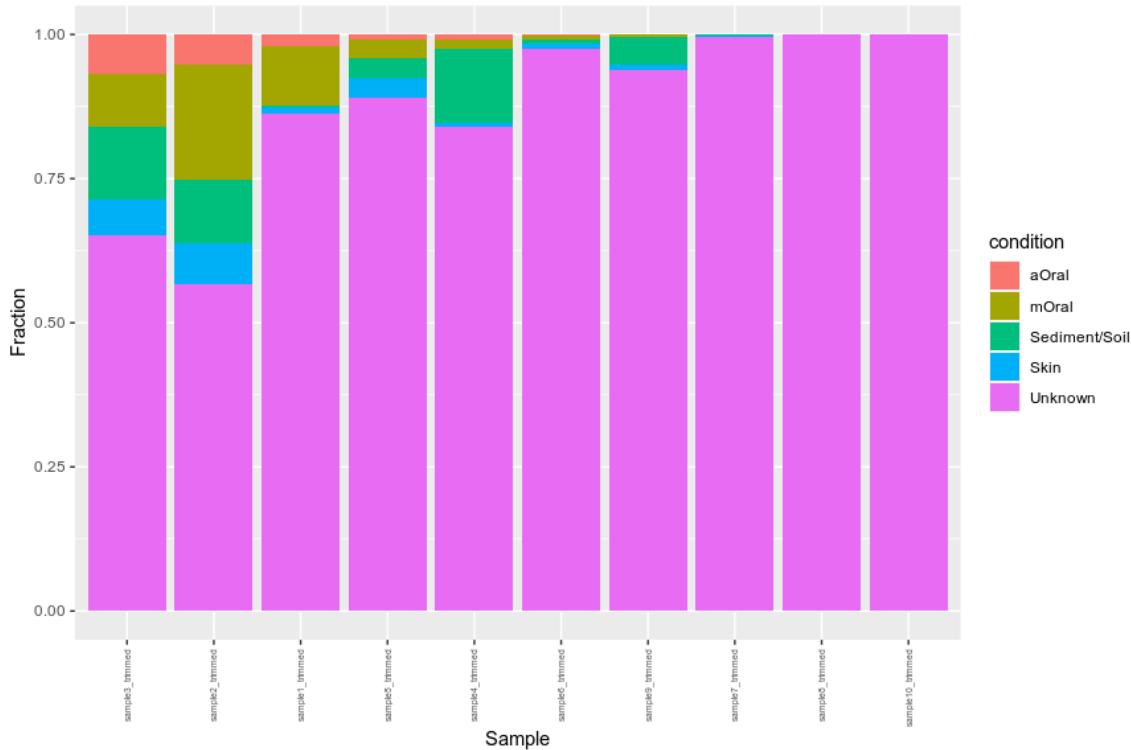
name <- rep(rownames(result), each = 5); value <- as.numeric(t(result))
condition <- rep(c("Sediment/Soil","Skin","a0ral","m0ral","Unknown"),
dim(result)[1])
data <- data.frame(name, condition, value)

library("ggplot2"); library("viridis")
ggplot(data, aes(fill=condition, y=value, x=reorder(name,seq(1:length(name))))) +
  geom_bar(position = "fill", stat = "identity") +
  theme(axis.text.x = element_text(angle=90, size = 5, hjust = 1, vjust = 0.5)) +
  xlab("Sample") + ylab("Fraction")
```

Once finished examining the plot you can quit R

```
## Press 'n' when asked if you want to save your workspace image.
quit()
```

## 17. Microbiome contamination correction



*decOM* has certain advantages compared to *SourceTracker* as its is a taxonomic classification / database free approach. However, it also appears to be very sensitive to the particular training / source data set. In the example above it can be seen that the microbial source tracking of sink samples is very much dominated by the Oral community, which was the training / source data set.

### 17.3. Summary

In this chapter we have learned that:

- Evenness of coverage is an important metric for validation of findings
- Deamination profile, DNA fragmentation, mapping quality, edit distance and PMD scores are other authentication / validation metrics to consider
- Negative controls are important for disentangling ancient / endogenous from modern / exogenous contamination
- Microbial source tracking is another layer of evidence that can facilitate interpretation of ancient metagenomic findings

## 17.4. Questions to think about

1. What is a false-positive microbial finding and how can we recognize it?
2. What is the difference between depth, breadth and evenness of coverage?
3. What is contamination and how can it bias ancient metagenomic analysis?
4. How can we separate ancient from modern DNA sequence?
5. What is a negative (blank) control sample and why is it useful to have?
6. What is microbial source tracking and how can it help with decontamination?

## 17.5. Readings

1. Clio Der Sarkissian, Irina M. Velsko, Anna K. Fotakis, Åshild J. Vågene, Alexander Hübner, and James A. Fellows Yates, Ancient Metagenomic Studies: Considerations for the Wider Scientific Community, *mSystems* 2021 Volume 6 Issue 6 e01315-21.
2. Warinner C, Herbig A, Mann A, Fellows Yates JA, Weiß CL, Burbano HA, Orlando L, Krause J. A Robust Framework for Microbial Archaeology. *Annu Rev Genomics Hum Genet.* 2017 Aug 31;18:321-356. doi: 10.1146/annurev-genom-091416-035526. Epub 2017 Apr 26. PMID: 28460196; PMCID: PMC5581243.
3. Orlando, L., Allaby, R., Skoglund, P. et al. Ancient DNA analysis. *Nat Rev Methods Primers* 1, 14 (2021). <https://doi.org/10.1038/s43586-020-00011-0>

## 17.6. Resources

1. **KrakenUniq:** Breitwieser, F. P., Baker, D. N., & Salzberg, S. L. (2018). KrakenUniq: confident and fast metagenomics classification using unique k-mer counts. *Genome Biology*, vol. 19(1), p. 1–10. <http://www.ec.gc.ca/education/default.asp?lang=En&n=44E5E9B1>
2. **Samtools:** Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, 1000 Genome Project Data Processing Subgroup, The Sequence Alignment/Map format and SAMtools, *Bioinformatics*, Volume 25, Issue 16, 15 August 2009, Pages 2078–2079, <https://doi.org/10.1093/bioinformatics/btp352>
3. **PMDtools:** Skoglund P, Northoff BH, Shunkov MV, Derevianko AP, Pääbo S, Krause J, Jakobsson M. Separating endogenous ancient DNA from modern day contamination in a Siberian Neandertal. *Proc Natl Acad Sci U S A.* 2014 Feb

## *17. Microbiome contamination correction*

- 11;111(6):2229-34. doi: 10.1073/pnas.1318934111. Epub 2014 Jan 27. PMID: 24469802; PMCID: PMC3926038.
4. **pyDamage:** Borry M, Hübner A, Rohrlach AB, Warinner C. PyDamage: automated ancient damage identification and estimation for contigs in ancient DNA de novo assembly. PeerJ. 2021 Jul 27;9:e11845. doi: 10.7717/peerj.11845. PMID: 34395085; PMCID: PMC8323603.
  5. **SourceTracker:** Knights D, Kuczynski J, Charlson ES, Zaneveld J, Mozer MC, Collman RG, Bushman FD, Knight R, Kelley ST. Bayesian community-wide culture-independent microbial source tracking. Nat Methods. 2011 Jul 17;8(9):761-3. doi: 10.1038/nmeth.1650. PMID: 21765408; PMCID: PMC3791591.
  6. **deCOM:** <https://www.biorxiv.org/content/10.1101/2023.01.26.525439v1>, doi: <https://doi.org/10.1101/2023.01.26.525439>
  7. **aMeta:** <https://www.biorxiv.org/content/10.1101/2022.10.03.510579v1>, doi: <https://doi.org/10.1101/2022.10.03.510579>
  8. **Bowtie2:** Langmead, B., Salzberg, S. Fast gapped-read alignment with Bowtie 2. Nat Methods 9, 357–359 (2012). <https://doi.org/10.1038/nmeth.1923>
  9. **cuperdec:** <https://cran.r-project.org/web/packages/cuperdec/index.html>
  10. **decontam:** <https://www.bioconductor.org/packages/release/bioc/html/decontam.html>

**Part IV.**

**Ancient Genomics**

## *Genome Mapping*

A natural extension to any ancient \_meta\_genomics project is to further investigate the specific genomes of the plethora of species and strains you may have detected. In this section of the book, we will look at the specific techniques used to reconstruct ancient genomes using standard genomics reference-based methods, but as always in the context of the short and damaged DNA fragments that are typical of ancient DNA.

## **Genome Mapping**

An important step in the reconstruction of full genomic sequences is mapping. Even relatively short genomes usually cannot be sequenced as a single consecutive piece. Instead, millions of short sequence reads are generated from genomic fragments. These reads can be several hundred nucleotides in length but are considerably shorter for ancient DNA (aDNA).

For many applications involving comparative genomics these ‘reads’ have to be aligned to one or multiple already-reconstructed reference genomes in order to identify differences between the sequenced genome and any given contextual dataset. Aligning millions of short reads to much longer genome sequences in a time-efficient and accurate manner is a bioinformatics challenge for which numerous algorithms and tools have been developed. Each of these programs comes with a variety of parameters that can significantly alter the results and default settings are often not optimal when working with aDNA. Furthermore, read mapping procedures are often part of complex computational genomics pipelines and are therefore not directly applied by many users.

In this chapter we will take a look at specific challenges during read mapping when dealing with aDNA. We will get an overview of common input and output formats and manually apply a read mapper to aDNA data studying the direct effects of variation in mapping parameters. We will conclude the session with an outlook on genotyping, which is an important follow-up analysis step, that in turn is very relevant for down-stream analyses such as phylogenetics.

## **Phylogenomics**

Phylogenetic trees are central tools for studying the evolution of microorganisms, as they provide essential information about their relationships and timing of divergence between microbial strains.

In this chapter, we will introduce basic phylogenetic concepts and definitions, and provide guidance on how to interpret phylogenetic trees. We will then learn how to reconstruct

## *Phylogenomics*

phylogenetic trees from DNA sequences using various methods ranging from distance-based methods to probabilistic approaches, including maximum likelihood and Bayesian phylogenetics. In particular, we will learn how to use ancient genomic data to reconstruct time-calibrated trees with BEAST2.

# 18. Genome Mapping

## 💡 Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate genome-mapping
```

## 18.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

## 18.2. Mapping to a Reference Genome

One way of reconstructing genomic information from DNA sequencing reads is mapping/aligning them to a reference genome. This allows for identification of differences between the genome from your sample and the reference genome. This information can be used for example for comparative analyses such as in phylogenetics. For a detailed explanation of the read alignment problem and an overview of concepts for solving it, please see <https://doi.org/10.1146/annurev-genom-090413-025358>.

In this session we will map two samples to the *Yersinia pestis* (plague) genome using different parameter sets. We will do this “manually” in the sense that we will use all necessary commands one by one in the terminal. These commands usually run in the background when you apply DNA sequencing data processing pipelines.

## 18. Genome Mapping

### 18.2.1. Preparation

The data and conda environment `.yaml` file for this practical session can be downloaded from here: <https://doi.org/10.5281/zenodo.6983174>. See instructions on page.

We will open a terminal and then navigate to the working directory of this session:

```
cd /<path>/<to>/genome-mapping/
```

Then, as already mentioned above, we need to activate the conda environment of this session. By doing this all the necessary tools can be accessed in the current terminal session:

```
conda activate genome-mapping
```

We will be using the Burrows-Wheeler Aligner (Li et al. 2009 – <http://bio-bwa.sourceforge.net>). There are different algorithms implemented for different types of data (e.g. different read lengths). Here, we use BWA backtrack (`bwa aln`), which is suitable for Illumina sequences up to 100bp. Other algorithms are `bwa mem` and `bwa sw` for longer reads.

### 18.2.2. Reference Genome

For mapping we need a reference genome in FASTA format. Ideally we use a genome from the same species that our data relates to or, if not available, a closely related species. The selection of the correct reference genome is highly relevant. E.g. if the chosen genome differs too much from the organism the data relates to, it might not be possible to map most of the reads. Reference genomes can be retrieved from comprehensive databases such as NCBI.

In your directory, you can find 2 samples and your reference. As a first step we will index our reference genome (make sure you are inside your directory).

The first index we will generate is for `bwa`.

```
bwa index YpestisC092.fa
```

The second index will be used by the genome browser we will apply to our results later on:

## 18. Genome Mapping

```
 samtools faidx YpestisC092.fa
```

We need to build a third index that is necessary for the genotyping step, which comes later after mapping:

```
 picard CreateSequenceDictionary R=YpestisC092.fa
```

### 18.2.3. Mapping Parameters

We will be using *bwa aln*, but we need to specify parameters. For now we will concentrate on the “seed length” and the “maximum edit distance”. We will use the default setting for all other parameters during this session. The choice of the right parameters depend on many factors such as the type of data and the specific use case. One aspect is the mapping sensitivity, i.e. how different a read can be from the chosen reference and still be mapped. In this context we generally differentiate between *strict* and *lenient* mapping parameters.

As many other mapping algorithms *bwa* uses a so-called “seed-and-extend” approach. I.e. it initially maps the first  $N$  nucleotides of each read to the genome with relatively few mismatches and thereby determines candidate positions for the more time-intensive full alignment.

A short seed length will generate more such candidate positions and therefore mapping will take longer, but it will also be more sensitive, i.e. there can be more differences between the read and the genome. Long seeds are less sensitive but the mapping procedure is faster.

In this session we will use the following two parameter sets:

#### Lenient

Allow for more mismatches → -n 0.01

Short seed length → -l 16

#### Strict

Allow for less mismatches → -n 0.1

Long seed length → -l 32

We will be working with pre-processed files (`sample1.fastq.gz`, `sample2.fastq.gz`), i.e. any quality filtering and removal of sequencing adapters is already done.

We will map each file once with lenient and once with strict parameters. For this, we will make 4 separate directories, to avoid mixing up files:

## 18. Genome Mapping

```
mkdir sample1_lenient sample2_lenient sample1_strict sample2_strict
```

### 18.2.4. Mapping Sample1

Let's begin with a lenient mapping of sample1.

Go into the corresponding folder:

```
cd sample1_lenient
```

Perform the *bwa* alignment, here for sample1, and specify lenient mapping parameters:

```
bwa aln -n 0.01 -l 16 ../YpestisC092.fa ../sample1.fastq.gz > reads_file.sai
```

Proceed with writing the mapping in *sam* format ([https://en.wikipedia.org/wiki/SAM\\_\(file\\_format\)](https://en.wikipedia.org/wiki/SAM_(file_format))):

```
bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ../YpestisC092.fa reads_file.sai
```

Note that we have specified the sequencing platform (Illumina) by creating a so-called “Read Group” (-r). This information is used later during the genotyping step.

Convert SAM file to binary format (BAM file):

```
samtools view -b -S reads_mapped.sam > reads_mapped.bam
```

For processing of *sam* and *bam* files we use *SAMtools* (Li et al. 2009 – <http://samtools.sourceforge.net/>).

-b specifies to output in BAM format. (-S specifies input is SAM, can be omitted in recent versions.)

Now we sort the *bam* file → Sort alignments by leftmost coordinates:

```
samtools sort reads_mapped.bam > reads_mapped_sorted.bam
```

The sorted bam file needs to be indexed → more efficient for further processing:

## 18. Genome Mapping

```
 samtools index reads_mapped_sorted.bam
```

Deduplication → Removal of reads from duplicated fragments:

```
 samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam
```

```
 samtools index reads_mapped_sorted_dedup.bam
```

Duplicated reads are usually a consequence of amplification of the DNA fragments in the lab. Therefore, they are not biologically meaningful.

We have now completed the mapping procedure. Let's have a look at our mapping results:

```
 samtools view reads_mapped_sorted_dedup.bam | less -S
```

(exit by pressing q)

We can also get a summary about the number of mapped reads. For this we use the *samtools idxstats* command (<http://www.htslib.org/doc/samtools-idxstats.html>):

```
 samtools idxstats reads_mapped_sorted_dedup.bam
```

### 18.2.5. Genotyping

The next step we need to perform is genotyping, i.e. the identification of all SNPs that differentiate the sample from the reference. For this we use the *Genome Analysis Toolkit (GATK)* (DePristo et al. 2011 – <http://www.broadinstitute.org/gatk/>)

It uses the reference genome and the mapping as input and produces an output in *Variant Call Format (VCF)* ([https://en.wikipedia.org/wiki/Variant\\_Call\\_Format](https://en.wikipedia.org/wiki/Variant_Call_Format)).

Perform genotyping on the mapping file:

```
 gatk3 -T UnifiedGenotyper -R .../YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output_m
```

Let's have a look...

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

(exit by pressing q)

### 18.2.6. Mapping and Genotyping for the other Samples/Parameters

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

#### 18.2.6.1. Sample2 Lenient

**i** Note

This is a larger file and lenient mapping takes longer so this file will likely take a few minutes. If you are short on time, proceed with the other sample/parameter settings first and come back to this later if there is time.

```
cd ..
cd sample2_lenient

bwa aln -n 0.01 -l 16 ../YpestisC092.fa ../sample2.fastq.gz > reads_file.sai

bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ../YpestisC092.fa reads_file.sai

samtools view -b -S reads_mapped.sam > reads_mapped.bam

samtools sort reads_mapped.bam > reads_mapped_sorted.bam

samtools index reads_mapped_sorted.bam

samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam

samtools index reads_mapped_sorted_dedup.bam

gatk3 -T UnifiedGenotyper -R ../YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output_mothur
```

#### 18.2.6.2. Sample1 Strict

```
cd ..
cd sample1_strict
```

## 18. Genome Mapping

```
bwa aln -n 0.1 -l 32 ../YpestisC092.fa ../sample1.fastq.gz > reads_file.sai  
bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ../YpestisC092.fa reads_file.sai  
samtools view -b -S reads_mapped.sam > reads_mapped.bam  
samtools sort reads_mapped.bam > reads_mapped_sorted.bam  
samtools index reads_mapped_sorted.bam  
samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam  
samtools index reads_mapped_sorted_dedup.bam  
gatk3 -T UnifiedGenotyper -R ../YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output_m
```

### 18.2.6.3. Sample2 Strict

```
cd ..  
cd sample2_strict  
  
bwa aln -n 0.1 -l 32 ../YpestisC092.fa ../sample2.fastq.gz > reads_file.sai  
bwa samse -r '@RG\tID:all\tLB:NA\tPL:illumina\tPU:NA\tSM:NA' ../YpestisC092.fa reads_file.sai  
samtools view -b -S reads_mapped.sam > reads_mapped.bam  
samtools sort reads_mapped.bam > reads_mapped_sorted.bam  
samtools index reads_mapped_sorted.bam  
samtools rmdup -s reads_mapped_sorted.bam reads_mapped_sorted_dedup.bam  
samtools index reads_mapped_sorted_dedup.bam  
gatk3 -T UnifiedGenotyper -R ../YpestisC092.fa -I reads_mapped_sorted_dedup.bam --output_m
```

### 18.2.7. Comparing Genotypes

In order to combine the results from multiple samples and parameter settings we need to aggregate and comparatively analyse the information from all the *vcf* files. For this we will use the software *MultiVCFAnalyzer* (<https://github.com/alexherbig/MultiVCFAnalyzer>).

It produces various output files and summary statistics and can integrate gene annotations for SNP effect analysis as done by the program *SnpEff* (Cingolani et al. 2012 - <http://snpeff.sourceforge.net/>).

Run *MultiVCFAnalyzer* on all 4 files at once. First `cd` one level up (if you type `ls` you should see your 4 directories, reference, etc.):

```
cd ..
```

Then make a new directory...

```
mkdir vcf_out
```

...and run the programme:

```
multivcfanalyzer NA YpestisC092.fa NA vcf_out F 30 3 0.9 0.9 NA sample1_lenient/mysnps.vcf
```

Let's have a look in the 'vcf\_out' directory (`cd` into it):

```
cd vcf_out
```

Check the parameters we set earlier:

```
less -S info.txt
```

(exit by pressing q)

Check results:

```
less -S snpStatistics.tsv
```

(exit by pressing q)

The file content should look like this:

## 18. Genome Mapping

```
SNP statistics for 4 samples.
Quality Threshold: 30.0
Coverage Threshold: 3
Minimum SNP allele frequency: 0.9
sample SNP Calls (all) SNP Calls (het) coverage(fold) coverage(percent)
refCall allPos noCall discardedRefCall discardedVarCall filteredVarCall unhandledGe
sample1_lenient 213 0 16.38 92.69
4313387 4653728 293297 46103 728 0 0
sample1_strict 207 0 16.33 92.71
4314060 4653728 293403 45633 425 0 0
sample2_lenient 1274 0 9.01 83.69
3893600 4653728 453550 297471 7829 0 4
sample2_strict 1218 0 8.94 83.76
3896970 4653728 455450 295275 4815 0 0
```

First we find the most important parameter settings and then the table of results. The first column contains the dataset name and the second column the number of called SNPs. The genome coverage and the fraction of the genome covered with the used threshold can be found in columns 4 and 5, respectively. For example, sample1 had 207 SNP calls with strict parameters. The coverage is about 16-fold and about 93% of the genome are covered 3 fold or higher (The coverage threshold we set was 3).

### 18.2.8. Exploring the Results

For visual exploration of mapping results so-called “Genome Browsers” are used. Here we will use the *Integrative Genomics Viewer (IGV)* (<https://software.broadinstitute.org/software/igv/>).

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

```
igv
```

Note that you cannot use the terminal while IGV is open. If you want to use it anyways, open a second terminal via the bar on the bottom.

Load your reference (`YpestisC092.fa`):

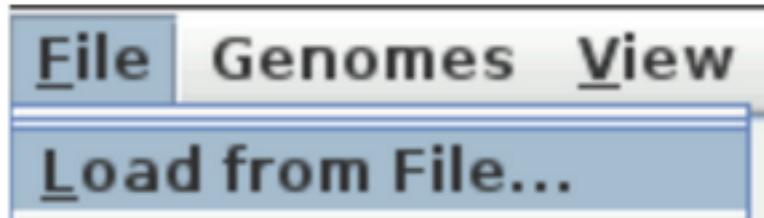
→ *Genomes* → *Load Genome from File*

## 18. Genome Mapping



Load your *bam* files (do this 4 times, once for each mapping):

→ *File* → *Load from File*



Try to explore the mapping results yourself. Here are some questions to guide you. Please also have a look at the examples below.

What differences do you observe between the samples and parameters?

Differences in number of mapped reads, coverage, number of SNPs

Do you see any global patterns?

Which sample is more affected by changing the parameters?

Which of the two samples might be ancient, which is modern?

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

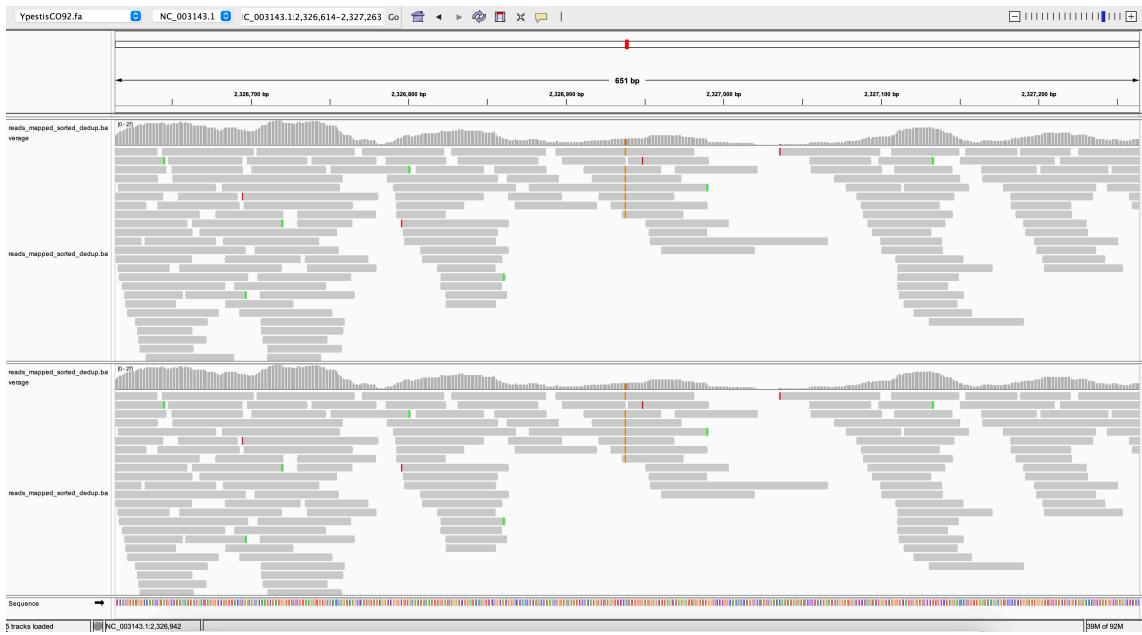
Can you identify SNPs that were called with lenient but not with strict parameters or vice versa?

Let's check out some of these in IGV. Do you observe certain patterns in these genomic regions?

### 18.2.9. Examples

Please find here a few examples for exploration. To get a better visualization we only loaded `sample2_lenient` (top track) and `sample2_strict` (bottom track):

## 18. Genome Mapping



You can see all aligned reads in the current genomic region as stacks of grey arrows. In the middle of the image you see brown dashes in all of the reads. This is a SNP. You also see sporadically green or red dashes in some reads but not all of them at a given position. These sporadic differences are DNA damage such as we typically find it for ancient DNA.

For jumping to a specific coordinate you need to enter it into the coordinate field at the top:



E.g. if you enter 2326942 after the colon in the coordinate field and hit enter, you will jump to the same position as in the screenshot above.

Let's have a look at some positions.

For example position 36472:

## 18. Genome Mapping



In the middle of the image you see a SNP (T) that was called with strict parameters (bottom) but not with lenient parameters (top). But why would it not be called in the top track? It is not called because there are three reads that cover the same position, but do not contain the T. We can see that these reads have other difference to the reference at other positions. That's why they are not mapped with strict parameters. It is quite likely that they originate from a different species. This example demonstrates that sensitive mapping parameters might actually lead to a loss of certain SNP calls.

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

## 18. Genome Mapping

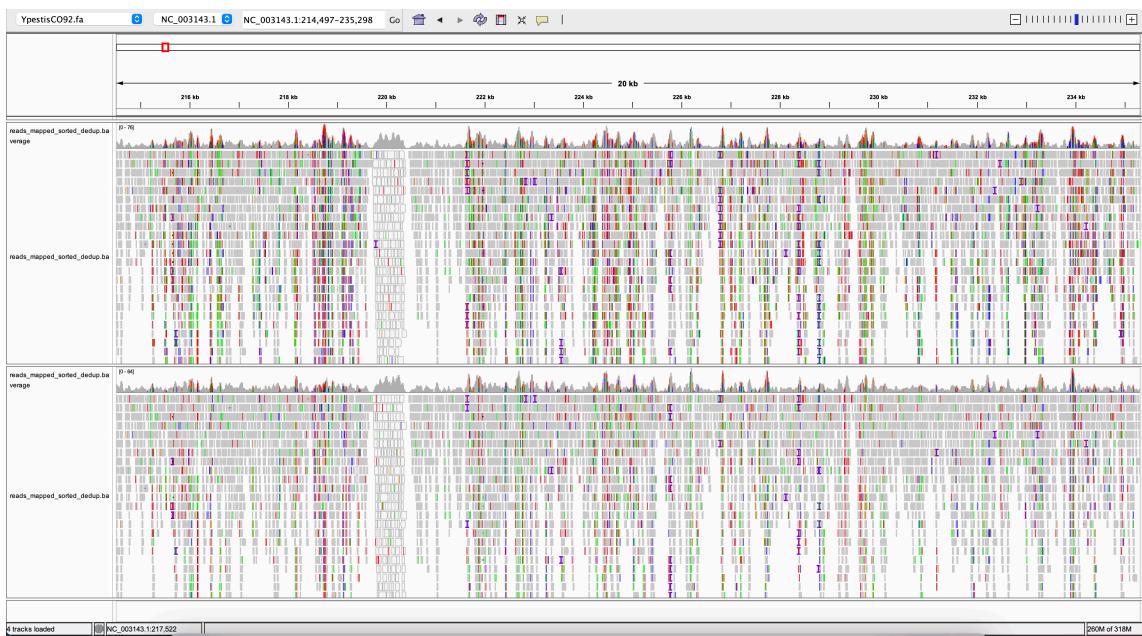


You might need to zoom out a bit using the slider in the upper right corner.

So, what is going on here? We see a lot of variation in most of the reads. This is reduced a bit with strict mapping parameters (bottom track) but the effect is still quite pronounced. Here, we see a region that seems to be conserved in other species as well, so we have a lot of mapping from other organisms. We can't compensate that with stricter mapping parameters and we would have to apply some filtering on genotype level to remove this variation from our genotyping. Removing false positive SNP calls is important as it would interfere with downstream analyses such as phylogenomics.

Such regions can be fairly large. For example, see this 20 kb region around position 224750:

## 18. Genome Mapping



### 18.2.10. Conclusions

- Mapping DNA sequencing reads to a reference genome is a complex procedure that requires multiple steps.
- Mapping results are the basis for genotyping, i.e. the detection of differences to the reference.
- The genotyping results can be aggregated from multiple samples and comparatively analysed e.g. in the context of phylogenomics.
- The chosen mapping parameters can have a strong influence on the results of any downstream analysis.
- This is particularly true when dealing with ancient DNA samples as they tend to contain DNA from multiple organisms. This can lead to mismapped reads and therefore incorrect genotypes, which can further influence downstream analyses.

# 19. Introduction to Phylogenomics



Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate phylogenomics
```

## 19.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

## 19.2. Practical

### 19.2.1. Preparation

The data and conda environment `.yaml` file for this practical session can be downloaded from [here](https://doi.org/10.5281/zenodo.6983184): <https://doi.org/10.5281/zenodo.6983184>. See instructions on page.

Change into the session directory

```
cd /<path>/<to>/phylogenomics/
```

The data in this folder should contain an alignment (`snpAlignment_session5.fasta`) and a `txt` file with the ages of the samples that we are going to be working with in this session (`samples.ages.txt`)

Load the conda environment.

```
conda activate phylogenomics
```

### 19.2.2. Visualize the sequence alignment

In this practical session, we will be working with an alignment produced as you learned in the practical *Genome mapping*.

#### What is in the data?

- the alignment is a SNP alignment (it contains only the variable genomic positions, not the full genomes)
- it contains 33 *Yersinia pestis* sequences and 1 *Yersinia pseudotuberculosis* sequence which can be used as an outgroup
- in this practical, we will investigate the phylogenetic position of four prehistorical *Y. pestis* strains that we have recently discovered: KZL002, GZL002, CHC004 and VLI092

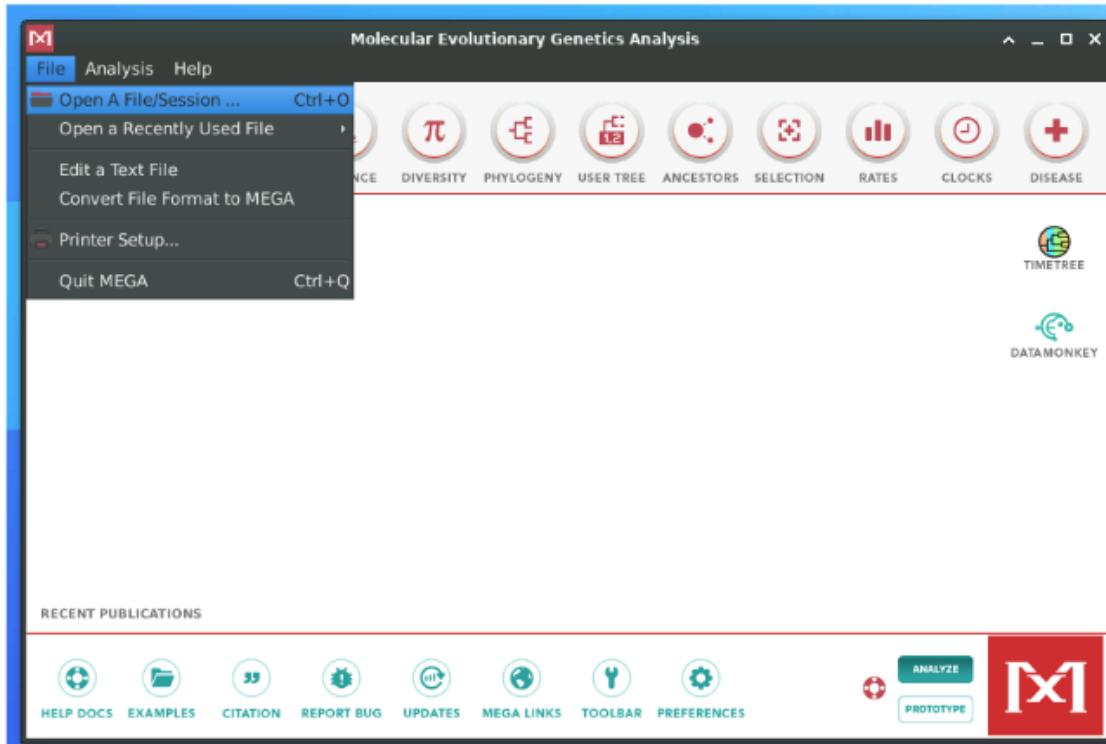
We start by exploring the alignment in *MEGA*. Open the *MEGA* desktop application by typing `mega &` in the terminal.

#### Tip

Adding “`&`” at the end of a commandline allows to run a program in the background while letting the terminal accessible. This particularly useful when starting a graphical interface from the terminal.

Then, load the alignment by clicking on File -> Open A File/Session -> Select the `snpAlignment_session5.fasta` (in the working directory of the session).

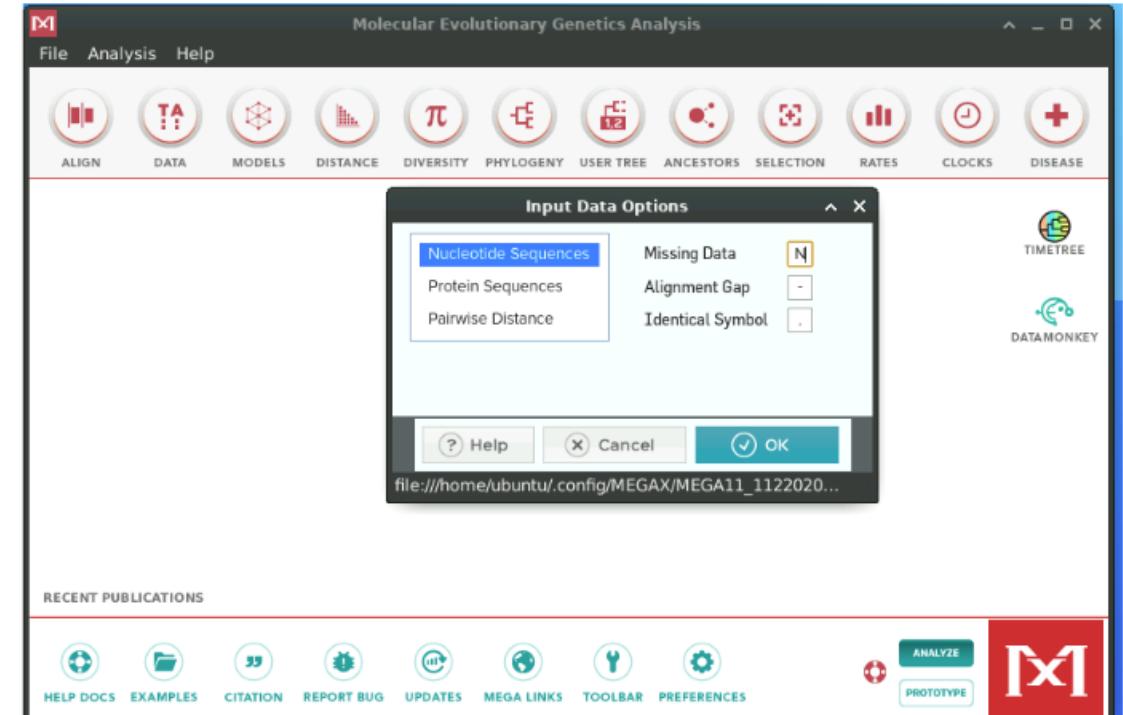
## 19. Introduction to Phylogenomics



It will ask you what you want to do with the alignment. In *MEGA* you can also produce an alignment, however, since our sequences are already aligned we will press on *Analyze*.

Then we will select *Nucleotide Sequences* since we are working with a DNA alignment. Note that *MEGA* can also work with Protein Sequences as well as Pairwise Distance Matrix (which we will cover shortly). In the same window, we will change the character for *Missing Data* to **N** and click in *OK*.

## 19. Introduction to Phylogenomics



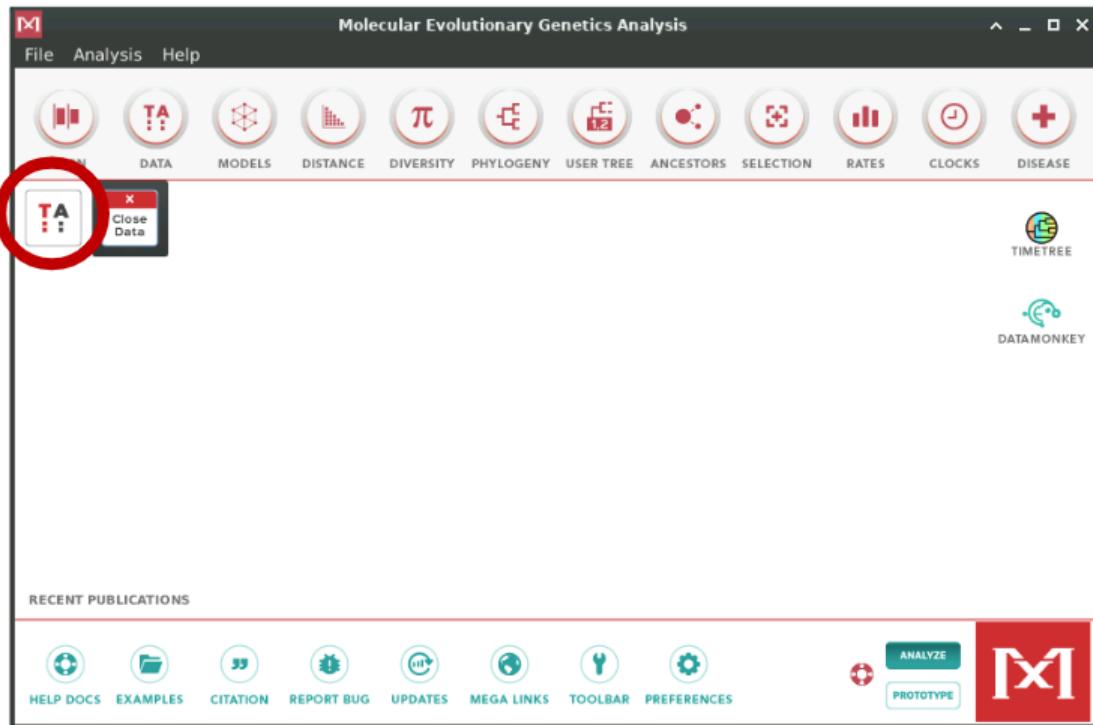
A window would open up asking if our alignment contains protein encoding sequences, and we will select *No*.

### 💡 Tip

If you had protein encoding sequences, you would have selected Yes. This will allow you to treat different positions with different evolutionary modes depending on their codon position. One can do this to take in account that the third codon position can change to different nucleotides without resulting in a different amino acid, while position one and two of the codon are more restricted.

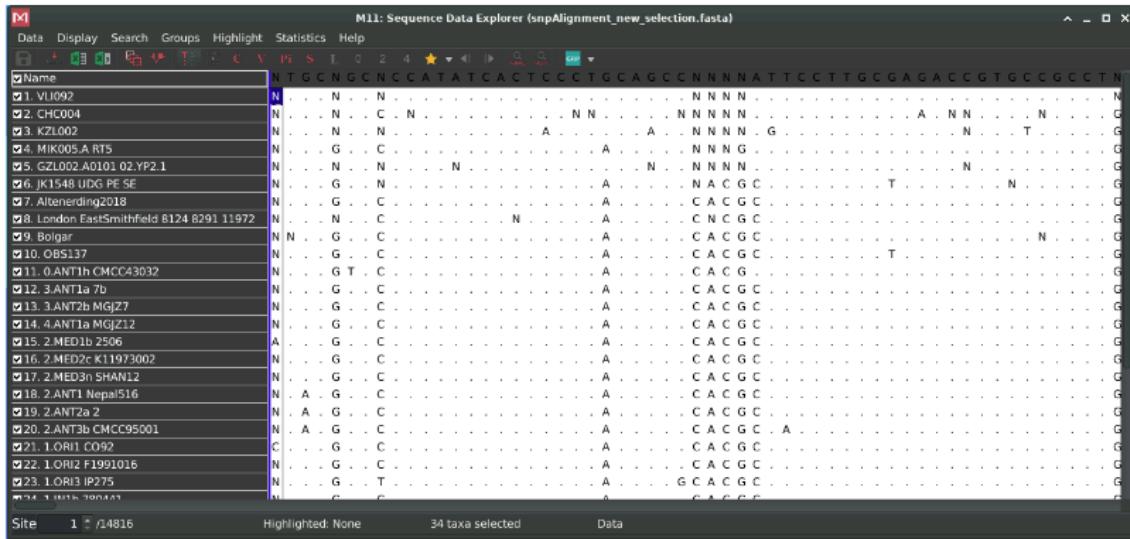
To explore the alignment, you will then click on the box with *TA*

## 19. Introduction to Phylogenomics



You will see an alignment containing sequences from the bacterial pathogen *Yersinia pestis*. Within the alignment, we have four sequences of interest (KZL002, GZL002, CHC004 and VLI092) that date between 5000-2000 years Before Present (BP), and we want to know how they relate to the rest of the *Yersinia pestis* genomes in the alignment.

## 19. Introduction to Phylogenomics



### Questions:

How many sequences are we analysing?

#### Answer

We are analysing 34 sequences.

What are the Ns in the sequences?

#### Answer

They represent positions where we have missing data. We told *MEGA* to encode missing positions as *N*

What do you think the dots represent?

#### Tip

The first line is a **consensus** sequence: it indicates the nucleotide supported by the majority of the sequences in the alignment (90% of the sequences should agree, otherwise an *N* is displayed)

**i** Answer

They represent positions that are the same as the consensus

Once you know this, can you already tell by looking at the alignment which sequence is the most divergent (scroll down)

**i** Answer

We can easily see that the last sequence in the alignment (*Y. pseudotuberculosis*) contains more disagreements to the consensus. This is normal since this is the only genome not belonging to the *Y. pestis* species: we will use it as an outgroup

### 19.2.3. Distance-based phylogeny: Neighbour Joining

The Neighbour Joining (NJ) method is an agglomerative algorithm which can be used to derive a phylogenetic tree from a pairwise distance matrix. In essence, this method will be grouping taxa that have the shortest distance together first, and will be doing this iteratively until all the taxa/sequences included in your alignment have been placed in a tree.

Here are the details of the calculations for a small NJ tree example with 6 taxa:

## 19. Introduction to Phylogenomics

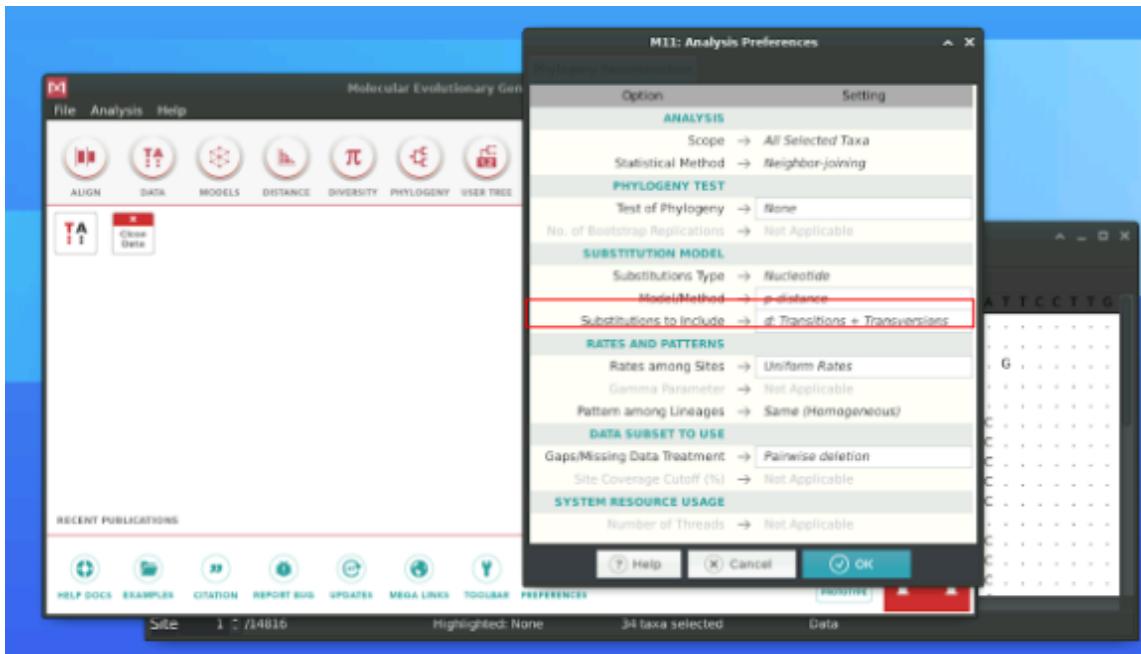
TABLE 27.11. Neighbor-joining example

|                 | Cycle 1  | Cycle 2   | Cycle 3   | Cycle 4  | Cycle 5   |
|-----------------|--|---|---|--|---|
| Distance matrix | A B C D E<br>B 5<br>C 4 7<br>D 7 10 7<br>E 6 9 6 5<br>F 8 11 8 9 8   | U <sub>1</sub> C D E<br>C 3<br>D 6 7<br>E 5 6 5<br>F 7 8 9 8  | U <sub>1</sub> C U <sub>2</sub><br>C 3<br>U <sub>2</sub> 3 4<br>F 7 8 6   | U <sub>2</sub> U <sub>3</sub><br>U <sub>3</sub> 2<br>F 6 6   | U <sub>4</sub><br>F 5   |
| Step 1          | S <sub>A</sub> = (5+4+7+6+8)/4 = 7.5<br>S <sub>B</sub> = (5+7+10+9+11)/4 = 10.5<br>S <sub>x</sub> = (sum all $D_{ij}$ )/(N - 2), where N is the # of OTUs in the set.<br>S <sub>C</sub> = (4+7+7+6+8)/4 = 8<br>S <sub>D</sub> = (7+10+7+5+9)/4 = 9.5<br>S <sub>E</sub> = (6+9+6+5+8)/4 = 8.5<br>S <sub>F</sub> = (8+11+8+9+8)/4 = 11 | S <sub>U<sub>1</sub></sub> = (3+6+5+7)/3 = 7<br>S <sub>C</sub> = (3+7+6+8)/3 = 8<br>S <sub>D</sub> = (6+7+5+9)/3 = 9<br>S <sub>E</sub> = (5+6+5+8)/3 = 8<br>S <sub>F</sub> = (7+8+6)/3 = 10.5   | S <sub>U<sub>2</sub></sub> = (3+3+7)/2 = 6.5<br>S <sub>C</sub> = (3+4+8)/2 = 7.5<br>S <sub>D</sub> = (3+4+6)/2 = 6.5<br>S <sub>E</sub> = (7+8+6)/2 = 10.5   | S <sub>U<sub>2</sub></sub> = (2+6)/2 = 8<br>S <sub>U<sub>3</sub></sub> = (2+6)/1 = 8<br>S <sub>F</sub> = (6+6)/1 = 12  | Because N - 2 = 0, we cannot do this calculation.   |
| Step 2          | Calculate pair with smallest ( $M_{ij}$ ), where $M_{ij} = D_{ij} - S_i - S_j$ . Choose one of these (AB here).  | Smallest are M <sub>AB</sub> = 5 - 7.5 - 10.5 = -13 M <sub>DE</sub> = 5 - 9.5 - 8.5 = -13 Choose one of these (DE here).  | Smallest is M <sub>CU<sub>1</sub></sub> = 3 - 7 - 8 = -12 Choose one of these (DE here).  | Smallest is M <sub>CU<sub>1</sub></sub> = 3 - 6.5 - 7.5 = -11 Choose one of these (DE here).   | Smallest is M <sub>U<sub>2</sub>,F</sub> = 6 - 8 - 12 = -14 M <sub>U<sub>3</sub>,F</sub> = 6 - 8 - 12 = -14 M <sub>U<sub>2</sub>,U<sub>3</sub></sub> = 2 - 8 - 8 = -14 Choose one of these (M <sub>U<sub>2</sub>,U<sub>3</sub></sub> here). |
| Step 3          | Create a node (U) that joins pair with lowest $M_{ij}$ such that S <sub>U<sub>i</sub></sub> = D <sub>ij</sub> /2 + (S <sub>i</sub> - S <sub>j</sub> )/2.   | U <sub>1</sub> joins A and B: S <sub>AU<sub>1</sub></sub> = D <sub>AB</sub> /2 + (S <sub>A</sub> - S <sub>B</sub> )/2 = 1 S <sub>BU<sub>1</sub></sub> = D <sub>AB</sub> /2 + (S <sub>B</sub> - S <sub>A</sub> )/2 = 4 U <sub>2</sub> joins D and E: S <sub>DU<sub>2</sub></sub> = D <sub>DE</sub> /2 + (S <sub>D</sub> - S <sub>E</sub> )/2 = 3 S <sub>EU<sub>2</sub></sub> = D <sub>DE</sub> /2 + (S <sub>E</sub> - S <sub>D</sub> )/2 = 2 | U <sub>3</sub> joins C and U <sub>1</sub> : S <sub>CU<sub>3</sub></sub> = D <sub>CU<sub>1</sub></sub> /2 + (S <sub>C</sub> - S <sub>U<sub>1</sub></sub> )/2 = 2 S <sub>U<sub>3</sub>U<sub>1</sub></sub> = D <sub>CU<sub>1</sub></sub> /2 + (S <sub>U<sub>1</sub></sub> - S <sub>C</sub> )/2 = 1 | U <sub>4</sub> joins U <sub>2</sub> and U <sub>3</sub> : S <sub>U<sub>2</sub>U<sub>4</sub></sub> = D <sub>U<sub>2</sub>U<sub>4</sub></sub> /2 + (S <sub>U<sub>2</sub></sub> - S <sub>U<sub>4</sub></sub> )/2 = 1 S <sub>U<sub>3</sub>U<sub>4</sub></sub> = D <sub>U<sub>3</sub>U<sub>4</sub></sub> /2 + (S <sub>U<sub>3</sub></sub> - S <sub>U<sub>4</sub></sub> )/2 = 1 | For last pair, connect U <sub>4</sub> and F with branch length = 5.   |
| Step 4          | Join i and j according to S above and make all other taxa in form of a star. Branches in black are of unknown length. Branches in red are of known length.   |   |   |  | <br>Note this is the same tree we started with (drawn in unrooted form here).   |
| Step 5          | Calculate new distance matrix of all other taxa to U with $D_{kU} = D_{ik} + D_{jk} - D_{ij}$ , where i and j are those selected from above.   |   | <br>Note this is the same tree we started with (drawn in unrooted form here).   | <br>Note this is the same tree we started with (drawn in unrooted form here).  |   |

From <http://www.icp.ucl.ac.be/~opperd/private/upgma.html>.

Luckily, you won't have to do this by hand since *MEGA* allows you to build a NJ tree. For that go back to *MEGA* and click on the *Phylogeny* symbol (toolbar of the main menu window) and then select *Construct Neighbour Joining Tree*. Type "Yes" when you are asked if you want to use the currently active date. In the window that pop up, you will then chance the *Model/Method* to *p-distance*. Then press *OK* and a window with the calculated phylogenetic tree will pop up.

## 19. Introduction to Phylogenomics



### i p-distances?

A NJ tree can be built from any type of distances. This includes:

- p-distances (also called raw distances): these are simply the proportion of differences between sequences
- corrected distances: these are based on an underlying substitution model (JC69, K80, GTR...) and account for multiple substitutions at the same sites (which would result in only one visible difference)
- p-distances and corrected distances should be similar when the number of substitutions is low compared to the genome length

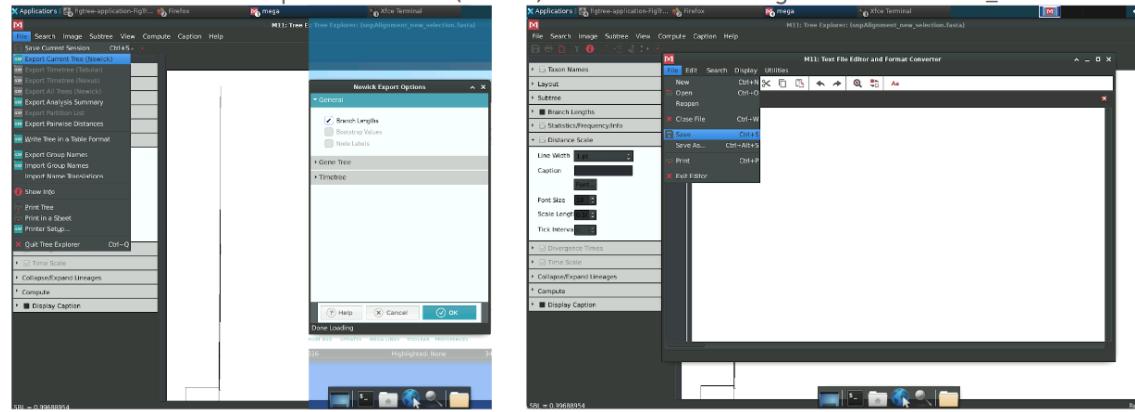
**note:** a “substitution” is a type of mutation in which a nucleotide is replaced by another.

Since the tree is not easily visualised in *MEGA*, we will export it in newick format (a standard text format for trees) and explore our tree in *FigTree*. This tool has a better interface for visually manipulating trees and allows us to interact with the phylogenetic tree.

To do that you will click on *File*, then *Export current tree (Newick)* and click on *Branch Lengths* to include those in the newick annotation. When you press *OK*, a new window

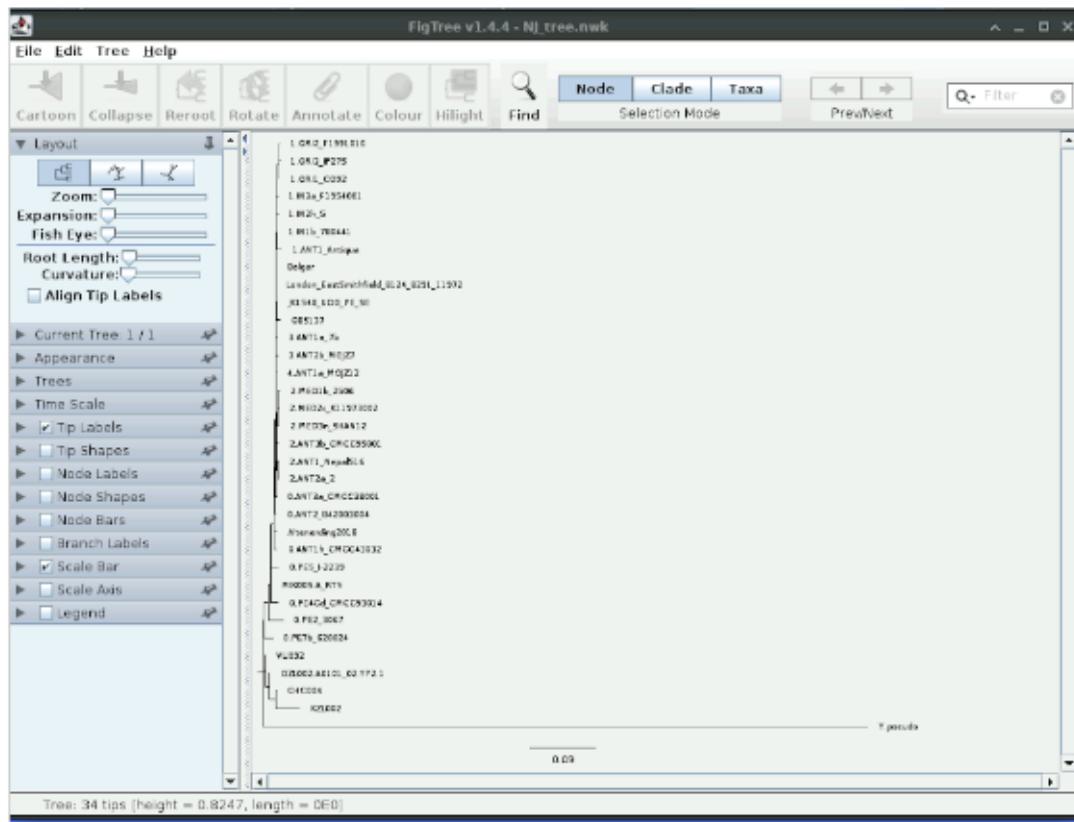
## 19. Introduction to Phylogenomics

with the tree in newick format will pop up and you will then press *File* -> *Save* and saved it as *NJ\_tree.nwk*. You can then close the text editor and tree explorer windows (no need to save the session).



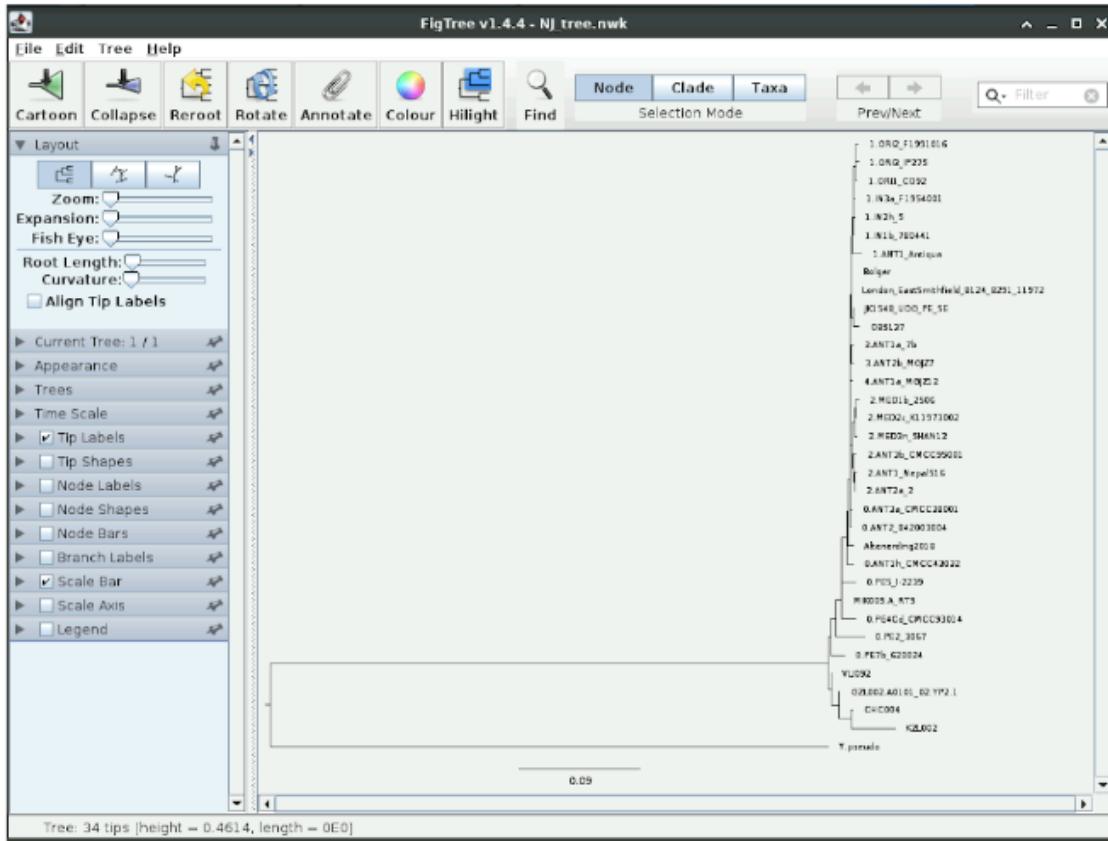
As said above, we will explore own NJ tree in *FigTree*. Open the software by typing `figtree &` in the terminal (if you use the same terminal window as the one in which you ran mega, you might have to press `enter` first). Then, open the NJ tree by clicking on *File* -> *Open* and selecting the file with the NJ tree *NJ\_tree.nwk*

## 19. Introduction to Phylogenomics



Note that even though a root is displayed by default in *FigTree*, NJ trees are actually **unrooted**. We know that *Yersinia pseudotuberculosis* (labelled here as *Y. pseudotuberculosis*) is an outgroup to *Yersinia pestis*. You can reroot the tree by selecting *Y.pseudotuberculosis* and pressing *Reroot*.

## 19. Introduction to Phylogenomics



Now we have a rooted tree.

### Questions:

How much time did the NJ-tree calculation take?

**i** Answer

~1 second

How many leaves/tips has our tree?

**i** Answer

34, i.e. the number of sequences in our SNP alignment.

Where are our taxa of interest? (KZL002, GZL002, CHC004 and VLI092)

## 19. Introduction to Phylogenomics

### **i** Answer

They all fall ancestral to the rest of *Yersinia pestis* in this tree.

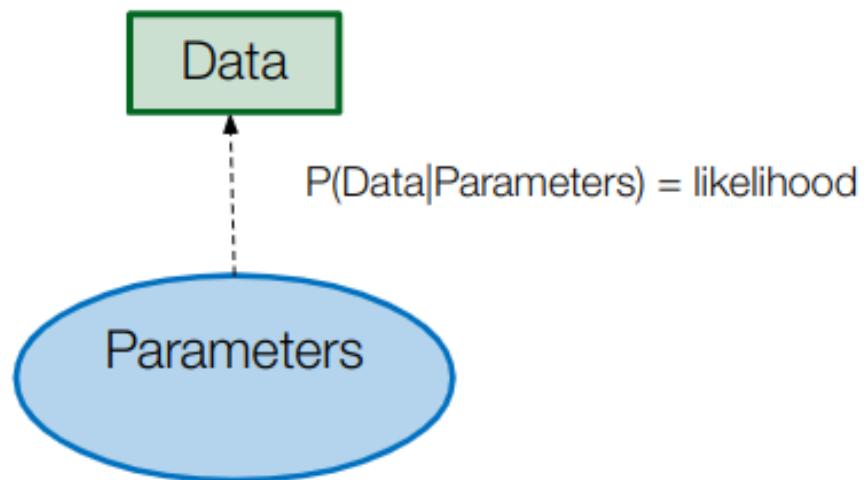
Do they form a monophyletic group (a clade)?

### **i** Answer

Yes, they form a monophyletic group. We can also say that this group of prehistoric strains form their own lineage.

### **19.2.4. Probabilistic methods: Maximum Likelihood and Bayesian inference**

These are the most commonly used approach today. In general, probabilistic methods are statistical techniques that are based on models under which the observed data is generated through a stochastic process depending on a set of parameters which we want to estimate. The probability of the data given the model parameters is called the likelihood.



**Question:**

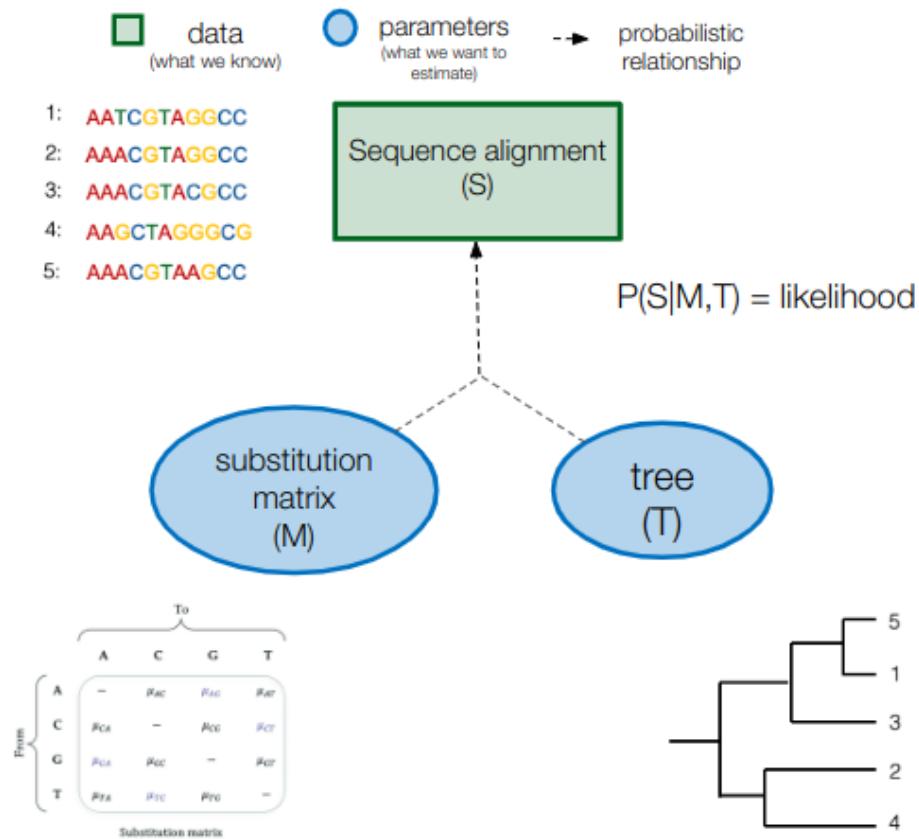
## 19. Introduction to Phylogenomics

In a phylogenetic probabilistic model, what are the data and what are the parameters?

### Answer

In a phylogenetic probabilistic model, the data is the sequence alignment and the parameters, are:

- the parameters of the chosen substitution model (substitution rates and base frequencies)
- the phylogenetic tree

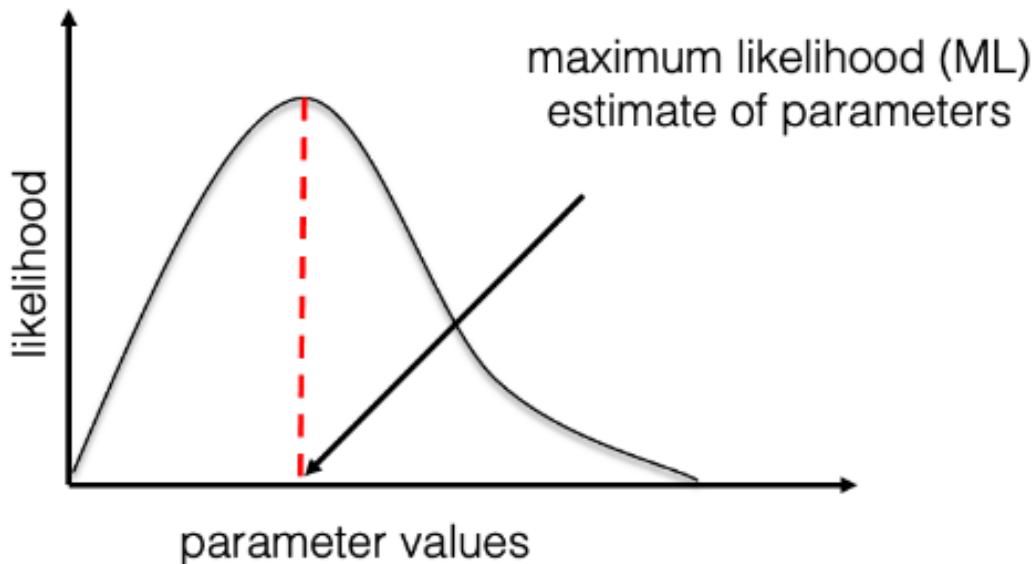


### 19.2.4.1. Maximum likelihood estimation and bootstrapping

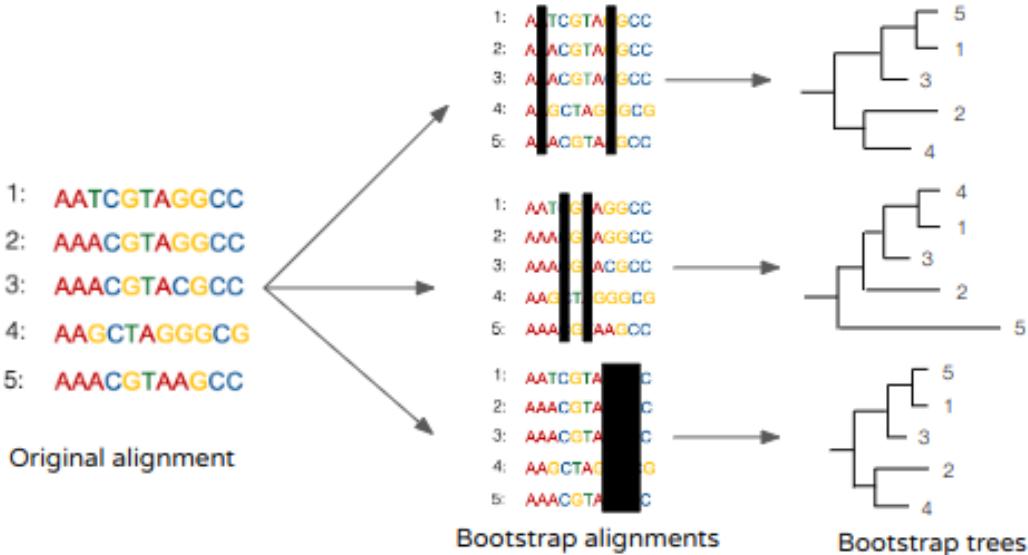
One way we can make inferences from a probabilistic model is by finding the combination of parameters which maximises the likelihood. These parameter values are called maximum

## 19. Introduction to Phylogenomics

likelihood (ML) estimates. We are usually not able to compute the likelihood value for all possible combinations of parameters and have to rely on heuristic algorithms to find the maximum likelihood estimates.



The Maximum likelihood estimates are point estimates, i.e. single parameter values (for example, a tree), which does not allow to measure the associated uncertainty. A classic method to measure the uncertainty in ML trees is bootstrapping, which consists in repeatedly “disturbing” the alignment by masking sites randomly and estimating a tree from each of these bootstrap alignments.



For each clade in the ML tree, a bootstrap support value is computed which corresponds to the proportion of bootstrap trees containing the clade. This gives an indication of how robustly the clade is supported by the data (i.e. whether it holds even after disturbing the dataset). Bootstrapping can be used to measure the topology uncertainty of trees estimated with any inference method.

**i Note**

Bootstrapping can be used to measure uncertainty with any type of inference method, including distance methods

**Let's make our own ML tree!**

Here is a command to estimate an ML phylogenetic tree together with bootstraps using *RAxML* (you may find the list of parameters in the *RAxML* manual):

```
raxmlHPC-PTHREADS -m GTRGAMMA -T 3 -f a -x 12345 -p 12345 -N autoMRE -s snpAlignment_session
```

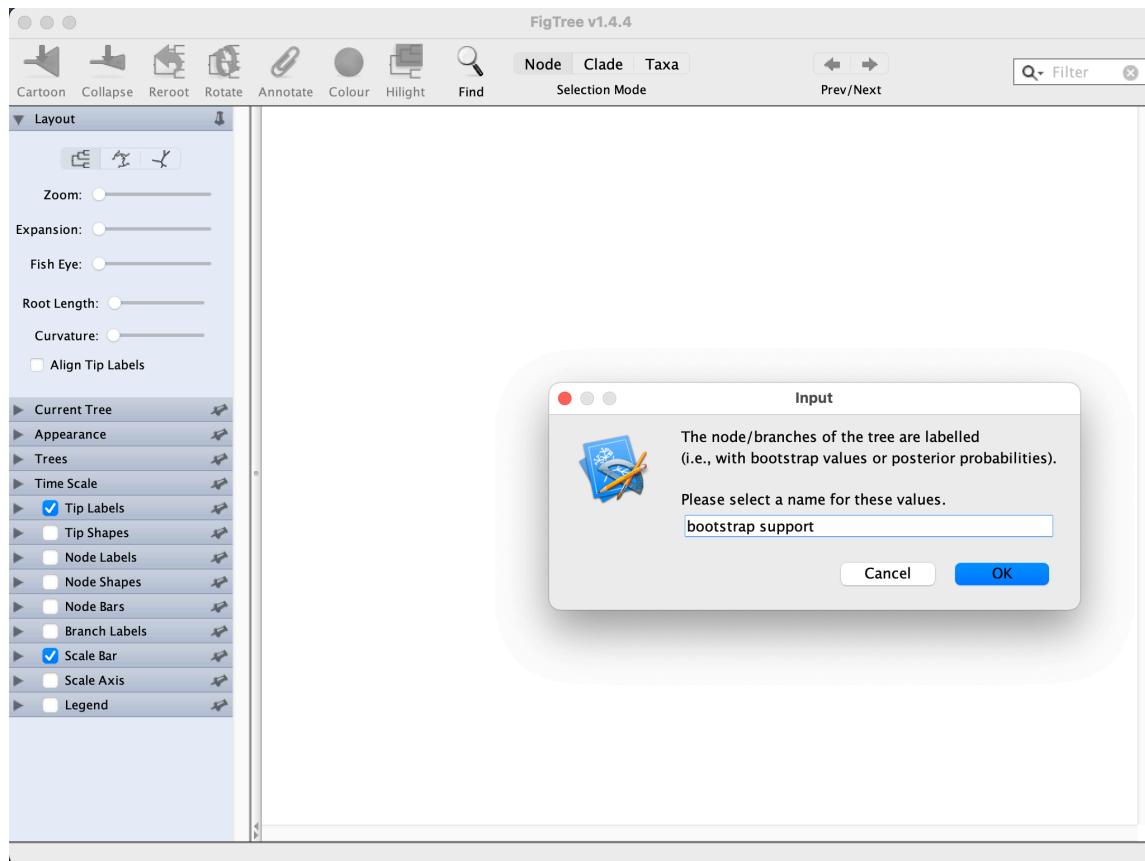
Here is the meaning of the chosen parameters:

## 19. Introduction to Phylogenomics

```
raxml -m GTRGAMMA -f a -x 12345 -p 12345 -N autoMRE -s snpAlignment.fasta -n full_dataset.tre
```

GTR (+GAMMA) substitution model    ML estimation + rapid bootstrapping    Random seeds    Criterium to determine the number of bootstraps    input    output suffix

Once the analysis has been completed, you can open the tree using *Figtree* (RAxML\_bipartitions.full\_dataset.tre file, change “label” to “bootstrap support” at the prompt).



The tree estimated using this model is a substitution tree (branch lengths represent genetic distances in substitutions/site). As for the NJ tree, it is not oriented in time: this is an unrooted tree (displayed with a random root in Figtree). You can reroot the tree in *Figtree* using *Y. pseudotuberculosis* as an outgroup, as previously.

## Questions:

## 19. Introduction to Phylogenomics

Can you confirm the position of our genomes of interest (KZL002, GZL002, CHC004 and VLI092)?

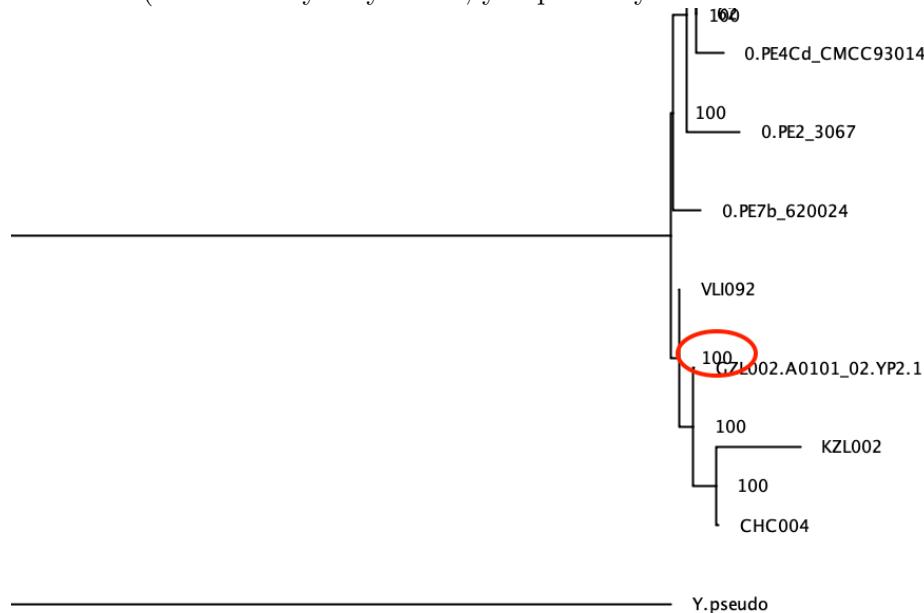
### Answer

Yes. Just as in the NJ tree, they form a clade which is basal to the rest of the *Y. pestis* diversity.

Is that placement well-supported? (look at the bootstrap support value: click on the “Node Labels” box and open the drop-down menu, change “Node ages” to “bootstrap support”)

### Answer

The placement is strongly supported as indicated by a bootstrap support of 100% for this clade (it is not very easy to see, you probably need to zoom in a bit)



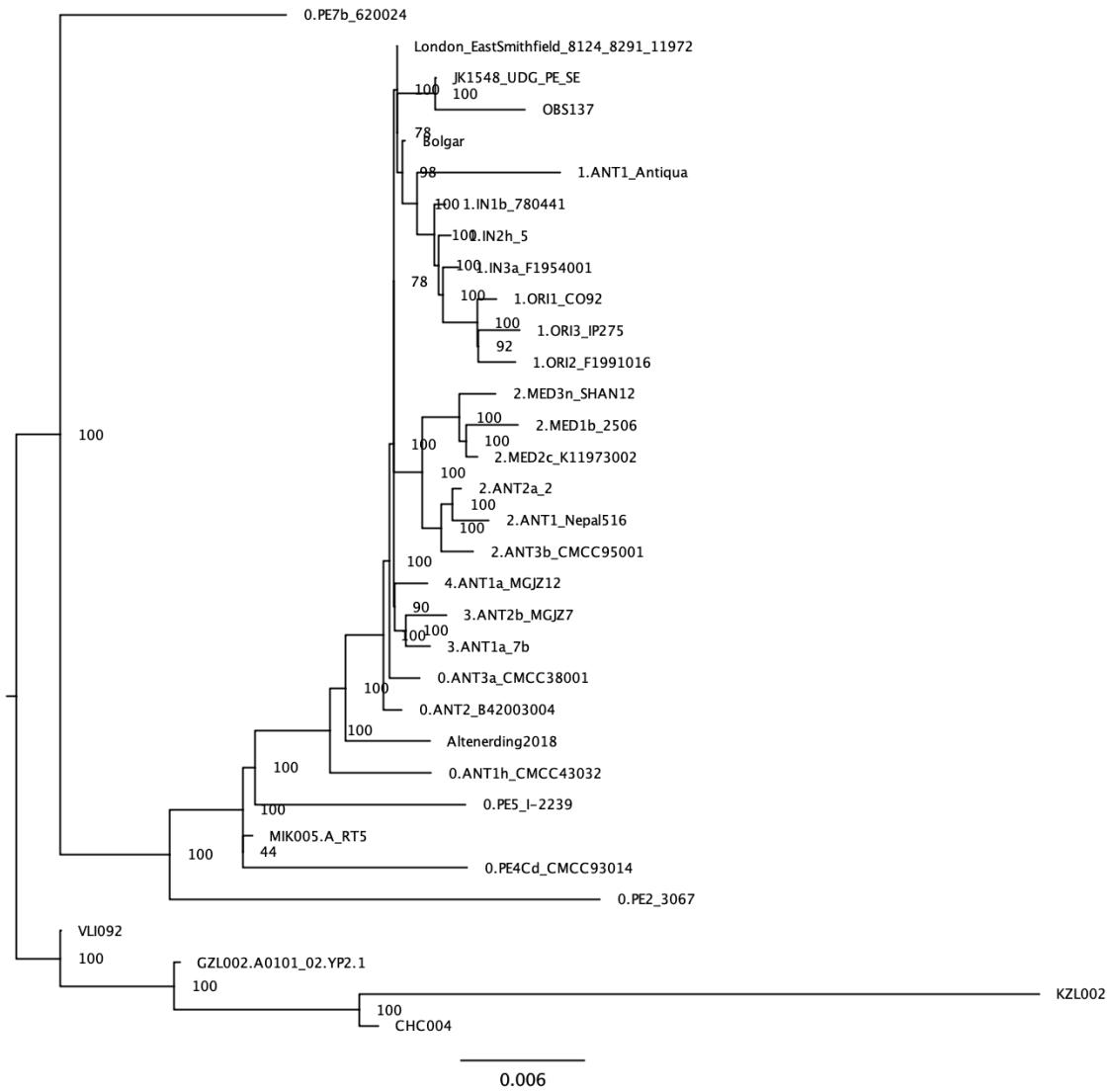
You can notice that the phylogeny is difficult to visualize due to the long branch leading to *Y. pseudotuberculosis*. Having a very distant outgroup can also have deleterious effects on the estimated phylogeny (due to the so-called “long branch attraction” effect). We can construct a new phylogeny after removing the outgroup:

- go back to the alignment in mega, unclick *Y.pseudotuberculosis*, and export in fasta format (“Data” -> “Export Data” -> change “Format” to “Fasta” and click “Ok”;

## 19. Introduction to Phylogenomics

you can save it as: “snpAlignment\_without\_outgroup.fas”

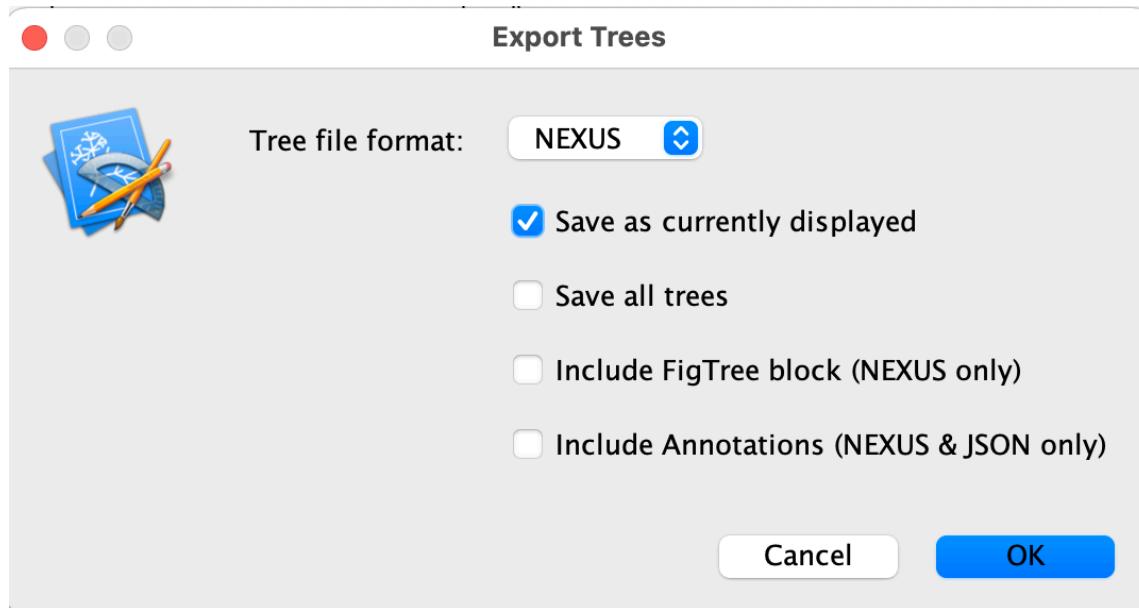
- run raxml on this new alignment (change input to “snpAlignment\_without\_outgroup.fas” and output prefix to “without\_outgroup” in the commandline)
- open the bipartition... file in figtree and reroot the tree based on the knowledge we have gained previously: place the root on the branch leading to the prehistoric Y. pestis strains (KZL002, GZL002, CHC004 and VLI092).



Lastly, we will export the rooted tree from figtree: File -> Export trees -> select the “save as currently displayed” box and save as “ML\_tree\_rooted.tre” (this will be useful for the

## 19. Introduction to Phylogenomics

section “Temporal signal assessment” at the end of this tutorial)



### 19.2.4.2. Estimating a time-tree using Bayesian phylogenetics (*BEAST2*)

Now, we will try to use reconstruct a phylogeny in which the branch lengths do not represent a number of substitutions but instead represent the time of evolution. To do so, we will use the dates of ancient genomes (C14 dates) to calibrate the tree in time. This assumes a molecular clock hypothesis in which substitutions occur at a rate that is relatively constant in time so that the time of evolution can be estimated based on the number of substitutions.

#### Note

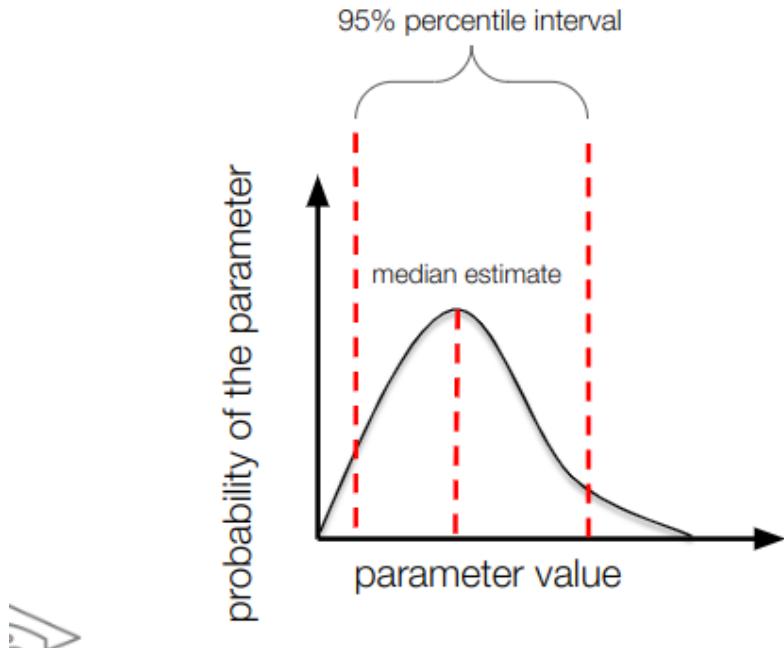
A great advantage of ancient pathogen genomes is that they provide key calibration points to estimate molecular clocks and dated phylogenies. This is more difficult to do with modern data alone.

We will estimate a time-tree from our alignment using Bayesian inference as implemented in the *BEAST2* software. Bayesian inference is based on a probability distribution that is different from the likelihood: the posterior probability. The posterior probability is the probability of the parameters given the data. It is easier to interpret than the likelihood because it directly contains all the information about the parameters: point estimates such

## 19. Introduction to Phylogenomics

as the median or the mean can be directly estimated from it, but also percentile intervals which can be used to measure uncertainty.

$$\begin{aligned} P(\text{Parameters}|\text{Data}) &\propto P(\text{Data}|\text{Parameters}) \times P(\text{Parameters}) \\ \text{Posterior} &\propto \text{Likelihood} \times \text{Prior} \end{aligned}$$



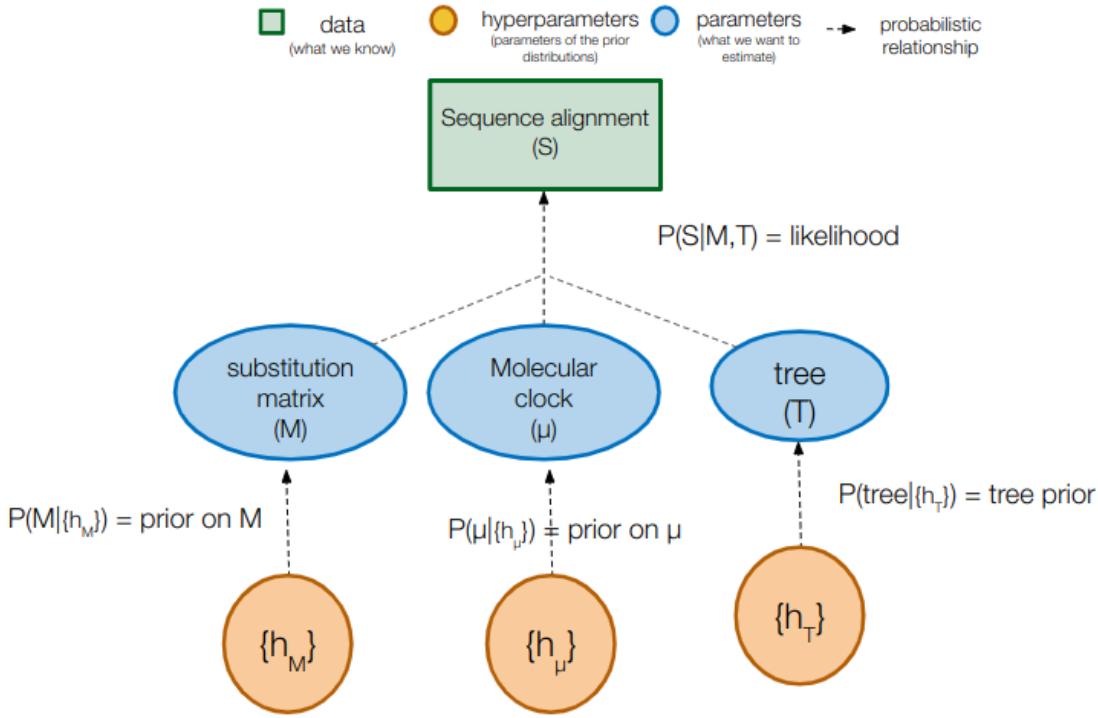
The Bayes theorem tells us that the posterior probability is proportional to the product of the likelihood and the “prior” probability of the parameters:

$$P(\text{parameters}|\text{data}) \propto P(\text{data}|\text{parameters}).P(\text{parameters})$$

The diagram illustrates the components of Bayes' theorem. It shows three arrows pointing upwards from labels to their corresponding terms in the equation. The first arrow points from the label "posterior probability" to the term  $P(\text{parameters}|\text{data})$ . The second arrow points from the label "likelihood" to the term  $P(\text{data}|\text{parameters})$ . The third arrow points from the label "prior probability" to the term  $P(\text{parameters})$ .

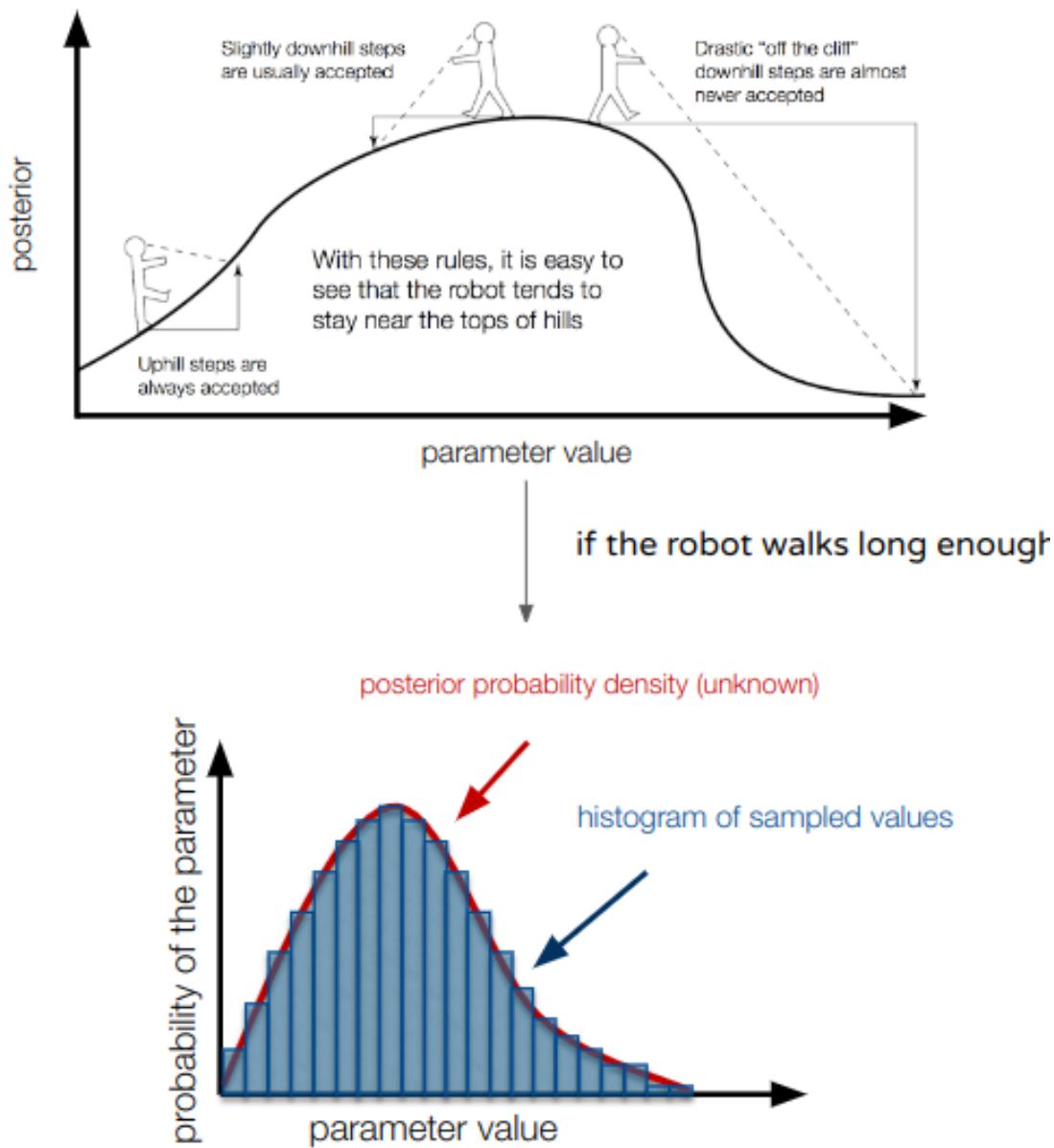
Therefore, for Bayesian inference, we need to complement our probabilistic model with prior distributions for all the parameters. Because we want to estimate a time tree, we also add another parameter: the molecular clock (average substitution rate in time units).

## 19. Introduction to Phylogenomics



To characterize the full posterior distribution of each parameter, we would need in theory to compute the posterior probability for each possible combination of parameters. This is impossible, and we will instead use an algorithm called Markov chain Monte Carlo (MCMC) to approximate the posterior distribution. The MCMC is an algorithm which iteratively samples values of the parameters from the posterior distribution. Therefore, if the MCMC has run long enough, the (marginal) posterior distribution of the parameters can be approximated by a histogram of the sampled values.

## 19. Introduction to Phylogenomics



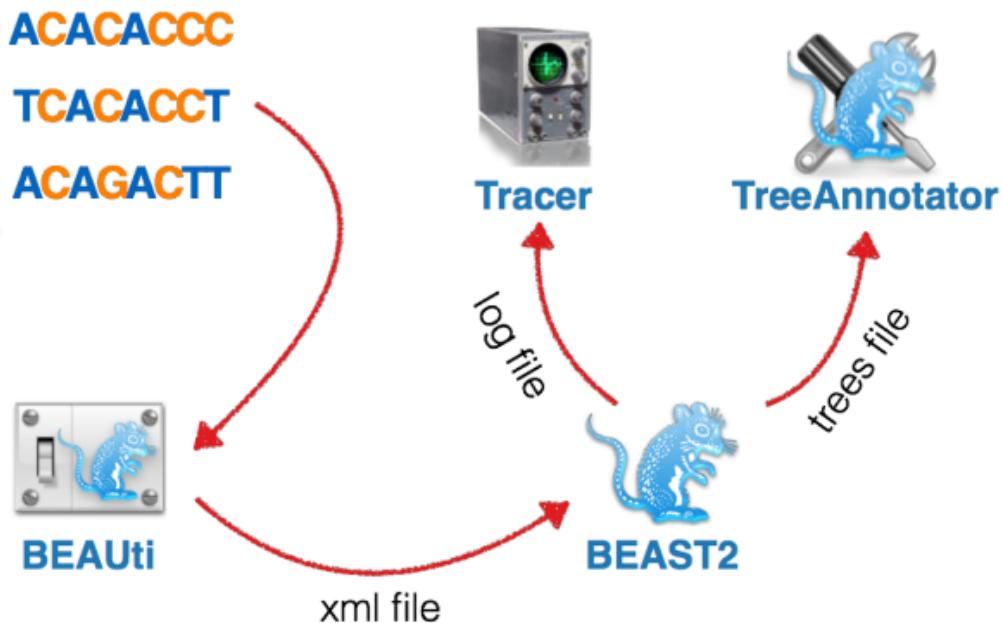
### 19.2.4.2.1. Set up a *BEAST2* analysis

## 19. Introduction to Phylogenomics

### Tip

The “taming the beast” website has great tutorials to learn setting a *BEAST2* analysis. In particular, the “Introduction to BEAST2”, “Prior selection” and “Time-stamped data” are good starts.

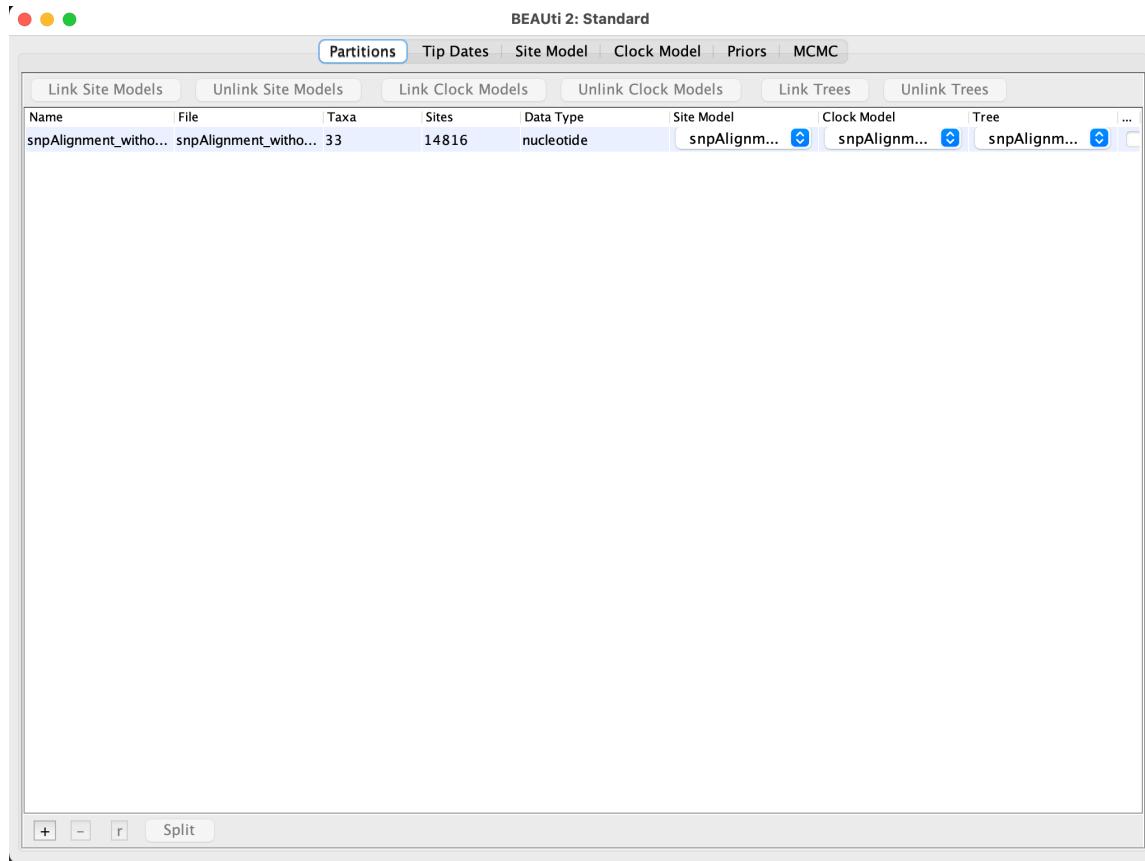
The different components of the *BEAST2* analysis can be set up in the program *BEAUTi*:



Open *BEAUTi* by typing `beauti &` in the terminal, and set up an analysis as followed:

- load the alignment without outgroup in the “Partitions” tab (“File” -> “Import alignment”; select “nucleotide”)

## 19. Introduction to Phylogenomics



- set the sampling dates in the “Tip dates” tab:
  - select “Use tip dates”
  - click on “Auto-configure” -> “read from file” and select the sample\_dates.txt file
  - change “Since some time in the past” to “Before present”

## 19. Introduction to Phylogenomics

BEAUTI 2: Standard

Partitions Tip Dates Site Model Clock Model Priors MCMC

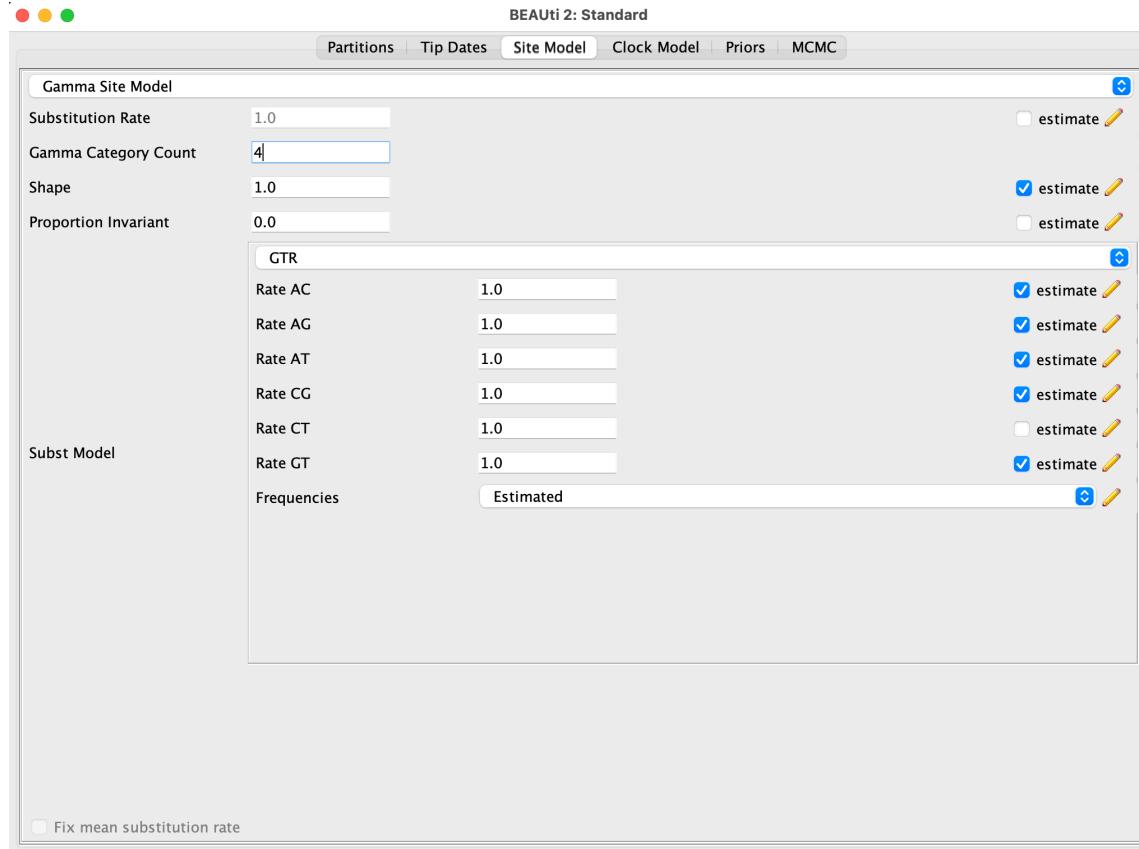
Use tip dates  
 Dates specified:  numerically as year  Before the present  
 as dates with format dd/M/yyyy ?

Auto-configure Clear

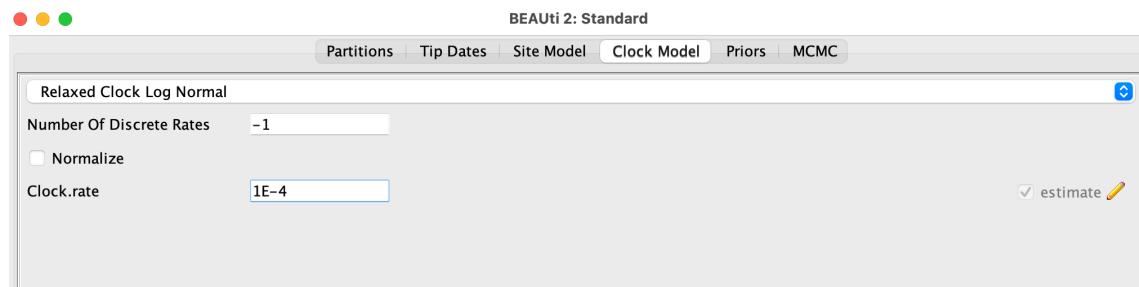
| Name                                  | Date (raw value) | Height |
|---------------------------------------|------------------|--------|
| 0.PE7b_620024                         | 0                | 0.0    |
| 1.ANT1_Antiqua                        | 0                | 0.0    |
| 1.IN1b_780441                         | 0                | 0.0    |
| 1.IN2h_5                              | 0                | 0.0    |
| 1.IN3a_F1954001                       | 0                | 0.0    |
| 1.ORI1_C092                           | 0                | 0.0    |
| 1.ORI2_F1991016                       | 0                | 0.0    |
| 1.ORI3_IP275                          | 0                | 0.0    |
| 2.ANT1_Nepal516                       | 0                | 0.0    |
| 2.ANT2a_2                             | 0                | 0.0    |
| 2.ANT3b_CMCC95001                     | 0                | 0.0    |
| 2.MED1b_2506                          | 0                | 0.0    |
| 2.MED2c_K11973002                     | 0                | 0.0    |
| 2.MED3n_SHAN12                        | 0                | 0.0    |
| 3.ANT1a_7b                            | 0                | 0.0    |
| 3.ANT2b_MGJZ7                         | 0                | 0.0    |
| 4.ANT1a_MGJZ12                        | 0                | 0.0    |
| Altenerding2018                       | 1453             | 1453.0 |
| Bolgar                                | 569              | 569.0  |
| CHC004                                | 3979             | 3979.0 |
| GZL002.A0101_02.YP2.1                 | 4430             | 4430.0 |
| JK1548_UDG_PE_SE                      | 394              | 394.0  |
| KZL002                                | 2596.5           | 2596.5 |
| London_EastSmithfield_8124_8291_11972 | 601              | 601.0  |
| MIK005.A_RT5                          | 3789             | 3789.0 |
| OBS137                                | 229              | 229.0  |
| VLI092                                | 4725.5           | 4725.5 |

- select the substitution model in the “Site model” tab:
  - chose a GTR model
  - use 4 Gamma categories for the Gamma site model: this is to account for variations of the substitution rate accross sites (site=nucleotide position)

## 19. Introduction to Phylogenomics



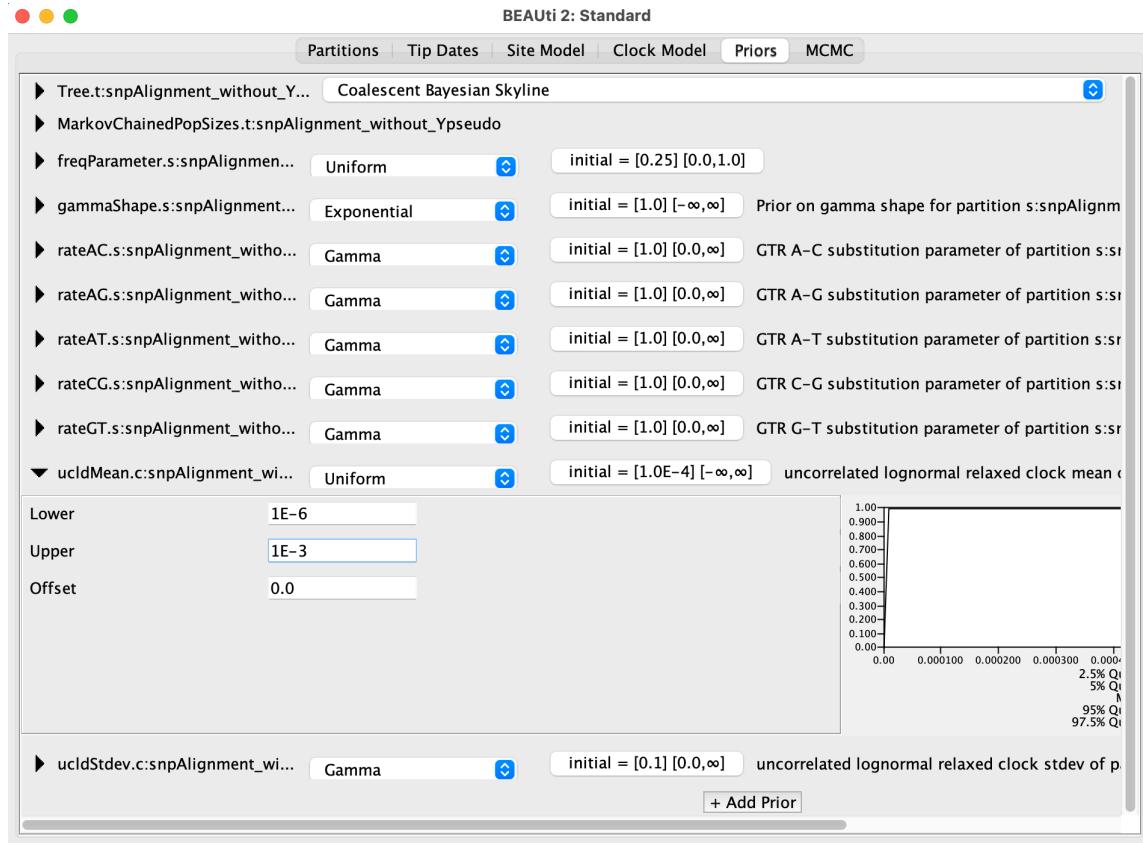
- choose the molecular clock model in the “Clock model” tab:
  - use a relaxed clock lognormal model (this is to allow for some variation of the clock rate across branches)
  - change the initial value of the clock rate to 10-4 substitution/site/year (**10-4 can be written 1E-4**)



- choose the prior distribution of parameters in the “Priors” tab:

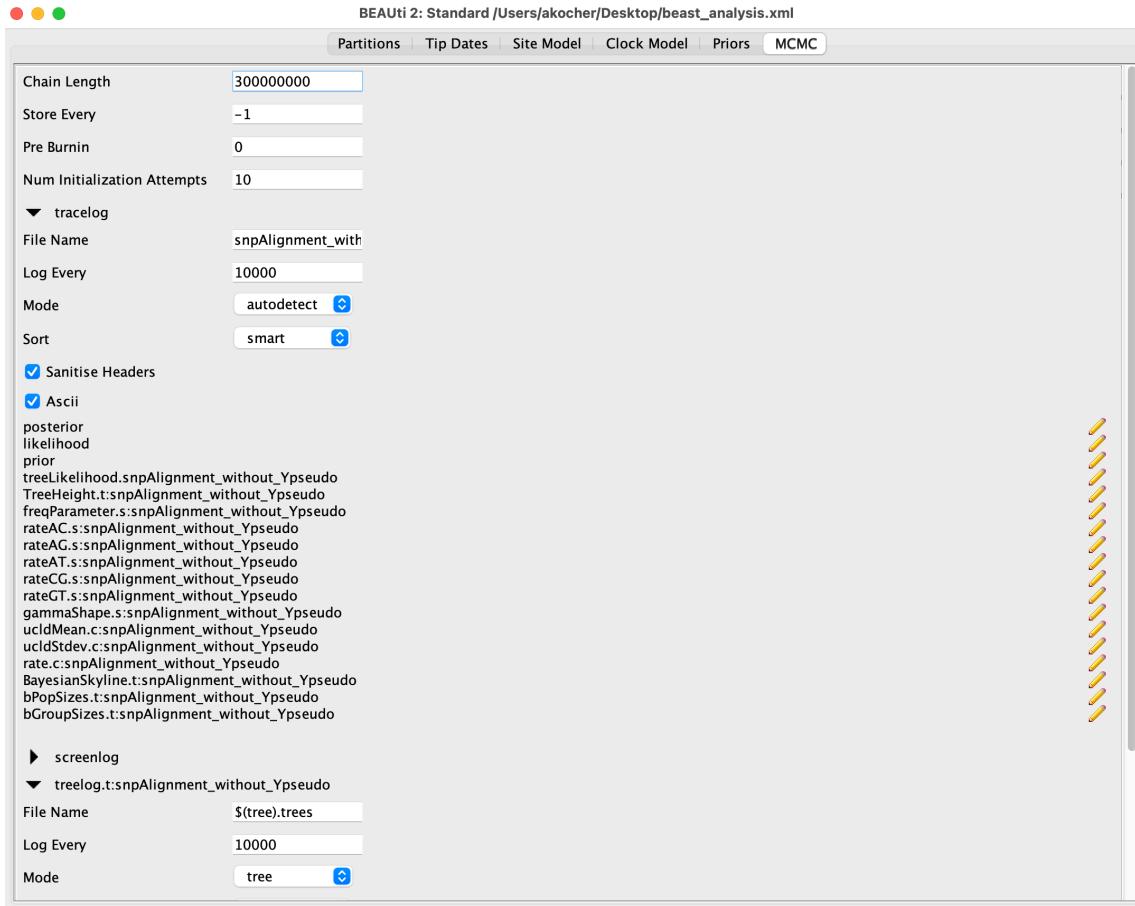
## 19. Introduction to Phylogenomics

- use a Coalescent Bayesian Skyline tree prior
- change the mean clock prior to a uniform distribution between 1E-6 and 1E-3 subst/site/year
- leave everything else to default



- set up the MCMC in the “MCMC” tab:
  - use a chain length of 300M
  - sample the monodimensional parameters and trees every 10,000 iterations (unfold “tracelog” and “treelog” menus and change “log every” to 10,000)

## 19. Introduction to Phylogenomics



- save the analysis setup as an xml file: “File” -> “Save as”; you can name the file “beast\_analysis\_Y\_pestis.xml”

Now that the analysis is setup, we can run it using BEAST:

```
beast beast_analysis_Y_pestis.xml
```

Once the analysis is running two files should have been created and are continuously updated:

- the “snpAlignment\_without\_outgroup.log” file which contains the values sampled by the MCMC for various monodimensional parameters such as the clock rate, as well as other values that are logged along the MCMC such as the posterior probability and the likelihood.

## 19. Introduction to Phylogenomics

- the “snpAlignment\_without\_outgroup.trees” file which contains the MCMC trees sampled by the MCMC

While the analysis is running, you can start reading the next section

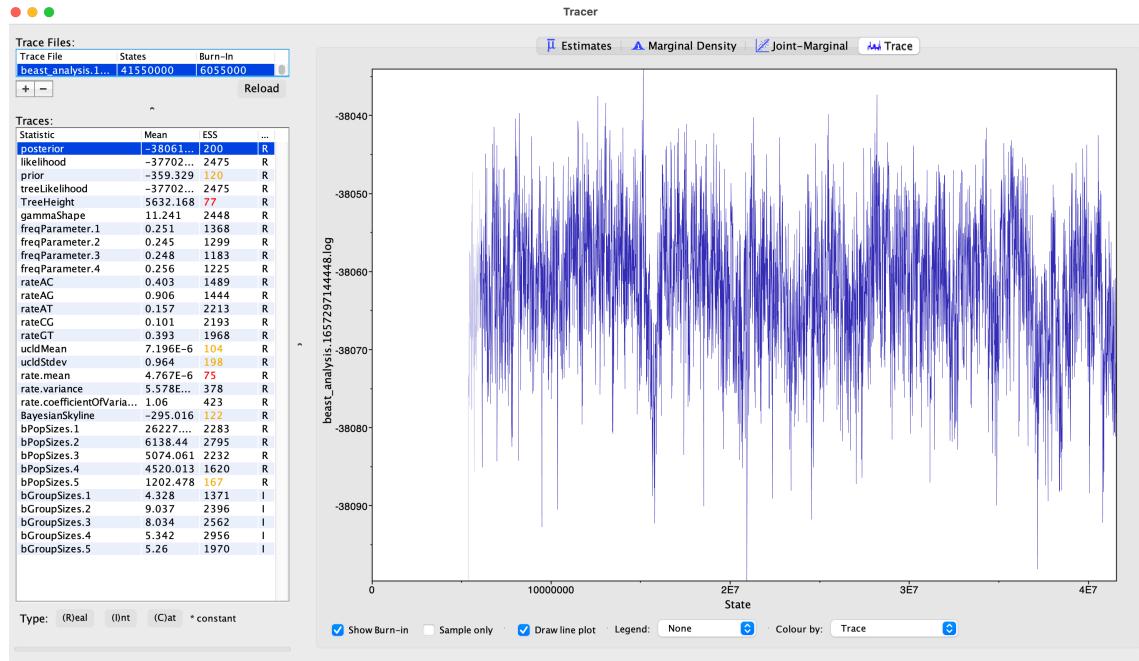
### 19.2.4.2.2. Assessing BEAST2 results

#### **i** Reminder

We are using an MCMC algorithm to sample the posterior distribution of parameters. If the MCMC has run long enough, we can use the sampled parameters to approximate the posterior distribution itself. Therefore, we have to check first that the MCMC chain has run long enough.

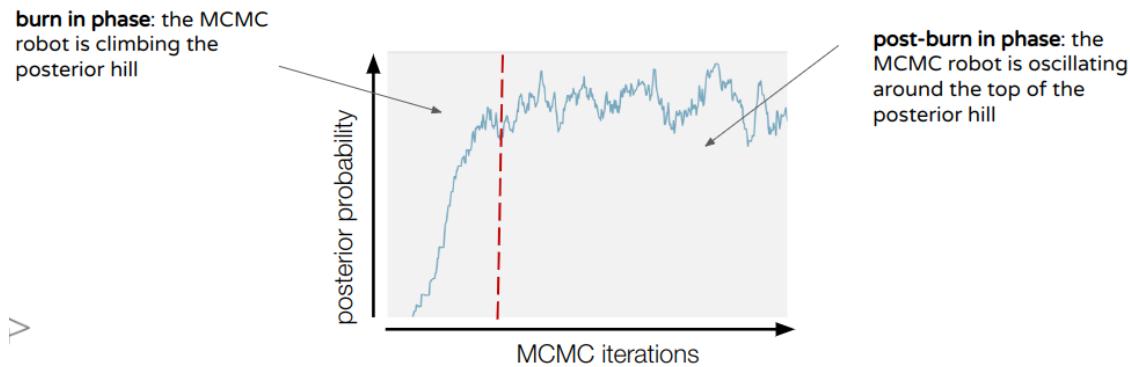
We can assess the MCMC sampling using the program *Tracer*. *Tracer* can read BEAST log files and generate statistics and plots for each of the sampled parameters. Most importantly, *Tracer* provides:

- trace plots:** show the sampled parameter values along the MCMC run. Trace plots are a very useful MCMC diagnostic tool.

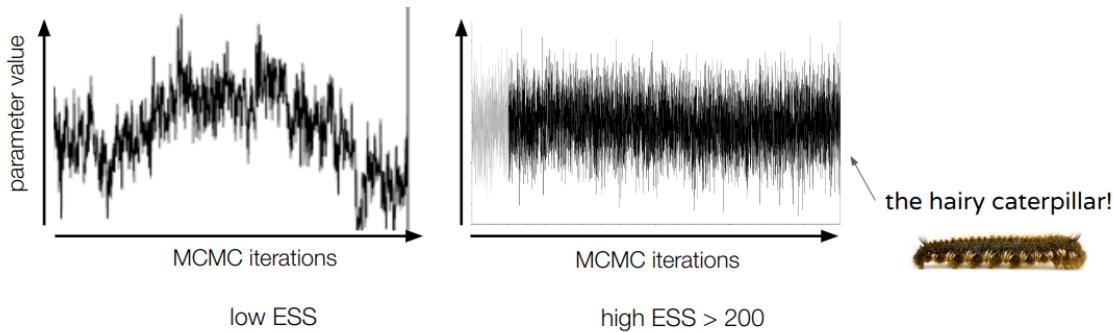


## 19. Introduction to Phylogenomics

The first thing that one needs to assess is whether the MCMC has passed the so called “burn-in” phase. The MCMC starts with a random set of parameters and will take some time to reach a zone of high posterior probability density. The parameter values that are sampled during this initial phase are usually considered as noise and discarded (by default, tracer discards the first 10% of samples). The burn-in phase can be visualized on trace plots as an initial phase during which the posterior probability of sampled parameters is constantly increasing before reaching a plateau:

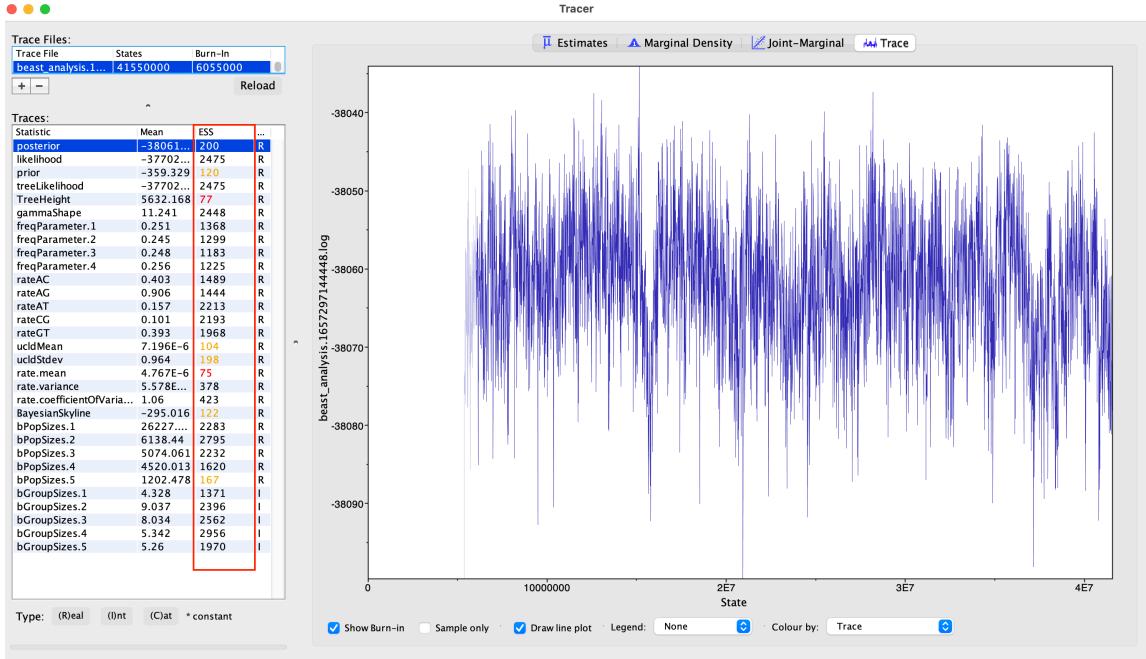


Once the burn-in phase is passed, one can look at the trace plots to assess if the parameters have been sampled correctly and long enough. Usually, when this is the case, the trace should be quite dense and oscillating around a central value (nice trace plots should look like “hairy caterpillars”). In the figure below, the trace on the left doesn’t look good, the one on the right does:



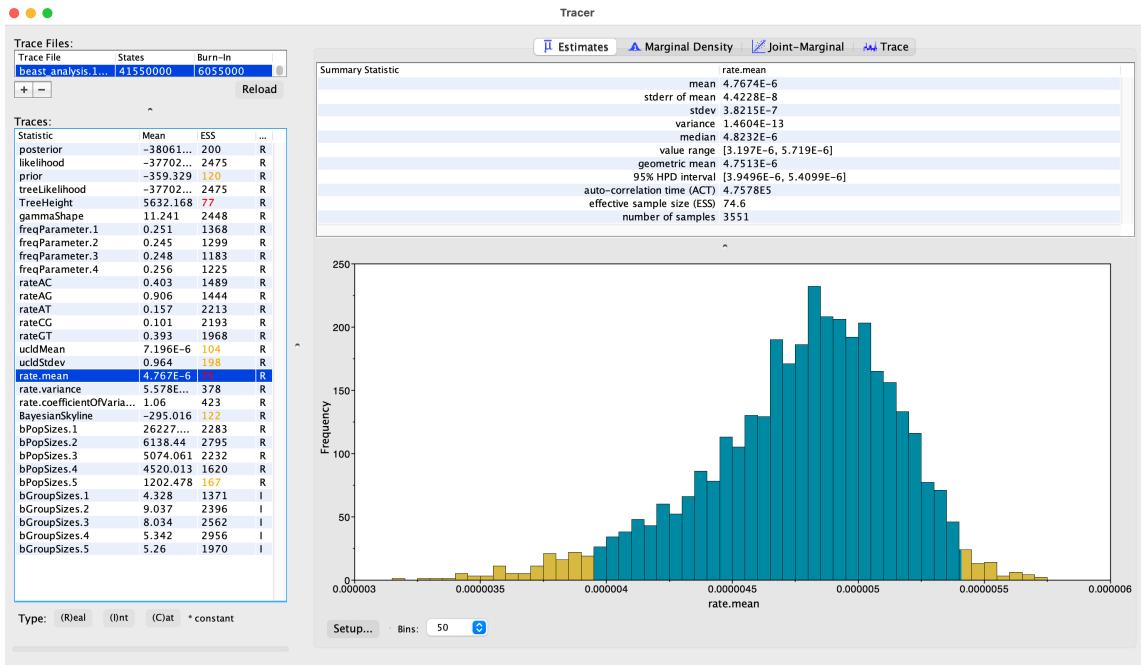
- **ESS values:** tracer also calculates effective sample sizes (ESS) for each of the sampled parameters. ESSs are estimates of the number of sampled parameter values after correcting for auto-correlation along the MCMC. As a rule of thumb, one usually considers that an MCMC is run long enough if all parameter’s ESS are  $> 200$ . Note that if the trace looks like a hairy caterpillar, the corresponding ESS value should be high.

## 19. Introduction to Phylogenomics



- **Parameter estimates:** *Tracer* also provides statistics and plots to explore the posterior distribution of the parameters. These should be considered only if the trace plot and ESS values look fine. In the “Estimates” tab, after selecting the chosen parameter in the left panel, the upper-right panel shows point estimates (mean, median) and measures of uncertainty (95% HPD interval), and the bottom-right panel shows a histogram of the sampled value:

## 19. Introduction to Phylogenomics



Let's now load the “snpAlignment\_without\_outgroup.log” file into *Tracer*. Open a new terminal tab, activate the conda environment with `conda activate phylogenomics`, open tracer with `tracer &`, and then “File” -> “Import trace file” -> select “snpAlignment\_without\_outgroup.log”. Note that one can load a BEAST2 log file into tracer even if the analysis is still running. This allows to assess if the MCMC is running correctly or has run long enough before it's completed.

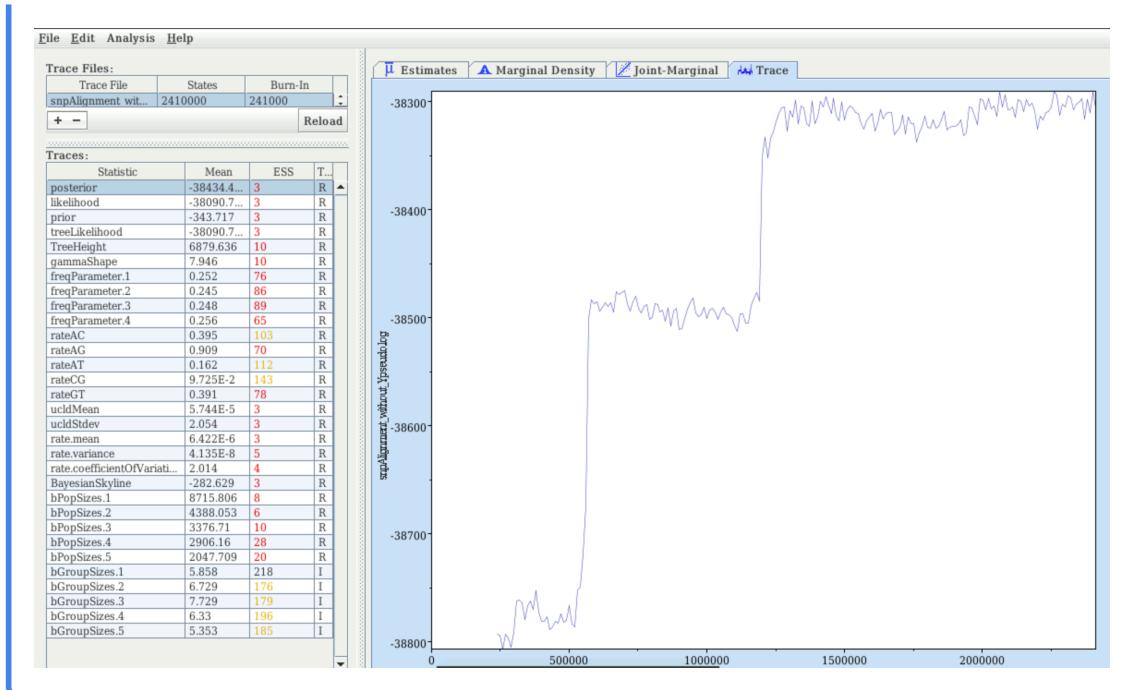
### Question:

Has the MCMC run long enough?

#### Answer

You have probably let your analysis run for 10-20 mins before looking at the log file, and this is definitely not sufficient: the burnin phase has recently been passed, the trace plots do not look very dense and ESS values are low. It would probably take a few hours for the analysis to complete. Luckily we have run the analysis in advance and saved the log files for you in the “intermediateFiles” folder: “snpAlignment\_without\_outgroup.log” and “snpAlignment\_without\_outgroup.trees”

## 19. Introduction to Phylogenomics



You can now load the “intermediateFiles/snpAlignment\_without\_outgroup.log” file into *Tracer*.

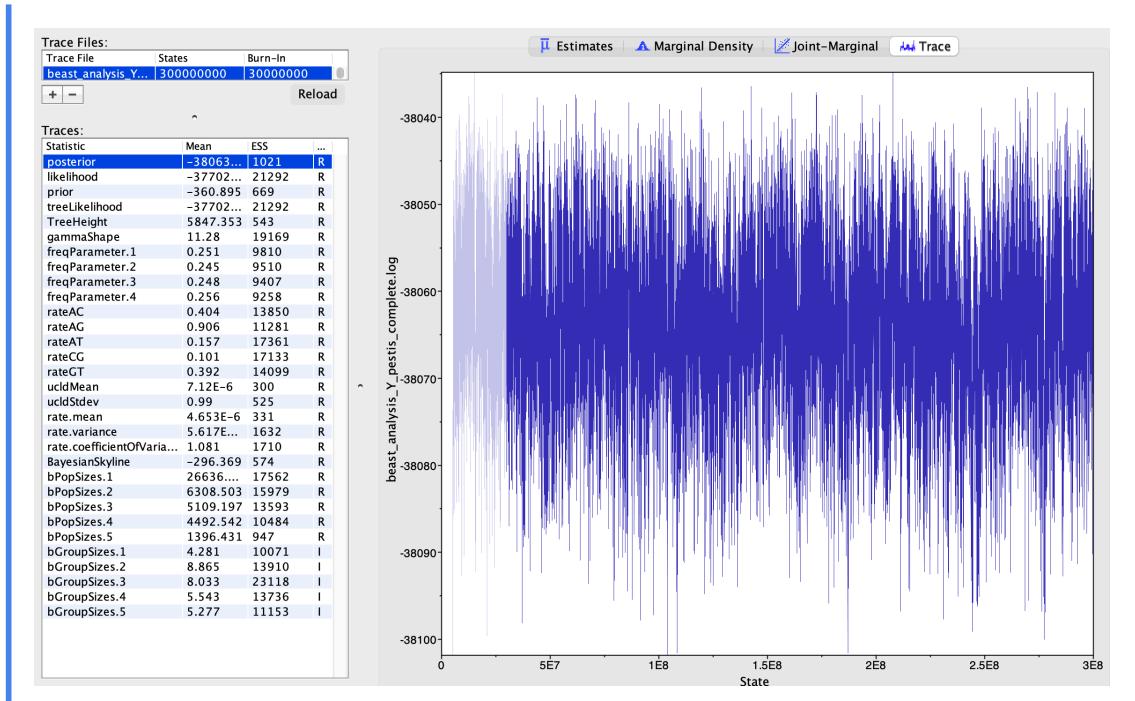
### Questions:

Has the MCMC run long enough?

#### i Answer

Yes! The trace plots look good and all ESSs are > 200

## 19. Introduction to Phylogenomics

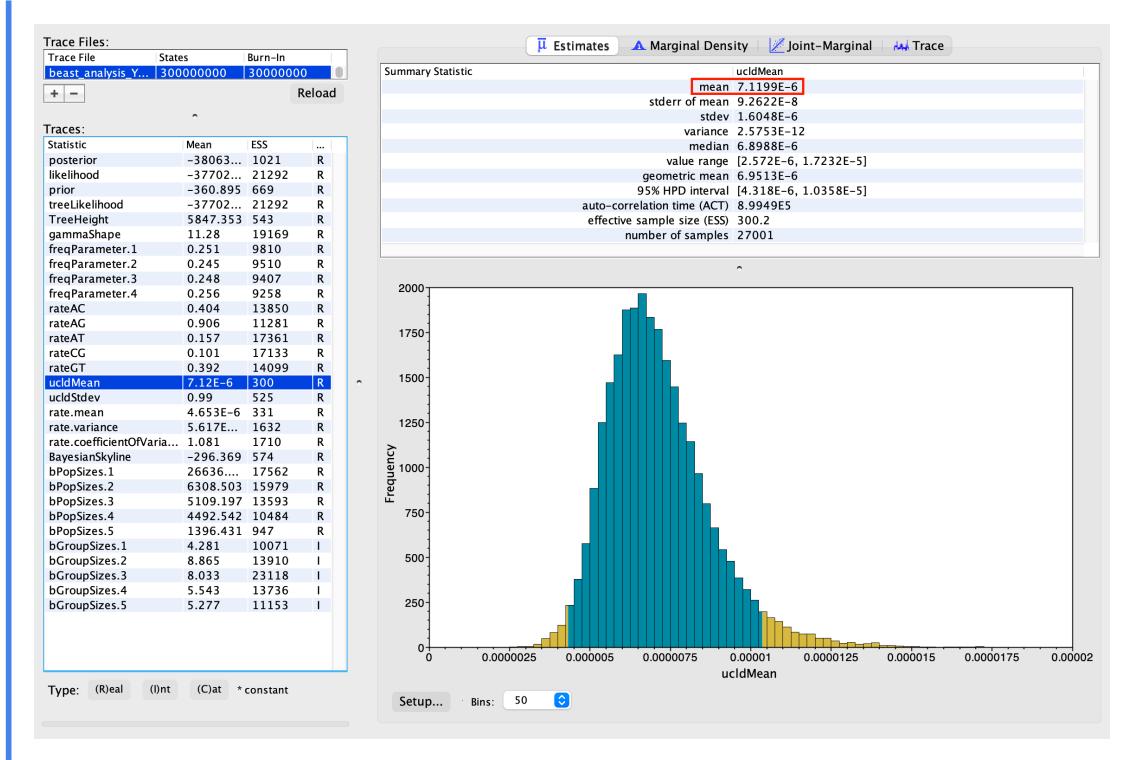


What is your mean estimate of the clock rate (ucld mean)?

### Answer

~7.10<sup>-6</sup> substitution/site/year. Note, however, that this estimate is largely biased since we used a SNP alignment containing only variable positions. In order to get an unbiased estimate of the substitution rate, we should have used the full alignment or account for the number of constant sites by using a “filtered” alignment (see here). In general, this is good practice since not accounting for conserved positions in the alignment can sometimes affect the tree as well (although this should usually be minor, which is why we didn’t bother to do this here).

## 19. Introduction to Phylogenomics

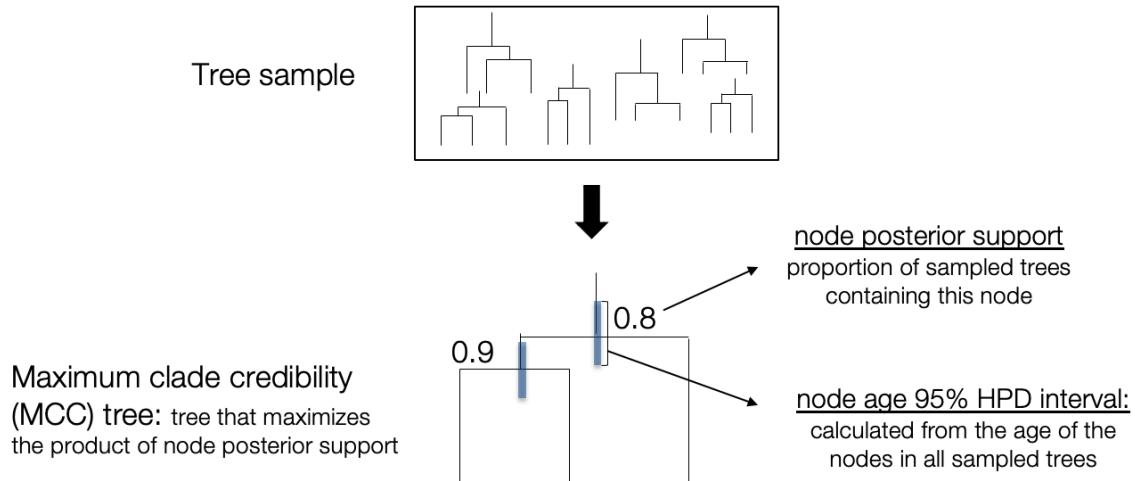


### 19.2.4.2.3. MCC tree

Since we are working in a Bayesian framework, we do not obtain a single phylogenetic tree as with Maximum likelihood, but a large set of trees which should be representative of the posterior distribution. In contrast with monodimensional parameters, a tree distribution cannot be easily summarized with mean or median estimates. Instead, we need to use specific tree-summarizing techniques. One of the most popular is the maximum clade credibility (MCC) tree, which works as follow:

- 1. For any node in any of the sampled trees, compute a **posterior support**: the proportion of trees in the sample which contain the node
- 2. Select the MCC tree: this is the tree in which the product of node posterior supports is the highest
- 3. Calculate node/branch statistics on the MCC tree: typically, the mean/median estimates and HPD interval of node ages are calculated from the full tree sample and annotated on the MCC tree

## 19. Introduction to Phylogenomics



Let's generate an MCC tree from our tree sample. We can do this using the *TreeAnnotator* software, which has both a commandline and graphical interface. Let's use the commandline here and run the following (using a burn-in of 10%):

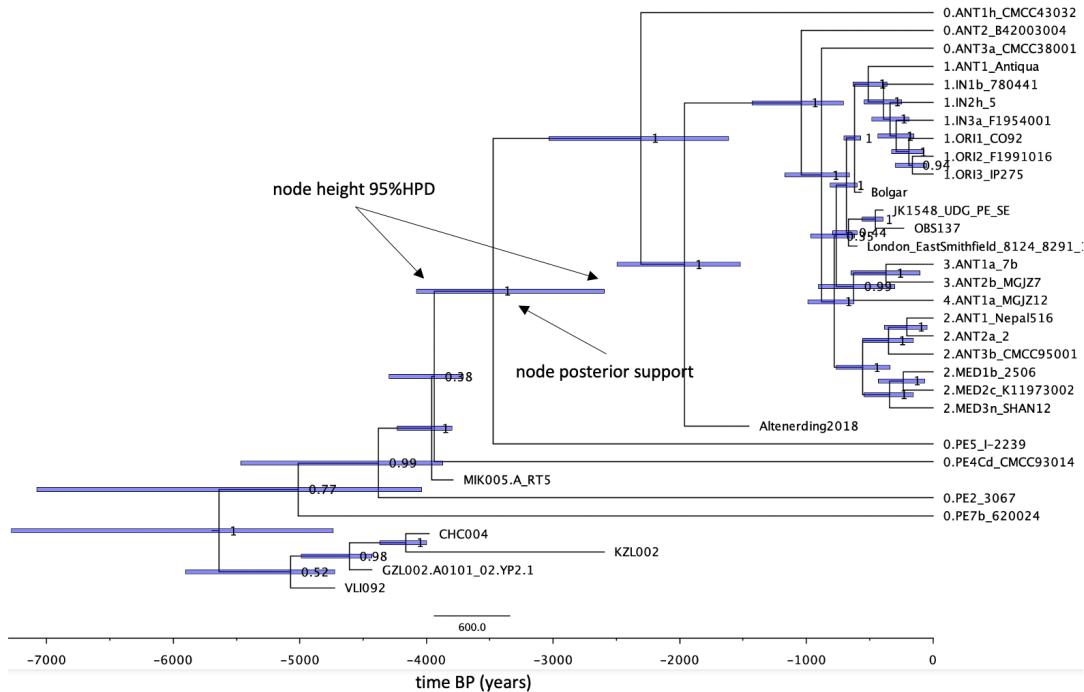
```
treeannotator -burnin 10 intermediateFiles/snpAlignment_without_outgroup.trees snpAlignment
```

Once this is completed, we can open the MCC tree with figtree. Let's then add a few elements to the plot:

- 1. Tick the “Scale Axis” box, unfold the corresponding menu, and select “Reverse axis” (now the timescale is in years BP)
- 2. Tick the “Node Labels” box, unfold the corresponding menu, and select “Display: posterior”. The posterior support of each node is now displayed. Note that the support value is a proportion (1=100%)
- 3. Tick the “Node Bars” box, unfold the corresponding menu, and select “Display: height\_95%\_HPD”. The 95% HPD intervals of node ages are now displayed.

## 19. Introduction to Phylogenomics

**Maximum clade credibility (MCC) tree**



### Questions:

Is the root of the tree consistent with what we found previously?

**i Answer**

Yes! The root is placed between our prehistoric strains and the rest of *Y. pestis* strains. Note that this time we didn't have to use an outgroup because we estimated a time-tree: the root is identified as the oldest node in the tree.

What is your estimate for the age of the most recent common ancestor of all *Y. pestis* strains?

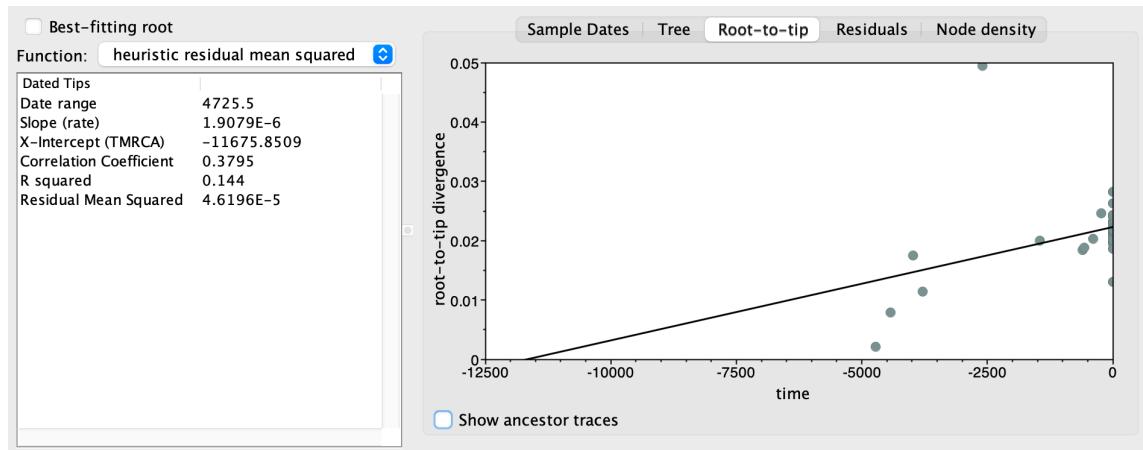
**i Answer**

~5800 years BP (HPD 95%: ~8000-4500 years BP)

### 19.2.4.3. Bonus: Temporal signal assessment

It is a good practice to assess if the genetic sequences that we analyse do indeed behave like molecular clocks before trying to estimate a time tree (i.e. we should have done this before the actual *BEAST2* analysis). A classic way to assess the temporal signal of a dataset is the root-to-tip regression. The rationale of the root-to-tip regression is to verify that the oldest a sequence is, the closer it should be to the root in a (rooted) substitution tree because there was less time for substitution to accumulate. In other words, there should be a correlation between sample age and distance to the root, which we can assess using a linear regression (root-to-tip regression). This can be done using the program *TempEst*:

1. open *TempEst* by typing `tempest &` and load the rooted ML tree that we produced previously (you should have saved it as “ML\_tree\_rooted.tre”)
2. click on “Import Dates” in the “Sample Dates” tab, select the sample\_age.txt file and click “OK”
3. still in the “Sample Dates” tab, change “Since some time in the past” to “Before present” (one might need to extend the *TempEst* window to see the pull down menu)
4. look at the “Root-to-tip tab”: is there a positive correlation between time and root-to-tip divergence as expected under the molecular clock hypothesis?



**Part V.**

**Ancient Metagenomic Resources**

Following the earlier chapters, you should now be familiar with the basic concepts of the practical aspects of the ancient metagenomic project. However, as with all bioinformaticians, we like to automate as much of our work as possible. In this chapter, we will introduce you to some of the dedicated resources to address this, covering both where and how to find ancient metagenomic data, but also how to speed up the analyses introduced earlier through automated pipelines.

## Accessing Ancient Metagenome Data

Finding relevant comparative data for your ancient metagenomic analysis is not trivial. While palaeogenomicists are very good at uploading their raw sequencing data to large sequencing data repositories such as the EBI’s ENA or NCBI’s SRA archives in standardised file formats, these files often have limited metadata. This often makes it difficult for researchers to search for and download relevant published data they wish to use to augment their own analysis.

AncientMetagenomeDir is a community project from the SPAAM community to make ancient metagenomic data more accessible. We curate a list of standardised metadata of all published ancient metagenomic samples and libraries, hosted on GitHub. In this chapter we will go through how to use the AncientMetagenomeDir repository and associated tools to find and download data for your own analyses. We will also discuss important things to consider when publishing your own data to make it more accessible for other researchers.

## Ancient Metagenomic Pipelines

Analyses in the field of ancient DNA are growing, both in terms of the number of samples processed and in the diversity of our research questions and analytical methods. Computational pipelines are a solution to the challenges of big data, helping researchers to perform analyses efficiently and in a reproducible fashion. Today we will introduce nf-core/eager, one of several pipelines designed specifically for the 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

### *Ancient Metagenomic Pipelines*

be ready to strike out into the world of nf-core/eager and build your own analyses from scratch!

# 20. Accessing Ancient Metagenomic Data

## 💡 Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate accessing-ancientmetagenomic-data
```

## 20.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

## 20.2. Introduction

In most bioinformatic projects, we need to include publicly available comparative data to expand or compare our newly generated data with.

Including public data can benefit ancient metagenomic studies in a variety of ways. It can help increase our sample sizes (a common problem when dealing with rare archaeological samples) - thus providing stronger statistical power. Comparison with a range of previously published data of different preservational levels can allow an estimate on the quality of the new samples. When considering solely (re)using public data, we can consider that this can also spawn new ideas, projects, and meta analyses to allow further deeper exploration of ancient metagenomic data (e.g., looking for correlations between various environmental factors and preservation).

Fortunately for us, geneticists and particularly palaeogenomicists have been very good at uploading raw sequencing data to well-established databases [[@Anagnostou2015-mz](#)].

## 20. Accessing Ancient Metagenomic Data

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'.

### 20.3. Finding Ancient Metagenomic Data

FAIR principles were defined by researchers, librarians, and industry in 2016 to improve the quality of data uploads - primarily by making data uploads more 'machine readable'. FAIR standards for:

- Findable
- Accessible
- Interoperable
- Reproducible

When we consider ancient (meta)genomic data, we are pretty close to this. Sequencing data is in most cases accessible (via the public databases like ENA, SRA), interoperable and reproducible because we use field standard formats such as FASTQ or BAM files. However *findable* remains an issue.

This is because the *metadata* about each data file is dispersed over many places, and very often not with the data files themselves.

In this case I am referring to metadata such as: What is the sample's name? How old is it? Where is it from? Which enzymes were used for library construction? What sequencing machine was this library sequenced on?

To find this information about a given data file, you have to search many places (main text, supplementary information, the database itself), for different types of metadata (as authors report different things), and also in different formats (text, tables, figures).

This very heterogenous landscape makes it difficult for machines to index all this information (if at all), and thus means you cannot search for the data you want to use for your own research in online search engines.

## 20.4. AncientMetagenomeDir

This is where the SPAAM community project ‘AncientMetagenomeDir’ comes in [Fellows\_Yates2021-rp]. AncientMetagenomeDir is a resource of lists of metadata of all publishing and publicly available ancient metagenomes and microbial genome-level enriched samples and their associated libraries.

By aggregating and standardising metadata and accession codes of ancient metagenomic samples and libraries, the project aims to make it easier for people to find comparative data for their own projects, appropriately re-analyse libraries, 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 2000 samples and 5000 libraries, spanning 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.) (Figure 20.1).

| project_name | publication_year | publication_doi     | site_name      | latitude | longitude | geo_loc_name | sample_name |
|--------------|------------------|---------------------|----------------|----------|-----------|--------------|-------------|
| Warinner2014 | 2014             | 10.1038/ng.2906     | Dalheim        | 51.565   | 8.84      | Germany      | B61         |
| Warinner2014 | 2014             | 10.1038/ng.2906     | Dalheim        | 51.565   | 8.84      | Germany      | G12         |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Gola Forest    | 7.657    | -10.841   | Sierra Leone | Chimp       |
| Weyrich2017  | 2017             | 10.1038/nature21674 | El Sidrón Cave | 43.386   | -5.328    | Spain        | ElSidron1   |
| Weyrich2017  | 2017             | 10.1038/nature21674 | El Sidrón Cave | 43.386   | -5.329    | Spain        | ElSidron2   |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Spy Cave       | 50.48    | 4.674     | Belgium      | Spy1        |

Figure 20.1.: Example few columns and rows of an AncientMetagenomeDir table, including project name, publication year, site name, latitude, longitude, country and sample name.

Critically, by standardising the recorded all metadata across all publications this makes it much easier for researchers to filter for particular time periods, geographical regions, or sample types of their interest - and then use the also recorded accession numbers to efficiently download the data.

At their core all different AncientMetagenomeDir tables must have at 6 minimum metadata sets at the sample level:

- Publication information (doi)
- Sample name(s)
- Geographic location (e.g. country, coordinates)
- Age
- Sample type (e.g. bone, sediment, etc.)
- Data Archive and accessions

Each table then has additional columns depending on the context (e.g. what time of microbiome is expected for host-associated metagenomes, or species name of the genome that was reconstructed).

The AncientMetagenomeDir project already has 9 major 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.

## 20.5. AMDirT

But how does one explore such a large dataset of tables with thousands of rows? You could upload this into a spreadsheet tool or in a programming language like R, but you would still have to do a lot of manual filtering and parsing of the dataset to make it useful for downstream analyses.

In response to this The SPAAM Community have also developed a companion tool ‘AMDirT’ to facilitate this. Amongst other functionality, AMDirT allows you to load different releases of AncientMetagenomeDir, filter and explore to specific samples or libraries of interest, and then generate download scripts, configuration files for pipelines, and reference BibTeX files for you, both via a command-line (CLI) or graphical user interface (GUI)!

### 20.5.1. Running AMDirT viewer

We will now demonstrate how to use the AMDirT graphical user interface to load a dataset, filter to samples of interest, and download some configuration input files for downstream ancient DNA pipelines.

#### Warning

This tutorial will require a web-browser! Make sure to run on your local laptop/PC or, if on on a server, with X11 forwarding activated.



Figure 20.2.: AMDirT logo: a cog with the SPAAM icon in the middle with the word AMDirT to the side

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

While in the `accessing-ancientmetagenomic-data` conda environment, run the following command to load the GUI into your web-browser. If the browser doesn't automatically load, copy the IP address and paste it in your browser's URL bar.

`AMDirT viewer`

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 (Figure 20.3), 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’ from `[@Kocher2021-vg, @fig-accessingdata-projectfilter]`.

Then scroll to the right, and filter the **geo\_loc\_name** to ‘United Kingdom’ (Figure 20.5).

## 20. Accessing Ancient Metagenomic Data

The screenshot shows the AMDirT viewer tool running in a web browser (localhost:8501). On the left, a sidebar displays the AMDirT version (1.4.6) and a dropdown menu for selecting an AncientMetagenomeDir release, currently set to v23.06.0. Below this is another dropdown for selecting a table, with 'ancientmetagenome-hostassociated' highlighted and selected. A note at the bottom of this sidebar states: 'Only ENA and SRA archives are supported for now'. The main content area shows a table titled 'Select samples to filter' with columns: project\_name, publication\_year, publication\_doi, and site\_name. The table lists 20 entries, mostly from Weyrich et al. (2017), with some from Warinner et al. (2014) and others from Al Khiday et al. (2017). The publication years range from 2014 to 2017, and the sites include Dalheim, Gola Forest, El Sidrón Cave, Spy Cave, Dudka, Stuttgart-Mühlhausen I, and Near Cape Town.

| project_name | publication_year | publication_doi     | site_name              |
|--------------|------------------|---------------------|------------------------|
| Warinner2014 | 2014             | 10.1038/ng.2906     | Dalheim                |
| Warinner2014 | 2014             | 10.1038/ng.2906     | Dalheim                |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Gola Forest            |
| Weyrich2017  | 2017             | 10.1038/nature21674 | El Sidrón Cave         |
| Weyrich2017  | 2017             | 10.1038/nature21674 | El Sidrón Cave         |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Spy Cave               |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Spy Cave               |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Dudka                  |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Dudka                  |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Stuttgart-Mühlhausen I |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Stuttgart-Mühlhausen I |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Stuttgart-Mühlhausen I |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Al Khiday              |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Al Khiday              |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Near Cape Town         |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Near Cape Town         |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Jewbury                |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Jewbury                |
| Weyrich2017  | 2017             | 10.1038/nature21674 | Near Cape Town         |

Figure 20.3.: Main AMDirT Viewer page on load. A toolbar on the left displays the AMDirT version, and dropdown menus for the AncientMetagenomeDir release, table, and downloading tool to use. The rest of the page shows a tabular window containing rows and columns corresponding to sample metadata and rows of samples with metadata such as project name, publication year and site name.

## 20. Accessing Ancient Metagenomic Data

The screenshot shows the AMDirT viewer tool running in a Firefox browser window. The title bar reads "AMDit viewer" and the address bar shows "localhost:8501". The main interface includes a logo, the title "AMDit viewer tool", and a sidebar with version information (Version: 1.4.6) and a dropdown for selecting an AncientMetagenomeDir release (v23.06.0). A dropdown menu for "Select a table" is open, showing options like "ancientsinglegenome-hostassociated" (which is selected), "ancientmetagenome-environmental", "ancientmetagenome-hostassociated", "ancientsinglegenome-hostassociated", and "test". A yellow box at the bottom of the sidebar states: "Only ENA and SRA archives are supported for now". On the right, a table titled "Select samples to filter" displays data from the "ancientsinglegenome-hostassociated" table. The columns are "project\_name", "publication\_doi", and "site\_name". A search bar in the header contains the text "Kocher2021". The table rows include entries such as Schuenemann2013, Schuenemann2018, AndradesValtuena2017, and AndradesValtuena2018, along with their corresponding DOIs and site names.

| project_name         | publication_doi         | site_name                      |
|----------------------|-------------------------|--------------------------------|
| Schuenemann2013      | 10.1126/science.1238286 | Sigtuna                        |
| Schuenemann2013      | 10.1126/science.1238286 | St. Jørgen cemetery, Odense    |
| Schuenemann2013      | 10.1126/science.1238286 | Refshale                       |
| Schuenemann2013      | 10.1126/science.1238286 | St. Mary Magdalene leprosarium |
| Schuenemann2013      | 10.1126/science.1238286 | St. Mary Magdalene leprosarium |
| Schuenemann2018      | 2018                    | Szentend-Kistőke               |
| Schuenemann2018      | 2018                    | Necropolis Vicenne Camp        |
| Schuenemann2018      | 2018                    | Prušánky                       |
| Schuenemann2018      | 2018                    | Great Chesterford              |
| Warinner2014         | 2014                    | Dalheim                        |
| Bos2014              | 2014                    | El Yaral                       |
| Bos2014              | 2014                    | El Algodonal                   |
| Bos2014              | 2014                    | Chiribaya Alta                 |
| AndradesValtuena2017 | 2017                    | Rashevatskiy                   |
| AndradesValtuena2017 | 2017                    | Beli Manastir-Popova zemlja    |
| AndradesValtuena2017 | 2017                    | Giyakarai                      |
| AndradesValtuena2017 | 2017                    | Kunila                         |
| AndradesValtuena2017 | 2017                    | Haunstetten Unterer Talbach    |
| AndradesValtuena2017 | 2017                    | Haunstetten Postillionstrasse  |

Figure 20.4.: The AMDit viewer with a filter menu coming out of the project name column, with just the Kocher 2021 in the search bar.

## 20. Accessing Ancient Metagenomic Data

The screenshot shows the AMDirT viewer tool interface running in a web browser (localhost:8501). On the left, there's a sidebar with configuration options: release (v23.06.0), table selection (ancientsinglegenome-hostassociated), rows per page (50), download method (curl), and a note about supported archives (Only ENA and SRA archives are supported for now). The main area displays a table titled "Select samples to filter" with columns: latitude, longitude, geo\_loc\_name, and sample\_host. The table contains four rows of data from the United Kingdom. A search bar at the top of the filter panel shows "United". Below the search bar, there are checkboxes for "(Select All)" and "United Kingdom", with "United Kingdom" being checked. The table footer shows "1 to 4 of 4" and navigation links.

| latitude | longitude | geo_loc_name   | sample_host |
|----------|-----------|----------------|-------------|
| 52.08    | 0.18      | United Kingdom | Homo sapii  |
| 52.08    | 0.18      | United Kingdom | Homo sapii  |
| 52.26    | 0.061     | United Kingdom | Homo sapii  |
| 52.9     | 0.544     | United Kingdom | Homo sapii  |

Figure 20.5.: The AMDirT viewer with a filter menu coming out of the geo location column, with just the United Kingdom written in the search bar and the entry ticked in the results.

## 20. Accessing Ancient Metagenomic Data

You should be left with 4 rows.

Finally, scroll back to the first column and tick the boxes of these four samples (Figure 20.6).

The screenshot shows a web browser window titled "AMDirT viewer" at "localhost:8501". The interface includes a logo, version information (v23.06.0), and dropdown menus for selecting a release (v23.06.0), a table (ancient single genome-host associated), and a data download method (curl). A note states: "Only ENA and SRA archives are supported for now". The main content area displays a table titled "Select samples to filter" with the following data:

| project_name | publication_year | publication_doi         | site_name |
|--------------|------------------|-------------------------|-----------|
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Hinxton   |
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Hinxton   |
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Oakington |
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Sedgeford |

At the bottom right of the table, there is a footer: "1 to 4 of 4" and navigation icons (< > Page 1 of 1).

Figure 20.6.: The AMDirT viewer with just four rows with samples from Kocher2021 that are located in the United Kingdom being displayed, and the tickboxes next to each one selected

Once you've selected the samples you want, you can press **Validate selection**. You should then see a series loading-spinner, and new a lot of buttons should appear (Figure 20.7)!

You should have four categories of buttons:

- Download AncientMetagenomeDir Library Table
- Download Curl sample download script
- Download <tool/pipeline name> input TSV
- Download Citations as BibText

## 20. Accessing Ancient Metagenomic Data

The screenshot shows the AMDirT viewer tool interface in a web browser. The title bar says "AMDit viewer" and the address bar shows "localhost:8501".

**AMDit viewer tool**  
Version: 1.4.6  
Select an AncientMetagenomeDir release: v23.06.0  
Select a table: ancientsinglegenome-hostassociated  
Number of rows to display: 50  
Data download method: curl  
Only ENA and SRA archives are supported for now

AncientMetagenomeDir release: v23.06.0  
Displayed table: ancientsinglegenome-hostassociated

Select samples to filter

| project_name | publication_year | publication_doi         | site_name |
|--------------|------------------|-------------------------|-----------|
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Hinxton   |
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Hinxton   |
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Oakington |
| Kocher2021   | 2021             | 10.1126/science.abi5658 | Sedgeford |

1 to 4 of 4 | < | > | Page 1 of 1 | >>

Validate selection

4 samples selected

Download AncientMetagenomeDir Library Table | Download Curl sample download script | Download nf-core/eager input TSV | Download nf-core/mag input CSV | Download nf-core/taxpr filer input CSV | Download aMeta input TSV

Citation information could not be resolved for the following DOIs: [redacted]

Figure 20.7.: The AMDit viewer after pressing the ‘validate selection’ button. A range of buttons are displayed at the bottom including a warning message, with the buttons offering download of a range of files such as download scripts, AncientMetagenomeDir library tables and input configuration sheets for a range of ancient DNA bioinformatics pipelines

## 20. Accessing Ancient Metagenomic Data

The first button is to download a table containing all the AncientMetagenomeDir metadata of the selected amples. The second is for generating a download script that will allow you to immediately download all sequencing data of the samples you selected. The third set of buttons generate (partially!) pre-configured input files for use in dedicated ancient DNA pipeline such as nf-core/eager [@Fellows\_Yates2021-jl], PALAEOMIX [@Schubert2014-ps], and/or and aMeta [@Pochon2022-hj]. Finally, the fourth 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. In this case the reference metadata for that particular publication isn't publicly available so there is a warning.

It's important to note you are not necessarily restricted to Curl for downloading the data. AMDirT aims to add support for whatever tools or pipelines requested by the community. For example, an already supported downloading tool alternative is the nf-core/fetchNGS pipeline. You can select these using the drop-down menus on the left hand-side.

Press the AncientMetagenomeDir library, curl download, and nf-core/eager buttons to download two `.tsv` files, one `*.sh` file and one '`*bib`' file. 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.

### 20.5.2. Inspecting AMDirT viewer 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_bibliography.bib
AncientMetagenomeDir_curl_download_script.sh
AncientMetagenomeDir_filtered_libraries.csv
AncientMetagenomeDir_nf_core_eager_input_table.tsv
```

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.

## 20. Accessing Ancient Metagenomic Data

```
$ cat AncientMetagenomeDir_curl_download_script.sh
#!/usr/bin/env bash
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/009/ERR6053619/ERR6053619.fastq.gz -o ERR6053619.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/008/ERR6053618/ERR6053618.fastq.gz -o ERR6053618.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/005/ERR6053675/ERR6053675.fastq.gz -o ERR6053675.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/006/ERR6053686/ERR6053686.fastq.gz -o ERR6053686.fastq.gz
```

By providing this script for you, AMDirT facilitates fast download of files of interest by replacing the one-by-one download commands for each sample with a *single* command!

```
$ bash AncientMetagenomeDir_curl_download_script.sh
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/009/ERR6053619/ERR6053619.fastq.gz -o ERR6053619.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/008/ERR6053618/ERR6053618.fastq.gz -o ERR6053618.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/005/ERR6053675/ERR6053675.fastq.gz -o ERR6053675.fastq.gz
curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR605/006/ERR6053686/ERR6053686.fastq.gz -o ERR6053686.fastq.gz
```

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_nf_core_eager_input_table.tsv
Sample_Name Library_ID Lane Colour_Chemistry SeqType Organism Strandedness UDG
I0157 ERR6053618 0 4 SE Homo sapiens double unknown ERX5692504_ERR6053618.fastq
I0161 ERR6053619 0 4 SE Homo sapiens double unknown ERX5692505_ERR6053619.fastq
OAI017 ERR6053675 0 4 SE Homo sapiens double half ERX5692561_ERR6053675.fastq
SED009 ERR6053686 0 4 SE Homo sapiens double half ERX5692572_ERR6053686.fastq
```

Finally, we can look into the BibTeX citations file (\*bib) which will provide you with the citation information of all the downloaded data and AncientMetagenomeDir itself.

### Warning

The contents of this file is reliant on indexing of publications on CrossRef. In some cases not all citations will be present (as per the warning), so this should be double checked!

```
$ cat AncientMetagenomeDir_bibliography.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 J. V{\~n}e
Becky Cribdon and Irina M. Velsko and Maxime Borry and Miriam J. Bravo-Lopez and Antonia
and Eleanor J. Green and Shreya L. Ramachandran and Peter D. Heintzman and Maria A. Spy
H\"ubner and Abigail S. Gancz and Jessica Hider and Aurora F. Allshouse and Valentina Zar
title = {Community-curated and standardised metadata of published ancient metagenomic s
journal = {Scientific Data}
}
```

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

### 20.5.3. AMDirT convert

If you're less of a GUI person and consider yourself a command-line wizard, you can also use the `AMDirT convert` command instead of the GUI version.

In this case you must supply your own filtered AncientMetagenomeDir samples table, and use command line options to specify which files to generate.

For example, lets say you used R to make the following filtered file (basically the same Kocher et al. 2021 samples from the UK, as filtered in the GUI part of this tutorial), you would supply this to `AMDirT convert` like so:

```
$ AMDirT convert -o . --bibliography --librarymetadata --curl --eager ancientsinglegenome-h
```

When running `ls` you should see the same resulting files as before with the GUI!

```
$ ls
AncientMetagenomeDir_bibliography.bib
AncientMetagenomeDir_curl_download_script.sh
AncientMetagenomeDir_filtered_libraries.tsv
```

## 20. Accessing Ancient Metagenomic Data

```
AncientMetagenomeDir_nf_core_eager_input_table.tsv  
ancientsinglegenome-hostassociated_samples_Kocher2021_UnitedKingdom.tsv
```

In this case you say where to save the output files, then use different flags to specify which scripts, bib, and configuration files you want, and finally your filtered AncientMetagenomeDir TSV file and which table it's derived from.

## 20.6. 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 and GitHub.

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 AncientMetagenomeDir publication, and read more about the workflow on the AncientMetagenomeDir wiki

As AncientMetagenomeDir is on GitHub, the means we can also use this repository to try out our Git skills we learnt in the chapter Introduction to Git(Hub)!

Your task is to complete the following steps however with `git` terms removed (indicated by quotes). You should be able to complete the steps and also note down what the *correct* Git terminology:

1. Make a ‘copy’ the jfy133/AncientMetagenomeDir repository to your account
  - We are forking a personal fork of the main repository to ensure you don’t accidentally edit the main dataset!
2. ‘Download’ the ‘copied’ repo to your local machine
3. ‘Change’ to the `dev` branch
4. Modify the file `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 the changes to the original jfy133/AncientMetagenomeDir repo
  - Make sure to put ‘Summer school’ in the title of the ‘Request’

## 20. Accessing Ancient Metagenomic Data

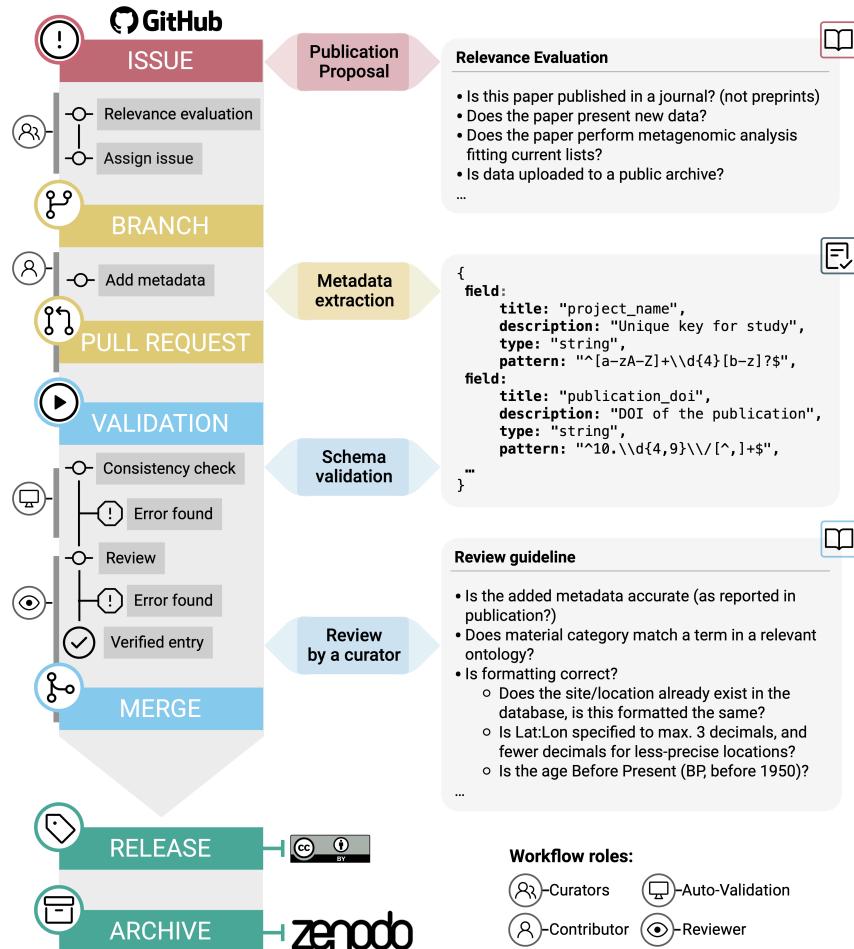


Figure 20.8.: Overview of the AncientMetagenomeDir contribution workflow. Potential publications are added as a GitHub issue where they undergo relevance evaluation. Once approved, a contributor makes a branch on the AncientMetagenomeDir GitHub repository, adds the new lines of metadata for the relevant samples and libraries, and once ready opens a pull request against the main repository. The pull request undergoes an automated consistent check against a schema and then undergoes a human-based peer review for accuracy against AncientMetagenomeDir guidelines. Once approved, the pull request is merged, and periodically the dataset is released on GitHub and automatically archived on Zenodo.

💡 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`
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

## 20.7. 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!

## 20.8. Resources

- AncientMetagenomeDir paper
- AncientMetagenomeDir repository
- AMDirT web server
- AMDirT documentation
- AMDirT repository

## 20.9. References

# 21. Ancient Metagenomic Pipelines



Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the `yml` file in the following link (right click and save as to download), and once created, activate the environment with:

```
conda activate ancient-metagenomic-pipelines
```

## 21.1. Lecture

Lecture slides and video from the 2022 edition of the summer school.

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

## 21.2. Introduction

A **pipeline** is a series of linked computational steps, where the output of one process becomes the input of the next. Pipelines are critical for managing the huge quantities of data that are now being generated regularly as part of ancient DNA analyses. In this chapter we will go through three dedicated ancient DNA pipelines - all with some (or all!) functionality geared to ancient metagenomics - to show you how you can speed up the more routine aspects of the basic analyses we've learnt about earlier in this text book through workflow automation.

We will introduce:

- **nf-core/eager** - a generalised aDNA 'workhorse' pipeline that can do both ancient genomics and (basic) metagenomics [[@Fellows\\_Yates2021-jl](#)]
- **aMeta** - a pipeline for resource efficient and accurate ancient microbial detection and authentication [[@Pochon2022-hj](#)]

## 21. Ancient Metagenomic Pipelines

- **nf-core/mag** - a *de novo* metagenomics assembly pipeline [@Krakau2022-we] that includes a dedicated ancient DNA mode for damage correction and validation.

Keep in mind that there are many other pipelines that exist, and picking which one often comes down to personal preference, such as which functionality they support, which language they are written in, and whether their computational requirements can fit in your available resources.

Other examples of other ancient DNA genomic pipelines include Paleomix [@Schubert2014-ps], and Mapache [@Neuenschwander2023-aj], and for ancient metagenomics: metaBit [@Louvel2016-jo] and HAYSTAC [@Dimopoulos2022-tp].

### 21.3. Workflow managers

All the pipelines introduced in this chapter utilise *workflow managers*. These are software that allows users to ‘chain’ together the inputs and outputs of distinct ‘atomic’ steps of a bioinformatics analysis - e.g. separate terminal commands of different bioinformatic tools, so that ‘you don’t have to’. You have already seen very basic workflows or ‘pipelines’ when using the bash ‘pipe’ (!) in the Introduction to the Command Line chapter, where each row of text the output of one command was ‘streamed’ into the next command.

However in the case of bioinformatics, we are often dealing with non-row based text files, meaning that ‘classic’ command line pipelines don’t really work. Instead this is where bioinformatic *workflow managers* come in: they handle the passing of files from one tool to the next but in a *reproducible* manager.

Modern computational bioinformatic workflow managers focus on a few main concepts. To summarise @Wratten2021-es, these areas are: data provenance, portability, scalability, re-entrancy - all together which contribute to ensuring reproducibility of bioinformatic analyses. **Data provenance** refers to the ability to track and visualise where each file goes and gets processed, as well as *metadata* about each file and process (e.g., What version of a tool was used? What parameters were used in that step? How much computing resources were used). **Portability** follows from data provenance where it’s not just can the entire execution of the pipeline be reconstructed - but can it also be run *with the same results* on a different machine? This is important to ensure that you can install and test the pipeline on your laptop, but when you then need to do *heavy* computation using real data, that it will still be able to execute on a high-performance computing cluster (HPC) or on the cloud - both that have very different configurations. This is normally achieved through the use of reusable software environments such as conda or container engines such as docker, and tight integration with HPC schedulers such as SLURM. As mentioned earlier, not having to run each command manually can be a great speed up to your analysis. However this

## 21. Ancient Metagenomic Pipelines

needs to be able to be **Scalable** that it the workflow is still efficient regardless whether you're running with one or ten thousands samples - modern workflow managers perform resource requirement optimisation and scheduling to ensure that all steps of the pipeline will be executed in the most resource efficient manner so it completes as fast as possible - but regardless of the number of input data. Finally, as workflows get bigger and longer, **re-entrancy** has become more important, i.e., the ability to re-start a pipeline run that got stuck halfway through due to an error.

All workflow managers have different ways of implementing the concepts above, and these can be very simple (e.g. Makefiles) to very powerful and abstract (e.g. Workflow Description Language). In this chapter we will use pipelines that use two popular workflow managers in bioinformatics, Nextflow and Snakemake.

This chapter will not cover how to write your *own* workflow, as this would require a whole other textbook. However it is recommended to learn and use workflow managers when carrying out repetitive or routine bioinformatic analysis (Nextflow and snakemake being two popular ones in bioinformatics). Use of workflow managers can help make your work more efficiently (as you only run one command, rather than each step separately), but also more *reproducible* by reducing the risk of user error when executing each step: the computer will do exactly what you tell it, and if you don't change anything, will do the exact same thing every time. If you're interested in writing your own workflows using workflow managers, many training and tutorials exist on the internet (e.g., for Nextflow there is the official training or from software carpentries, or the official training for snakemake).

### 21.4. What is nf-core/eager?

nf-core/eager is a computational pipeline specifically designed for preprocessing and analysis of ancient DNA data (Figure 21.1). It is a reimplementation of the previously published EAGER (Efficient Ancient Genome Reconstruction) pipeline [@Peltzer2016-ov] written in the workflow manager Nextflow. In addition to reimplementing the original genome mapping and variant calling pipeline, in a more reproducible and portable manner the pipeline also included additional new functionality particularly for researchers interested in microbial sciences, namely a dedicated genotyper and consensus caller designed for low coverage genomes, the ability to get breadth and depth coverage statistics for particular genomic features (e.g. virulence genes), but also automated metagenomic screening and authentication of the off-target reads from mapping (e.g. against the host reference genome).

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](#).

## 21. Ancient Metagenomic Pipelines

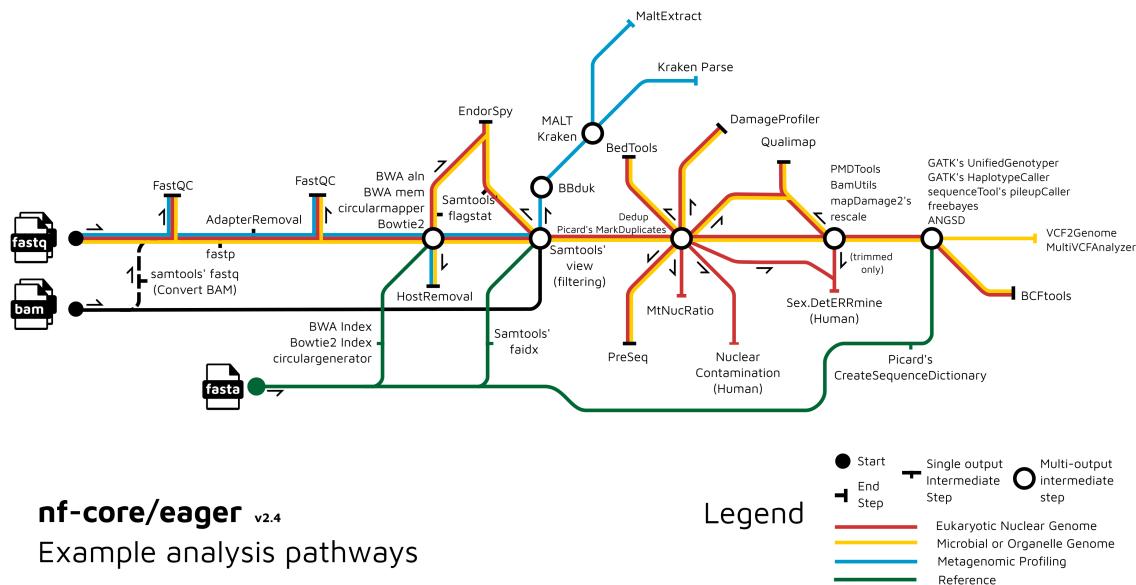


Figure 21.1.: nf-core/eager workflow summary in the form of a ‘metro map’ style diagram. FASTQ or BAM input for red and yellow lines (representing eukaryotic and prokaryotic workflows) goes through FastQC, Adapter Removal, alignment to a reference FASTA input with a range of alignments, before going through BAM filtering, deduplication, *in silico* damage removal, and variant calling. Multiple statistics steps come out of the deduplication ‘station’. The blue line represents the metagenomic workflow, where offtarget reads come out of the BAM filtering ‘station’, and goes through complexity filtering with BBduk, MALT or Kraken2, and optionally into MaltExtract for MALT output.

## 21. Ancient Metagenomic Pipelines

Briefly, nf-core/eager takes at a minimum standard input file types that are shared across the genomics field, i.e., raw FASTQ files or aligned reads in bam format, and a reference fasta. nf-core/eager performs preprocessing of this raw data, including adapter clipping, read merging, and quality control of adapter-trimmed data. nf-core/eager then carries mapping using a variety of field-standard shot-read alignment tools with default parameters adapted for short and damaged aDNA sequences. The off-target reads from host DNA mapping can then go into metagenomic classification and authentication (in the case of MALT). After genomic mapping, BAM files go through deduplication of PCR duplicates, damage profiling and removal, and finally variant calling. A myriad of additional statistics can be generated depending on the users preference. Finally, nf-core eager uses MultiQC to create an integrated html report that summarizes the output/results from each of the pipeline steps.

### 21.4.1. Running nf-core/eager

For the practical portion of this chapter, 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 . We will use nf-core/eager to perform a typical microbial *genomic* analysis, i.e., reconstruction of an ancient genome to generate variant calls that can be used for generating phylogenomic trees and other evolutionary analysis, and gene feature coverage statistics to allow insight into the functional aspects of the genome.

These four libraries come from two ancient individuals, GLZ002 and KZL002. GLZ002 comes from the Neolithic Siberian site of Glazkovskoe predmestie [@Yu2020-zw] and was radiocarbon dated to 3081-2913 calBCE. KZL002 is an Iron Age individual from Kazakhstan, radiocarbon dated to 2736-2457 calBCE [@Andrades\_Valtuena2022-tq]. Both individuals were originally analysed for human population genetic analysis, but when undergoing metagenomic screening of the off-target reads, both set of authors identified reads from *Yersinia pestis* from these individuals - the bacterium that causes plague. Subsequently the 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. Generate statistics on gene features (e.g. virulence factors)
6. Merge data by sample and perform genotyping on the combined dataset
7. Review quality control data to evaluate the success of the previous steps

## 21. Ancient Metagenomic Pipelines

Let's get started!

First, activate the conda environment that we downloaded during setup:

```
conda activate ancient-metagenomic-pipelines
```

And change into our practical session directory:

```
cd /<path>/<to>/ancient-metagenomic-pipelines
```

 Warning

nf-core/eager v2 requires an older version of Nextflow - if installing manually, ensure you do not have Nextflow version any greater than v22.10.6!

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
```

Next we can re-visit AMDirT (see Accessing Ancient Metagenomic Data) to download a pre-prepared configuration file for nf-core/eager

1. Load AMDirT ([AMDirT viewer](#)), and select the latest release and the ‘ancientsinglegenome-hostassociated’ table
2. Filter the `sample_name` column to just show KZL002 and GLZ002, and select these two rows
  - Press the burger icon on the column
  - Press the filter tab and deselect everything
  - Search GLZ002 and select in filter menu
  - Search KLZ002 and select in filter menu
  - Close filter menu and select the two rows
3. Press the Validate Selection button
4. Press the ‘Download Curl sample download script’, ‘Download nf-core/eager input TSV’, and ‘Download Citations as BibTex’ buttons
5. Move the follows into `eager/` of this tutorial’s directory

## 21. Ancient Metagenomic Pipelines

### 💡 Tip

We won't actually use the BibTex citations file for anything in this tutorial, but it is good habit to *always* to record and save all citations of any software or data you use!

To download the FASTQ files

1. Move into the `eager/` directory
2. Run `bash AncientMetagenomeDir_curl_download_script.sh` to download the files (this may take ~3 minutes)

Next we now inspect the AMDirT generated input TSV file for nf-core/eager!

```
cat AncientMetagenomeDir_nf_core_eager_input_table.tsv
```

| Sample_Name | Library_ID | Lane | Colour_Chemistry | SeqType | Organism     | Strandedness | UDG  |                       |
|-------------|------------|------|------------------|---------|--------------|--------------|------|-----------------------|
| GLZ002      | ERR4093961 | 0    | 4                | PE      | Homo sapiens | double       | half | ERR4093961_1.fastq.gz |
| GLZ002      | ERR4093962 | 0    | 4                | PE      | Homo sapiens | double       | full | ERR4093962_1.fastq.gz |
| KZL002      | ERR8958768 | 0    | 4                | PE      | Homo sapiens | double       | half | ERR8958768_1.fastq.gz |
| KZL002      | ERR8958769 | 0    | 4                | PE      | Homo sapiens | double       | half | ERR8958769_1.fastq.gz |

Here we see 10 columns, all pre-filled. The first two columns correspond to sample/library IDs that will be used for data provenance and grouping. When you have sequencing multiple lanes you can speed up preprocessing these independently before merging, so lane can specify this (although not used in this case as we have independent libraries per sample). You can indicate the colour chemistry to indicate whether your data requires additional preprocessing to remove poly-G tails, and then also strandedness and UDG damage treatment status of the libraries if you require further damage manipulation. Finally you provide paths to the FASTQ files or BAM files

Other than the raw FASTQ files, we will need a reference genome and annotation coordinates of genes present on the genome. In this case you will use a *Yersinia pestis* (accession: GCF\_001293415.1) reference genome (.fna) and gene feature file (.gff) from NCBI Genome. This is placed in the `reference/` directory.

### ℹ️ Self guided: reference download and preparation

To download the required reference genome and annotation file run the following command to download from the NCBI Genome database.

## 21. Ancient Metagenomic Pipelines

```
## Download from NCBI
curl -OJX GET "https://api.ncbi.nlm.nih.gov/datasets/v2alpha/genome/accession/GCF_001293415.1_ASM129341v1_genomic.gff"
unzip *.zip
mv ncbi_dataset/data/GCF_001293415.1/* .

## We have to sort the gff file to make it eager compatible
gffread genomic.gff GCF_001293415.1_ASM129341v1_genomic.gff
```

With all of these, we can run the pipeline!

First lets enter a screen session

```
screen -R eager
conda activate ancient-metagenomic-pipelines
```

Now we can construct an eager command from within the `data/` directory so that it looks like this:

```
nextflow run nf-core/eager -r 2.4.7 \
-profile docker \
--fasta ../reference/GCF_001293415.1_ASM129341v1_genomic.fna \
--input AncientMetagenomeDir_nf_core_eager_input_table.tsv \
--anno_file ../reference/GCF_001293415.1_ASM129341v1_genomic.gff \
--run_bam_filtering --bam_unmapped_type discard \
--skip_preset \
--run_genotyping --genotyping_tool ug --gatk_ug_out_mode EMIT_ALL_SITES \
--run_bcftools_stats --run_bedtools_coverage
```

### Tip

We don't normally recommend running analyses in the directory your data is in! It is better to keep data and analysis results in separate directories. Here we are just running eager alongside the data for convenience (i.e., you don't have to modify the downloaded input TSV)

So what is this command doing? The different parameters do the following:

1. Tell nextflow to run the nf-core/eager pipeline with version 2.4.7
2. Specify which computing and software environment to use with `-profile`

## 21. Ancient Metagenomic Pipelines

- In this case we are running locally so we don't specify a computing environment (such as a configuration for an institutional HPC)
  - We use `docker` as our container engine, which downloads all the software and specific versions needed for nf-core/eager in immutable 'containers', to ensure nothing gets broken and is as nf-core/eager expects
3. Provide the various paths to the input files (TSV with paths to FASTQ files, a reference fasta, and the reference fasta's annotations)
  4. Activate the various of the steps of the pipeline you're interested in
    - We turn off preseq (e.g. when you know you can't sequence more)
    - We want to turn on BAM filtering, and specify to generate unmapped reads in FASTQ file (so you could check off target reads e.g. for other pathogens)
    - You turn on genotyping using GATK UnifiedGenotyper (preferred over HaplotypeCaller due to compatibility with that method to low-coverage data)
    - We turn on variant statistics (from GATK) using bcftools, and coverage statistics of gene features using bedtools

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

And now we wait...



### Tip

As a reminder, to detach from the screen session type `ctrl + a` then `d`. To reattach `screen -r eager`

Specifically for ancient (meta)genomic data, the following parameters and options may be useful to consider when running your own data:

- **--mapper circularmapper:**
  - This aligner is a variant of `bwa aln` that allows even mapping across the end of linear references of circular genomes
  - Allows reads spanning the start/end of the sequence to be correctly placed, and this provides better coverage estimates across the entire genome
- **--hostremoval\_input\_fastq:**
  - Allows re-generation of input FASTQ files but with any reads aligned to the reference genome removed
  - Can be useful when dealing with modern or ancient data where there are ethical restrictions on the publication of host DNA
  - The output can be used as 'raw data' upload to public data repositories

## 21. Ancient Metagenomic Pipelines

- `--run_bam_filtering --bam_unmapped_type:`
  - A pre-requisite for performing the metagenomic analysis steps of nf-core/eager
  - Generates FASTQ files off the unmapped reads present in the reference mapping BAM file
- `--run_bedtools_coverage --anno_file <path>/<to>/<genefeature>.bed`
  - Turns on calculating depth/breadth of annotations in the provided bed or GFF file, useful for generating e.g. virulence gene presence/absence plots
- `--run_genotyping --genotyping_tool ug --gatk_ug_out_mode EMIT_ALL_SITES`
  - Turns on genotyping with GATK UnifiedGenotyper
  - A pre-requisite for running MultiVCFAnalyzer for consensus sequencing creation
  - It's recommend to use either GATK UnifiedGenotyper or freeBayes (non-MultiVCFAnalyzer compatible!) for low-coverage data
  - GATK HaplotypeCaller is *not* recommended for low coverage data as it performs local *de novo* assembly around possible variant sites, and this will fail with low-coverage short-read data
- `--run_multivcfanalyzer --write_allele_frequencies`
  - Turns on SNP table and FASTA consensus sequence generation with MultiVCFAnalyzer (see pre-requisites above)
  - By providing `--write_allele_frequencies`, the SNP table will also provide the percentage of reads at that position supporting the call. This can help you evaluate the level of cross-mapping from related (e.g. contaminating environmental) species and may question the reliability of any resulting downstream analyses.
- `--metagenomic_complexity_filtering`
  - An additional preprocessing step of raw reads before going into metagenomic screening to remove low-complexity reads (e.g. mono- or di-nucleotide repeats)
  - Removing these will speed up and lower computational requirements during classification, and will not bias profiles as these sequences provide no taxon-specific information (i.e., can be aligned against thousands to millions of genomes)
- `--run_metagenomic_screening`
  - Turns on metagenomic screening with either MALT or Kraken2
  - If running with MALT, can supply `--run_malteextract` to get authentication statistics and plots (damage, read lengths etc.) for evaluation.

### 21.4.2. Top Tips for nf-core/eager success

#### 1. Screen sessions

Depending on your input data, infrastructure, and analyses, running nf-core/eager can take hours or even days. To avoid crashed due to loss of power or network connectivity, try running nf-core/eager in a `screen` or `tmux` session:

```
screen -R eager
```

#### 2. Multiple ways to supply input data

In this tutorial, a tsv file to specify our input data files and formats. This is a powerful approach that allows nf-core eager to intelligently apply analyses to certain files only (e.g. merging for paired-end but not single-end libraries). However 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 2.4.7 --profile docker --fasta ../reference/GCF_001293415  
--input "data/*_{1,2}.fastq.gz" <...> \  
--udg_type half
```

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 researchers who might be less comfortable with the command line, check out the nf-core/eager launch GUI! The GUI also provides a full list of all pipeline options with short explanations for those interested in learning more about what the pipeline can do.

#### 5. When something fails, all is not lost!

When individual jobs fail, nf-core/eager will try to automatically resubmit that job with increased memory and CPUs (up to two times per job). When the whole pipeline crashes, you can save time and computational resources by resubmitting with the `-resume` flag. nf-core/eager will retrieve cached results from previous steps as long as the input is the same.

## 21. Ancient Metagenomic Pipelines

### 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.

#### 21.4.3. nf-core/eager output

##### 21.4.3.1. Results files

In the `results/` directory of your nf-core/eager run, you will find a range of directories that will contain output files and tool log of all the steps of the pipeline. nf-core/eager tries to only save the ‘useful’ files.

Everyday useful files for ancient (microbial) (meta)genomics typically are in folders such as:

- `deduplication/` or `merged_bams`
  - These contain the most basic BAM files you would want to use for downstream analyses (and used in the rest of the genomic workflow of the pipeline)
  - They contain deduplicated BAM files (i.e., with PCR artefacts removed)
  - `merged_bams/`
    - \* This directory will contain BAMs where multiple libraries of one sample have been merged into one final BAM file, when these have been supplied
- `damage_rescaling/` or `trimmed bam/`
  - These contain the output BAM files from *in silico* damage removal, if you have turned on e.g. BAM trimming or damage rescaling
  - If you have multiple libraries of one sample, the final BAMs you want will be in `merged_bams/`
- `genotyping/`
  - This directory will contain your final VCF files from the genotyping step, e.g. from the GATK or freeBayes tools
- `MultiVCFAnalyzer/`
  - This will contain consensus sequences and SNP tables from `MultiVCFAnalyzer`, which also allows generation of ‘multi-allelic’ SNP position statistics (useful for the evaluation of cross-mapping from contaminants or relatives)
- `bedtools/`

## 21. Ancient Metagenomic Pipelines

- This directory will contain the depth and breadth statistics of genomic features if a `gff` or `bed` file has been provided to the pipeline
- The files can be used to generate gene heatmaps, that can be used to visualise a comparative presence/absence of virulence factor across genomes (e.g. for microbial pathogens)
- `metagenomic_classification` or `malt_extract`
  - This directory contains the output RMA6 files from MALT, the profiles and taxon count tables from Kraken2, or the aDNA authentication output statistics from maltExtract.

Most other folders contain either intermediate files that are only useful for technical evaluation in the case of problems, or statistics files that are otherwise summarised in the run report.

So, before you delve into these folders, it's normally a good idea to do a 'quality check' of the pipeline run. You can do this using the interactive MultiQC pipeline run report.

### 21.4.3.2. MultiQC General Statistics

```
cd multiqc/
```

 Note

If you're impatient, and your nf-core/eager run hasn't finished yet, you can cancel the run with `ctrl + c` (possibly a couple of times), then you can open a premade file in `ancient-metagenomic-pipelines/multiqc_report.html`

In here you should see a bunch of files, but you should open the `multiqc_report.html` file in your browser. You can either do this via the commandline (e.g. for firefox `firefox multiqc_report.html`) or navigate to the file using your file browser and double clicking on the HTML file.

Once opened you will see a table, and below it many figures and other tables (Figure 21.2). All of these statistics can help you evaluate the quality of your data, pipeline run, and also possibly some initial results!

Typically you will look at the General Statistics table to get a rough overview of the pipeline run. If you hover your cursor over the column headers, you can see which tool the column's numbers came from, however generally the columns go in order of left to right,

## 21. Ancient Metagenomic Pipelines

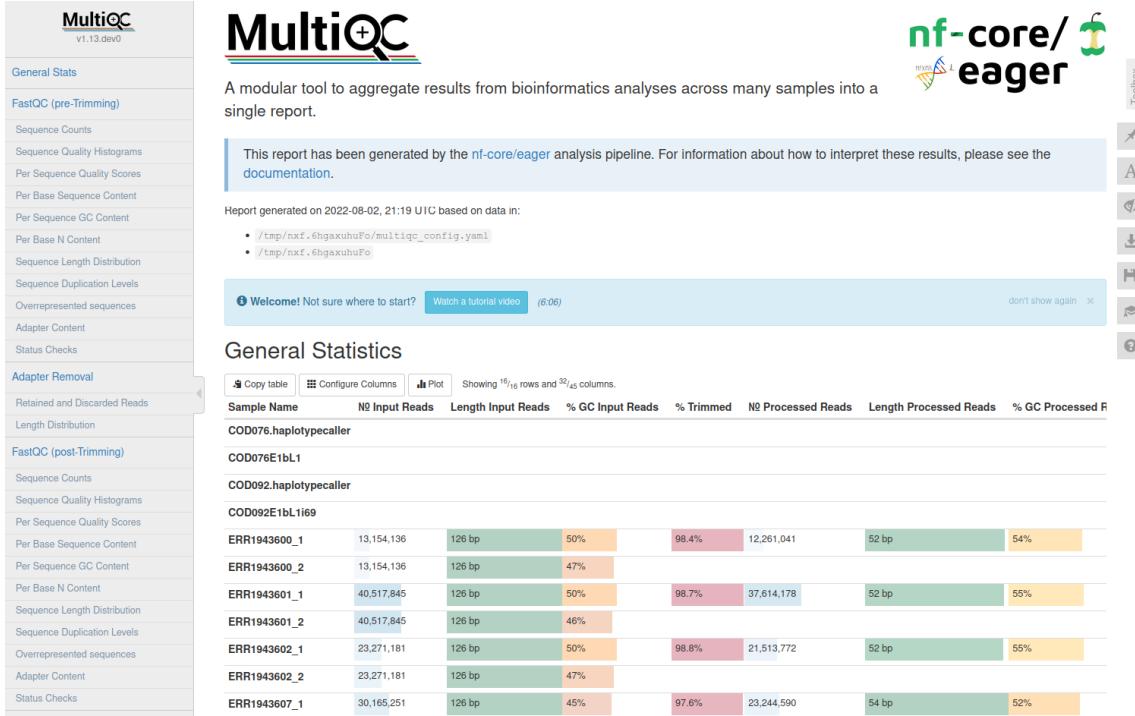


Figure 21.2.: Screenshot of initial view of a MultiQC report. The left hand sidebar provides links to various sections of the report, most of them containing summary images of the outputs of all the different tools. On the main panel you see the MultiQC and nf-core/eager logos, some descriptive information about how the report was generated, and finally the top of the ‘General Statistics’ tables, that has columns such as Sample Name, Nr. Input reads, length of input reads, %trimmed, etc. Each numeric column contains a coloured barchart to help readers to quickly evaluate the number of a given sample against all others in the table

## 21. Ancient Metagenomic Pipelines

where the left most columns are from earlier in the pipeline run (e.g. removing adapters), to variant calling statistics (e.g. number of variants called). You can also configure which columns to display using the **Configure columns** button. It's important to note that in MultiQC tables, you may have duplicate rows from the same library or sample. This is due to MultiQC trying to squish in as many statistics from as many steps of the pipeline as possible (for example, statistics on each of a pair of FASTQ files, and then statistics on the single merged and mapped BAM file), so you should play around with the column configuration to help you visualise the best way to initially evaluate your data.

The bulk of the MultiQC report is made up of per-tool summary plots (e.g., barcharts, linecharts etc). Most of these will be interactive, i.e. you can hover over lines and bars to get more specific numbers of each plot. However the visualisations are aimed at helping you identify possible outliers that may represent failed samples or libraries.

Evaluating how good the data is and how well the run went will vary depending on the dataset and the options selected. However the nf-core/eager tutorials have a good overview of questions you can ask from your MultiQC report to see whether your data is good or not. We shamelessly copy these questions here (as the overlap authors of both the the nf-core/eager documentation and this text book is rather high).

Once completed, you can try going through the MultiQC report the command you executed above, and compare against the questions below. Keep in mind you have a sample  $N$  of two, so many of the questions in regards to identifying ‘outliers’ may be difficult.

The following questions are by no means comprehensive, but rather represent the ‘typical’ questions the nf-core/eager developers asked of their own data and reports. However they can act as a good framework for thinking critically about your own results.

### 21.4.3.2.1. General Stats Table

- Do I see the expected number of raw sequencing reads (summed across each set of FASTQ files per library) that was requested for sequencing?
- Does the percentage of trimmed reads look normal for aDNA, and do lengths after trimming look short as expected of adNA?
- Does the Endogenous DNA (%) columns look reasonable (high enough to indicate you have received enough coverage for downstream, and/or do you lose an unusually high reads after filtering )
- Does ClusterFactor or ‘% Dups’ look high (e.g.  $>2$  or  $>10\%$  respectively - high values suggesting over-amplified or badly preserved samples i.e. low complexity; note that genome-enrichment libraries may by their nature look higher).
- Do you see an increased frequency of C>Ts on the 5' end of molecules in the mapped reads?

## *21. Ancient Metagenomic Pipelines*

- Do median read lengths look relatively low (normally  $\leq 100$  bp) indicating typically fragmented aDNA?
- Does the % coverage decrease relatively gradually at each depth coverage, and does not drop extremely drastically?
- Does the Median coverage and percent  $>3x$  (or whatever you set) show sufficient coverage for reliable SNP calls and that a good proportion of the genome is covered indicating you have the right reference genome?
- Do you see a high proportion of % Hets, indicating many multi-allelic sites (and possibly presence of cross-mapping from other species, that may lead to false positive or less confident SNP calls)?

### **21.4.3.2.2. FastQC (pre-AdapterRemoval)**

- Do I see any very early drop off of sequence quality scores suggesting problematic sequencing run?
- Do I see outlier GC content distributions?
- Do I see high sequence duplication levels?

### **21.4.3.2.3. AdapterRemoval**

- Do I see high numbers of singletons or discarded read pairs?

### **21.4.3.2.4. FastQC (post-AdapterRemoval)**

- Do I see improved sequence quality scores along the length of reads?
- Do I see reduced adapter content levels?

### **21.4.3.2.5. Samtools Flagstat (pre/post Filter)**

- Do I see outliers, e.g. with unusually low levels of mapped reads, (indicative of badly preserved samples) that require downstream closer assessment?

### **21.4.3.2.6. DeDup/Picard MarkDuplicates**

- Do I see large numbers of duplicates being removed, possibly indicating over-amplified or badly preserved samples?

#### 21.4.3.2.7. MALT (metagenomics only)

- Do I have a reasonable level of mappability?
  - Somewhere between 10-30% can be pretty normal for aDNA, whereas e.g. <1% requires careful manual assessment
- Do I have a reasonable taxonomic assignment success?
  - You hope to have a large number of the mapped reads (from the mappability plot) that also have taxonomic assignment.

#### 21.4.3.2.8. PreSeq (genomics only)

- Do I see a large drop off of a sample's curve away from the theoretical complexity? If so, this may indicate it's not worth performing deeper sequencing as you will get few unique reads (vs. duplicates that are not any more informative than the reads you've already sequenced)

#### 21.4.3.2.9. DamageProfiler (genomics only)

- Do I see evidence of damage on the microbial DNA (i.e. a % C>T of more than ~5% in the first few nucleotide positions?) ? If not, possibly your mapped reads are deriving from modern contamination.

#### 21.4.3.2.10. QualiMap (genomics only)

- Do you see a peak of coverage (X) at a good level, e.g.  $\geq 3x$ , indicating sufficient coverage for reliable SNP calls?

#### 21.4.3.2.11. MultiVCFAnalyzer (genomics only)

- Do I have a good number of called SNPs that suggest the samples have genomes with sufficient nucleotide diversity to inform phylogenetic analysis?
- Do you have a large number of discarded SNP calls?
- Are the % Hets very high indicating possible cross-mapping from off-target organisms that may confounding variant calling?

As above evaluating these outputs will vary depending on the data and or pipeline settings, and very much. However the extensive output documentation of nf-core/eager can guide you through every single table and plot to assist you in continuing any type of ancient DNA project, assisted by fun little cartoony schematic diagrams (Figure 21.3)!

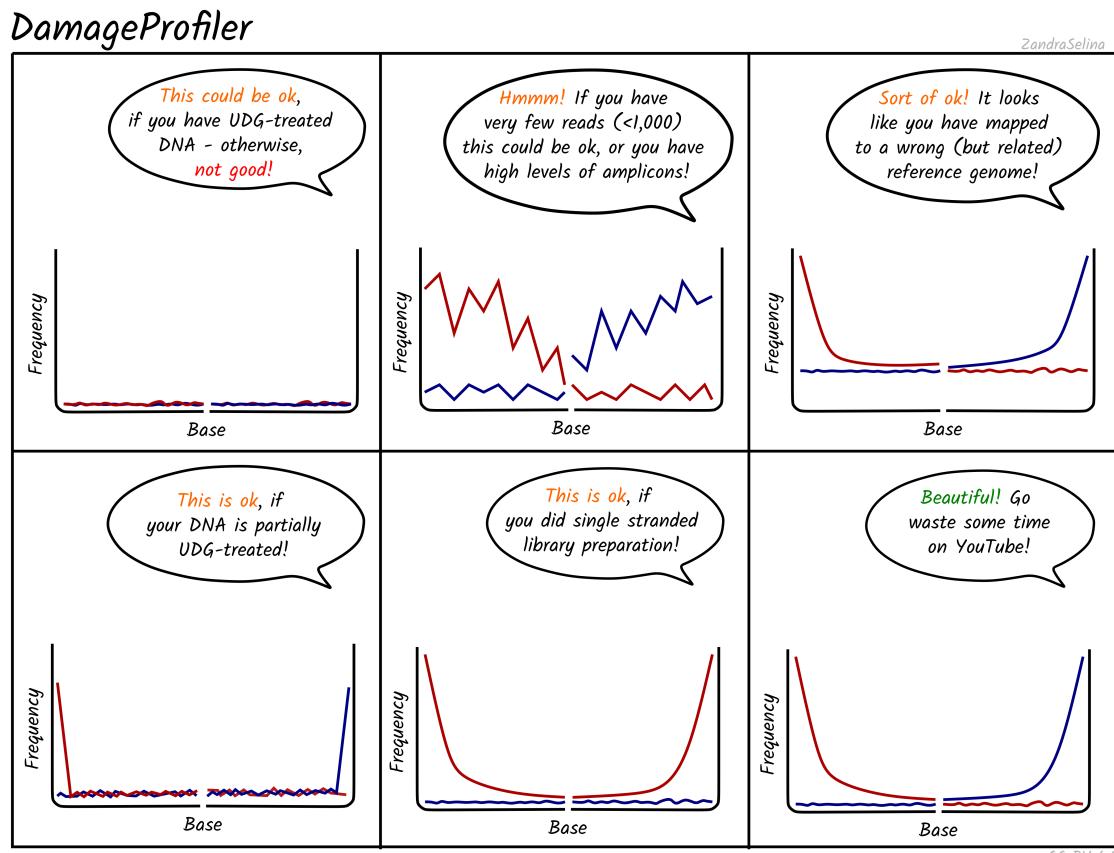


Figure 21.3.: Example of cartoon schematic diagram of the output from DamageProfiler in different contexts. The six panels show different types of ancient DNA damage line-plots from having no damage (flat red/blue lines), however with a speech bubble noting that if the library was UDG treated, that the flat lines might be valid, all the way to the ‘classic’ ancient DNA damage plot with both red and blue lines showing an exponential decay from the ends of reads to the middle, with a motivational speech bubble. Source: Zandra Fagernäs, CC-BY 4.0 via nf-core/eager documentation

#### 21.4.4. Clean up

Before continuing onto the next section of this chapter, you will need to deactivate from the conda environment

```
conda deactivate
```

You may also need to delete the contents of the eager directory

```
cd /<path>/<to>/ancient-metagenomic-pipelines/eager  
rm -r *
```

### 21.5. What is aMeta?



#### Tip

For this chapter's exercises, if not already performed, you will need to create the conda environment from the following `yml` file, and activate the environment:

```
conda activate aMeta
```

While nf-core/eager is a solid pipeline for microbial genomics, and can also perform metagenomic screening via the integrated HOPS pipeline or Kraken2, in some cases you may wish to have a more accurate and resource efficient pipeline. In this section, we will demonstrate an example of using aMeta, a snakemake workflow proposed by @Pochon2022-hj that aims to minimise resource usage by combining both low-resource requiring k-mer based taxonomic profiling as well as accurate read-alignment (Figure 21.5).

Rather than the very computationally heavy HOPS pipeline [@Hubler2019-qw], that requires extremely large computational nodes with large RAM (>1 TB) to load MALT databases into memory, aMeta does this via a two step approach. Firstly it uses KrakenUniq (a k-mer based and thus memory efficient method) to do a screening of sequencing reads against a broad generalised microbial database. Once all the possible taxa have been detected, aMeta will then make a new database of just the genomes of the taxa that were reported from KrakenUniq (i.e. a specific database) but using MALT. MALT on thus much reduced database is then used to perform computationally much heavier alignment against the reference genomes and LCA taxonomic reassignment. The output from MALT is then sent to the MaltExtract program of the HOPS pipeline for ancient DNA authentication statistics.

## 21. Ancient Metagenomic Pipelines

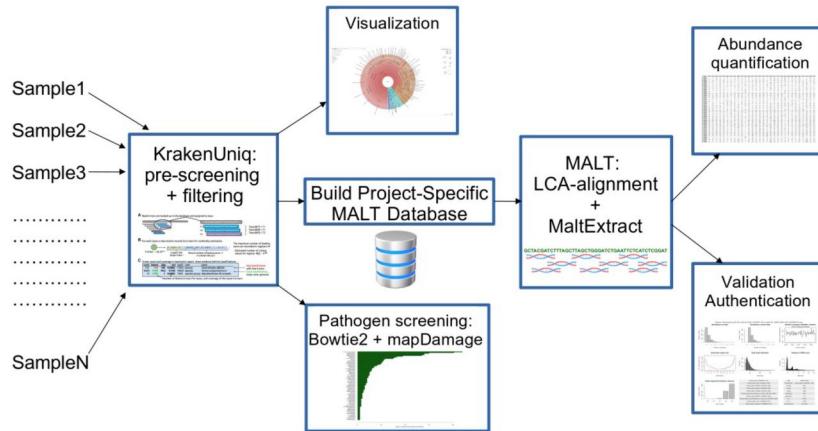


Figure 21.4.: Schematic diagram of the aMeta pipeline. Input samples initially undergo generalised screening using the K-mer based KrakenUniq. For every hit that the reads match inside this database, then sees the genome of that hit then constructed into a MALT database where the reads undergo a mapping step to generate alignments, a lowest-common-ancestor (LCA) algorithm step to refine taxonomic assignments, and ancient DNA authentication statistics generation

### 21.5.1. Running aMeta

In this tutorial we will try running the small test data that comes with aMeta.

aMeta has been written in Snakemake, which means running the pipeline has to be installed in a slightly different manner to the `nextflow pull` command that can be used for nf-core/eager.

```
cd ancient-metagenomic-pipelines/aMeta
```

#### i Note

The code repository, conda environment and database construction has already been performed for you. If you are running this manually you'll need to run the steps in the note block below

**i** Self guided: setting up aMeta for tutorial

### 21.5.2. Setting up aMeta manually

## 21. Ancient Metagenomic Pipelines

```
<!-- TODO: replace with link to before-you-start -->
git clone https://github.com/NBISweden/aMeta
cd aMeta
conda env create -f workflow/envs/environment.yaml
conda activate aMeta
cd ancient-metagenomic-pipelines/aMeta

## Change into ~/.test to set up all the required test resources (Databases etc.)
cd ~/.test

## Set up conda envs
snakemake --use-conda --show-failed-logs -j 2 --conda-cleanup-pkgs cache --conda-create-envs-on-copy
source $(dirname $(dirname $CONDA_EXE))/etc/profile.d/conda.sh

## Build dummy KrakenUniq database
env=$(grep krakenuniq .snakemake/conda/*yaml | awk '{print $1}' | sed -e "s/.yaml://g")
conda activate $env
krakenuniq-build --db resources/KrakenUniq_DB --kmer-len 21 --minimizer-len 11 --jellyfish
conda deactivate

## Get Krona taxonomy tax dump
env=$(grep krona .snakemake/conda/*yaml | awk '{print $1}' | sed -e "s/.yaml://g" | head -n 1)
conda activate $env
cd $env/opt/krona
./updateTaxonomy.sh taxonomy
cd -
conda deactivate

## Adjust malt max memory usage
env=$(grep hops .snakemake/conda/*yaml | awk '{print $1}' | sed -e "s/.yaml://g" | head -n 1)
conda activate $env
version=$(conda list malt --json | grep version | sed -e "s/\"//g" | awk '{print $2}')
cd $env/opt/malt-$version
sed -i -e "s/-Xmx64G/-Xmx3G/" malt-build.vmoptions
sed -i -e "s/-Xmx64G/-Xmx3G/" malt-run.vmoptions
cd -
conda deactivate

touch .initdb

## Run a quick test
snakemake --use-conda --show-failed-logs --conda-cleanup-pkgs cache -s ../workflow/Snakefile

snakemake -s ../workflow/Snakefile425report --report-stylesheet ../workflow/report/custom.css

## Now we move back into the main repository where we can symlink all the database files
cd ../
cd resources/
ln -s ../../.test/resources/*
cd ../config
```

### 21.5.3. aMeta configuration

```
sample fastq
foo data/bar.fq.gz
bar data/foo.fq.gz
```



Make sure when copy pasting into your test editor, tabs are not replaced with spaces, otherwise the file might not be read!

```
samplesheet: "config/samples.tsv"

krakenuniq_db: resources/KrakenUniq_DB

bowtie2_db: resources/ref.fa
bowtie2_seqid2taxid_db: resources/seqid2taxid.pathogen.map
pathogenomesFound: resources/pathogenomesFound.tab

malt_nt_fasta: resources/ref.fa
malt_seqid2taxid_db: resources/KrakenUniq_DB/seqid2taxid.map
malt_accession2taxid: resources/accession2taxid.map

ncbi_db: resources/ncbi

n_unique_kmers: 1000
n_tax_reads: 200
```

## 21. Ancient Metagenomic Pipelines

### Note

As this is only a dummy run (due to the large-ish computational resources required for KrakenUniq), we re-use some of the resource files here. While this will produce nonsense output, it is used here to demonstrate how you would execute the pipeline.

### 21.5.4. Prepare and run aMeta

```
conda activate aMeta
```

```
cd /<path/<to>/ancient-metagenomic-pipelines/ameta/aMeta/
```

```
snakemake --snakefile workflow/Snakefile --use-conda -j 10 --conda-frontend conda
```

### Note

You can modify `-j` to represent the number of available CPUs you have on your machine.

### 21.5.5. aMeta output

## 21. Ancient Metagenomic Pipelines

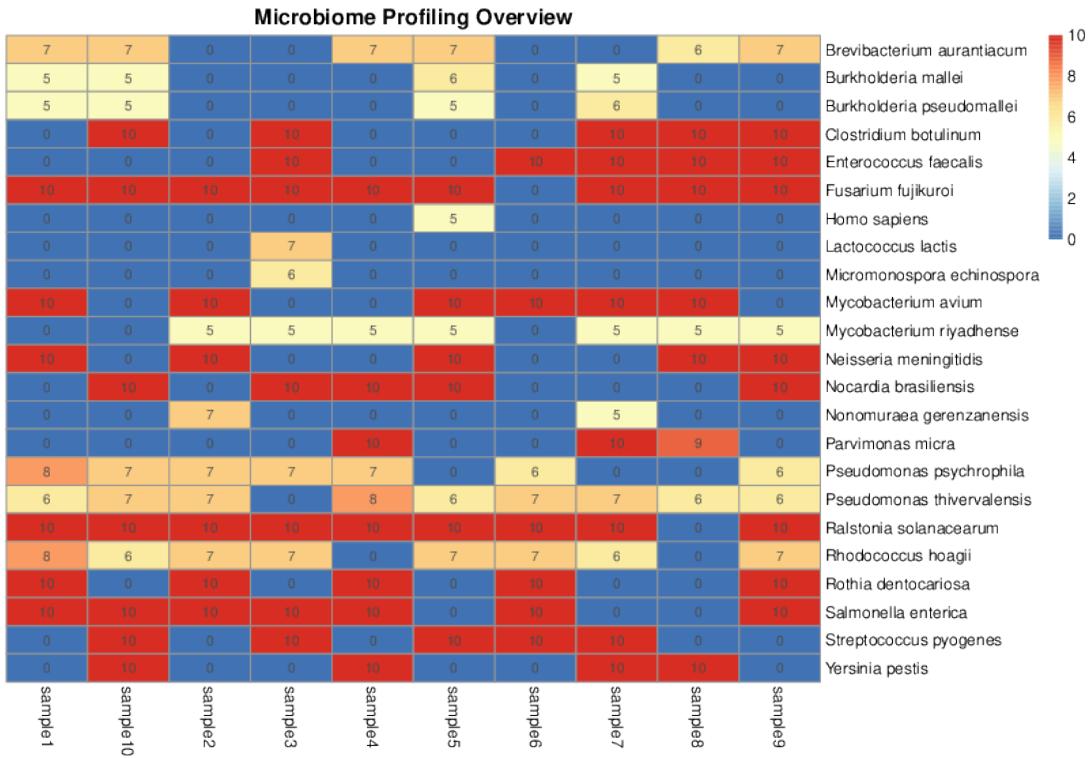


Figure 21.5.: Example microbiome profiling summary heatmap from aMeta. The columns represent different samples, and the rows of different species. The cells of the heatmap are coloured from blue, to yellow, to red, representing aMeta authentication scores from 0 to 10, with the higher the number the more confident of the hit being both the correct taxonomic assignment and that it is ancient. Numbers in the coloured cells also provide the direct score number.

- 
- 
- 
- 
- 
- 
- 

#### 21.5.6. Clean up

```
conda deactivate
```

### 21.6. What is nf-core/mag?

## 21. Ancient Metagenomic Pipelines

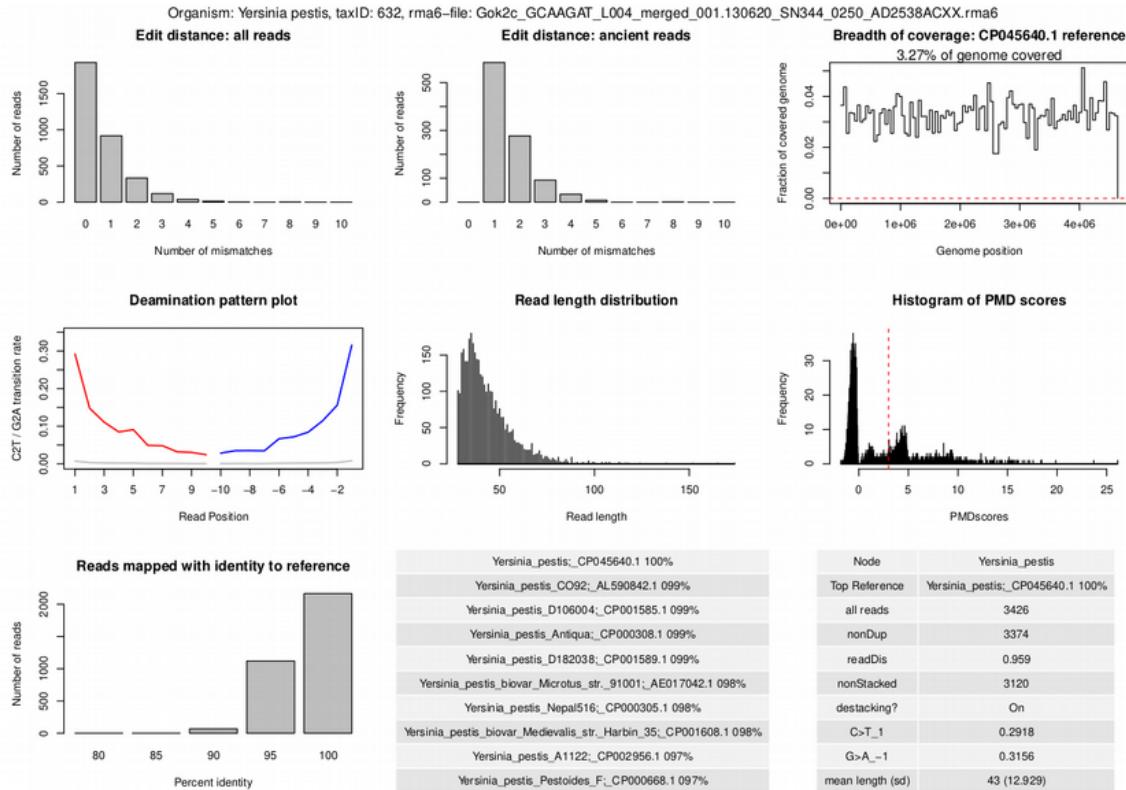


Figure 21.6.: Example of sample/hit specific PDF output from aMeta, 9 panels represent different figures that are useful for evaluating the authenticity of an ancient metagenomic hit. From Left to Right, Top from bottom, the panels consist of: 1. Edit distance (all reads) bar plot, 2. Edit distance (ancient reads) bar plot, 3. Breadth of coverage line plot, 4. Deamination line plot, 5. Read length distribution bar plot, 6. PMD score histogram, 7. Percent identity bar plot, 8. table of similarity to hits from closest hit to least closest, and 10. a general statistics table including the name of the taxonomic node, number of reads, duplicates, and mean read length etc.

## 21. Ancient Metagenomic Pipelines

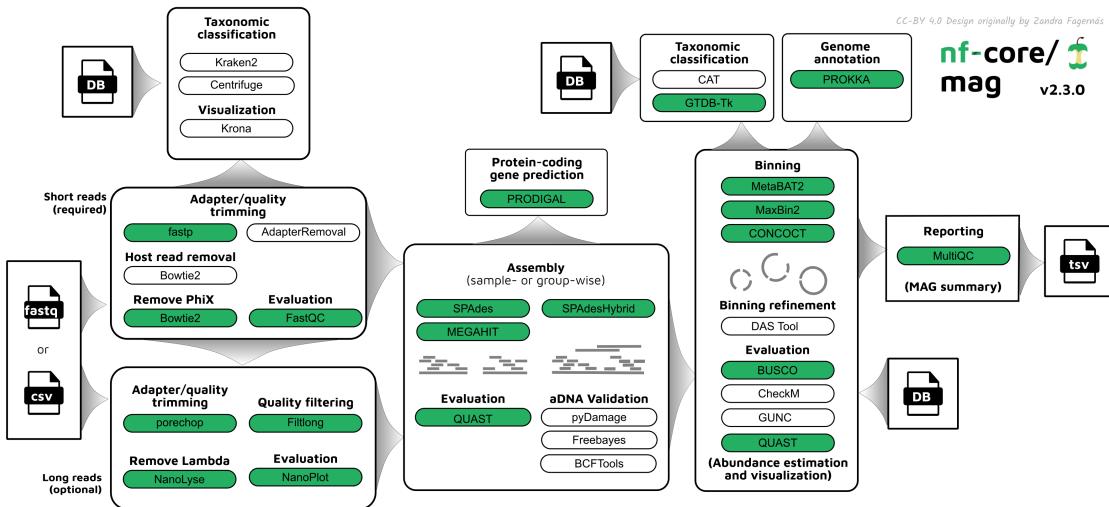


Figure 21.7.: Overview diagram of the main steps and tools of the nf-core/mag pipeline.

Taking reads in FASTQ format or a samplesheet as input, reads can go through adapter and/or quality trimming for specific tools for both short and long reads. These reads can optionally go into taxonomic classification and visualisation, at the same time as the prepared reads go into sample- or group wise assembly, evaluation, as well as the ancient DNA validation subworkflow. Resulting contigs can be functionally annotated at the same time as optionally going through binning and binning refinement and evaluation (with statistics generation). Bins and refined bins can be taxonomically classified and annotated, with all final reports going into MultiQC.

### 21.6.1. Running nf-core/mag

**i** Self guided: reference download and preparation

This tutorial will re-use the data from the *de novo* assembly chapter. If you have not run that practical, or deleted the data, you can retrieve these with

```
## IF YOU'VE INTO RUN/DELETE DENOVO ASSEMBLY THEN RUN THIS
cd /<path>/<to>/denovo-assembly/
wget https://share.eva.mpg.de/index.php/s/CtLq2R9iqEcAFyg/download/2612_R1.fastq.gz
wget https://share.eva.mpg.de/index.php/s/mc5JrpDWdL4rC24/download/2612_R2.fastq.gz
wget https://data.ace.uq.edu.au/public/CheckM_databases/checkm_data_2015_01_16.tar.gz
```

Once you have done this, we will need to download the single copy marker gene reference database for CheckM, which is used to assess completeness of genome bins.

## 21. Ancient Metagenomic Pipelines

```
cd /<path>/<to>/ancient-metagenomic-pipelines/
mkdir -p mag/data
cd mag/data

mkdir checkm_data_2015_01_16
tar --directory checkm_data_2015_01_16/ -xzvf checkm_data_2015_01_16.tar.gz

cd ..
```

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

```
## Don't run this yet!
nextflow run nf-core/mag -r 2.3.2 \
-profile test,docker \
--input '../denovo-assembly/*{R1,R2}.fastq.gz' --outdir ./results # -c custom.config \
--ancient_dna --binning_map_mode own \
--binqc_tool checkm --checkm_db data/checkm_data_2015_01_16/ \
--centrifuge_db false --kraken2_db false --skip_krona --skip_spades --skip_spadeshybrid --
```

- 1.
2.
  - 
  -

## 21. Ancient Metagenomic Pipelines

3.

4.

### ⚠ Binning mapping mode with ancient DNA

The aDNA mode of nf-core/mag *only* supports mapping reads back to the original contigs. The other mapping modes are when doing co-assembly, where you assume there are similar organisms across all your metagenomes, and wish to map reads from all samples in a group to a single sample's contig (for example). Doing this on aDNA reads would risk making false positive damage patterns, as the damaged reads may derive from reads from a different sample with damage, that are otherwise not present in the sample used for generating the given contig.

5.

### ℹ nf-core pipelines and reference files

Most nf-core pipelines will actually download reference databases, build reference indices for alignment, and in some cases reference genomes for you if you do not specify them as pipeline input. These download and reference building steps are often very time consuming, so it's recommended that once you've let the database download it once, you should move the files somewhere safe and you can re-use in subsequent pipeline runs.

6.

### ℹ Information on skipped tests

The steps we are skipping are: host/arefact removal (`Bowtie2` removal of phiX), taxonomic classification of reads (`centrifuge`, `kraken2`, `krona`), the additional assemblers (`metaSPAdes` and `SPAdeshybrid`, as these require very large computational resources), additional bidders (`MaxBin2` and `CONCOCT`, while they are used in the *de novo* assembly chapter, they take a long time to run), skip raw contig annotation (with `Prodigal`, as we do this at the bin level), and contig taxonomic classification (with `GTDB` and `BUSCO` as they require very large

databases).

### 21.6.2. Configuring Nextflow pipelines

```
process {

    withName: FASTP {
        cpus = 8
    }

    withName: BOWTIE2_PHIX_REMOVAL_ALIGN {
        cpus = 8
    }

    withName: BOWTIE2_ASSEMBLY_ALIGN {
        cpus = 8
    }

    withName: PYDAMAGE_ANALYZE {
```

## 21. Ancient Metagenomic Pipelines

```
    cpus = 8
}

withName: PROKKA {
    cpus = 4
}

withName: MEGAHIT {
    cpus = 8
}

withName: CHECKM_LINEAGEWF{
    memory = 24.GB
    cpus   = 8
    ext.args = "--reduced_tree"
}

}
```

```
nextflow run nf-core/mag -r 2.3.2 \
-profile test,docker \
--input '../denovo-assembly/*{R1,R2}.fastq.gz' --outdir ./results
--ancient_dna --binning_map_mode own \
--binqc_tool checkm --checkm_db data/checkm_data_2015_01_16/ \
--centrifuge_db false --kraken2_db false --skip_krona --skip_spades --skip_spadeshybrid --s
-c custom.config
```

 Tip

Configuration files don't need to be personal nor custom!

You can also generate a HPC / server specific profile which can be used on all nf-core pipelines (and even your own!).

See [nf-co.re/configs](#) for all existing institutional configs

### 21.6.3. nf-core/mag output

#### 21.6.3.1. Results Files

- -
- -
- -
- -
- -
- -

*21. Ancient Metagenomic Pipelines*

**21.6.3.2. Report Table**

- 
- —
- 
- 
- —
- 

- 
- 
- 
- 
-

## *21. Ancient Metagenomic Pipelines*

### **21.6.4. Clean up**

```
conda deactivate
```

```
cd /<path>/<to>/ancient-metagenomic-pipelines/mag  
rm -r *
```

### **21.7. Questions to think about**

- 1.
- 2.
- 3.

### **21.8. References**

# Summary

1 + 1

## **Part VI.**

# **Appendices**

## **22. Resources**

### **22.1. Introduction to NGS Sequencing**

•  
•

## **23. Tools**

### **23.1. Introduction to R and the Tidyverse**

- 
- 
- 

### **23.2. Introduction to Python and Pandas**

- 
- 

### **23.3. Introduction to Git(Hub)**

- 

### **23.4. Functional Profiling**

- 
- 
- 
-

## 23.5. *De novo* assembly

- • • • • • • • • • • • • • •

## 23.6. Genome Mapping

- ● ●

## 23.7. Phylogenomics

- • •

## 23.8. Ancient Metagenomic Pipelines

- ● ●