Introduction to Ancient Metagenomics

James A. Fellows Yates and Christina Warinner

2025-05-21

Table of contents

Introduction

Ancient metagenomics is an emerging field of research that applies the techniques of metagenomics to ancient DNA. Metagenomics is defined as the study of the collection of genes and genomes from an assemblage of microorganisms present in a defined environment (Marchesi and Ravel 2015), but in practice it often considers all DNA present in a given environment. Metagenomes are typically obtained by untargeted, so-called "shotgun" DNA sequencing, followed by analysis using reference-based mapping techniques or reference-free genome assembly techniques.

Ancient metagenomics applies these techniques to ancient DNA recovered from the archaeological and palaeontological records in order to reconstruct information about past ecologies, whether that is the structure and composition of past ecosystems, the microbial communities present in ancient microbiomes, the aetiology and pathogenesis of ancient infectious diseases, or the microbial make-up of ancient fermented foods.

Because ancient DNA is more degraded than modern DNA, many techniques and approaches used in metagenomics must be adapted or modified in order to allow for the study of ancient organisms. This can range from small parameter tweaks to existing methods to more large-scale changes, such as new laboratory protocols or specialised software tools. Ancient metagenomics studies also face distinct challenges with respect to authenticating results and managing post-mortem or post-depositional microbial contamination, and knowing these challenges, as well as strategies for their solution, is critical to the success of ancient metagenomics projects.

Fortunately, many new protocols, software tools, and pipelines have been developed to address many of the problems that challenge ancient metagenomics, and over the past decade the rapidly growing field of ancient metagenomics has leveraged these advancements to reveal an astonishing wealth of new information about human and natural history, from revealing how past ecosystems changed in response to long-term climatic and anthropogenic change, to identifying the causes of devastating pandemics, 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.

With such a fast-moving field, it can be difficult to know where to start, and this is true both for incoming students planning their first research projects and for established metagenomics researchers looking to pivot to ancient DNA. To help orient new people to the field, we have developed an online course and companion textbook that serves as a broad theoretical and practical introduction to the field of ancient metagenomics. The textbook is divided into five theoretical chapters and 12 practical exercise modules. The goal of the theoretical chapters is to provide an introduction to key concepts in genetics and genomics, ancient DNA, metagenomics and ecology, taxonomy and phylogeny, and functional genomics in order to establish the foundational knowledge necessary to plan and carry out ancient metagenomics research. The goal of the practical modules is to equip readers with a software toolkit and a practical understanding of the strategies used to analyse ancient metagenomics datasets.

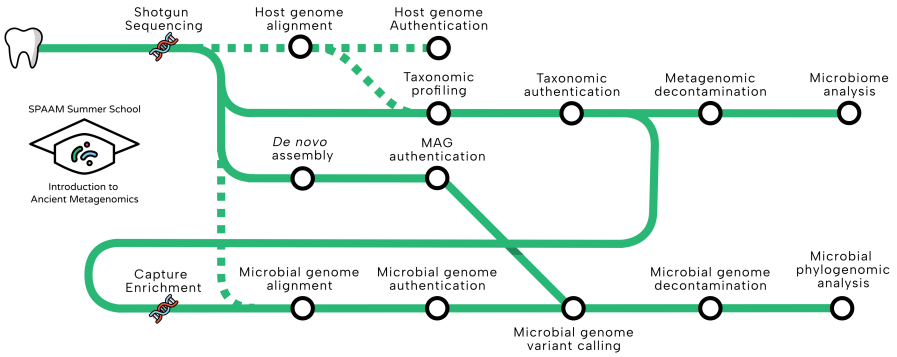


Figure 1: A metro-map style diagram showing the major steps of an ancient metagenomic study workflow, with a green line starting from a sample (indicated by a tooth), sequencing (indicated by a DNA strand) then going through multiple 'stations': host genome alignment, host genome authentication, taxonomic profiling, taxonomic authentication, metagenomic decontamination, microbiome analysis, *de novo* assembly, MAG authentication, capture enrichment, microbial genome alignment, microbial genome authentication, microbial genome variant calling, microbial genome decontamination, microbial phylogenomic analysis.

Focusing on host-associated ancient metagenomics, readers will learn the main steps of ancient metagenomic bioinformatic workflows, become familiar with the command line, learn how to process next-generation-sequencing (NGS) data, conduct taxonomic profiling of datasets, carry out ecological analyses, and perform de novo metagenomic assembly. 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, reliable, and reproducible manner (Figure **??**).

All material was originally developed for the SPAAM Summer School: Introduc-

tion to Ancient Metagenomics

References

Citing this book

The source material for this book is located on GitHub:

https://github.com/SPAAM-community/intro-to-ancient-metagenomics-book.

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

> Fellows Yates, J. A., Warinner, C., Andrades Valtueña, A., Borry, M., Guellil, M., Herbig, A., Hiß, A., Hübner, A., Kocher, A., Lamnidis, T. C., Michel, M. E., Nota, K., Oskolkov, N., Pach, A. L., Pérez, V., Schmid, C., Velsko, I. M., Warner, R., Zampirolo, G., & Zeibig, T. (2024). Introduction to Ancient Metagenomics (2024.2). Zenodo. https://doi.org/10.5281/zenodo.13784555

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. Contributors to the textbook are listed below.

2022-2024

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

2022-2024

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 microbiome.

2024

Aleksandra Laura Pach is a bioinformatician currently working on a PhD at the Schroeder and Racimo groups at section for Ecology and Evolution, Globe institute, University of Copenhagen. Her research centers around ancient metagenomics specifically focusing on inference of eukaryotes and methodologies.

2022-2024

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

2022-2024

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

2022-2024

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

2022-2024

Arthur Kocher initially trained as a veterinarian. He then pursued a PhD in the field of disease ecology, during which he studied the impact of biodiversity changes on the transmission of zoonotic diseases using molecular tools such as DNA metabarcoding. During his Post-Docs, he extended his research focus to evolutionary aspects of pathogens, which he currently investigates using ancient genomic data and Bayesian phylogenetics.

2022-2024

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.

2024

Giulia Zampirolo is a postdoctoral researcher specialised in ancient metagenomics, with a background in both archaeology and paleogenomics. During her PhD at the Centre for GeoGenetics (University of Copenhagen), she utilised sedimentary ancient DNA across different types of deposits to investigate past human-environment interactions. Currently based at the Globe Institute, University of Copenhagen, she works in Kristine Bohmann's research group, contributing to the project SEACHANGE, which explores the evolution of past marine ecosystems and biodiversity. Her research interests lie in the interdisciplinary intersection of archaeology,

2022-2024

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

2022-2024

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

2022-2024

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

2023-2024

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.

2023-2024

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.

2024

Tessa Zeibig is a trained Microbiologist, she's currently pursuing her PhD in the Computational Pathogenomics Group led by Alexander Herbig at the Max Planck Institute for Evolutionary Anthropology. Her focus is on human viruses, seeking to understand their evolution and development, using ancient genomic data.

2024

Vilma Pérez is a microbial ecologist at the Australian Centre for Ancient DNA, University of Adelaide. Her research focuses on reconstructing microbial communities from present and past environments using environmental DNA techniques. Her objective is to use this information as bioindicators, providing insights into how environments have changed or responded to disturbances over time.

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.

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.

2022

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

2022

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.

Foreword

This textbook and companion course was conceived in 2020 during the depths of the global coronavirus pandemic. James Fellows Yates and I were discussing the need for more training resources in the emerging field of ancient metagenomics, and, together with Pierre Stallforth at the Leibniz HKI, I had just received generous funding from the Werner Siemens Foundation to advance new techniques and methodologies in the emerging research area of microbial paleobiotechnology. Trapped at home and interacting with the world primarily through screens, we began to think seriously about how we might harness our new digital connectivity and our new funding to expand computational training opportunities, irrespective of nationality or institution, for a new generation of students eager to pursue a career in the young field we and others were striving to build: ancient metagenomics.

When I began graduate school in 2004, the idea that one day we would be able to reconstruct the genomes and ecological communities of microbes that lived thousands or even millions of years ago seemed a remote possibility. This was still the era of PCR and Sanger sequencing, and each newly reported microbial ancient DNA sequence was met with deep skepticism and vigorously debated. By decade's end, however, emerging high throughput sequencing technologies began to change that outlook, and in 2011 the first complete ancient microbial genome was published. The study of ancient microbes started to gain momentum, and after another decade of research starts and stops, false turns, dead ends, and dramatic breakthroughs, the seemingly impossible was starting to happen - we were developing an archaeology of microbes. As the field grew, so did its applications–expanding beyond its origins in pathogen genomics to include microbiome studies, fermented food studies, clinical metagenomics, and environmental microbial ecology.

Around this time, people finally stopped asking why studies of ancient microbes belonged in the fields of archaeology and anthropology and started understanding how fundamental microbes are to the long arc of human health, diet, and culture. In 2019, I co-organized a Wenner Gren Symposium on *Cultures of Fermentation*, an interdisciplinary meeting that brought together social anthropologists, biological anthropologists, and archaeologists to examine the role of microbes in human cultures and societies from the Paleolithic to the present, and

we won the 2023 American Anthropological Association's Cross-Fields Prize for our companion article about the symposium in *Current Anthropology*. It was a milestone in gaining recognition for the field and the first major anthropological prize awarded for the study of microbes.

Microbial ancient metagenomics was finally coming into its own, and a vibrant new research community was taking shape. Much of this community centers around a group affectionately known as SPAAM (https://spaam-community.org), which stands for Standards Precautions and Advances in Ancient Metagenomics. It was the first in what is now a long list of cheeky food-themed acronyms in biomolecular archaeology, which includes the tools MALT (MEGAN Alignment Tool) and HOPS (Heuristic Operations for Pathogen Screening), as well as other ancient biomolecules interest groups in general like HAAM (Human Ancient DNA, Ancestry, and Mobility) and PAASTA (Palaeoproteomics And Archaeology, Society for Techniques and Advances).

Alexander Herbig and Johannes Krause organized the first SPAAM conference in 2016, and its first major output was a 2017 article in the journal Annual Review of Genomics and Human Genetics in which we proposed a research framework for the emerging field of microbial archaeology. Recognizing the potential for SPAAM to serve as a hub for fostering education, networking, and collaborative research in ancient metagenomics, James Fellows Yates built upon this initial effort by further organizing this nascent community of researchers and students around the world, coordinating subsequent SPAAM conferences, and building up resources and workshops to support the far-flung members of this growing field. In 2022, SPAAM joined the International Society for Biomolecular Archaeology (ISBA) as an affiliate group.

Reflecting on our own trajectories of education and training, James and I had been talking for some time about how our field needed a more formalized way of training new researchers in the field and getting students up to speed on the very quickly evolving methods, concepts, and especially bioinformatics strategies that underlie our work. The Werner Siemens Foundation grant I had received included funds for education and outreach, and James and I hatched the idea of developing an online ancient metagenomics course to further these goals. The Jena School for Microbial Communications (JSMC) at Friedrich Schiller University signed on to accredit the course, as did the Max Planck-Harvard Research Center for the Archaeoscience of the Ancient Mediterranean (MHAAM). More than a dozen group leaders, postdocs, and grad students at the MPI-EVA and other institutions volunteered to help develop the course, and in 2022 we launched our first Introduction to Ancient Metagenomics Summer School, a week-long intensive online practical course open to students worldwide for free.

From the beginning, we had the idea of making all of the teaching material for the course available on a website, and the current textbook grew organically out of our experiences teaching the material to students with diverse educational backgrounds and skill levels. The book is written with a target audience in mind

of a Master's student or early stage PhD student who is highly motivated to begin work in the field of ancient metagenomics, but who may be transitioning from a different field, whether that is biology, computer science, or even archaeology. We aim to provide both a broad overview and a solid foundation in the most essential concepts needed to begin work in ancient metagenomics, and we encourage our readers to follow up with the citations provided in the textbook to gain deeper understanding of the many topics we cover in an abbreviated form.

First and foremost, we wanted this textbook to be hands-on and practical, and so we introduce not only theoretical concepts but also provide datasets, pipelines, and code to allow students to go through the full process of analyzing ancient microbial DNA datasets–from fastq files to phylogenies and functional genomics.

This textbook is a labor of love produced through the efforts of many people who have generously volunteered their time to create a resource for future generations, to help give new students a jumpstart in the field, and to promote a research community that is collaborative, supportive, and committed to advancing knowledge and understanding of the microbial world around us and how it came to be.

Christina Warinner Harvard University, Cambridge, Massachusetts, USA, May 2024

Thank you

I would like to extend a special thank you to the following people who contributed to the development of the course and companion textbook:

- Aida Andrades Valtueña
- Maxime Borry
- Keri Burge
- Remí Densie
- Sebastian Duchene
- Jasmin Frangenberg
- Meriam Guellil
- Alexander Herbig
- Alina Hiß
- Alexander Hübner
- Arthur Kocher
- Thiseas Lamnidis
- Megan Michel
- Nikolay Oskolkov
- Betsy Nelson
- Gunnar Neumann
- Kevin Nota
- Aleksandra Laura Pach

- Vilma Pérez
- Clemens Schmid
- Irina Velsko
- Robin Warner
- Christina Warinner
- James Fellows Yates
- Guilia Zampirolo
- Tessa Zeibig

Acknowledgements

We are very grateful to Matthias Meyer and Pierre Stallforth for generously reading our early drafts of sections of our textbook and providing valuable feedback and suggestions.

Acknowledgements

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

Financial Support

WSS
WERNER SIEMENS-STIFTUNG

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

Institutional 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. We describe here to approximately recreate the computing platform used during the summer schools.

Basic requirements

> **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 following 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 be 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 currently the most popular distribution source of bioinformatics software for conda.

For each chapter, if it requires pre-prepared data, the top of the page will

have a box called 'Self guided: chapter envionment setup' that link to a `.tar` archive. This contain the raw data will be available to download for following the chapter Furthermore, the same box will describe how to use a conda `.yml` file that specifies the software environment for that chapter will also be available for you to install.

See the rest of this page on how to install `conda` (if not already available to you), and also how to create `conda` software environments.

Software Environments

Before loading the environment for the exercises, the software environment will need to be created 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.

If you've not yet installed `conda`, please follow the instructions in the box below.

Quick guide to installing conda

These instructions have been tested on Ubuntu 22.04, but should apply to most Linux operating systems. For OSX you may need to download a different file from here.

- Change directory to somewhere suitable for installing a few gigabytes of software, e.g. `mkdir ~/bin/ && cd ~/bin/`
- Download miniconda

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

- Run the install script

```
bash bash Miniconda3-latest-Linux-x86_64.sh
```

  - Review license
  - Agree to license
  - Make sure to install miniconda to the correct directory! e.g. `/home/<YOUR_USER>/bin/miniconda3`
  - Yes to running `conda init`
  - Copy the `conda config` command
  - Close the terminal (e.g. with `exit` or ctrl + d)
  - Open the terminal again and run the command you copied (i.e., `conda config --set auto_activate_base false`)
  - Exit and open the terminal again
  - Type `conda --version` to check conda is installed and working
  - Set up bioconda

```
conda config --add channels bioconda
conda config --add channels conda-forge
```

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 carry out the following steps.

1. Download and unpack the `conda` env file the top of the chapter by right clicking on the link and pressing 'save as'. Once uncompressed, change into the directory.
2. Then you can run the following conda command to install the software into it's dedicated environment

```
conda env create -f /<PATH/<TO>/<DOWNLOADED_FILE>.yml
```

> **Note**
>
> You only have to run the environment creation once for each chapter/environment!

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

```
conda env list
```

4. To load the relevant environment, you can run

```
conda activate <NAME_OF_ENVIRONMENT>
```

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, run `conda remove --name <NAME_OF_ENV> --all -y`

> **Note**
>
> If at any point you have issues of running out of space on your machine, you can first try running
>
> ```
> conda clean --all
> ```
>
> And you can answer 'yes' to all the prompts to remove unused packages and caches.
> If you are still having issues, you can remove the environments that you don't need any more with the following.
> `conda env list`

To list all your existing conda environments, and then delete the directory of the unneeded environment with `rm`.

```
rm -r /<path>/<to>/<conda_install>/envs/<environment_name>
```

Additional Software

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

Introduction to the command line

- 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 -c bioconda biopython=1.68 bwa=0.7.17 maxbin2=2.2.7 metabat2 samtool
cd /<path>/<to>/denovo-assembly
git clone https://github.com/bxlab/metaWRAP.git
## don't forget to update path/to!
export PATH=$PATH:/<path>/<to>/metaWRAP/bin
```

Warning

If you close your terminal halfway through the chapter, when opening the terminal again to continue the chapter, you MUST re-export the path to the metaWRAP bin directory to have access to the software!

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 /<path>/<to>/func
```

Authentication

- pip

- metaDMG
  - Make a conda environment file called `metadmg.yml` containing the following.

```
name: metaDMG
channels:
- conda-forge
- bioconda
dependencies:
- conda-forge::python=3.9.15
- bioconda::htslib=1.17
- conda-forge::eigen=3.4.0
- conda-forge::cxx-compiler=1.5.2
- conda-forge::c-compiler=1.5.2
- conda-forge::gsl=2.7
- conda-forge::iminuit=2.17.0
- conda-forge::numpyro=0.10.1
- conda-forge::joblib=1.2.0
- conda-forge::numba=0.56.2
- conda-forge::flatbuffers=22.9.24
- conda-forge::psutil=5.9.4
```

  - Create the environment with

```
conda env create -f metadmg.yml
```

  - Change to the chapter's directory

```
cd /<path>/<to>/authentication
```

  - Activate the new environment

```
conda activate metaDMG
```

  - Clone the latest version of metaDMG, and compile

```
git clone https://github.com/metaDMG-dev/metaDMG-cpp.git
cd metaDMG-cpp
make clean && make CPPFLAGS="-L${CONDA_PREFIX}/lib -I${CONDA_PREFIX}/include" HTSSRC=sys
```

  - Install some patches, and some additional modules

```
pip install git+https://github.com/metaDMG-dev/metaDMG-core #@stopiferrors_branch
pip install metaDMG[viz]
```

  - Deactivate the conda environment

```
conda deactivate
```

  - Make the metaDMG executable available in the environment

```
export PATH="$PATH:/<path>/<to>/authentication/metaDMG-cpp/"
```

> **Warning**
>
> If you close your terminal halfway through the chapter, when opening the terminal again to continue the chapter, you MUST re-export the path to the metaDMG-cpp directory to have access to the software!

Phylogenomics

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

    ```
    cd /<path>/<to>/phylogenomics
    tar -xvf TempEst_v1.5.3.tgz
    cd TempEst_V1.5.3
    export PATH="$PATH:/<path>/<to>/phylogenomics/TempEst_v1.5.3/bin/"
    ```

  - If you get an error like `Exception in thread "main" java.lang.UnsatisfiedLinkError: Can't load library: /usr/lib/jvm/java-11-openjdk-amd64/lib/libawt_xawt.so`, make sure you have Java installed e.g.

    ```
    sudo apt install openjdk-11-jdk
    ```

- MEGAX (v11.0.11)

> **Warning**
>
> If you close your terminal halfway through the chapter, when opening the terminal again to continue the chapter, you MUST re-export the path to the Tempest bin directory to have access to the software!

Ancient metagenomic pipelines

- 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 -
    sudo chmod a+r /etc/apt/keyrings/docker.gpg
    echo \
    "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.g
    "$(. /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-plugi
    ```

```
## 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
```

- aMeta (make sure you've already downloaded the data directory as per the chapter instructions)

```
cd /<path>/<to>/ancient-metagenomic-pipelines/
git clone https://github.com/NBISweden/aMeta
cd aMeta
## We have to patch the environment to use an old version of Snakemake as aMeta is not compa
sed -i 's/snakemake-minimal>=5.18/snakemake <=6.3.0/' workflow/envs/environment.yaml
conda env create -f workflow/envs/environment.yaml
```

Part I

Theory

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

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

Lectures

Introduction to NGS Sequencing

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

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

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

Introduction to Ancient DNA

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

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

non-proofreading polymerases and note key steps in NGS library preparation during which enzyme selection is critical in ancient DNA studies.

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

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 eDNA

This chapter introduces the field of environmental DNA (eDNA), with a particular focus on ancient environmental DNA preserved in sediments (sedimentary ancient DNA or sedaDNA). We begin by defining what is environmental DNA and discussing its inherent complexity, especially when working with ancient eDNA.

Next, we untangle this complexity by exploring the different components found in sedimentary DNA records. We then examine the processes that influence how

these records are formed and preserved (i.e., taphonomic processes), emphasising their importance for the reliability of sedimentary DNA data. We also address some limitations of this method and offer key considerations for overcoming them.

Finally, we highlight the potential of sedaDNA as a tool for reconstructing past environments and tracking ecological changes over time, using studies to illustrate the type of insights that can be gained.

Chapter 1

Introduction to NGS Sequencing

> **!** Important
>
> This page is still under construction

> **!** Important
>
> Parts of this text are a raw transcription of the 2022 lecture from the Werner Siemens-Stiftung funded SPAAM Summer School: Introduction to Ancient Metagenomics (2022). Extensive editing is still underway. Readability will be low.

Next generation sequencing (NGS) revolutionised the whole of biology by providing rapid and cheap access to huge amounts of DNA sequence data. One unexpected benefit of this technology was that the technique used by Illumina sequencers was that it was also *perfect* for sequencing ancient DNA.

In this chapter, we will do a brief overview of how DNA is structured, how DNA sequencing works, how most NGS sequenced DNA sequences are digitally represented, and then cover some of the important considerations for ancient metagenomic sequencing datasets.

1.1 Basic structure of DNA

To understand how sequencing works, we need to understand the basic structure of DNA. Deoxyribonucleic acid (DNA) is a biological molecule famous for it's

'double helix' structure (Figure **??**), and is present in the vast majority of cells in your body . It encodes all the genetic information required for the development, growth, and function of living organisms.

The each strand of the double helix structure consist of four main components called nucleotides, which bind together in different orders. These nucleotides consist of two groups:

- Pyrimidines: cytosine (C) and thymine (T)
- Purines: guanine (G) and adenine (A)

The nucleotides on one strand are connected via a 'sugar phosphate backbone', however the two strands are held together by the interaction of the hydrogen bond pairing of four different nucleobases on each strand (Figure **??**).

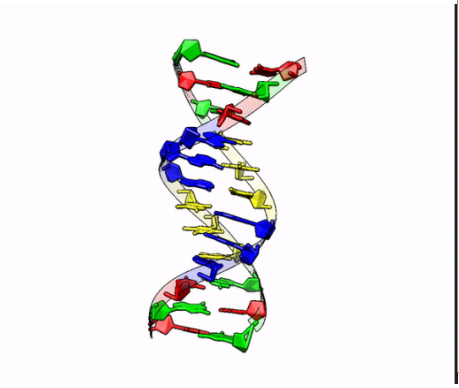


Figure 1.1: Still from animation of 3D DNA double helix. Source: Erin Rod, CC BY 4.0, via Wikimedia Commons

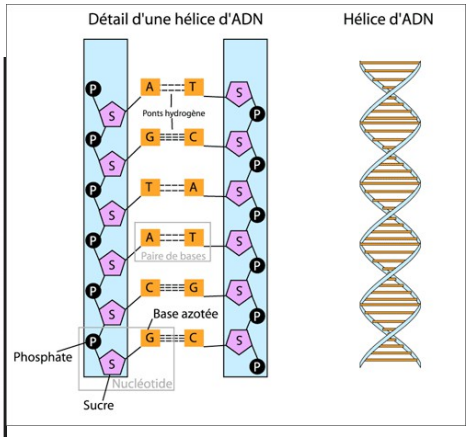


Figure 1.2: 2D molecular diagram of DNA helix, with sugar-phosphate backbone and amine groups labelled indicated. Source: Pradana Aumars, CC BY-SA 4.0, via Wikimedia Commons

This pairing consists of one pyrimidine and one purine, which are complementary to each other.

- C with G (to remember: thing CGI in animated movies)
- A and T (to remember: think AT-AT Walker from Star Wars, Figure **??**)

Therefore, whenever you find a C on one strand, you will normally find a G on the other strand (and vice versa), and the same for A and T. If you find an A on one strand, you will find a T on the other strand, or if a T on one strand, you will have another.

What this means is that because of this complementary pairing, depending on which strand you are reading, you can always reconstruct the order bases on

the *other* strand.

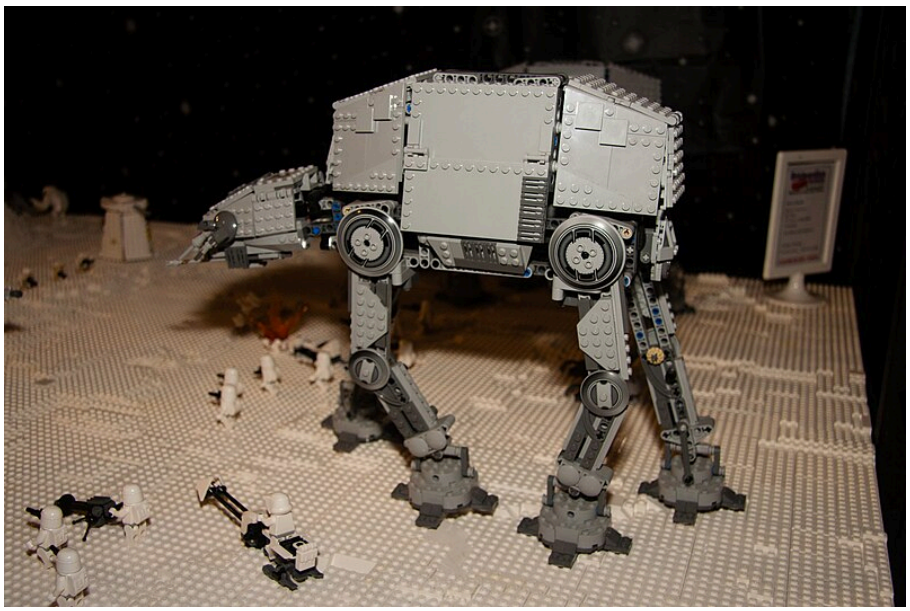


Figure 1.3: Lego construction of AT AT walker from Star Wars. Source: Tim Moreillon, CC BY-SA 2.0 Generic, via Wikimedia Commons

1.2 DNA replication

This characteristic of the DNA molecule is a critical component on how DNA gets replicated (or copied) within a cell.

At it's core, DNA replication consists of:

1. Unwinding the DNA helix and separate the two strands by breaking the hydrogen bonds between the complementary bases using a helicase enzyme
2. Use a polymerase enzyme to add complementary new 'free' nucleotides floating in the cell to the exposed base on the strand (repeating for each base until a 'stop' signal is received)
3. Construct bonds on the sugar-phosphate backbone of the new strand using a ligase enzyme

A graphical representation can be seen in using a helicase enzyme in the image below (Figure **??**).

This concept is important because it is the basis of how DNA sequencing works, and how we can reconstruct the sequence of a DNA molecule.

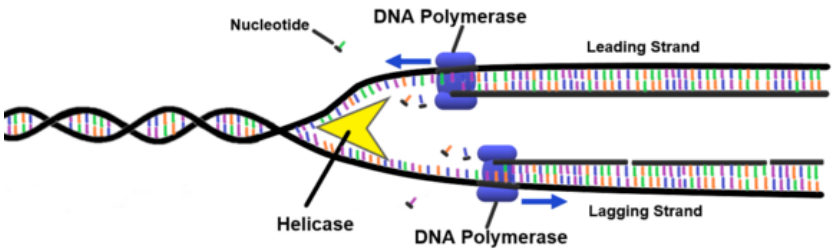


Figure 1.4: Schematic diagram of DNA replication, with a helicase unwinding DNA strand with DNA polymerases on the leading and lagging strand incorporating free nucleotides. Source: Christine Imiller, CC BY-SA 4.0, via Wikimedia Commons

1.3 Extracting DNA

To understand why NGS sequencing revolutionised the field of palaeogenomics, we need to quickly need to compare a difference between how we get modern and ancient DNA.

1.3.1 Modern DNA

To get DNA from 'modern' samples (i.e., living organisms), you first get your tissue. You then have to break down (lyse) the cell membrane and/or walls, to release the molecular contents of the cell. Extraction protocols then use a variety of enzymes to degrade the other biomolecules in the cell (e.g., proteins, lipids, RNA) so that they do not 'interfere' with the extraction of the DNA itself. Finally, the DNA is separated and pulled out from the rest of the now-broken cell contents (purification).

This extracted DNA are typically in very long and intact strands that you can sort of imagine as long, soft spaghetti pasta like structures (Figure **??**).

1.3.2 Ancient DNA

In contrast, while the concept of ancient DNA extraction is the similar, the end product of the extraction is different.

Taking a skeleton as an example, once you have your bone sample, cell lysis is normally not necessary as the cells are already broken down. However, we have to instead 'liberate' the DNA from the mineral matrix of the bone - so

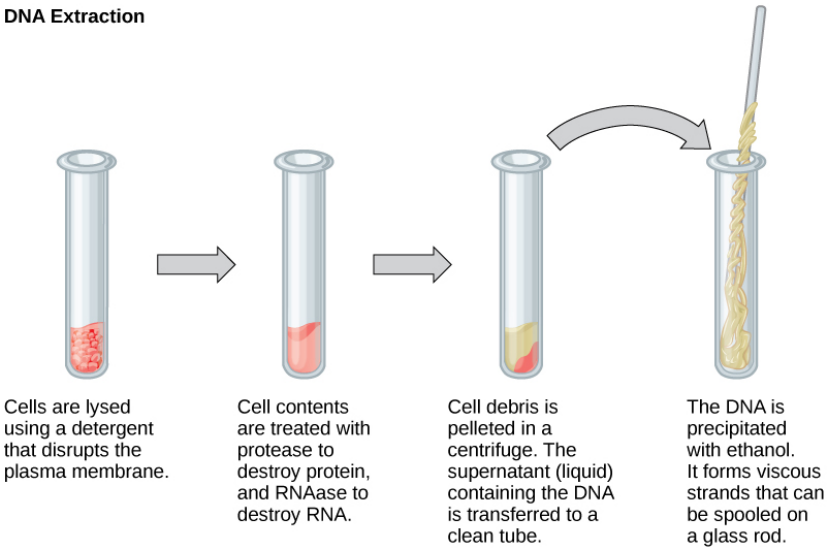


Figure 1.5: Simplified schematic diagram of the process of DNA extraction from living cells. Source: CNX OpenStax, CC BY 4.0 International, via Wikimedia Commons

palaeogenomicsts first demineralise the bone to release the DNA and then then degrade the remaining more robust molecules in the sample (typically just protein) before purifying the DNA.

The resulting DNA molecules are quite different from the modern DNA. Rather than well cooked, soft spaghetti structures - ancient DNA molecules are more like extremely overcooked spaghetti. These molecules are highly degraded, broken down to very small fragments, and also often have 'damage' at the ends in the form of 'modified nucleotides' that do not represent the original sequence (see the Introduction to Ancient DNA chapter for more information).
Finally, the small amount of tiny and damaged DNA molecules typically sits in a 'soup' of 'contaminating' high-quality modern DNA from the surrounding burial and storage environment (Figure **??**).

As we will find out in the next section, this short fragments of DNA is not a disadvantage for NGS sequencing, but rather a benefit.

1.4 DNA sequencing

Now that we have our DNA we want to be able to 'read' the order of the nucleotide sequence that makes up the molecule. Sequencing is the processing of converting the sequence of the four chemical nucleotides to the `ACTG` you can see on your computer screen. The result of this process is the essentially what

Figure 1.6: © Lucy Reading-Ikkanda for The Scientist Magazine, June 2015 (reused here with permission). Very small fragmented 'endogenous' DNA is preserved in bone, but alongside more recent and modern contaminating microbial and recent human DNA

we do pretty much all of our analyses on in genetics and genomics.

The core concept of most NGS sequencing methods [1] is to replicate the strand, but when adding a nucleotide, you instead supply a modified nucleotide with a fluorophore attached to it. This fluorophore is a small molecule that emits a specific colour when excited by a laser. As you have four different nucleotides, each will emit a different colour. Therefore if you record the colour emitted light each time the modified nucleotide is added to a strand, you can reconstruct the sequence of the DNA molecule (Figure **??**).

1.4.1 Sanger sequencing

Historically, the first mass-production sequenced method was Sanger sequencing (Figure **??**) (Sanger, Nicklen, and Coulson 1977a). This was the method used for sequenced the first human genome under the Human Genome Project (Heather and Chain 2015)

The concept of Sanger sequencing involves making lots of copies of a single DNA molecule using replication. But when adding the nucleotides, you include within the mixture of (normal) free nucleotides, you include a few modified nucleotides that include fluorophore, but also that also have a 'blocking' component that prevents any additional nucleotides from being added to the strand of that *particular copy* by the polymerase. Critically, the point at which a blocking nucleotide is incorporated is random. As there is only a small proportion of blocking nucleotides in amongst pool of free nucleotides,the incorporation of the blocking nucleotides occurs at a slow rate and at different points during the replication process. This results in many replicated molecules but all of different lengths (Figure **??**).

Once you have the replicated molecules, Sanger sequencing involves running the mixture through a capillary gel. As all the molecules are of different lengths, they will move through the gel at different speeds, and therefore separate out. A laser is then fired at the gel at regular intervals, and the light emitted from the fluorophores is detected. Shorter molecules will move faster through the gel, and therefore will be detected first, whereas longer molecules will be detected later. Therefore, by detecting the light emitted as the incrementally longer molecules pass through the gel, you can reconstruct the sequence of the nucleotides along the module.

The result is a chromatogram, which is a graph that records the strength of each colour at each point in the gel band (Figure **??**).

While this method was used for the first sequencing human genome, it was very slow and very expensive - it does not scale well due to having to make many copies of a single molecule and sequence each nucleotide sequence one at a time.

Furthermore, the method requires a large amount of starting DNA. However

[1] Newer techniques such as Nanopore sequencers use different methods.

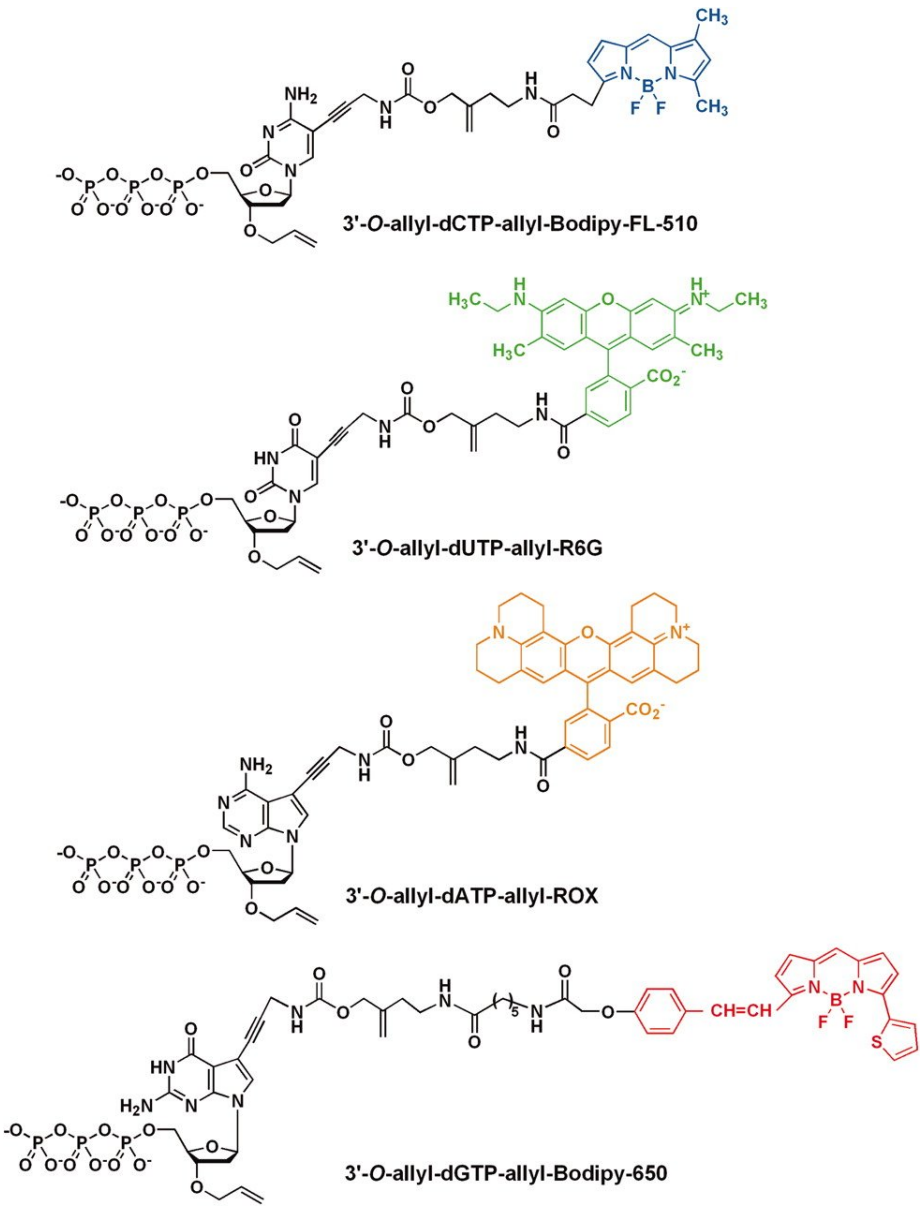


Figure 1.7: Molecular diagram of the four DNA nucleotides with coloured fluorophore amine groups. Source: (Ju et al. 2006)

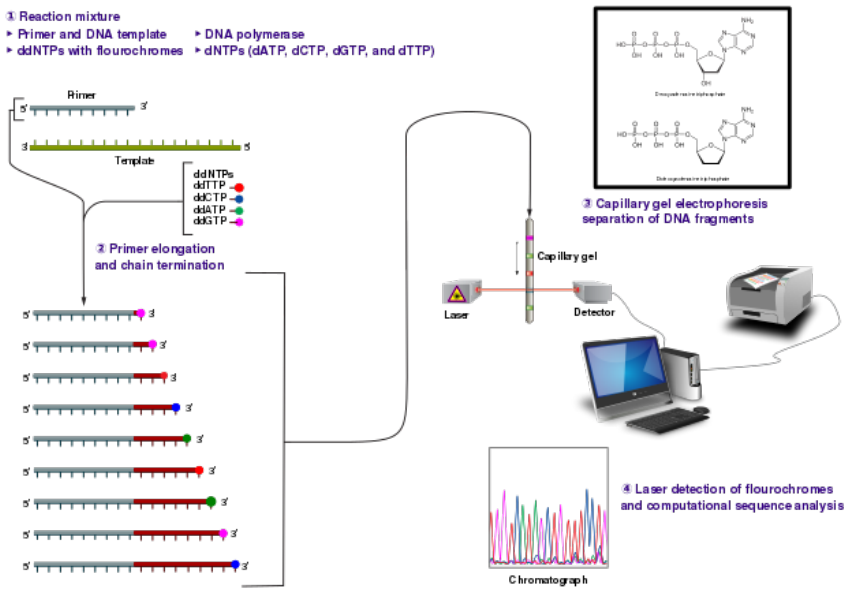


Figure 1.8: Diagram of sanger sequencing, with a primer, template and free blocking nucleotides being added to ends of molecules at random lengths, and then being sent through a size-selecting capillary with a laser dector to identify which of the four blocking nucleotides are passing through the capillary and thus ready ina chromatopgraph. Source: Estevezj, CC BY-SA 3.0 Unported via Wikimedia Commons

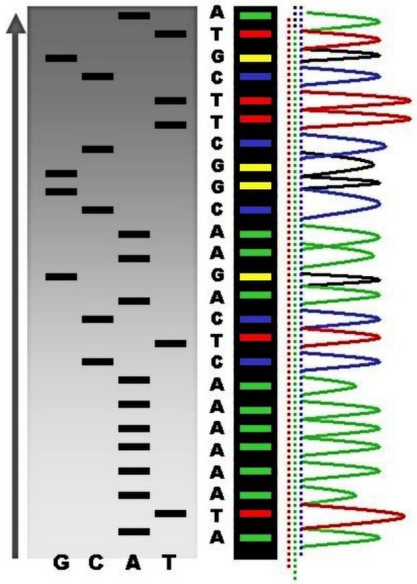


Figure 1.9: Example of gel-based chromatogram with four columns with bands indicating which colour is the blocking nucleotide of that particular sized DNA molecule, and on the size an intensity graph of each colour given off by each flourophore to indicate which colour it is. Source: Morse Phoque (Abizar), CC BY-SA 3.0 Unported via Wikimedia Commons

due to the degraded nature of ancient DNA, it is very rare to have sufficient concentration of DNA to be able to use Sanger sequencing.

1.4.2 Next generation sequencing

However, in 2005, the first 'Next Generation Sequencing' (NGS) machines were released [2] in the form of Roche's 454 Pyrosequencing (Heather and Chain 2015).

These new sequencers were able to quickly and cheaply sequence millions of DNA molecules at once, revolutionising genetics and enabling 'accessible' genomics to a much wider set of scientists.

While a few companies emerged with their own NGS machines, such as PacBio and IonTorrent, sequencers from Illumina have been by far the been the most successful. These machines have been so successful, they have almost exclusively sequenced the entire corpus of ancient DNA sequence data. Therefore the rest of this chapter will focus just on Illumina sequencing.

1.5 Illumina sequencing

Illumina sequencing relies on a process termed 'Sequencing by synthesis' (SBS) using reversible terminators (Bentley et al. 2008). This follows a similar concept as Sanger Sequencing in that you add modified nucleotides with fluorophores, however instead of using multiple molecules with different blocking separated via gel, it uses nucleotides that can be reversibly blocked and thus continue incorporating nucleotides in a single molecule.

This works by immobilising the DNA so it stays on one location on a special type of plate. The machine then performs out a typical DNA replication, adding the flourophore modified nucleotides one at a time, and then taking a photo of the emitted light. However the key difference is the blocking part of the oligo is removed, so the next modified oligo is added.

To visualise how this looks like, you can imagine a plate upon which you see millions of coloured tiny dots, each cycling in realtime through different colours as each new nucleotide is added.

To follow this however, the DNA needs to be prepared in a specific way.

1.5.1 Flow cells and adapters

To ensure the DNA molecules do not move while taking photos, in Illumina sequencing, the DNA molecules are bound to a special glass slide called a flowcell. This also ensures the DNA does not get washed away during the sequencing process as different solutions wash through the machine between each step.

[2] With the more recent emergence of Nanopore sequencing, really NGS should be equated to 'second generation sequencing', a term you will see in some contexts.

To immobilise the DNA, the flow cell is coated with a 'lawn' of synthetic oligonucleotides (i.e., custom sequences made in a lab). By attaching (ligating) the *complementary* sequence of the synthetic oligonucleotides to your DNA molecules from your sample, when your DNA molecules flow over the lawn, the complementary bonds between the nucleotide pairs will form. This then will hold the DNA in place.

The process of adding the complementary synthetic oligonucleotides (often referred to as oligos), called 'Adapters', is called 'Library preparation'. These adapters sequences do not only include the complementary sequence to flowcell oligos, but also an additional sequence that acts as a priming site for polymerases to start the replication process Furthermore, researchers have exploited the highly-multiplex nature of Illumina sequence to allow them to sequence multiple samples at once. They do this by adding additional sequences to the adapter sequence construct that consist of short sample-specific 'indices' or 'barcodes'. These indices allow DNA molecules from different samples to be sequenced at the same time and then separated back into the groups of molecules from the same samples during data analysis. .

In the image above

1.5.2 Clustering

Now that each DNA molecule is bound to it's own section of the flowcell, we have the problem that the light emitted from a single tiny fluorophore will not be strong enough to be captured by the camera in the sequencer.

Illumina machines get around this through a process called 'clustering'. The clustering process will make many copies of the exact same DNA molecule right next to the original template molecule. When the cluster of DNA copies emit the light from the same nucleotide along their sequence at the same time, collectively the light emitted will now be sufficiently strong to be captured by the camera.

To make copies sufficient copies of the 'template' molecule, clustering is carried out using 'bridge amplification' (Bentley et al. 2008).

The process of bridge amplification is as follows:

1. A single-stranded DNA molecule is immobilised on the flow cell via the oligo adapter on *one* end of the module
2. The DNA is bent over
3. The oligo adapter on the other end of the molecule binds to the other complementary flow-cell oligo adapter in the close vicinity to the template (forming a bridge)
4. A priming sequence, free nucleotides, and a polymerase is added to the flow cell
5. The reverse complement of the DNA molecule is made via replication, and the polymerase mixture is washed away

6. The two strands are then separated to form two single-stranded DNA molecules (the new one the reverse complement of the original template molecule)
7. Repeat 1-5 for the two new strands until sufficient copies are made

This process is also depicted in Figure **??**.

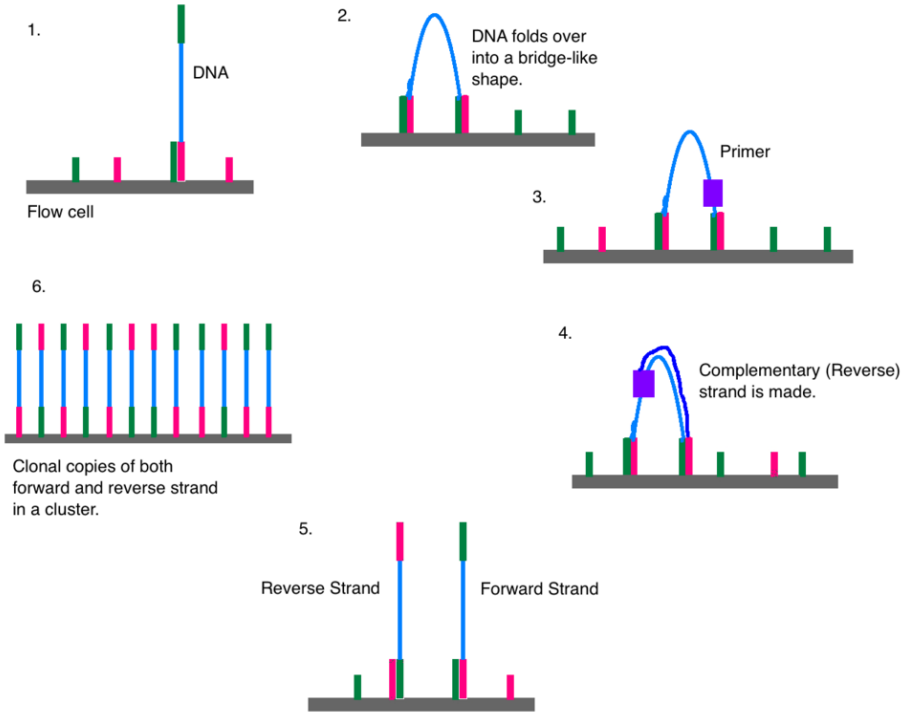


Figure 1.10: Diagram of bridge amplification. Source: DMLapato, CC BY-SA 4.0 International via Wikimedia Commons

Of course, at the stage that you've had many cycles of replication, you have in a single cluster both forward and reverse complement copies of the same DNA molecule. As this would emit different colours at each position along the sequence, one of the strand directions are removed but 'trimming' the flow cell lawn at one of the complementary oligo adapter sequences

1.5.3 Sequencing-by-synthesis

With our cluster of many copies of the same DNA molecule, we can now start the 'Sequencing by synthesis' (SBS) process itself.

As with replication and Sanger sequencing, the process SBS involves adding free nucleotides to the template strand (with only the complementary base being able

to be incorporated to the exposed strand), some of which are modified to emit light when excited with a laser, and taking a picture.

The main difference with Sanger sequencing is you can actually reuse the same DNA molecule to add more nucleotides along the same strand - rather than 'discarding' it with permanently blocking nucleotides. This makes the SBS technique more resource sufficient, both in reagents (you need fewer copies of the molecule and recycle the same original copies of template molecules), but also in space - you don't have to separate out the molecules in a gel - they are all fixed on the flowcell in a tightly packed layout.

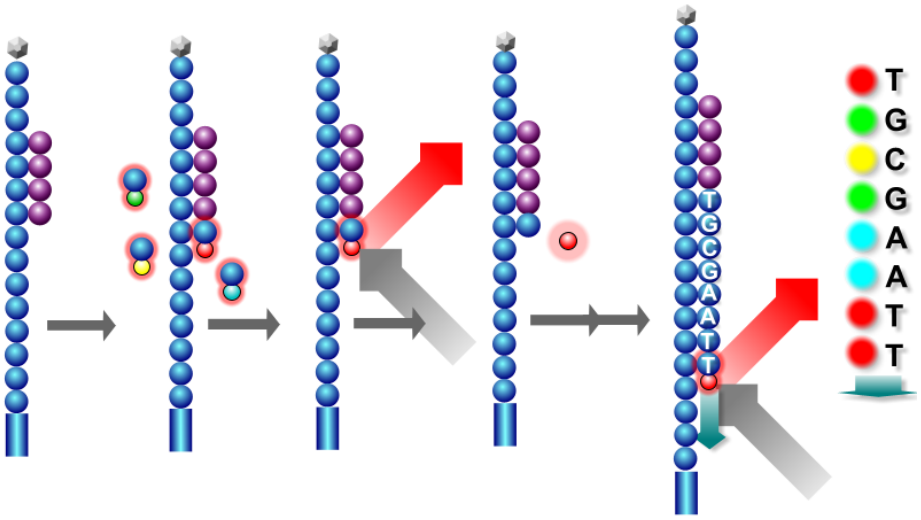


Figure 1.11: Diagram of sequencing by synthesis. Source: Abizar Lakdawalla, CC BY-SA 3.0 Unported via Wikimedia Commons

The process as in Figure **??** can be broken down to:

1. On a single stranded molecule, bind a primer on the molecule's adapter priming site
2. Add fluorescently labelled nucleotides (with a reversible terminator) to the flow cell
3. Only complementary nucleotides will bind to the template strand at the first exposed base
4. Wash away unbound nucleotides
5. Fire a laser to excite the modified nucleotides to emit the corresponding colour, and take a picture
6. Clip off the fluorophore part of the nucleotide
7. Repeat 2-6 until the entire strand is sequenced

On Illumina sequencers, the number of repetitions (known as cycles) and images typically happens either 50, 75, or 125 times, depending on the machine and

the type of sequencing chemistry kit.

The reason why Illumina sequencers (and similar NGS technologies) have been so successful is that this process is happening across millions of clusters at the same time.

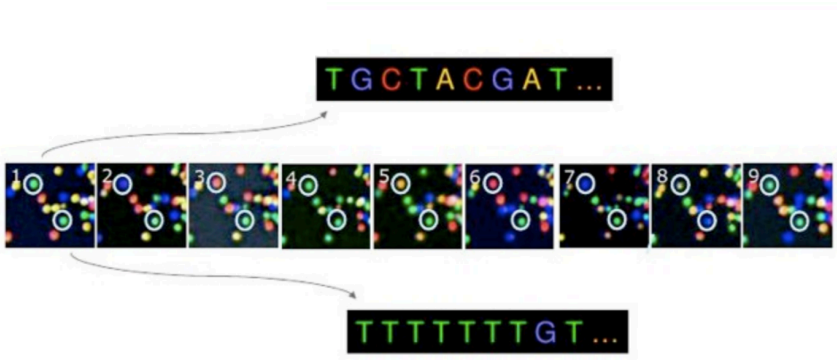


Figure 1.12: Diagram of sequencing by synthesis. Source: EBI Training, CC BY 4.0 via EBI Training

You can see a small fraction of this flow cell in Figure **??**. Each coloured dot corresponds to a cluster of DNA molecules. At each cycle (each photo), a new nucleotide is added to the strand, and a laser is fired to excite the fluorophores. You can see two different clusters emit different lights, as they are different DNA molecules thus have different nucleotides at that particular 'cycle' of the 'replication' process. By converting the emitted light to the known corresponding A, C, G, T, at each photo you can reconstruct the sequence of the DNA molecule.

By now you probably are starting to feel comfortable with the concept for one-colour equals one nucleotide. To throw a curve ball, Illumina sequencers actually have *three* different 'colour chemistries' they use on different machines.

1.5.4 Colour chemistry

To be able to offer cheaper machines, Illumina came up with a system that requires the number of colours that need to be detected. These variants are often referred to different 'colour chemistries'.

A list of Illumina platforms and their colour chemistries as follows below:

- Four-colour chemistry
  - Illumina HiSeq
  - Illumina MiSeq
- Two-colour chemistry
  - Illumina NovaSeq
  - Illumina NextSeq

- One-colour chemistry
  - Illumina iSeq

In the vast majority of cases, your data will be sequenced on only four- and two-colour chemistries, so we will focus on these two.

In the case of the Illumina HiSeq and MiSeq platforms, the system for sequencing is as described previously: each of the four bases of the nucleotides that make up DNA will have a reversible-terminator variant, and each of these variants will have a fluorophore that will emit one of four colours. Therefore, the camera on the machine is designed to be able to pick up each of the four different wavelengths through different imaging channels.

In contrast, two-colour chemistry machines will only have imaging channels that detect have two dye colours - red and green If the machine detects green being emitted from a cluster on the flowcell, this correspond to T. If the machine detects red, it records a C. If the machine detects *both* red and green wavelengths being emitted from a cluster - this corresponds to an A. If no colour is emitted, this is *assumed to be a G*.

This characteristic of two colour-chemistry thus means you must be aware of the type of sequencer your data was generated on, as this will require slightly different preprocessing as we will see later on in the section Low sequence diversity.

1.5.5 Base quality and paired-end sequencing

Another consideration with Illumina sequencing kits is not just the number of cycles, but which kit to use based on its 'pairment'.

As I'm sure everyone readying this book has experienced, biology is not like physics where everything is perfect and 'easy'. Biology is messy, things do not stay stable. The same applies to sequencing - as the sequencer goes through each cycle of adding a new nucleotide to each DNA strand in a cluster, mistakes will increasingly start happening as polymerases don't bind properly, fall off, skip nucleotides, or even add the wrong base. The accumulated effects of these mistakes across all DNA strands in a cluster is that the emitted light during the imaging part of the cycle becomes less clearly one colour as mixtures of wavelengths are emitted. Thus, the confidence of the *correct* nucleotide has been recorded at that position of the DNA molecule becomes less.

(Illumina) sequencers address this in two ways.

Firstly, when measuring the particular base being recorded, all sequencers will at the same time record a 'base quality' that is associated with that call. This corresponds to the confidence (via a probability) that the recorded base was the correct one. For example, if the wavelength was mostly one colour (let's say red for C) but the camera picked up a bit of another colour (lets say green for T), it will record C in the output, but the base quality score will be lower. If

the sequencer's camera picks up wavelengths of all four colours from the same cluster at the same time - or even no light is detected at all - then the base quality for that position in that DNA molecule will be recorded as an N and a base quality of 0.

Secondly, specifically for Illumina sequencers, there is the concept of 'paired-end' sequencing. As there is an overall drop in base quality (i.e., confidence we have the right nucleotide) the closer we get to the end of the molecule due to the accumulated mistakes over time, we can freshly sequence the molecule again but in the *reverse* direction. This way, we can 'merge' or average the base quality scores of two independent 'observations' of the nucleotide at that position. So if the last base when reading the molecule in the forward direction is an N (due to ambiguous wavelength emission), we can correct or 'recover' this last base by making it the *first* base we call in when sequencing in the reverse direction from scratch with fresh reagents.

This works as, Illumina sequencers can conveniently re-use the bending over approach as performed by the 'bridge' amplification method of clusterting we saw earlier in this chapter. I.e., after bending over, the opposite flowcell olgio from the forward reverse flowcell anchor is cut so the 'end' of the original molecule is now bound to the flow cell.

This paired-end sequencing technique means you get a pair of files - both with the same sequence from each cluster, but one has the molecule 'read' in the forward direction, and the other file with the 'reverse' direction.

Another added benefit of paired-end sequencing is that you can recover longer molecules. In cases where the DNA molecule is *longer* than the number of cycles the chemistry kit being used, by flipping the over the molecule and sequencing from the other end you can recover the end of the molecule. Of course if the molecule is longer than the sum of both set of cycles (e.g. a 120bp molecule, even though the sequencing kit is only two times 50bp), you will miss the 'middle' part of the molecule. However this is counteracted by the sheer number of independent molecules sequenced, so the 'missing' part can be filled in from another sequencing DNA molecule.

It should be noted, however, the added benefit of sequencing longer molecules via paired-end sequencing is not so relevant to ancient DNA or ancient metagnomics. This is because most 'true' DNA molecules are often *shorter* than the sequencing cycles of a typical Illumina kit.

1.5.6 Base calling and demultiplexing

We should now understand how the machine itself 'reads' the DNA molecule clusters on the flow cell.

But how does these images get converted to a format that you can read on your computer screen? And if you have sequenced multiple samples at the same time (each with their own sample-specific indicies), how do you separate these out?

Base calling is the process of converting the images to digital text-based `A`, `C`, `T`, and `G`s. This is not something the vast majority of researchers have to do as it nowadays happens on the sequencer itself and thus the process is not particularly relevant for us.

However, once the file with the digital representations of the sequences is taken off the machine, typically a sequencing technician or bioinformatician at the sequencing center, but sometimes a student, will perform something called 'demultiplexing'. This is where the person doing the demultiplexing will run the file through a tool that identifies the known index sequence at the beginning of each read entry in the file, and put all reads containing the same sample-specific index into it's own independent file.

While this is not something most researchers will do themselves, it is useful to know as errors detected in downstream analysis can be traced back to suboptimal demultiplexing (such as incorrect assignment of indices, if you have not the number of sequenced reads you expected).

1.5.7 FASTQ File

The primary output from demultiplexing is called FASTQ file. This is a text-based (semi-)standard format for storing biological sequences, and their corresponding base quality scores. These files are typically compressed to save space, but can be very large (gigabytes in size, even when compressed).

The structure of a FASTQ file is a repeated set of four specific lines:

1. A metadata or ID line
2. The sequence itself
3. A 'spacer' line
4. The base quality score (corresponding to the base in the same position as in line 2)

Let's break this down.

The metadata line records a range of information from the sequencer and also potentially the demultiplexing. Typicall this line will record the sequencing machine ID, a run number, the ID of the flow cell, coordinate information from which cluster on the flow cell the particular sequence is from, and then extra information that depends on the sequencing center (e.g., some will record the sample-specific index pairs here, or certain settings of the demultiplexing tool).

The sequence line will contain `A`, `C`, `T`, `G`, and `N` characters in the order of the DNA molecule as recorded by the sequencer. In the figure you can see an example of an `N` call in the first position. It is actually quite common to see in the first or second position of reads, as the sequencer's camera is still calibrating itself.

The third

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 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.

1.6 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.

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

1.7.1 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.

1.7.2 Index hopping

Another thing during sequencing, you have to consider is something called index hopping. So this is a problem, particularly with Illumina Sequ, 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, 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.

1.7.3 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.

1.7.4 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 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.

1.7.5 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, 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.

1.7.6 Q and A

1.7.6.1 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.

1.7.6.2 How many indicies 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.

1.7.6.3 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 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 bbases, 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.

1.7.6.4 What is full genome seuqencing

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.

1.7.6.5 Tools for generating indicies

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.

1.8 Readings

1.8.1 Reviews

(Schuster 2008)

(Shendure and Ji 2008)

(Slatko, Gardner, and Ausubel 2018)

(Dijk et al. 2014)

1.8.2 Sequencing Library Construction

(Kircher, Sawyer, and Meyer 2012)

(M. Meyer and Kircher 2010a)

1.8.3 Errors and Considerations

(Ma et al. 2019)

(Sinha et al. 2017)

(Valk et al. 2019)

1.9 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?

1.10 References

Chapter 2

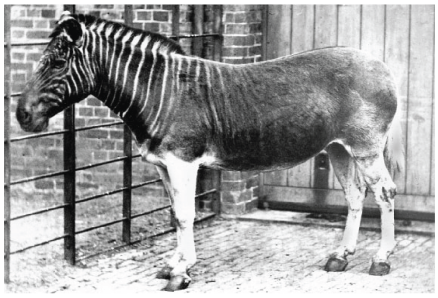
Introduction to Ancient DNA

The field of palaeogenomics focuses on the study of ancient DNA, and traces its origins to the 1980s. This chapter provides a brief overview of the methodological history of the field and reviews the fundamentals of how ancient DNA is recovered, sequenced, and authenticated today. Emphasis is placed on how DNA degrades and how DNA degradation patterns are used to help authenticate ancient DNA at the level of sequences, genomes, and samples, as well as how DNA damage creates both opportunities and downstream challenges when genotyping or generating phylogenies.

2.1 Ancient DNA beginnings

The origins of the field of palaeogenomics can be traced back to 1984 with the publication of the first ancient DNA study by Russell Higuichi and colleagues (Higuchi et al. 1984). Working in Allan Wilson's lab at the University of California at Berkeley, Higuchi investigated DNA from a museum soft tissue specimen of the quagga, a subspecies of zebra that had gone extinct in the late 19th century (Figure **??**). Compared to today, 1984 was a very different world in terms of technological capabilities within biology and genetics. To help put this into perspective, Figure **??** shows the DNA sequences that were published in this first paper. They consist of partial DNA sequences from two mitochondrial genes – cytochrome oxidase I and a second unidentified gene that was later determined to be NADH dehydrogenase I. Sequencing these two short segments of DNA required an enormous amount of time and effort to achieve, and all of the work was analog. In fact, most of the digital tools we take for granted today did not yet exist in 1984. Journals did not have websites, manuscripts were submitted by post, and genetic data was mostly tabulated and analysed

by hand. Microsoft Excel, which is today a kind of ubiquitous software for basic spreadsheet data manipulation, wasn't even invented and released until 1985 (for Mac) and 1987 (for Windows).



Unidentified reading frame 1

```
Quagga    C CCA ATC CTG CTC GCC GTA GCA TTC CTC ACA CTA GTT GAA CGA
Zebra     . ... ... ... ..T ... ..G ... ... ..T ... ... ... ... ...

Quagga    AAA GTC TTA GGC TAC ATA CAA CTT CGT AAA GGA CCC AAC ATC
Zebra     ... ... ... ... ... ... ... ... ..C ... ... ... ... ...

Quagga    GTA GGC CCC TAT GGC CTA CAA CCC ATT AC
Zebra     ... ... ... ... ... ... ... ... ... G.
```

Cytochrome oxidase I

```
Quagga    A GGA GGA TTC GTT CAC TGA TTC CCT CTA TTC TCA GGA TAC ACA
Zebra     G ... ... ..T ... ... ... ... ... ... ... ... ..G ... ...

Quagga    CTC AAC CAA ACC TGA GCA AAA ATT CAC TTT ACA ATT ATA TTC
Zebra     ... ... ... ... ... ... ... ..C ... ... ... ... ... ...

Quagga    GTA GGG GTC AAC ATA ATT TTC TTC CCA
Zebra     ... ..A ... ..T ... .C. ... ... ...
```

Figure 2.1: The first ancient DNA sequences. In 1984, ancient DNA was successfully sequenced from a museum specimen of muscle tissue from a quagga, an extinct subspecies of zebra. Short DNA sequences from two genes were obtained using Sanger Sequencing, resulting in a total of 226 bp of reconstructed quagga DNA. Quagga image by Frederick York, 1870, Wikipedia, public domain. DNA sequences from (Higuchi et al. 1984).

This first ancient DNA study, published at the very beginning of the era of genetic sequencing in the journal *Nature*, illustrates what an important achievement it was in the 1980s to be able to sequence even this small amount of quagga DNA. However, the methods they used are very different from the methods we use today. Higuchi and colleagues achieved the genetic sequencing reported in their paper using a technique called primed-synthesis dideoxynucleoside chain-termination sequencing, a method that had been developed by Frederick Sanger and colleagues only seven years before in 1977 (Sanger, Nicklen, and Coulson 1977b). Sanger's method, which came to be known as **Sanger Sequencing**, was the first truly successful and useful method for sequencing DNA, and it became the primary method of DNA sequencing for the next 35 years. This was a breakthrough moment in the history of genetics, and it is worth exploring how the method works.

In the previous chapter, you learned some of the basic concepts of DNA sequencing that are used for **next generation sequencing (NGS)**. NGS builds upon the basic principles of the original Sanger method, and so it is worth examining the Sanger method in more detail here. The Sanger method works by using a DNA polymerase to copy a large pool of identical DNA templates using a mixture of ordinary nucleotides and nucleotides containing a blocking component. As the polymerase extends along each DNA template molecule, it incorporates the available nucleotides, and when it incorporates a blocking nucleotide it prematurely stops. The random incorporation of blocking nucleotides at different points along the template molecules results in DNA of different extension lengths, which can then be separated and ordered by size using gel elec-

trophoresis. In the original Sanger method, each round of sequencing required four separate reactions - each containing blocking nucleotides for a different base. Following gel electrophoresis, the different banding patterns of the four reactions could then be used to determine the base present at each particular position in the sequence, and thus determining the overall DNA sequence of the template.

Figure **??** shows an example of how this works from another early ancient DNA study by Matthias Höss and Svante Pääbo on a different extinct equid (Höss and Pääbo 1993). The bands are "read" from bottom to top, which represents the 5' to 3' orientation of the template molecule. The four lanes of the gel represent the four separate polymerase extension reactions, each containing a mixture of ordinary nucleotides and blocking nucleotides corresponding to the base annotated at the top of each lane. The shortest DNA sequences travel fastest through the gel and are visualised as bands at the bottom of the gel; these represent the beginning of the DNA sequence. The longest DNA sequences travel more slowly through the gel and are visualised as bands at the top; these represent the end of the DNA sequence. Determining the DNA sequence was initially an entirely analogue process generally carried out with a ruler and pencil. Band by band, the bases would be recorded from bottom to top, resulting in a consensus sequence. This process, setting aside the prior laboratory work necessary to produce sufficient template DNA for sequencing, would itself have taken a full workday in order to set up the reactions, perform the extensions, run the gel, image the gel, develop the film, and then manually read out the DNA sequences. Although automated sequencing and basecalling first became available for the Sanger method in 1987 with the release of the Applied Biosystems ABI 370 instrument, which utilised fluorophores instead of radioactive molecules to detect the DNA bases (making it safer) and a personal computer for digital basecalling (making it faster), analog methods continued to be widely used well into the 1990s.

Remarkably, the original quagga sequences were obtained using Sanger Sequencing without the use of **polymerase chain reaction (PCR)**. Although PCR was invented by Kary Mullis in the 1980s (Saiki et al. 1985, 1988), it did not become widely accessible until the early 1990s (Bartlett and Stirling 2003; Mullis et al. 1992). Thus Higuchi and colleagues achieved the first ancient DNA sequences through even more laborious methods based on restriction enzymes and bacterial cloning. In total, their study reports two ancient DNA sequences, one 114 bp long and the other 112 bp long, from one ancient sample. Now contrast this to today, when it is possible to routinely generate >50 billion high quality ancient DNA sequences every 48 hours on an Illumina NovaSeq X instrument (https://emea.illumina.com/systems/sequencing-platforms/novaseq-x-plus.html).

Ancient DNA sequencing has come a long way since the 1980s, and grappling with these changes is the rationale for developing this textbook and companion course on Ancient Metagenomics, which focuses on bioinformatic coding and scripting. During the Sanger sequencing era, from the 1980s to the 2000s, the

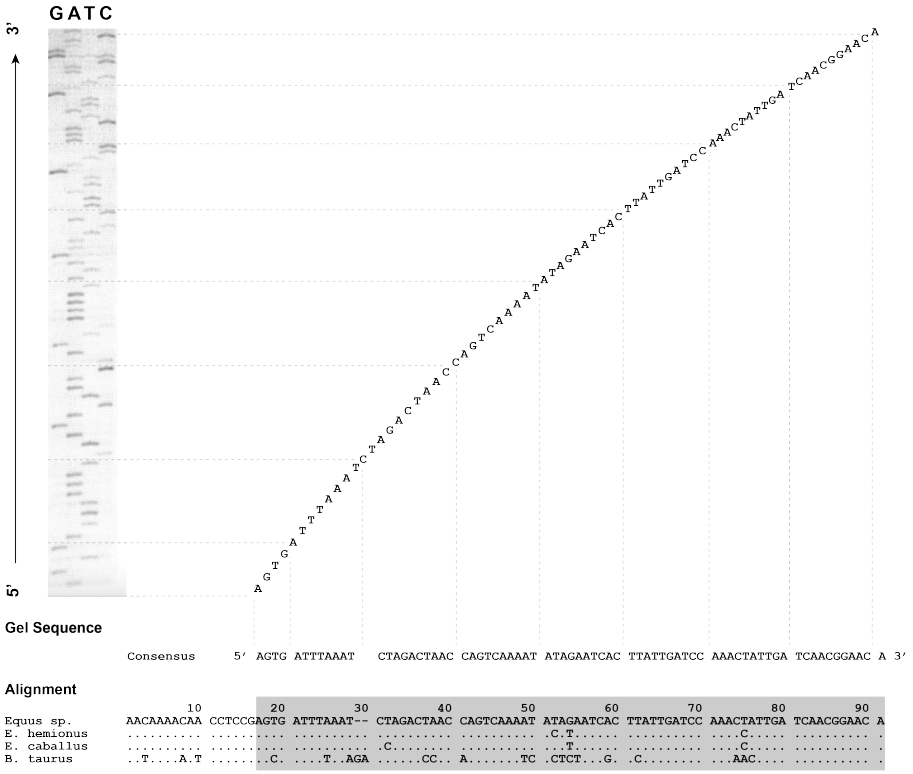


Figure 2.2: Example of ancient DNA sequenced by the Sanger method. In 1993, mitochondrial DNA was successfully sequenced from a 25,000-year-old horse bone using Sanger Sequencing. The portion of the DNA sequence alignment visible on the gel is highlighted in gray. Adapted from (Höss and Pääbo 1993).

vast majority of the hours of work that went into ancient DNA analysis was in the laboratory and consisted of extensive wet lab preparation of samples to reach the point where a relatively short DNA sequence could be generated. It is no wonder then that this era demanded intensive training in biochemistry and molecular biology. The situation is very different today, in which the laboratory methods have become highly standardised but the data output is enormous. Since the rise of high-throughput sequencing in 2010s, the main challenge for ancient DNA researchers has become how to manage this deluge of genetic data, and this requires specialised computational and bioinformatics skills to be able to handle, process, and interpret the vast amounts of data that make up today's ancient DNA datasets.

2.2 From quagga to ancient microbes

How did we get from quagga to ancient microbes? It has been an incredible journey, made all the more remarkable by the fact that nearly everything we now know about ancient microbes has been achieved through the study of ancient DNA. The rest of this chapter will focus on how genetic technologies are used to study ancient microbes. But first, let's start with some important questions: Where exactly does one find the DNA of ancient microbes? How does one archaeologically recover organisms that are too small to see? It turns out there are several reliable sources of ancient microbes that have proven to be productive and informative about the past. Here, we will review six sources that are particularly important in archaeology: teeth, bones, historic medical specimens, palaeofaeces, cultural objects, and sediments (Figure **??**).

2.2.1 Teeth

Perhaps the single greatest source of ancient microbial DNA is teeth. As the only part of the skeleton normally visible during life, teeth are also the only part of the skeleton typically colonised by microbes. Figure **??** shows the mandible (lower jaw) of a woman who died around a thousand years ago at the medieval monastic site of Dalheim in Germany (Warinner et al. 2014; Radini et al. 2019). Several important microbially-associated features are visible. First, there are mineral accretions on the surface of her teeth. These accretions are called dental calculus, and they form through the periodic calcification of dental plaque (Fagernäs and Warinner 2023; Warinner, Speller, and Collins 2015). Below the calculus, the alveolar bone (which holds the teeth) is thickened, recessed, and porous - indications that the periodontium (the soft and hard tissues supporting the teeth) was chronically inflamed by dental plaque bacteria during life. Left untreated, this inflammation has caused partial destruction of the alveolar bone, a medical condition known as periodontitis. Thus, even from a simple visual inspection we can already see signs of ancient microbial activity that occurred during this woman's life.

Figure **??** shows a closer view of one of the teeth in cross-section using scanning

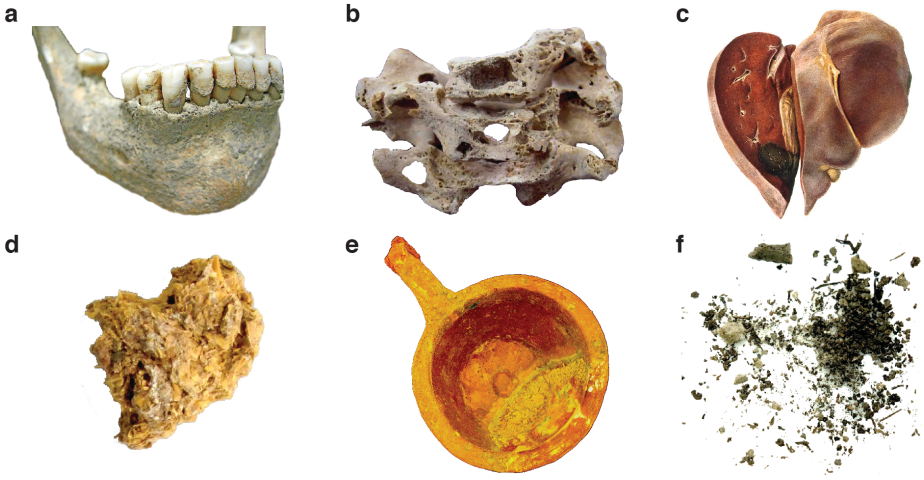


Figure 2.3: Sources of ancient microbial DNA. (a) teeth; (b) pathological bone; (c) historic soft tissue medical specimens; (d) palaeofaeces; (e) cultural artifact residues; (f) sediments. Images a-b and d-f are courtesy of the author. Images a-b, d-f are courtesy of the author Christina Warinner. Image c depicts a detail of an 1897 medical illustration of liver hepatitis; the image is licensed under a CC-BY 4.0 license and comes from Wellcome Images (image 577059i), a website operated by Wellcome Trust, a global charitable foundation based in the United Kingdom.

electron microscopy (SEM). There are three main sources of microbial DNA that can be obtained from this tooth. On the surface of the enamel, which appears as white, the mineral accretions of dental calculus are visible. Within the dental calculus are thousands, if not millions, of individual bacterial cells that were calcified in situ (Figure **??**, panel A). They are, in a sense, frozen in time, along with their DNA (Fagernäs and Warinner 2023). The process by which dental plaque calcifies into dental calculus happens at regular intervals over an individual's lifetime, building up a calcified biofilm that can become several millimetres thick. This process occurs during life, and in fact dental calculus is the only part of your body that routinely fossilises while you are still alive. What you see in Figure **??** are the calcified remains of the oral microbiome, and specifically the dental plaque bacteria that grow on the surfaces of teeth. Although most bacteria within dental calculus are commensals, respiratory and other pathogens affecting the oral cavity have also been identified within calculus, including Klebsiella pneumoniae (R. M. Austin et al. 2022) and Mycobacterium leprae (Fotakis et al. 2020).

Figure **??** (panel B) shows a close-up view of the pulp chamber of the tooth. During life, the pulp chamber of the tooth is vascularised. There are blood vessels that run through the root canal and feed the dental pulp, which is at the center of each tooth. This connects the teeth to the broader circulatory system. During life, if a person has a blood-borne infection, any pathogens that are circulating in the blood will also circulate through the teeth in the dental pulp chamber. If this person dies while the infection is active, the pathogens will remain in the tooth and decay in place, and their DNA will end up becoming smeared along the walls of the dental pulp chamber. The dental pulp chamber is one of the best known sources of DNA from pathogens involved in infectious blood-borne diseases around the time of death (Spyrou et al. 2019). Several pathogens have been identified archaeologically by analysing ancient DNA in the dental pulp chamber, including Yersinia pestis (Bos et al., 2011), Salmonella enterica Paratyphi C (Vågene, Herbig, Campana, Robles García, et al. 2018), smallpox (Mühlemann, Vinner, Margaryan, Wilhelmson, Fuente Castro, Allentoft, Barros Damgaard, Hansen, Holtsmark Nielsen, et al. 2020), and hepatitis B virus (Kocher et al. 2021).

Figure **??** (panel C) shows a part of the tooth dentine that is actively undergoing decomposition. This decomposition appears as discolouration in the SEM, and tissue degradation gives the dentine a ragged appearance. Such changes are caused by the activity of the necrobiome, the bacteria that contribute to the decay and decomposition of the teeth. Within the figure, many small dark bacterial cells are visible. They are dark in colour because they are not mineralised, and they are not mineralised because they are alive. These are the many living bacteria that are slowly breaking down and decomposing the tooth. Most necrobiome bacteria are environmental bacteria that originate from the burial sediments (Warinner et al. 2014), but in some cases oral bacteria can also contribute to decomposition (Mann et al. 2018). One thing that is important to keep in mind about the necrobiome is that these bacteria can also be quite

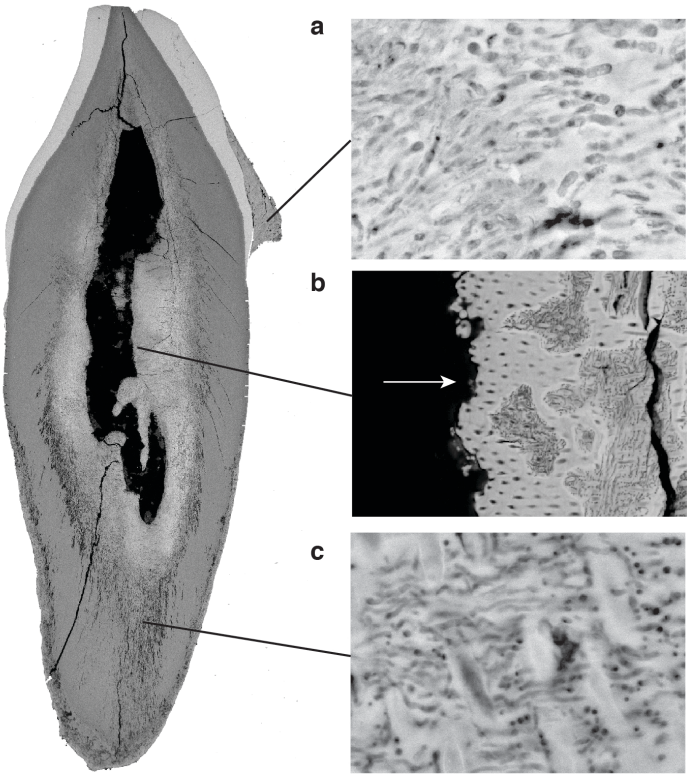


Figure 2.4: Archaeological tooth in cross-section and magnified views of different sources of microbial DNA. (a) Semi-fossilised oral microbiome bacteria in dental calculus; (b) pathogen DNA preserved on the walls of the dental pulp chamber; (c) necrobiome bacteria contributing to the decomposition of tooth dentine. Images produced using scanning electron microscopy (SEM) (Warinner 2022).

old. Bacteria invade the body and start to break it down within hours or days of death. When these bacteria die, their DNA will also accumulate damage and appear ancient. When analysing the necrobiome, it is typical to observe a mixture of DNA from bacteria of different ages. Some of the necrobiome DNA will originate from dead bacteria that contributed to the early stages of decomposition, and these dead bacteria may be nearly as old as the tooth itself and will show signs of DNA damage. Other necrobiome DNA will originate from living bacteria, including those that have more recently colonised the tooth.

2.2.2 Bone

Bone, and especially pathological bone, is another important source of ancient microbial DNA (Figure **??**, panel B). Some diseases that infect the skeleton, especially those that are chronic and progress over many months or years, may cause lesions or other characteristic alterations in bone. Chronic pulmonary infections, such as tuberculosis, can escape the lungs and infect neighbouring tissues, forming lesions on the interior of the ribs or infecting the thoracic vertebrae. Invasion of the vertebrae by the bacterial pathogen Mycobacterium tuberculosis can cause large lesions and cavities to form within the vertebral bodies (Figure **??**, panel B), which weakens and destabilises the spine. This can then result in the vertebrae collapsing forward, a condition called kyphosis. Kyphosis is responsible for the characteristic hunching of the back that accompanies advanced spinal tuberculosis, and it is depicted in a wide range of ancient art. Examples of spinal tuberculosis causing kyphosis have been described at the Chiribaya sites along the southern coast of Peru dating to ca. 1000 ya (Bos et al. 2014a). Leprosy is another disease that produces characteristic alterations to the skeleton. Leprosy is caused by Mycobacterium leprae, and as the disease progresses it results in a widening of the nasal aperture, destruction of the bone around the maxillary dentition, and the destruction and loss of bones in the extremities, especially fingers and toes. These features can be recognised osteologically in the archaeological record and have led to the recovery of *M. leprae* DNA from skeletons across Eurasia (Schuenemann et al. 2018).

2.2.3 Historic medical specimens

Historic medical specimens represent a large and varied source of ancient microbial DNA (Figure **??**, panel C). During the 18th, 19th, and early 20th centuries, many hospitals and medical schools amassed large collections of pathological specimens, which range from biopsies to complete organs. Such specimens were typically preserved and stored in alcohol or formalin, although precise documentation of their treatment is often lacking. Specimens from the early and mid-20th century also include histology slides and tissue blocks, often formalin-fixed and paraffin embedded (FFPE). RNA and DNA can be recovered from such specimens, although success rates are often low due to DNA cross-linking caused by the formalin treatment (Gryseels et al. 2020; Stiller et al. 2016). Investigation of historic medical specimens, such as vaccination kits, have led to

important discoveries regarding the diversity of vaccinia virus and orthopoxvirus strains used in 19th century smallpox vaccination efforts (Duggan et al. 2020), and the study of ancient RNA in FFPE tissue blocks has shown that HIV was locally circulating in Central Africa prior to 1960, more than two decades before its global outbreak (Gryseels et al. 2020; Worobey et al. 2008; Zhu et al. 1998).

2.2.4 Palaeofaeces

palaeofaeces are an important source of ancient microbial DNA relating to the gut microbiome and gastrointestinal pathogens (Figure **??**, panel D). faeces do not preserve in most environments, but specific conditions that immobilise or eliminate water, such as freezing temperatures, extreme dryness, high salinity, or mineralization, can lead to the long-term preservation of archaeological palaeofaeces and coprolites (Shillito et al. 2020). Permafrost, dry caves (Wibowo et al. 2021), and salt mines (Maixner, Sarhan, Huang, Tett, Schoenafinger, Zingale, Blanco-Míguez, Manghi, Cemper-Kiesslich, Rosendahl, Kusebauch, et al. 2021) are among the best environments for the long-term preservation of palaeofaeces. In addition to microbial DNA, palaeofaeces are also a good source of dietary and parasite DNA.

2.2.5 Cultural artifact residues

Residues within cultural artifacts such as serving dishes and food and beverage containers are potential sources of ancient culinary bacteria (Figure **??**, panel E). For example, food residues from baskets and ceramic bowls in western China have yielded molecular evidence for a variety of yeasts and lactic acid bacteria involved in the making of dairy products (Xie et al. 2016) and sourdough bread (Shevchenko et al. 2014).

2.2.6 Sediments

Finally, archaeological sediments are a rich source of environmental DNA (eDNA) containing the genetic remains of ancient microbes (Figure **??**, panel F). Sediment samples can be collected through coring, and resin impregnated sediment blocks previously prepared for micromorphology analysis have also been shown to successfully yield ancient DNA (Massilani et al. 2022a). The vast majority of research conducted on ancient sediments to date has focused on plant and mammalian DNA in order to reconstruct ancient ecosystems (Linderholm 2021; Eric Capo et al. 2022a; Kjær et al. 2022a; Zavala et al. 2021). However, interest in characterizing ancient microorganisms, such as plankton, is increasing (L. H. Armbrecht 2020), and the analysis of ancient bacteria is within reach (Fernandez-Guerra et al. 2023), although distinguishing between live and dead pools can be challenging (Ellegaard et al. 2020; Wegner et al. 2023).

2.3 Defining ancient DNA

Having defined the main sources of ancient microbial DNA, we might next ask: What makes something *ancient*? What is "ancient DNA" as opposed to just DNA? What makes it specifically ancient? Ancient DNA can be defined as any DNA from a non-living source that shows evidence of molecular degradation. You will notice this definition does not specify a particular range of time. That is because ancient DNA is not defined by a fixed age, but rather by its condition. For example, 100,000-year-old Neanderthal oral microbiome DNA from dental calculus (Klapper et al. 2023b) would obviously qualify as ancient DNA. And so would 5,000-year-old hepatitis B virus DNA from teeth (Kocher et al. 2021), 3,000-year-old gut microbiome DNA from palaeofaeces (Maixner, Sarhan, Huang, Tett, Schoenafinger, Zingale, Blanco-Míguez, Manghi, Cemper-Kiesslich, Rosendahl, Kusebauch, et al. 2021), 600-year-old plague DNA from skeletons (Bos et al. 2014a), oral bacteria from 19th century great apes in a museum (Fellows Yates, Velsko, et al. 2021), vaccinia virus DNA from 19th century medical specimens (Duggan et al. 2020), and even leprosy DNA from mid-20th century FFPE tissue blocks (Blevins et al. 2020). These are all examples of ancient DNA because they all show similar types of DNA damage that require special handling to be able to analyse. Essentially, ancient DNA is DNA that has undergone specific forms of degradation and damage.

2.4 Genome basics

Before going further, let's first take a step back and ask: Why does this matter? Why is ancient DNA defined by its state of preservation rather than its chronological age? DNA degradation matters when considering the diverse composition, organization, size, and copy number of the genomes we are trying to reconstruct and understand. So before moving on, we will now review some genome basics.

First, let's consider viruses. Viral genomes can be made up of DNA or RNA, and they can be single-stranded, double-stranded, or partially double-stranded. Their genome can consist of a single molecule of nucleic acid or many molecules, and their genomes can be linear or circular. Figure **??** shows the seven different types of viral genomes classified within the Baltimore system of viral classification (Koonin, Krupovic, and Agol 2021). The genomes of viruses are very diverse in structure, organization, and composition.

Bacterial genomes are more consistent in their organization (Bobay and Ochman 2017). A bacterial genome typically consists of one long, continuous circle of double-stranded DNA (Figure **??**), although some bacteria, such as *Streptomyces*, have linear genomes (Galperin 2007; Hinnebusch and Tilly 1993). A bacterial genome is often referred to as a chromosome, and while this terminology is not universally accepted due to differences in the compaction and staining properties of prokaryotic and eukaryotic genomes, the term **chromo-**

some is widely used by large data archives and platforms, including members of the International Nucleotide Database Collaboration (INSDC). In addition to the main genome, bacteria can also have accessory DNA called **plasmids**, which are typically smaller circles of DNA that can be exchanged with other bacteria.

The genomes of animals, which are eukaryotes, are more complex. Animals have two genomes: a **nuclear genome** and a **mitochondrial genome** (Figure **??**). The nuclear genome is composed of multiple linear strings of double-stranded DNA, which are called chromosomes (after the Greek chroma, meaning colour, which refers to the staining properties of DNA-protein complexes in eukaryotic chromatin). Each chromosome is one very long linear piece of DNA that is tightly coiled around histones and packaged within the cell nucleus. Animal cells are usually **diploid**, meaning they have two of each chromosome in their nuclear genome. The mitochondrial genome (mitogenome) is made up of circular double-stranded DNA, and multiple mitogenome copies are present in each of the mitochondria, the energy-producing organelles in the cell cytoplasm. Each cell moreover contains many mitochondria, resulting in an average of 1,000-5,000 mitogenomes per animal cell, depending on cell type (Moraes 2001). The reason mitochondria have circular DNA is because they originate from bacteria that entered into other cells at the beginning of the formation of eukaryotes (Dyall, Brown, and Johnson 2004; Martin, Garg, and Zimorski 2015). While they have undergone substantial genome reduction and transfer of genes to the nuclear genome over time, they themselves have retained the circular organization of their DNA within eukaryotic cells for more than a billion years.

Plant cells are similar in many ways to animal cells, but they also contain additional genomes (Figure **??**). Like animals, plants have a nuclear genome made up of multiple linear strings of double-stranded DNA folded up within their cell nucleus, but unlike animal cells they are often **polyploid**, meaning that they can have many copies of each chromosome in their nuclear genome - typically two (diploid), four (tetraploid), six (hexaploid), or eight (octoploid). Like animals, plant cells also have a mitochondrial genome that is circular and double-stranded and present in many copies per cell, with the largest number of copies in root cells (Preuten et al. 2010). However, plants also have additional genomes within other organelles, such as chloroplasts. The plant **chloroplast genome** is double-stranded and partially circular (Arnold J. Bendich 2004). It originates from a photosynthetic cyanobacteria that entered into cells of plant ancestors early in the development of eukaryotes (Dyall, Brown, and Johnson 2004; Martin, Garg, and Zimorski 2015).

In sum, the term genome is used to refer to a variety of different configurations of nucleic acids within viruses, prokaryotes, and eukaryotes. What are the relative genome sizes of these different types of organisms? Viruses have the smallest genomes, with most viral genomes falling between about 5 and 100 thousand bp long (Campillo-Balderas, Lazcano, and Becerra 2015). Bacteria are bigger. Bacterial genomes are, on average, about 1 to 10 million bp long

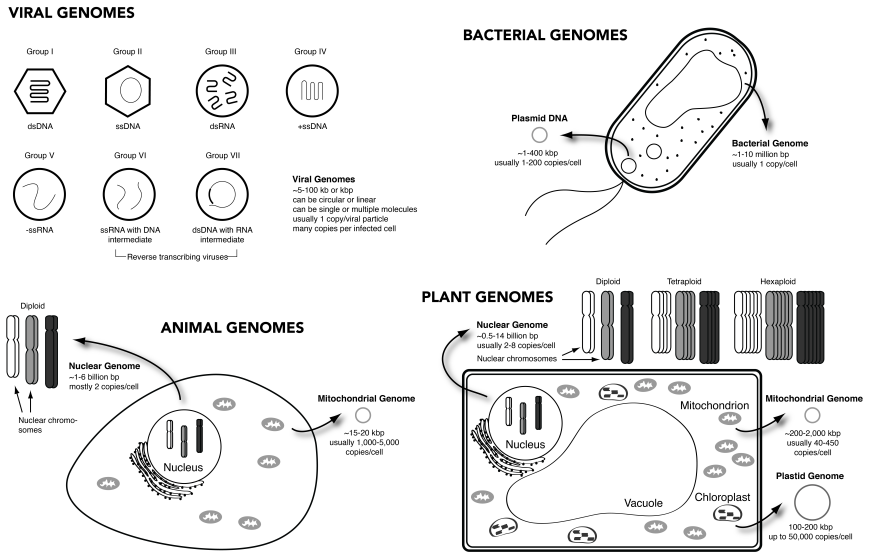


Figure 2.5: Genome basics. There are major differences in the composition, organization, size, and copy number of genomes in viruses, bateria, animal cells, and plant cells. Image by Christina Warinner under a CC-BY-SA 4.0

(Westoby et al. 2021), and their plasmids can vary in size from 1 to 400 thousand bp long (Thomas and Summers 2020). Large plasmids tend to be present in fewer copies than small plasmids, and plasmid copy number is usually 1-200 per bacterial cell (Ilhan et al. 2019; Thomas and Summers 2020). Overall, eukaryotes have the largest genomes, and whereas viral and bacterial genome sizes tend to scale linearly with their number of genes, no such correlation exists in eukaryotes (Bobay and Ochman 2017), a phenomenon that has been described as the C-value paradox or C-value enigma (Gregory 2005). Among animals, birds and mammals generally have nuclear genomes of about 1 to 6 billion bp long (Kapusta, Suh, and Feschotte 2017) and smaller mitogenomes that are 15-20 thousand bp long but present in 1,000-5,000 copies per cell (Boore 1999; Moraes 2001). Plants clock in with an average nuclear genome size of 0.5 to 14 billion base pairs (Michael 2014) and rather large mitogenomes (200 thousand to 2 million bp) and chloroplast genomes (100-200 thousand bp) that are each present in multiple copies per cell (A. J. Bendich 1987; B. R. Green 2011; Morley and Nielsen 2017; Preuten et al. 2010). It is thus clear that genome sizes vary enormously among these different types of organisms, with viruses and microbes typically having much smaller genomes than animals or plants.

Despite these trends, there are nevertheless many genomic outliers that have much larger or smaller genomes than expected. For example, the world's largest known virus, *Pandoravirus salinus*, has a genome of 2.5 million bases, which puts it firmly within the typical genome size range of bacteria (Campillo-Balderas,

Lazcano, and Becerra 2015). The world's smallest known bacterial genome, the endosymbiont *Carsonella ruddi*, is barely larger than a virus at 160 thousand bases long (Nakabachi et al. 2006), while the largest known bacterial genome, *Sorangium cellulosum*, is 13 million bases long (Schneiker et al. 2007). The largest known animal genome is that of the lungfish *Protopterus aethiopicus*, at 130 billion bases (Hidalgo et al. 2017). And among plants, the largest known genome is that of a small flowering plant, *Paris japonica*, at 149 billion bases (Hidalgo et al. 2017). Among eukaryotes, the size of the genome is not related to the complexity of the organism, and currently the organism with the largest known genome is a microscopic single-celled amoeba called *Polychaos dubium*, with an estimated genome size of 670 billion bases (Parfrey, Lahr, and Katz 2008), although its precise genome size is disputed and difficult to measure (Hidalgo et al. 2017). When trying to reconstruct ancient genomes from short, degraded ancient DNA, the genome size of the target organism matters a lot.

Next, let's put this information about genome sizes in context. Let's take the human genome as an example because it is the genome with which we are most familiar. The haploid size, or C-value, of the human genome is approximately 3 gigabase pairs, so 3 billion bp. Our body cells are diploid, so each of our cells contains two copies of the human genome, for a total of 6 billion bp. Humans have 23 chromosome pairs, for a total of 46 chromosomes, and each chromosome ranges in size between 50 and 250 million bp. And recall that chromosomes are really just long linear strings of DNA - very, very, very, very long DNA strings Humans also have a mitogenome, which by comparison is very tiny. It is only about 16,500 bases long, but it is present in thousands of copies per cell. On average, each cell contains 1,000-5,000 mitogenomes, but a mature human egg cell may contain as many as 1.5 million copies of the mitogenome (Cecchino and Garcia-Velasco 2019).

Have you ever wondered how chromosome 1 came to be named chromosome 1, and why chromosome 12 is called chromosome 12? The chromosomes are numbered in order of their length, or at least the length as they were originally calculated using cytogenetics (Figure **??**). Thus, chromosome 1 is the largest chromosome in the human genome, and it measures nearly 250 million bp long. In contrast, the shortest chromosomes in the human genome, chromosomes 21 and 22, are each around 50 million bp long.

Today, sequencing and analysing high molecular weight genomic DNA from living cells is relatively straightforward, especially with the recent advent of long-read sequences like PacBio and Oxford Nanopore (Koren and Phillippy 2015; Kovaka et al. 2023). However, ancient DNA is very different from the DNA shown in the karyogram in Figure **??**. Ancient DNA is broken and degraded into thousands, if not millions, of pieces. Let's take, for example, chromosome 1. In life, it is one continuous string of DNA that is 248,956,422 bp long. After death, it fragments into pieces that are each approximately 50 bp long. So to use a metaphor, I like to refer to ancient DNA as the world's worst jigsaw puzzle. Because if you had to put together a jigsaw puzzle of chromosome 1 from ancient

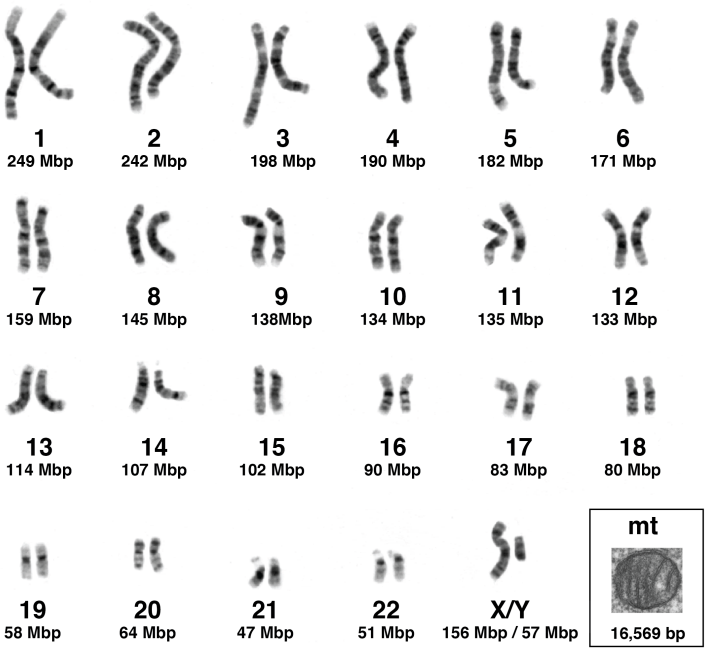


Figure 2.6: Karyogram micrograph showing the autosomal and sex chromosomes of a human male. Inset shows micrograph of human mitochondrion at a different scale. Chromosome lengths are from (Piovesan et al. 2019). Karyogram photo credit: National Human Genome Research Institute; image is in public domain. Mitochondrial TEM photo credit: Louisa Howard; image is in public domain.

DNA fragments, it would be a puzzle with over 5 million pieces. Hopefully this puts into perspective the kinds of challenges researchers face when working with ancient DNA.

2.5 How DNA degrades

How does DNA degrade? What kind of damage accumulates in ancient DNA over time? Some forms of DNA damage occur during life, for example during disease progression, immune response, and tissue necrosis. Such DNA damage is largely driven by the activity of nucleases, which create a variety of double-stranded breaks (Bokelmann, Glocke, and Meyer 2020; Vladimir V. Didenko 2008; Vladimir V. Didenko, Ngo, and Baskin 2003; V. V. Didenko and Hornsby 1996; Harkins et al. 2020). Other forms of DNA damage occur after death and are largely driven by chemical changes. In 1993, Tomas Lindahl was the first to examine the latter forms of postmortem damage in detail (Lindahl 1993). Lindahl presented a theoretical model of DNA degradation based on the chemical properties of DNA and identified **hydrolytic** and **oxidative** decomposition as the most likely types of damage to accumulate in fossil DNA. He also predicted that conditions of high ionic strength, adsorption to hydroxyapatite, and partial dehydration may contribute to long-term DNA preservation. Michael Hofreiter and colleagues expanded Lindahl's model of ancient DNA degradation in an influential review article in 2001, noting that the empirically observed excess of C->T an G->A miscoding lesions in PCR-amplified ancient DNA likely results from deamination of cytosine residues in ancient DNA template molecules (Hofreiter et al. 2001).

The chemical bonds within DNA that are most vulnerable to degradation are indicated with arrows in Figure **??**. Although DNA is vulnerable in many places, subsequent research has shown that some of these bonds undergo much faster rates of degradation than others, and that DNA damage generally proceeds in a predictable sequence.

2.5.1 Depurination and nicking

The first step in DNA degradation is typically a hydrolytic attack of the bond attaching the nitrogenous base of purines to the nucleotide's deoxyribose sugar, a process called **depurination** (Figure **??**). Guanines and adenines are purines, so this step leads to the removal of G and A bases, producing abasic sites along the DNA molecule. These abasic sites act like holes along the DNA molecule, and they make the phosphate backbone more exposed and therefore susceptible to attack.

Cleavage of the phosphate backbone occurs next, and this typically occurs 3' to the abasic site (Figure **??**). This process is called **nicking**. These nicks occur randomly along the DNA wherever there are abasic sites. Because they only affect one strand at any given position, they are called single-stranded nicks.

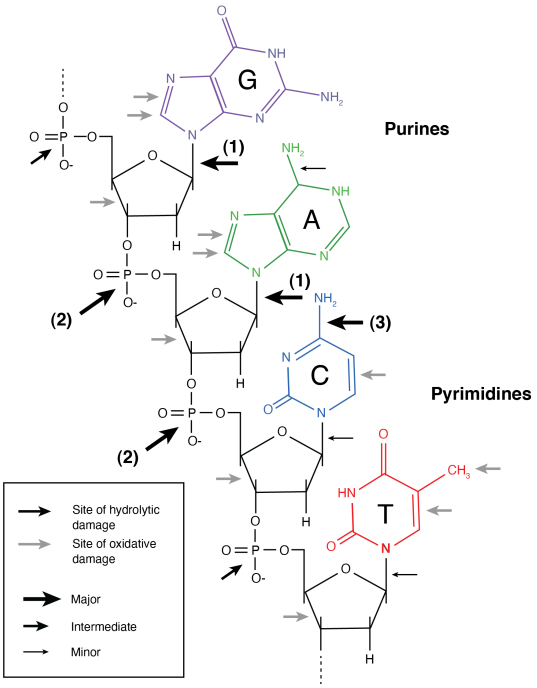


Figure 2.7: DNA degradation. Model of DNA degradation highlighting chemical bonds that are more susceptible to chemical attack and are thus most likely to show evidence of damage in ancient DNA. Four nucleotides are shown with the sequence: 5'-GACT-3'. The phosphate backbone and deoxyribose sugar parts of the DNA molecule are shown in black. The nitrogenous bases are coloured: guanine (G), purple; adenine (A), green; cytosine (C), blue; thymine (T), red. G and A are purines; C and T are pyrimidines. Sites of hydrolytic damage are indicated by black arrows; sites of oxidative damage are indicated by gray arrows. The typical order of damage accumulation at major damage sites (large arrows) is indicated by numbers: (1) depurination; (2) nicking; (3) cytosine deamination. Adapted from (Hofreiter et al. 2001; Lindahl 1993)

As nicks accumulate along the DNA molecule, the DNA becomes destabilised and the hydrogen bonds between the complementary bases become the primary force holding the DNA together. If two nicks form close to each other, the hydrogen bonds between the nicks may not be strong enough to hold the DNA together and the double helix may separate, a process known as melting. Melting occurs when the vibrational energy of the two DNA strands exceeds that of the hydrogen bonds holding them together. Because two nicks rarely form at the same position on the two DNA strands, most nicking results in DNA fragments with **single-stranded DNA overhangs** (Figure **??**).

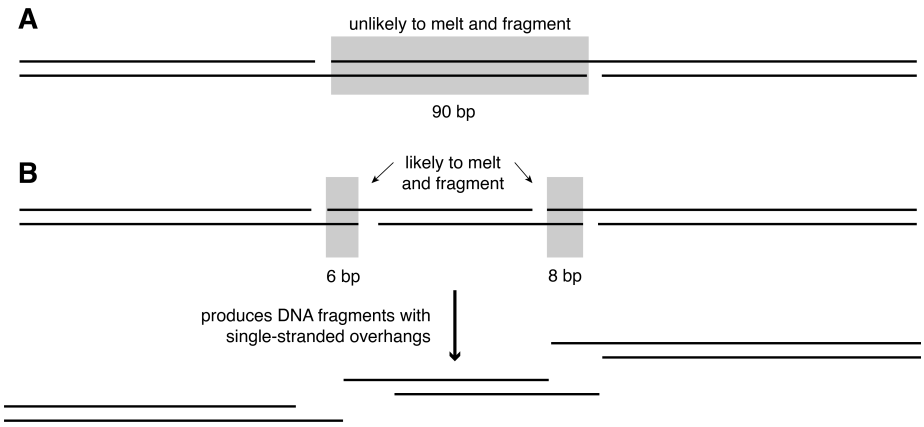


Figure 2.8: Process of DNA fragmentation. Depurination exposes the phosphate backbone of DNA to hydrolytic attack, which leads to nicking 3' to the abasic sites. (A) When nicks are spaced far apart, the DNA between the nicks is unlikely to melt. (B) When nicks are spaced close together, the DNA between the nicks is likely to melt, producing DNA fragments with single-stranded overhangs. Image by Christina Warinner under a CC-BY-SA 4.0.

DNA has different local melting points depending on its composition. G and C form a triple hydrogen bond, while A and T form a double hydrogen bond. Consequently, the melting point of GC rich sequences is higher than AT rich sequences. For example, at neutral pH a 12 bp fragment of DNA with the sequence 5'-GCGCGCGCGCGC-3' will have a melting point of 63.9 ˚ C, while a 12 bp fragment of DNA with the sequence 5'-ATATATATATAT-3' will have a melting point of 7.8 ˚ C[1]. Thus, if two single stranded nicks form within 12 bp of each other on opposite strands of the double helix, the DNA is likely to melt and fragment if the DNA is locally AT rich. When nicks are even closer together, the temperature required to melt the strands is even lower. For example, a 6 bp fragment of DNA with the sequence 5'-GCGCGC-3' will have a melting point of 24 ˚ C, while a 6 bp fragment of DNA with the sequence 5'-ATATAT -3' will have a melting point of 0 ˚ C. Changes in pH and ion concentration can

[1]Temperatures calculated using the IDT Oligoanalyser tool (https://eu.idtdna.com/calc/analyser).

affect the precise melting temperature, but closely spaced nicks are the major cause of ancient DNA fragmentation at ambient temperatures. Empirically, most ancient DNA overhangs have been found to be very short, with one- and two-nucleotide overhangs being most common, but overhangs can also extend 20 or more nucleotides into the ancient DNA molecule (Bokelmann, Glocke, and Meyer 2020).

The cumulative effect of nicking over time is a high degree of DNA fragmentation. A smoothed histogram of bacterial DNA fragment lengths measured from ancient dental calculus is shown in Figure **??**. The mode DNA length is around 50-60 bp long, but the curve sharply declines such that there is almost no DNA present at lengths 150 bp and higher. Almost all DNA within archaeological dental calculus is short, and this is the direct result of nicking and the melting of DNA between nicks. This is how time chops up high molecular weight modern DNA into tiny fragmented ancient DNA.

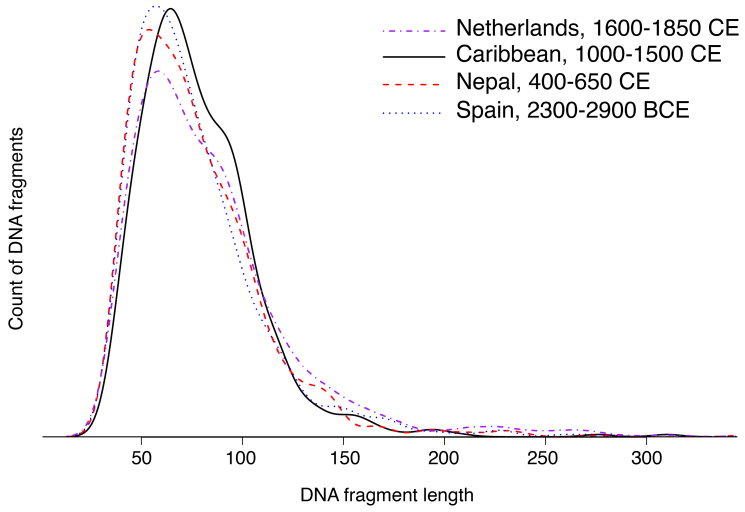


Figure 2.9: Smoothed histogram of bacterial DNA fragment lengths from ancient dental calculus. Ancient DNA quickly fragments into pieces <150 bp long, after which the rate of DNA fragmentation appears to slow. Here DNA ranging in age from 150 years old to nearly 5,000 years old all show a similar fragment size distribution. Data from (Mann et al. 2018).

2.5.2 Cytosine deamination

Once fragmented, nucleotides within the single-stranded overhangs become exposed to further chemical attack. The amine group of the cytosine nitrogenous base is particularly vulnerable to hydrolytic attack when not hydrogen-bonded to its complementary guanine, and it undergoes cytosine deamination Figure **??**.

During cytosine deamination, the NH_2 group is lost as ammonia, turning the cytosine nitrogenous base into a uracil Figure **??**. Uracils are not normally found in DNA, and are instead usually found in RNA. Although cytosine deamination may occur at any time in DNA, the rate of its formation is >100 times higher in single-stranded DNA (Frederico, Kunkel, and Shaw 1993; Lindahl and Nyberg 1974). Consequently, ancient DNA molecules tend to accumulate uracils at fragment termini, where they are single-stranded (Bokelmann, Glocke, and Meyer 2020). Most DNA polymerases are not able to correctly recognise uracils, and they generally interpret uracil as a thymine during DNA extension. As a result, ancient DNA sequences contain an overrepresentation of thymines at positions where cytosines are expected. This process produces the characteristic patterns of C->T transitions observed in ancient DNA datasets, as well as the corresponding G->A transitions in the complementary strand.

One important variant of the cytosine deamination process occurs when the cytosine is methylated, as is common in CpG islands involved in epigenetic DNA regulation. Methylated cytosines can also undergo deamination, and when that occurs the cytosine turns into a thymine, not a uracil (Figure **??**). Using enzymes that specifically target uracils in DNA, some ancient DNA researchers have taken advantage of differences between unmethylated and methylated cytosine deamination products to investigate ancient epigenetics (Hanghøj and Orlando 2019; Briggs et al. 2010; Gokhman et al. 2014).

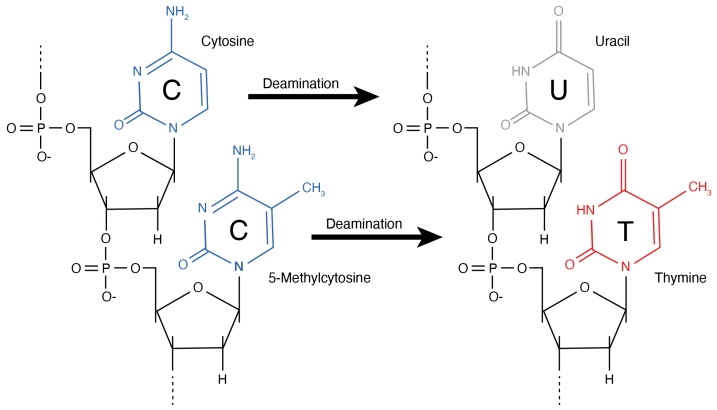


Figure 2.10: Deamination of cytosine and 5-methylcytosine. Deamination of cytosine results in the formation of uracil. Deamination of 5-methylcytosine results in the formation of thymine. Image by Christina Warinner under a CC-BY-SA 4.0.

2.5.3 Damage review

Let's review the typical progression of DNA damage Figure **??**. First, DNA undergoes depurination, which results in the random loss of G and A bases. Next, the phosphate backbone undergoes hydrolytic attack at the newly formed

abasic sites resulting in single-stranded nicking. If the nicks are closely spaced, the hydrogen bonding between the bases will not be strong enough to hold the double helix together, and the strands will melt. This produces DNA fragments with single-stranded overhangs. The length of the overhang corresponds to the distance between the nicks in the melted strands. Cytosines along the single-stranded overhangs undergo faster rates of cytosine deamination. Unmethylated cytosines deaminate into uracils, while methylated cytosines deaminate into thymines. DNA polymerases interpret both forms of deaminated cytosines as thymines, resulting in an excess of thymines in ancient DNA datasets, particularly at the ends of DNA sequences.

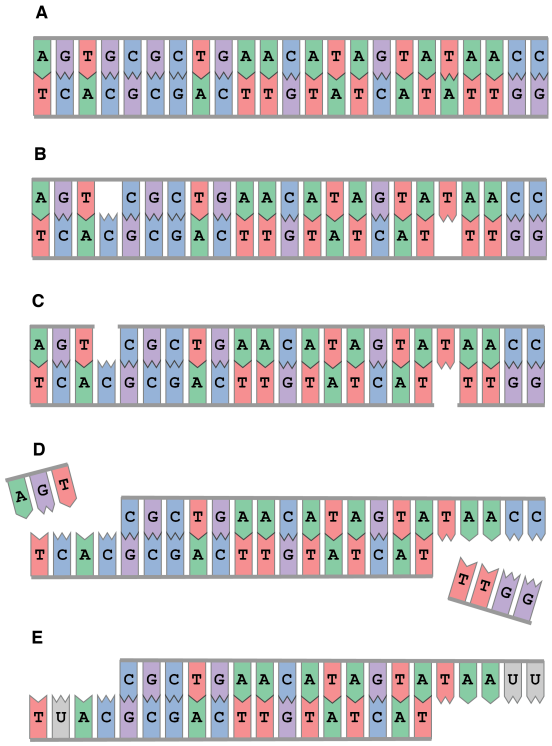


Figure 2.11: Deamination of cytosine and 5-methylcytosine. Deamination of cytosine results in the formation of uracil. Deamination of 5-methylcytosine results in the formation of thymine. Image by Christina Warinner under a CC-BY-SA 4.0.

2.6 Getting a handle on damage in ancient DNA

But how was this all figured out? How do we know how ancient DNA decays? The empirical process by which this was determined is a fascinating story, and it occurred relatively recently. Prior to the advent of NGS, it was known that

ancient DNA was fragmented, but the length distribution of ancient DNA could not be precisely measured. As recently as 2004, there were still large uncertainties about the typical length of ancient DNA (Pääbo et al. 2004). This was in large part because at the time there were no effective methods for measuring the length of very short, very low abundance DNA. Short DNA is very easily lost during DNA extraction, and many common DNA extraction methods, such as salting-out without a carrier (Gaillard and Strauss 1990), are completely unsuitable for recovering ancient DNA for this reason (Jeong et al. 2016). Low quantities of unamplified ancient DNA also do not visualise well using gel electrophoresis, and if modern soil DNA is also present in the sample, such contaminant DNA tends to overwhelm and obscure the visualization of any ancient DNA. Consequently, prior to 2007 there was great uncertainty about just how long or short ancient DNA really was. Many articles published in the 1990s and early 2000s provide estimates of around 100-500 bp, but those were really just educated guesses. Scientists could tell that ancient DNA was short, but they did not really know how short.

Early ancient DNA studies primarily used a targeted PCR approach to try to amplify DNA templates of about 300-500 bp, but these attempts had very high PCR failure rates and vexing contamination problems (J. J. Austin et al. 1997; Champlot et al. 2010; Hagelberg and Clegg 1991; Handt et al. 1994). Today we know why - it is because ancient DNA is much shorter than this - but at the time this was not yet known. PCR amplification is very sensitive and it can reliably generate amplicons from as few as 4 template molecules under a variety of conditions (Forootan et al. 2017). This makes PCR highly susceptible to even trace amounts of contamination, and if PCR reactions are designed to amplify templates that exceed the length of ancient DNA, the only molecules that will amplify are contaminants (Figure **??**, panel A). Ancient DNA studies during the 1990s and 2000s, when direct PCR was the primary method of ancient DNA analysis, were thus plagued with intractable problems of irreproducibility and contamination (Cooper and Poinar 2000). Nevertheless, there were successful examples of PCR amplification of ancient DNA, and researchers noted the occasional presence of C->T and G->A miscoding lesions in seemingly authentic ancient DNA amplicons, but they could not yet empirically confirm the precise mechanisms of how they got there (Gilbert et al. 2003; Hofreiter et al. 2001).

At this point in the pre-NGS era, damage was just seen as a problem for the ancient DNA community. It created uncertainty in DNA sequence reconstructions, and it was not recognised to have a benefit. This changed with the development of NGS. NGS represented a radical redesign of the methods used to amplify and sequence ancient DNA, and it eliminated most of the problems that had mired earlier PCR-based ancient DNA studies in ambiguity and uncertainty (Figure **??**, panel B). With NGS, it was no longer necessary to design PCR primer-binding sites on the actual ancient DNA template, which meant that much shorter DNA fragments could be analysed and sequenced. Instead, oligos containing universal priming sites were simply ligated onto the ends of the ancient DNA molecules. This made it possible for the first time to recover

all of the ancient DNA in a sample, and, in most cases, to sequence each ancient DNA molecule in its entirety–from beginning to end.

Once all of the DNA in a sample could be analysed, it became possible to measure the true size of ancient DNA and to determine the processes of DNA degradation in order. This was first achieved in 2007 in a study of Neanderthal DNA by Adrian Briggs and colleagues at the Max Planck Institute for Evolutionary Anthropology (MPI-EVA) in Leipzig, Germany (Briggs et al. 2007). Subsequent improvements in ancient DNA recovery during DNA extraction by Jesse Dabney and colleagues at the MPI-EVA showed that ancient DNA was much shorter than previously thought (Dabney et al. 2013), and that most DNA extraction protocols were simply not capable of recovering such short DNA fragments. The biased recovery of only relatively long DNA molecules using most extraction protocols had given the false impression that ancient DNA was in low abundance but relatively long (~100-500 bp). Together, these studies instead revealed that archaeological samples actually contain much more ancient DNA than previously thought, but that this ancient DNA is highly fragmented and degraded and therefore requires specialised protocols to recover, sequence, and analyse.

Today, we know that most ancient DNA is very short - on the order of about 30 to 50 bp long on average. Shorter DNA than this is very difficult to recover (Glocke and Meyer 2017), and also not desired since sequences <30 bp cannot be unambiguously identified to a specific taxonomic group or region of the genome (Filippo, Meyer, and Prüfer 2018). The process of piecing together the evidence to understand the progression of DNA degradation was a key achievement in the history of palaeogenomics. It turns out that DNA damage is quite predictable, and this has allowed scientists to turn the problem of DNA damage into a solution. There are now a number of software tools that specifically identify and quantify DNA damage as a way of authenticating ancient DNA and distinguishing it from modern DNA contaminants (Jónsson et al. 2013a; Skoglund et al. 2014). This work has been crucial for reconstructing extinct hominin genomes, and it is now used today for samples of all ages, depending on context, to separate ancient and modern DNA and to authenticate well-preserved archaeological samples.

2.6.1 Authenticating the Vindija Neanderthal DNA

Let's take a closer look at the 2007 study on ancient DNA damage that proved to be such a game-changer in the field (Briggs et al. 2007). Adrian Briggs and colleagues were trying to solve a problem - how do you distinguish Neanderthal DNA from the DNA of all of the humans who have touched the Neanderthal bone? And how do you recognise real Neanderthal DNA once you have it? It was a pressing problem because the first Neanderthal DNA sequences, published just a year before (R. E. Green et al. 2006; Noonan et al. 2006), had undergone intense scrutiny (Wall and Kim 2007) and were shown to be compromised by contamination. Using data generated by a newly available NGS

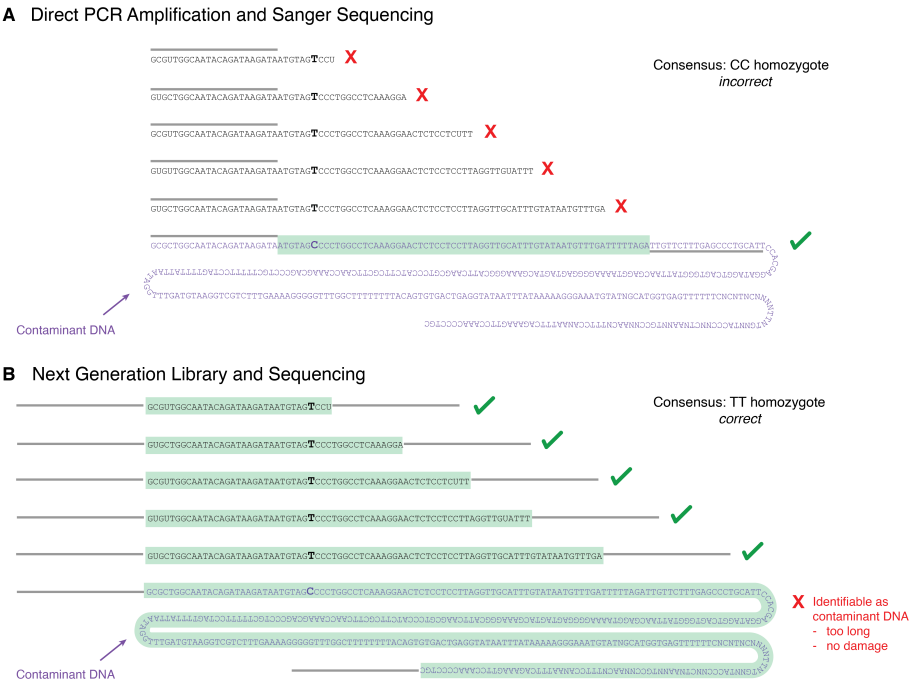


Figure 2.12: Comparison of PCR and NGS methods for analysing a pool of 6 ancient and contaminant DNA molecules. DNA sequences are from intron 13 of the MCM6 gene in humans, which contains a polymorphic site conferring lactase persistence (bold). Ancient DNA molecules are shown in black; contaminant DNA molecules are shown in purple; primers are shown as gray bars; sequenceable portions of the DNA template using each method are highlighted in green. (A) Using direct PCR, none of the ancient DNA molecules would amplify because the primer design requires a 111 bp template on which both primers can bind, and the ancient DNA template molecules are too short. Only the contaminant sequence would amplify, yielding an incorrect genotype (CC). (B) Using NGS, all of the DNA molecules would amplify and yield sequences. Downstream data analysis would allow the identification and removal of the contaminant sequence due to its excessive length (>300 bp) and lack of DNA damage. The remaining authenticated ancient DNA sequences could then be used to determine the correct genotype (TT). Image by Christina Warinner under a CC-BY-SA 4.0.

technique developed by 454 Life Sciences known as **pyrosequencing**, the team analysed the patterns of DNA damage present in the DNA of the Vindija Neanderthal. First, they examined the DNA sequence context around strand breaks in ancient DNA, something that could not be done using data generated using conventional targeted PCR. They aligned the 454-sequenced ancient DNA fragments to a reference genome and found that there was a bias towards G and A nucleotides in the -1 position, meaning one position upstream of the ancient DNA sequence in the alignment. In other words, the ancient DNA seemed to have disproportionately broken right after a purine, either G or A (Figure **??**). A different pattern was observed at the 3' end of the ancient DNA molecule, where the +1 position showed an overrepresentation of C and T nucleotides. This can be attributed to repair steps occurring during the construction of the NGS-sequencing library, such that the observed pattern at the 3' +1 position is simply the reverse complement of the 5' -1 position. We'll return to this topic later in the chapter.

The observations of Briggs and colleagues matched predictions previously made by Lindahl that purines were particularly susceptible to hydrolytic attack and depurination, after which the sugar phosphate backbone undergoes hydrolysis 3' to the depurinated site (Lindahl 1993). This pattern observed in the Vindija Neanderthal was then confirmed to be present in mammoth and cave bear DNA datasets, suggesting it was a generalised pattern characteristic of ancient DNA fragmentation.

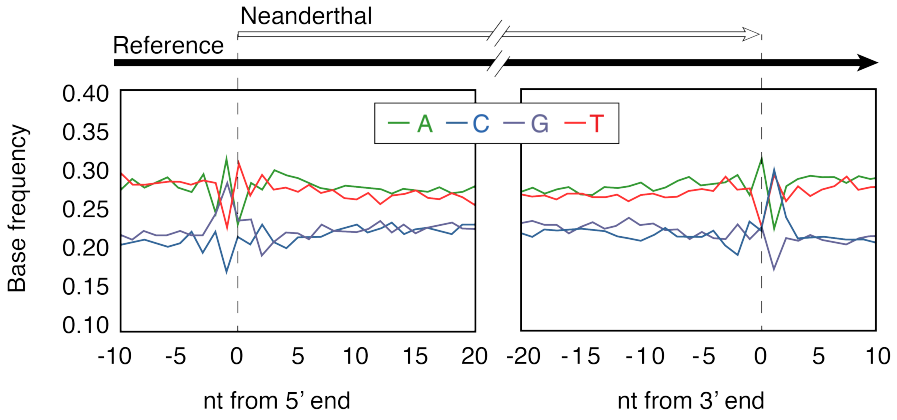


Figure 2.13: Empirically observed base frequencies in the DNA sequence adjacent to strand breaks in ancient DNA from the Vindija Neanderthal. Alignment to the reference genome showed that strands breaks disproportionately follow purines in the reference sequence, a pattern predicted by Lindahl's model of DNA degradation (Lindahl 1993). Figure adapted from (Briggs et al. 2007).

Briggs and colleagues also described a second damage-related pattern present in the Vindija Neanderthal DNA sequences: an excess of errors or differences from

the reference at the ends of each DNA sequence (Figure **??**). Specifically, they observed a high substitution frequency of C->T at the 5' end and of G->A at the 3' end of the DNA molecule. Corresponding again to Lindahl's predictions, the 5' overrepresentation of T could be explained as accelerated cytosine deamination in the single-stranded overhang at the sequence 5' terminus, while the 3' overrepresentation of A was simply the reverse complement. The deamination pattern was strikingly clear, and the elevated 5' C->T substitution frequency declined to baseline values by approximately the 10th bp of the DNA molecule, suggesting that the single-stranded overhangs were mostly less than 10 bp long - a pattern in agreement with melting temperature predictions. The plot was nicknamed the **smile plot** because it resembles the shape of a smile.

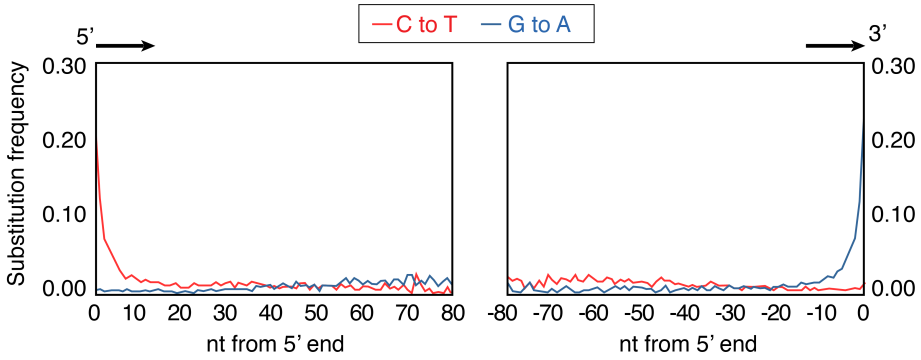


Figure 2.14: Damage plot of DNA from the Vindija Neanderthal. An elevated substitution frequency of C->T at the 5' end of the DNA molecules is consistent with accelerated cytosine deamination in the single stranded overhangs at the ends of ancient DNA molecules. Note that due to repair steps during NGS library construction, the 3' G->A substitutions are simply the reverse complement of the 5' C->T substitutions. Figure adapted from (Briggs et al. 2007).

2.7 DNA damage: from problem to solution

With the advent of NGS sequencing and the revelation that ancient DNA undergoes empirically predictable patterns of DNA degradation, DNA damage went from being an annoying and intractable problem to a powerful solution for DNA authentication. Next we will review the smile plot in detail, including how it is influenced by NGS library construction methods and how software tools designed to characterise deamination patterns are used to authenticate ancient DNA today.

2.7.1 The 'smile plot'

When interpreting ancient DNA damage patterns, it is critical to know that the characteristic shape of the ancient DNA smile plot is the product of two distinct processes. The first is the DNA degradation process itself, whereby depurination leads to nicking and melting, which produces single-stranded overhangs at the ends of DNA molecules where cytosines deaminate at a rate >100 times faster than in double stranded DNA. The second process relates to the asymmetric behaviour of repair enzymes used during NGS library construction, which we will discuss now.

Up until now, we have focused on the DNA damage patterns present on the 5' ends of the DNA fragments, and we have mostly ignored the DNA damage on the 3' ends. Why? It is because the damage observed on the 3' ends of ancient DNA molecules depends on the NGS library construction method. In order to understand this, it is important to recall that each strand of DNA has an orientation, or reading order, that runs from the 5' end of the molecule to 3' end of the molecule, and that the orientation of the two strands that make up the DNA double helix run in opposite directions. The 5' to 3' nomenclature refers to the position of the carbon atom in the 2-deoxyribose sugar that connects to the phosphate backbone (Figure **??**). The 5' to 3' orientation of DNA is important because most enzymes involved in DNA copying and repair only work in one direction of the orientation. So, for the enzymes we use to manipulate DNA during DNA amplification and sequencing it matters very much which side of the DNA fragment is 5' and which side is 3'.

Let's next examine a hypothetical pool of double-stranded DNA as it degrades into ancient DNA (Figure **??**, panels A-C). In this pool, each ancient DNA molecule can exhibit one or more forms of damage in different combinations. For example, two nicks may occur on the same DNA strand, resulting in one strand having both 5' and 3' overhangs and the other strand having none; or nicks may occur on opposite DNA strands, resulting in both strands having a 5' overhang or both strands having a 3' overhang (Figure **??**, panel B). Over time, cytosines then deaminate to form uracils on these single stranded overhangs (Figure **??**, panel C). The presence of these overhangs interferes with the steps needed to prepare the DNA for sequencing, so the first step of DNA library construction is to **end-repair** the DNA to make the strands fully double-stranded with blunt ends (Figure **??**, panels D-E). This makes the DNA suitable for ligating on adapters that make the DNA "readable" by the sequencing instrument. DNA repair is performed with **T4 polymerase**, a type of polymerase that originally came from the T4 virus, a bacteriophage that infects Escherichia coli. T4 polymerase has important properties that must be kept in mind in order to understand how its repair function will affect downstream sequences. When encountering DNA with single-stranded overhangs, T4 polymerase will trim back any 3' overhangs (Figure **??**, panel D), and fill in any 5' overhangs (Figure **??**, panel E). This removes approximately half of the uracils in the pool of ancient DNA, and it adds an adenine opposite the remaining uracils (recall

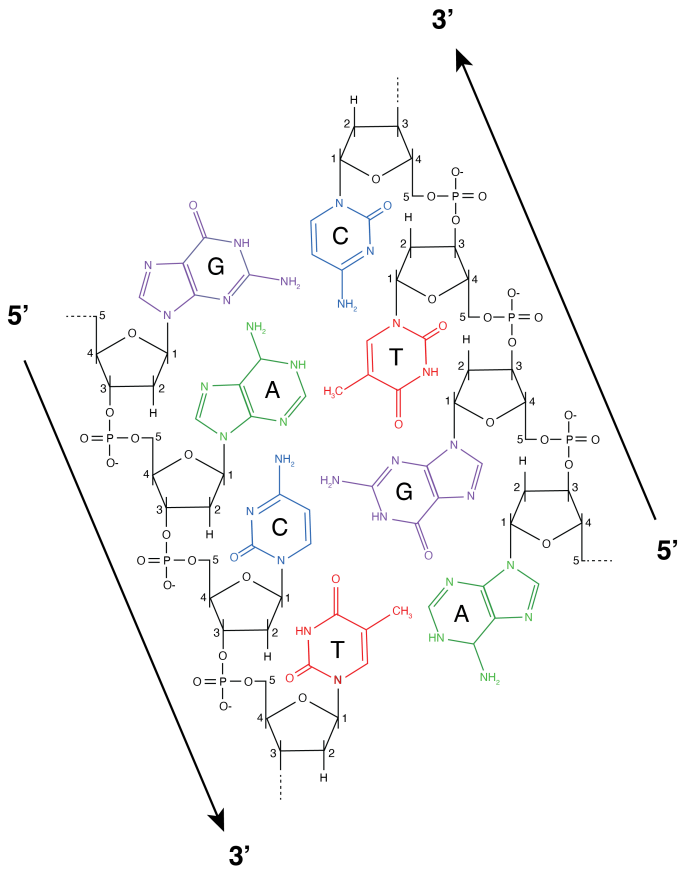


Figure 2.15: Model of DNA orientation. Carbon atoms within the 2-deoxyribose sugar component of each nucleotide are numbered to show the opposing 5' to 3' orientation of the DNA strands in the double helix. Image by Christina Warinner under a CC-BY-SA 4.0.

that polymerases treat uracils like thymines). The result is that uracils only remain on the 5' ends of the molecules, while the complementary adenines are found only on the 3' prime ends of the molecule - a pattern that is made even more visible if we imagine rearranging all the strands in the same orientation (Figure **??**, panel F). This asymmetric behaviour of T4 polymerase is what produces the asymmetric appearance of the ancient DNA smile plot, with an excess of C→T substitutions on the 5' (left) side of the plot, and an excess of G→A substitutions on the 3' (right) side of the plot (Figure **??**).

To review, the end result of the T4 polymerase damage repair process of ancient DNA is a set of fully double-stranded DNA with blunt ends (no overhangs) and an asymmetrical removal of half of the damage present in the original ancient DNA molecules. Some DNA molecules (those with damage only on their 3' overhangs) even end up with no damage at all after repair. An important consequence of this asymmetric loss of DNA damage on 3' overhangs means that ancient DNA prepared using a double-stranded preparation method using T4 polymerase will never reach a terminal substitution rate of 1.0, and instead the maximum substitution rate observed under normal conditions of DNA degradation is 0.5.

2.7.2 DNA damage tools

The predictable patterns of DNA decay combined with knowledge of how DNA repair mechanisms modify these patterns allow for the development of software to authenticate ancient DNA sequences. The first such publicly available automated tool was mapDamage (Ginolhac et al., 2011), which was later updated to mapDamage 2.0 (Jónsson et al. 2013a). Developed in the group of Ludovic Orlando, the tool aligned and compared ancient DNA sequences to reference genomes to visualise patterns of depurination and cytosine deamination, much like those originally presented in the study of the Vindija Neanderthal (Briggs et al. 2007). The same principles were incorporated into PMDtools (Skoglund et al. 2014), a suite of software tools developed by Pontus Skoglund to enable the separation of damaged and non-damaged DNA sequences within a mixture, an application that is helpful in removing modern DNA sequences from heavily contaminated ancient DNA datasets. More recently, the tool DamageProfiler (Neukamm, Peltzer, and Nieselt 2021b) was developed to allow a similar level of damage characterization, but in a more computationally efficient manner that decreases runtime and uses less memory. However, because each of these tools was designed with the goal of characterizing DNA damage for a single species of interest, they can be difficult to apply to large microbial metagenomics datasets, such as those generated from ancient microbiomes or sediments. The tool PyDamage (Borry et al. 2021b), which allows damage profiling for multiple genomes within a sample, was developed to solve this problem, and the tool can also provide reference-free estimates of DNA damage when combined with de novo assembly.

Collectively, DNA damage tools are an essential resource in ancient DNA anal-

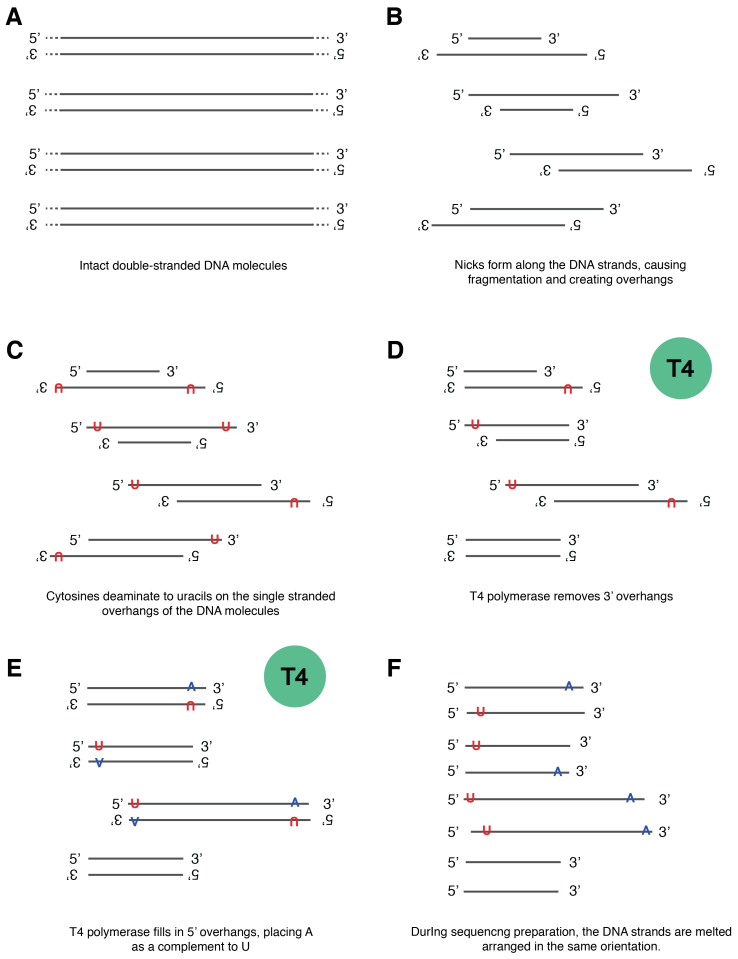


Figure 2.16: Processes of DNA decay and DNA repair that contribute to the DNA damage patterns observed in an ancient DNA "smile plot". As intact DNA (A) decays, it accumulates single stranded nicks that produce fragments with different patterns of single-stranded overhangs (B). Cytosine deamination along the overhangs leads to the accumulation of uracils at the 5' and 3' ends of the ancient DNA molecules (C). In the lab, T4 polymerase is used to repair the ancient DNA, which it does by trimming off 3' overhangs (D) and filling in 5' overhangs (E). This removes uracils from the 3' ends of the molecules and adds a complementary adenine opposite the remaining 5' uracils, introducing 3' adenines. This asymmetric repair behaviour of the T4 polymerase produces the characteristic asymmetric appearance of a "smile plot", in which C→T substitutions are observed on the 5' ends of ancient DNA molecules, and complementary G→A substitutions are observed on the 3' ends (F). Image by Christina Warinner under a CC-BY-SA 4.0.

ysis, as they allow researchers to authenticate ancient DNA sequences, estimate sample preservation, and troubleshoot potential problems (Figure **??**). For example, the absence of DNA damage observed in Figure **??** (panel A) indicates that the DNA present in this dataset is likely modern. The spiky damage pattern in Figure **??** (panel B) indicates that too few sequences were used to perform the analysis. In general, damage analysis works best with >1,000 aligned sequences, so observing this pattern simply means that more sequencing is needed. A similar pattern can also result if **ultrashort DNA sequences** (<30 bp) are not removed from the dataset prior to analysis; because ultrashort DNA sequences of this length are taxonomically non-specific, they can misalign to the reference genome, introducing noise. The data in Figure **??** (panel C) exhibits an ancient DNA damage pattern at sequence termini but it also has an elevated damage baseline, which suggests that the wrong reference genome was used. This is a common problem in studies of ancient bacteria in which the correct reference genome might not be known or available. For example, imagine that a sample of archaeological dental calculus contains *Streptococcus sanguinis*, but you align the data to the reference genome for *Streptococcus mitis*, a different but related species. The reference genome is close enough to produce a damage plot, but the elevated baseline alerts you to the fact that the selected reference genome is incorrect. To remedy this, a closer reference genome should be selected. However, if not available, the aDNA can also be assembled into contiguous sequences, or **contigs**, and the reads mapped back to these contigs to provide a damage estimate (Borry et al. 2021b); this method requires deep metagenomic sequencing, but it is the most reliable way to estimate damage for ancient microbial taxa with inadequate genomic representation in public data repositories. Finally, Figure **??** (panel D) shows the expected pattern of DNA damage for a dataset that contains authentic ancient DNA, is sufficiently powered (has enough DNA sequences for analysis), and is aligned to the correct reference genome.

At this point, you may be wondering whether it is possible to use DNA damage as a clock. If so, that would make DNA damage very useful for dating ancient remains; however, the answer seems to be sort of, but not really. DNA damage is more like a clock that just says "today" or "a while ago". The relationship between time and DNA damage is not linear, but rather is highly dependent on environmental factors like local temperature and humidity that speed up or slow down the processes of hydrolytic attack and depurination that cause DNA damage. Examining real examples of ancient DNA damage plots makes this clear. Figure **??**, for example, shows cytosine deamination patterns that have been reported for Neanderthal and human remains from Europe. The Neanderthal remains are >40,000 years old and they have a 0.45 terminal substitution rate, indicating that ~90% of the original DNA fragments contained damaged cytosines. Younger remains from Neolithic Scandinavia dating to around 5,000 years ago had a lower terminal substitution rate of ~ 0.25-0.35 (Skoglund et al. 2014). In contrast, far younger samples in Costa Rica dating to only around 1,000 years ago have terminal substitution rates of >0.45 (Morales-Arce et al. 2017), indicating that as much or more deamination damage has accumulated

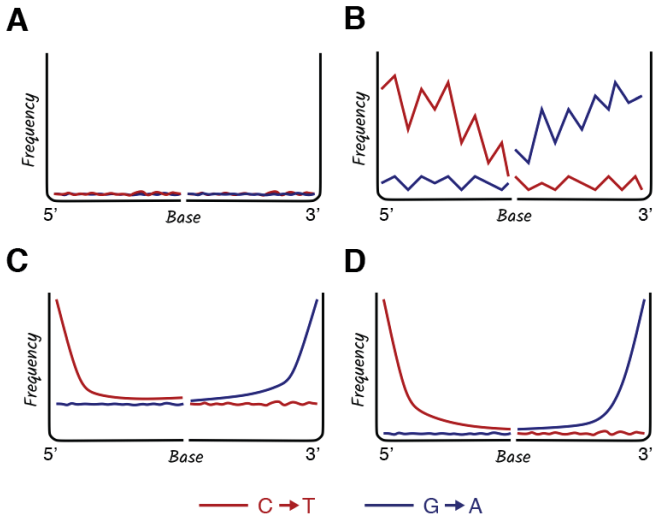


Figure 2.17: Common patterns of nucleotide misincorporation observed for double-stranded DNA libraries made from archaeological samples. (A) modern DNA; (B) too few sequences for analysis; (C) incorrect reference genome; (D) authentic ancient DNA. Figure modified from https://nf-co.re/eager/2.5.0, by Zandra Fagernäs under CC-BY 4.0 License, (Fellows Yates, Lamnidis, et al. 2021a).

in these tropical individuals in just one thousand years as in Neanderthals who are >40,000 years older (Figure **??**). Thus, while DNA damage accumulates over time, it is heavily influenced by temperature and humidity in a way that prevents it from being used as a simple clock.

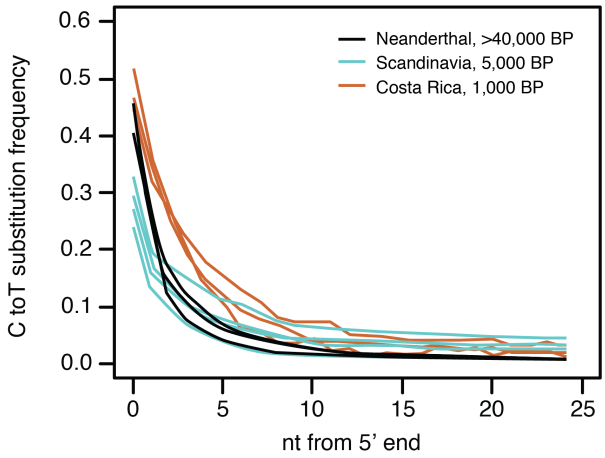


Figure 2.18: C to T substitution frequency observed in ancient DNA from different regions and periods. Cytosine deamination accumulates over time, but the rate of this hydrolytic damage is highly dependent on local environmental factors like temperature and humidity. In temperate Europe, 40,000 year-old Neanderthal DNA will have more damage than 5,000 year-old Scandinavian DNA from temperate Europe, but may have less damage than even very young samples from tropical regions like Costa Rica. For these reasons, aDNA damage cannot be used as a simple molecular clock. Adapted from (Morales-Arce et al. 2017; Skoglund et al. 2014).

Efforts have been made to model these factors in order to better predict DNA damage rates in different locations and under different conditions (Allentoft et al. 2012; Kistler et al. 2017; Smith et al. 2003), and while some broad patterns are clear, such as DNA degrading faster in hot and humid environments than in permafrost and dry caves, accurately predicting specific rates of DNA decay is notoriously difficult (Bokelmann, Glocke, and Meyer 2020). Many environmental factors operate at hyperlocal scales and can cause variation in DNA preservation within a single burial context (Jeong et al. 2016), across a single sediment block (Massilani et al. 2022a), among bones and teeth within a single skeleton (Gamba et al. 2014; Parker et al. 2020; Hansen et al. 2017), or even within a single bone (Prüfer et al. 2017; Hajdinjak et al. 2018).

Another important factor to keep in mind is that not all organisms undergo DNA degradation and fragmentation at the same rate, even if they originate from the same sample. This can be readily seen by examining the DNA fragment rate (Kistler et al. 2017) in archaeological samples that are known to contain mul-

tiple species (Warinner et al. 2017). For example, in an examination of a 19th century herbarium specimen of potato (Solanum tuberosum) infected with fungal pathogen known as potato blight (*Phytophthora infestans*), a similar DNA fragmentation rate was observed for both species (Figure **??**, panel A). However, in a human tooth infected with the plague pathogen *Yersinia pestis*, *Y. pestis* DNA was found to fragment at a faster rate than human DNA (Figure **??**, panel B). In dental calculus, which contains the genomes of many species, it has been observed that DNA from oral bacteria such as *Streptococcus* spp. fragments at a faster rate than DNA from oral archaea such as *Methanobrevibacter* spp. (Figure **??**, panel C), suggesting that differences in cell membrane stereochemistry between bacteria and archaea may influence the rate of DNA decay. However, in the same dental calculus samples no relationship was observed between the DNA fragmentation rates of *Methanobrevibacter* spp. and human host DNA (Figure **??**, panel D). Thus, while DNA damage does accumulate over time, the rate of degradation can vary across time and space, and even among different organisms within the same sample (Warinner et al. 2017). For these reasons, patterns of DNA damage are typically only compared in a relative manner and not used to predict absolute sample age.

2.7.3 Removing DNA damage

Up to this point, we have primarily discussed how molecular damage can be useful for authenticating ancient DNA. However, once the authenticity of the DNA sequences has been established, it may be useful or even necessary to remove some of this damage - specifically cytosine deamination - before proceeding to downstream analyses like taxonomic classification, metagenomic assembly, strain separation, genome phasing, genotyping, or tree-building. Fortunately, there are multiple laboratory-based and bioinformatic-based strategies available to remove molecular damage from your ancient DNA sequences.

Laboratory-based strategies focus primarily on removing or reducing cytosine deamination from the physical ancient DNA fragments themselves. In 2010, Adrian Briggs and colleagues developed the first enzymatic process for removing uracil-bearing DNA from ancient DNA fragments, focusing initially on ancient DNA from mammoths and Neanderthals (Briggs et al. 2010). In this approach, ancient DNA fragments are phosphorylated using **polynucleotide kinase (PNK)** and then a two enzyme cocktail named **Uracil-Specific Excision Reagent (USER)** is applied. The first enzyme, **uracil-DNA glycosylase (UDG)**, locates and removes uracils from the DNA fragments, leaving abasic sites. The second enzyme, **endonuclease-VIII**, then clips the phosphate backbone at these sites. Because most uracils in ancient DNA are found on single-stranded overhangs, this effectively clips off these damaged ends of the molecules. USER treatment can be advantageous because it removes all of the uracils from a DNA extract, but it also shortens the molecules at the same time. After USER treatment, only the DNA sequences between the innermost uracils of the original aDNA template molecules are likely to be built into a successful

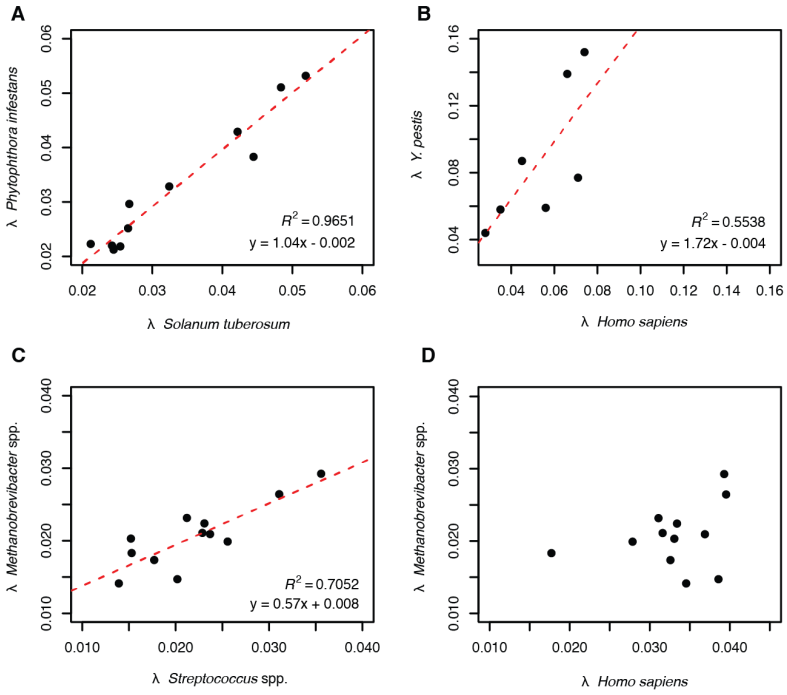


Figure 2.19: Comparison of DNA fragmentation rates ( ) among pairs of organisms within the same sample. (A) Herbarium sample of potato (*S. tuberosum*) infected with potato blight (*P. infestans*); (B) Archaeological human teeth (*H. sapiens*) infected with plague (*Y. pestis*); Archaeological human dental calculus, showing comparisons between (C) common oral microbes of archaeal (*Methanobrevibacter* spp.) and bacterial (*Streptococcus* spp.) origin; and between (D) host (*H. sapiens*) and bacteria (*Streptococcus* spp.). Adapted from (Warinner et al. 2017).

DNA library (Figure **??**, panel A). Damage profiling after USER treatment results in plots with no visible enrichment of base misincorporations (Figure **??**, panel B).

More recently, a modified form of USER treatment known as partial-USER treatment was developed to remove most, but not all, of the uracils in an ancient DNA extract (Rohland et al. 2015). This is achieved by not adding PNK prior to USER treatment and also using a shorter USER incubation time. UDG efficiently excises uracils at most positions, but experimental studies have shown that it will not excise the 5' terminal uracil if it is unphosphorylated (Varshney and Sande 1991). By excluding PNK from the USER treatment, these terminal uracils are selectively retained in the ancient DNA fragment, whereas most other uracils are removed (Figure **??**, panel C). Partial USER treatment also fails to remove terminal 3' uracils, but because 3' overhangs are removed by T4 polymerase during double-stranded library end repair, they are not observed in downstream damage plots. Within ancient DNA, most 5' terminal uracils are believed to be phosphorylated, even without PNK treatment, and thus are excised during partial USER treatment. However, a minority of 5' terminal uracils are unphosphorylated and thus are shielded from excision during partial USER treatment. Through this selective uracil excision, the partial UDG approach provides the advantage of retaining a damage signal in the sequences (for authentication), but limiting it to a predictable location where it can be easily bioinformatically masked or removed for downstream data analysis. As a result, the same DNA library can be used for both authentication and data analysis. Damage profiling after partial UDG treatment results in a plot where only the terminal base at each end of the DNA molecule shows evidence of misincorporation (Figure **??**, panel D). For ancient DNA, partial UDG-treated libraries will always have a terminal misincorporation rate and an average fragment length that is intermediate between USER-treated and non-USER treated libraries. Typically, the 5' terminal misincorporation rate of a partial USER-treated library is approximately one third that of a non-USER treated library (Rohland et al. 2015).

In addition to removing DNA damage, USER treatment also provides an additional benefit in ancient DNA studies by allowing states of epigenetic transcription regulation, such as gene silencing, to be identified. Within mammals, cytosine-guanine dinucleotide repeats are abundant within gene promoters, and 70-80% of the cytosines in these pairs are methylated in the form of 5-methylcytosine (Blackledge and Klose 2011). Concentrations of these repeats, known as **CpG islands**, play important roles in transcription regulation, and the presence of many 5-methylcytosines within these islands is associated with **gene silencing** (Newell-Price, Clark, and King 2000). As shown in Figure **??**, while age-related deamination of cytosine produces uracil, deamination of 5-methylcytosine produces thymine. Thus, if USER treatment is applied to ancient DNA, C→T miscoding lesions due to cytosine deamination are removed, but C–>T miscoding lesions due to 5-methylcytosine deamination remain, allowing patterns of gene silencing to be explored in ancient genomes (Briggs et

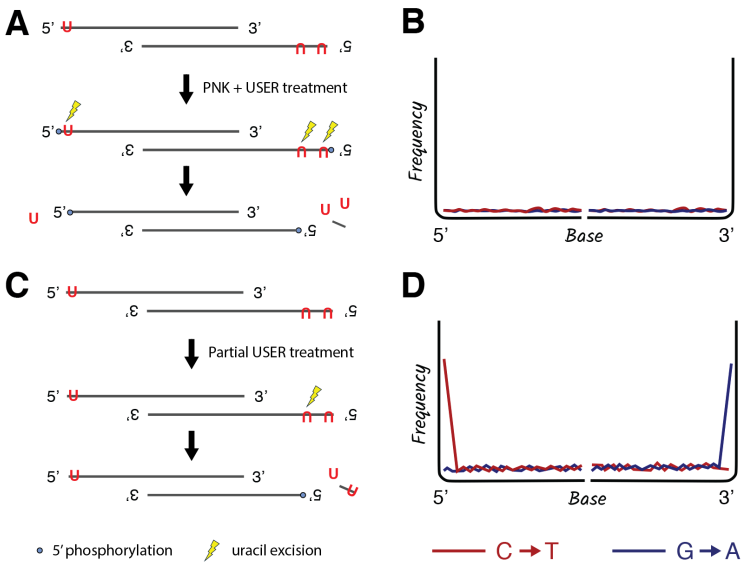


Figure 2.20: Effects of USER and partial USER treatments on ancient DNA. (A) Steps of USER treatment uracil excision; (B) nucleotide misincorporation plot for USER-treated libraries made using a double-stranded preparation method, showing no observable misincorporations; (C) steps of partial-USER treatment uracil excision; (D) nucleotide misincorporation plot for partial USER-treated libraries made using a double-stranded preparation method, showing an excess of misincorporations at a subset of terminal positions only. (B) and (D) are modified from https://nf-co.re/eager/2.5.0, by Zandra Fagernäs under CC-BY 4.0 License, (Fellows Yates, Lamnidis, et al. 2021a).

al. 2010; Hanghøj and Orlando 2019). If necessary, one can even reconstruct damage patterns from USER-treated libraries by quantifying 5-methylcytosine deamination at CpG sites; however, this generally requires many aligned sequences to be feasible.

2.7.4 Single-stranded library preparation

Until now, we have focused our discussion on ancient DNA libraries constructed using a **double-stranded preparation method**, the standard approach for building DNA libraries for high-throughput sequencing. This approach was adapted into a protocol specifically for ancient DNA by Matthias Meyer and Martin Kircher in 2010 (M. Meyer and Kircher 2010b), and today there are several double-stranded library protocols suitable for ancient DNA in use (Carøe et al. 2018; Fellows Yates, Aron, et al. 2021; Harkins et al. 2020). These approaches are very similar to those used for modern DNA but with a few key differences: first, they generally employ blunt-end ligation because T/A ligation has been shown to induce undesired effects such as GC bias and other artifacts in library construction for ancient DNA (Seguin-Orlando et al. 2013); and second, they typically do not employ a mechanical or enzymatic shearing step (M. Meyer and Kircher 2010b). Shearing is generally unnecessary for ancient DNA because it is already fragmented to sizes within the optimal range for short read sequencing (i.e., <600 bp), and the avoidance of shearing further prevents high molecular weight contaminant DNA from being successfully built into libraries, thereby reducing modern contamination in the libraries. Other protocol steps sometimes applied during modern DNA library construction (e.g., enzymatic tagmentation, size-selective purification, use of uracil-bearing hairpin adapters, commercial single-tube protocols) are not suitable for ancient DNA as they are likely to fail, introduce biases, or enrich for contamination when applied to highly fragmented, low molecular weight ancient DNA (Carøe et al. 2018; Kapp, Green, and Shapiro 2021).

While double-stranded preparation methods can be used to build successful DNA libraries from ancient DNA, they include several steps that disadvantage ancient DNA and can lead to reduced library construction efficiency. For example, because the blunt-end ligation step is non-directional, approximately half of the library will be improperly constructed and consist of DNA molecules containing either only a P5 (IS1/IS3) or only P7 (IS2/IS3) adapter configuration instead of the desired combination of one P5 (IS1/IS3) and one P7 (IS2/IS3) adapter[2] (Figure **??**). Such improperly constructed molecules will not be sequenced. While such a loss is generally trivial for modern DNA, it can be highly consequential for ancient DNA, particularly for samples with very low DNA abundance or low molecular complexity. Further losses occur for ancient DNA molecules containing intra-strand nicks, as these will also fail to amplify properly during the library amplification stage, resulting in further library loss.

[2]See (M. Meyer and Kircher 2010b) for an explanation of adapter and primer nomenclature for double-stranded library preparation.

Finally, the many purification steps required during double-stranded library preparation methods lead to yet more DNA losses, as short DNA fragments are only inefficiently retained by binding to silica or carboxylated beads during these cleaning steps. The development of new single-tube protocols substituting heat denaturation for washing during double-stranded library preparation (Carøe et al. 2018) mitigates some but not all of these losses.

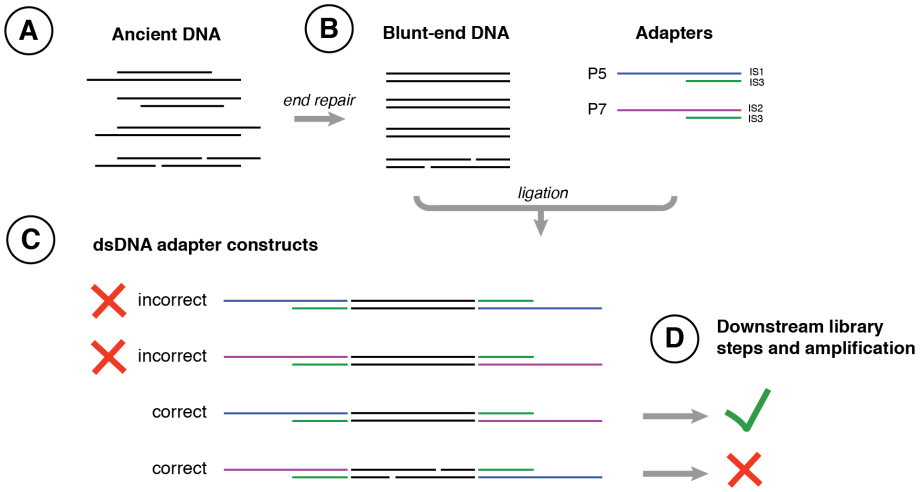


Figure 2.21: Sources of DNA loss during double-stranded library preparation methods. (A) Ancient DNA is end-repaired, resulting in the removal of 3' overhangs and the filling in of 5' overhangs. (B) P5 and P7 adapters and the now blunt-ended ancient DNA molecules are ligated together, resulting in (B) a mixture of incorrect (P5-P5 or P7-P7) and correct (P5-P7 or P7-P5) dsDNA adapter constructs. Of the correctly formed adapter constructs, only those without single-stranded nicks will successfully amplify to produce viable DNA libraries for sequencing (C). Image by Christina Warinner under a CC-BY-SA 4.0.

To address such issues, a **single-stranded library preparation method** was developed in 2012 for use on highly degraded Denisova remains (M. Meyer et al. 2012), and then formalised as a protocol by Marie-Theres Gansauge and Matthias Meyer in 2013 (Gansauge and Meyer 2013), and later refined into protocol ssDNA2.0 in 2017 (Gansauge et al. 2017) and automated for robotics in 2020 (Gansauge et al. 2020). This method first heat denatures ancient DNA into single-stranded fragments and then tightly binds these fragments to streptavidin beads, allowing a substantially more controlled and efficient library construction (Figure **??**). Optionally, ancient DNA can be pretreated with Endonuclease VIII to create nicks at abasic sites prior to single-stranded library preparation in order to avoid polymerase stalling during library amplification (Gansauge and Meyer 2013; Psonis, Vassou, and Kafetzopoulos 2021), but this step generally provides little empirical benefit and so it is no longer implemented in current

protocols (Gansauge et al. 2017). In studies of a wide variety of ancient DNA sources (Bennett et al. 2014; Gansauge and Meyer 2013; Wales et al. 2015), the single-stranded library preparation method was found to result in major improvements in DNA recovery, and in some cases an improved endogenous DNA recovery. However, single-stranded library preparation methods are also known to produce libraries with shorter median DNA fragment lengths (Bennett et al. 2014), likely due to the improved retention of very short DNA using streptavidin binding during washing steps and the recovery of nicked DNA that was otherwise lost during double-stranded DNA library preparation methods. Libraries produced by single-stranded library preparation methods are typically 10-40 nucleotides shorter than libraries prepared using double-stranded library preparation methods, and while this poses few disadvantages for studies of large-bodied eukaryotes with well characterised reference genomes, it can introduce challenges for studies of microorganisms where DNA length is important for species deconvolution and de novo genome assembly from large metagenomics datasets (Koren and Phillippy 2015; Kingsford, Schatz, and Pop 2010; Jiang et al. 2012). Currently, multiple single-stranded library preparation protocols are in use for ancient DNA (Gansauge et al. 2020; Gansauge and Meyer 2019; Kapp, Green, and Shapiro 2021).

2.7.5 Damage patterns in DNA prepared using single-stranded library protocols

An important feature of single-stranded library preparations is that they do not involve an end-repair step, and consequently damage on the 3' end of the DNA molecule is retained. As a result, uracils can be present on both the 5' and 3' ends of the DNA molecule and adenines are not misincorporated on the 3' end. Thus, damage plots for ancient DNA prepared using single-stranded library protocols will show an enrichment of C to T substitutions on both the 5' and 3' ends (Figure **??**). Optionally, USER treatment or partial-USER treatment can be applied to ancient DNA prior to single-stranded library preparation (Psonis, Vassou, and Kafetzopoulos 2021; Gansauge and Meyer 2013), although this is less commonly performed for single-stranded library preparations than for double-stranded library preparations, in part because it can result in libraries with high proportions of DNA <30 bp long.

2.7.6 Damage plot review

Having now examined DNA damage in detail, let's now review again the most common damage patterns likely to be encountered in ancient DNA studies and what they signify (Figure **??**). The first plot (Figure **??**, panel A) shows no misincorporation; this is the pattern expected for modern DNA or ancient DNA that has been USER-treated. The second plot (Figure **??**, panel B) shows a spiky profile indicating that the analysis is likely underpowered and requires more DNA sequences to make an accurate damage plot. Alternatively, there may be residual ultrashort DNA sequences (<30 bp) in the dataset causing

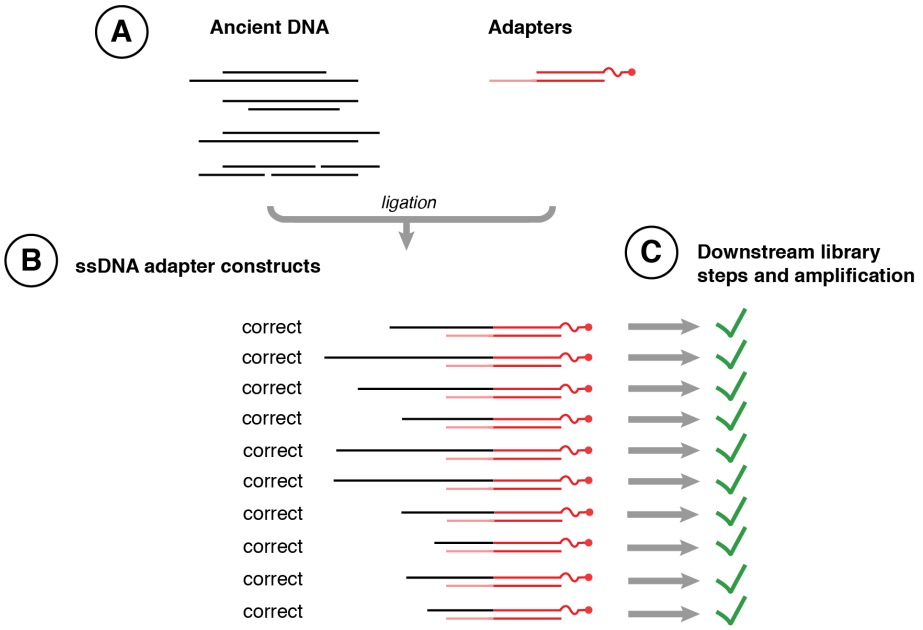


Figure 2.22: Improved retention of ancient DNA during single-stranded library preparations. (A) Biotinylated adapters and a pool of ancient DNA molecules are ligated together at high temperature, resulting in denaturation of the ancient DNA strands and (B) correct ssDNA adapter constructs for each single-stranded DNA molecule, including those formed by nicks. All resulting adapter constructs are suitable for downstream library steps, amplification, and sequencing (C). Image by Christina Warinner under a CC-BY-SA 4.0.

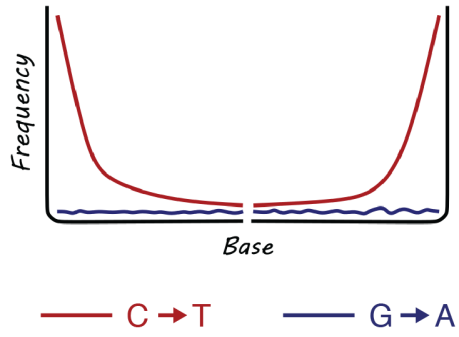


Figure 2.23: DNA damage pattern for DNA prepared using single-stranded library protocols. C to T substitutions are observed on both the 5' and 3' ends of the DNA molecule. Modified from https://nf-co.re/eager/2.5.0, by Zandra Fagernäs under CC-BY 4.0 License

noise; if present, these should be removed and the analysis repeated. The third plot (Figure **??**, panel C) looks like an expected ancient DNA profile for a library prepared using a double-stranded library preparation method, except that it has an elevated baseline. This indicates that the reference genome used for the analysis is incorrect. To remedy this, a closer reference genome should be selected, or may be created through metagenomic assembly. The fourth plot (Figure **??**, panel D) shows an expected ancient DNA profile for a library made with a double-stranded library preparation method and using an appropriate reference genome. The fifth plot (Figure **??**, panel E) shows a typical DNA damage pattern for ancient DNA treated with the partial-USER protocol, followed by a double-stranded library preparation. Finally, the sixth plot (Figure **??**, panel F) shows an expected DNA damage profile for ancient DNA that has been prepared using a single-stranded library preparation method.

2.7.7 Enzyme alert

At this point, we have only briefly discussed the role of DNA polymerases in DNA library preparation methods, but selection of appropriate DNA polymerases is a critical step in all ancient DNA studies. This is in large part because DNA polymerases greatly differ in how they respond to uracil, a key form of DNA damage present in ancient DNA (Heyn et al. 2010). In life, uracils are an essential component of RNA, where they are one of the four bases that make up the RNA code. Its counterpart in DNA is thymine, a chemically similar base that differs only in a single methyl group (Figure **??**). Although not typical of DNA, uracil can spontaneously form in DNA through various enzymatic errors and chemical damage, but such uracils are rapidly removed by cellular repair enzymes, including uracil-DNA glycosylases (UDG) (Kavli et al. 2007; Krokan, Drabløs, and Slupphaug 2002; Visnes et al. 2009).

Because of the close chemical similarity between uracil and thymine, most DNA polymerases are unable to distinguish between the two bases, treating both as a thymine. Such DNA polymerases are called **non-proofreading enzymes**. T4 polymerase is one such non-proofreading enzyme, and this is why T4 polymerase incorporates a complementary adenine for each uracil it encounters when repairing 5' overhangs, thus preserving the C to T damage signal. However, not all DNA polymerases treat thymines and uracils as equivalent. Some DNA polymerases simply stall at uracil positions, either stopping or disengaging from the DNA strand. Such enzymes belong to a group known as **proofreading enzymes**. Proofreading enzymes are characterised by their ability to recognise misincorporated nucleotides during DNA replication. Only archaea produce the proofreading DNA polymerases capable of recognizing uracil (Wardle et al. 2008), and most proofreading enzymes used in molecular biology are derived from the archaeal species *Pyrococcus furiosus* (the source of Pfu and Phusion enzymes).

It is important to know that many commercial library preparation kits developed for modern DNA cannot be used for ancient DNA because they incorporate

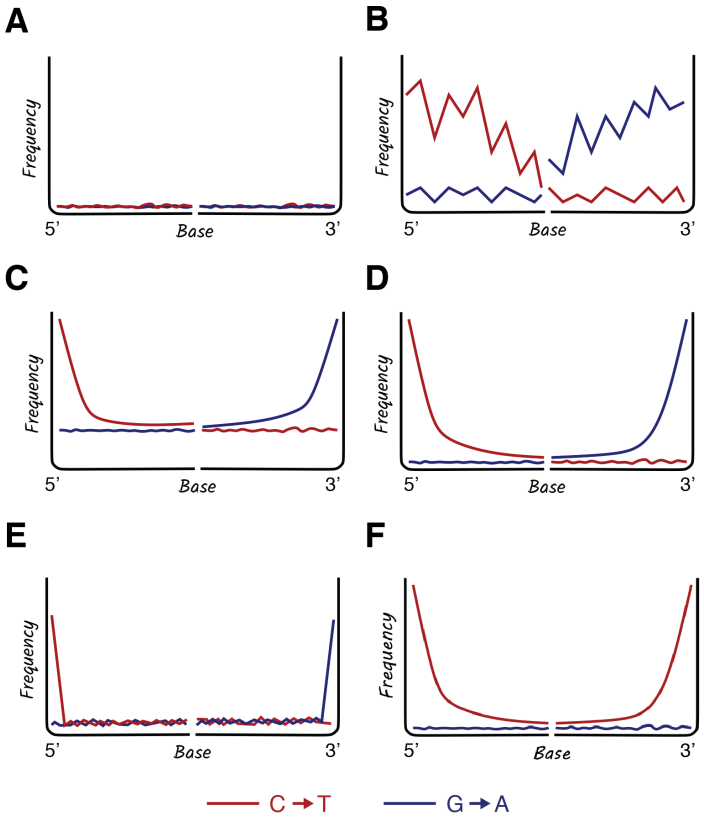


Figure 2.24: Expected DNA damage profiles for different DNA treatment methods and library preparation protocols. (A) Modern DNA or USER-treated ancient DNA; (B) insufficient DNA for analysis or inclusion of DNA <30 bp; (C) ancient DNA prepared using a double-stranded library preparation method, but aligned to the wrong reference genome; (D) ancient DNA prepared using a double-stranded library preparation method; (E) partial USER-treated ancient DNA prepared using a double-stranded library preparation method; (F) ancient DNA prepared using a single-stranded library preparation method. Model plots are adapted from (https://nf-co.re/eager/2.5.0), by Zandra Fagernäs under CC-BY 4.0 License, (Fellows Yates, Lamnidis, et al. 2021a).

proofreading enzymes that stall at uracil sites (Kapp, Green, and Shapiro 2021), causing ancient DNA libraries to become depleted in damaged sequences and resulting in library failure, skewed damage profiles, or a bias towards undamaged contaminant sequences. It is essential to ensure that only non-proofreading DNA polymerases are used during the end-repair and subsequent PCR steps of ancient DNA library preparation, and it is only after the damage signal has been "locked in" by placing complementary adenines opposite each uracil during these repair/replication steps that proofreading enzymes can be used for subsequent steps. USER-treated ancient DNA represents a special case; if uracils are fully removed during USER treatment, USER-treated ancient DNA is compatible with the use of proofreading DNA polymerases during all stages of library preparation. Knowing how different DNA polymerases react to uracil and choosing the appropriate enzyme to use at each step of DNA library preparation is critical for the successful recovery and sequencing of ancient DNA.

After learning of the dangers in using proofreading enzymes during ancient DNA library preparation, one might wonder why proofreading enzymes are used at all. The reason is because they are more accurate, and when used judiciously they can improve the sequence fidelity of your DNA datasets. In a typical ancient DNA workflow, end-repair is performed using the non-proofreading enzyme T4 polymerase, and the indexing PCR is performed using a non-proofreading DNA polymerase, such as Platinum Taq or Pfu Turbo Cx (an archaeal enzyme that has been modified to remove its uracil recognition domain). Once these steps are complete, the resulting DNA library can be subsequently treated like modern DNA and high fidelity proofreading enzymes, such as Herculase II Fusion or Phusion HS II, can be used for all subsequent amplifications, reamplifications, and reconditioning steps in preparation for downstream analyses, such as DNA-capture enrichment or DNA sequencing.

2.8 Ancient microbes: a 60,000 piece jigsaw puzzle

Earlier, we discussed how ancient DNA is like the world's worst jigsaw puzzle and that reconstructing human chromosome 1 from ancient DNA is like trying to assemble a puzzle from 5 million pieces. The problem is not quite so immense for microbes because their genomes are much smaller. So rather than having a 5-million-piece puzzle, reconstructing a typical bacterial genome is more like working with a 60,000-piece puzzle. However, this is admittedly somewhat misleading because ancient microbial species are almost never found in isolation; instead ancient microbes are usually found communities comprising tens to thousands of distinct species, so a more realistic metaphor would be one in which you attempt to reconstruct a hundred mixed-up 60,000-piece puzzles simultaneously. And, making things even worse, some of these puzzles are present more than once.

Taking a step back, let's examine the important role of DNA damage in studies of ancient DNA. Why does DNA damage matter so much? DNA damage is useful because it allows for the authentication of ancient DNA at the level of species (Jónsson et al. 2013a), contigs (Borry et al. 2021b), and even individual sequences (Skoglund et al. 2014). In studies of plague, for example, DNA damage has been used to support the ancient origin and authenticity of reconstructed Yersinia pestis genomes (Andrades Valtueña et al. 2022a; Rascovan et al. 2019; Rasmussen et al. 2015; Spyrou et al. 2018). Likewise, DNA damage has been used to support the metagenomic assemblies of fecal bacteria in ancient palaeofaeces (Borry et al. 2020; Maixner, Sarhan, Huang, Tett, Schoenafinger, Zingale, Blanco-Míguez, Manghi, Cemper-Kiesslich, Rosendahl, Kusebauch, et al. 2021; Wibowo et al. 2021), and to authenticate atypical taxa of Pleistocene origin in ancient dental calculus (Klapper et al. 2023b). DNA damage can also aid in the correct identification of low abundance pathogens by confirming that damaged sequences show appropriate edit distance profiles for the taxon of interest (Hübler et al. 2019a).

Nevertheless, DNA damage also poses taxonomic challenges for studies of ancient microbes, especially as it relates to the correct identification of specific taxa, the accurate mapping of microbial genomes, and successful reconstruction of metagenomically assembled genomes (MAGs). However, it has become apparent that the greatest challenge for reconstructing ancient microbes and their communities is not cytosine deamination, but rather DNA fragmentation. Cytosine deamination has been shown to only minimally impact taxonomic assignment (Velsko et al. 2018), and some de novo assemblers (e.g., MEGAHIT) are relatively robust to substitution errors (Klapper et al. 2023b). Moreover, cytosine deamination can be removed or mitigated through USER-treatment, or identified and corrected bioinformatically through deeper sequencing and high coverage genome generation. DNA fragmentation, however, remains a serious and insoluble problem. Ancient DNA is simply very short, and this makes ancient metagenomics more difficult.

Very short sequences often cannot be uniquely assigned to specific strains or species, leading to ambiguous or false taxonomic assignments in read-based analyses, and sequences less than 250 bp typically produce complex assembly graphs with short contigs and a low **N50** (length of the shortest contig needed to account for at least 50% of the total assembly). DNA sequence length is also one of the most important factors affecting ancient DNA assembly success. While some sources of ancient microbial DNA, such as palaeofaeces in very dry caves, have been found to have long median DNA fragment lengths of ~175 bp (Wibowo et al. 2021), most ancient microbial DNA is very short and similar to other sources of ancient DNA. The tipping point for ancient DNA studies occurs when DNA fragments become shorter than 30 bp. Such ultrashort fragments lack sufficient specificity for taxonomic or even genetic placement as they align to too many genomes with no phylogenetic coherence. Thus, DNA degradation imposes some hard limits on the study of ancient microbial DNA.

2.8.1 The limits of ancient DNA

What are the theoretical limits of ancient DNA? DNA decay modeling has predicted an approximately 3-million-year limit for ancient DNA studies – not because DNA does not survive beyond 3 million years, but rather because it is unlikely to survive in fragments of >30 bp beyond 3 million years, even under ideal conditions. In a study of DNA decay rates, Morten Allentoft and colleagues estimated the half-life of a 30 bp DNA fragment to be 158,000 years at -5 ° C (Allentoft et al. 2012). For a genetic sample with a starting amount of 1 million DNA molecules, this would mean that no DNA fragments of this length would be left after a period of ca. 3.3 million years. Moreover, they estimate that after 6.8 million years, the average length of ancient DNA would be 1 bp. Such models place deep time palaeobiology outside the realm of palaeogenomics, at least using current in-solution methods in which any spatial information still retained by DNA molecules in its parent substrate is lost during extraction, library preparation, and sequencing.

Three recent ancient DNA studies, each focusing on remains in permafrost, seem to confirm these estimates. The first, a study of mammoths dating to the Early and Middle Pleistocene (Valk et al. 2021), reported that the recovered DNA was exceptionally short (mostly <35 bp) and nearly saturated with cytosine deamination at the terminal 5' position. The two oldest mammoths in the study, dating to ca. 1.1 Mya, are the currently oldest eukaryotic genomes to be successfully reconstructed using ancient DNA. The two other studies focused on ca. 2 Mya eDNA recovered from deep subsurface permafrost sediments in Greenland, and they identified diverse ancient plant and animal (Kjær et al. 2022a) and microbial and viral (Fernandez-Guerra et al. 2023) DNA sequences. Analysis of the ancient plant and animal DNA has allowed the reconstruction of Late Pliocene and Early Pleistocene environments in Greenland, while the recovered microbial and viral DNA provides insights into the soil microbial ecology of this period. As with the mammoth study, eukaryotic DNA from the sediment cores was found to be mostly <35 bp in length and nearly saturated with cytosine deamination at the terminal 5' position, suggesting the ancient DNA in these samples falls near the limit of sufficient DNA preservation for palaeogenomic analysis. The microbial DNA was slightly more intact, likely reflecting an extended period of microbial survival in subsurface habitats, as is expected for environmental microbes in geological contexts.

2.9 Summary

The study of ancient DNA has changed enormously since its beginnings in the 1980s. Gone are the days of electrophoretic gels and rulers for ancient DNA sequencing. Today we have instruments capable of churning out more than 50 billion ancient DNA sequences in a single 48-hour run. This means that aspiring archaeogeneticists need to have a solid foundation in computer science and a proficiency in coding and scripting in languages such as R and Python. During

life, genomes are large high molecular weight molecules, but they fragment into thousands or millions of pieces once an organism dies. The very short and ultra-short fragment lengths of ancient DNA makes taxonomic identification, genome mapping, and metagenomic assembly more challenging than for modern DNA, and ancient DNA analysis generally requires the use of specialised software or parameter modifications to account for these differences. Robust and powerful tools now exist to characterise the chemical damage that accumulates in ancient DNA, and factors such as fragment length and cytosine deamination can be used to authenticate ancient DNA, but not provide a precise age estimate. Studies of ancient DNA require specialised laboratories and library protocols in order to appropriately manipulate and handle ancient DNA, and we now have options to remove or mitigate some forms of DNA damage through USER-treatment, or we can choose to maximise recovery of DNA damage through the use of single-stranded DNA library protocols. DNA fragmentation remains the biggest challenge in ancient metagenomics, but tremendous strides in ancient DNA laboratory and bioinformatics methods, in parallel with massive gains in sequencing technology, have broadened the scope of palaeogenomics studies to an enormous diversity of sample types, environments, and applications over the past 3 million years.

2.10 References

Chapter 3

Introduction to Metagenomics

> **!** Important
>
> This page is still under construction

This chapter is still under construction. For the slides and recorded lecture version of this session, please see the Werner Siemens-Stiftung funded SPAAM Summer School: Introduction to Ancient Metagenomics website.

Chapter 4

Introduction to Microbial Genomics

Microbial genomics is the study of microbial genomes, their structure, composition and evolution. In this chapter, we will briefly go over the basics of microbial genomics and what you need to know to study microbial genomes using ancient DNA.

4.1 Introduction

Microbial species come from diverse groups of organisms but are generally defined as organisms which are not visible to the naked eye. The 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). Another category of viruses are **retroviruses**. These will always 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 viral elements** (EVEs).

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 specialised 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.

4.2 Larger Genomic Elements

When using reference sequences from public depositories, such as NCBI, you will often be confronted with multiple sequences 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 extra-chromosomal 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 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.

4.3 Sampling & Sequencing

For ancient DNA samples, screening for organisms of interest is usually performed on "shotgun" sequencing data, which can then be either further genomically 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 (but could still reactivate later). 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 in your data 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 vascularised 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.

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 records 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, genome size and complexity. While a bacterium is generally easier to detect, their large genome size and sequence conservation, can make 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/unbiased sequencing is key, and because of 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

simultaneously.
- 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 regard to authentication if enrichment is performed on UDG/Half-UDG genomic libraries.

i 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 regard 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.

4.4 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 using taxonomic 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 related 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 sampled tissue and based on your laboratory workflow.

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

1. Authentication of the data as aDNA using deamination signatures and fragment lengths.
2. Comparative or competitive mappings to relevant reference sequences (target organism and closely related pathogenic/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 second one you will see low coverage across the whole sequence. The first one is likely to be a false positive, as your detection is probably based on intervals from either a commensal or an environmental organism which have very high sequence similarity to your reference genome. This can be further worsened by low complexity intervals. These peaks will usually also be reflected 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.

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 of 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 specialised 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. Additionally, some viruses are highly divergent within the same species.

Finally, intervals or mutations specific to the target organism should be identified and investigated. This can range from plasmids, genes, and 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.

4.5 Genomes

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).

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 section 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 ab-

sence 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.:
`AAAAAAAAAAAAAAAAAAAA` » *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.:
GCCCCCCCGGAATGGACCCGCGCCT » *80% GC*
ATTTTGAACCTAATTTATATAGCAA » *20% GC*

4.5.1 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 specialised (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 strains have lost, but the ancestral genomes might still have retained.

4.5.2 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.
- **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 specialised organisms. So while they are much more plastic than strictly clonal species, the available genetic pool for exchange will be smaller.

4.5.3 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 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 an annotated reference sequence and SNP/MNP/Indel calls. The problem when using 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, and consider potential issues.

Important exception are known and heavily conserved mutations, which for example are involved in **pseudogenisation**. Pseudogenisation 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 a critical part of the genome reduction process of highly specialised bacteria.

4.5.4 Virulence Associated Intervals

One of the questions 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 the host or the taxon's ecological niche.

4.6 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 & aetiopathology:

- 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.

4.7 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/

4.8 Questions to think about

Get to know your genome!

- Is the genome circular or linear?
- Does the species carry any plasmids?
- How genomically 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 sample grouping within modern diversity or basal too 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 & aetiopathology:

- 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?

4.9 Summary

- There are many different types of microbial organisms with distinct genomic structure and evolutionary dynamics!
- Species need to be carefully validated before genomic analysis
- Potential research questions will depend on the taxon (its genomic structure and evolutionary dynamics) and the available modern data

Chapter 5

Introduction to Environmental DNA

5.0.1 What is environmental DNA?

DNA can be recovered from virtually anywhere. In fact, DNA can be found in the environment from cellular material shed by organisms (e.g., skin flakes, urine, faeces, hair, mucus, or their natural demise) that have accumulated in the surrounding water, soil, air, sediment (Figure **??**). We call this type of DNA, **environmental** (eDNA).

Environmental DNA (eDNA) refers to the genetic material that can be extracted and studied from environmental samples without the need to isolate target organism first (Taberlet et al. 2018).

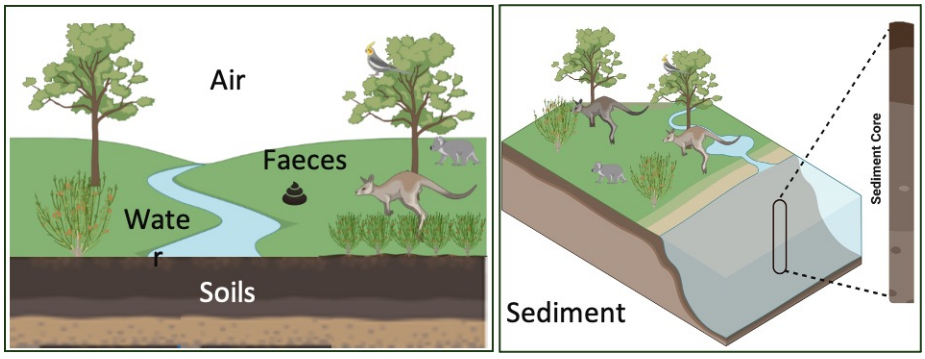


Figure 5.1: Examples of where environmental DNA can be recovered from the environment.

A defining feature of eDNA is its complexity. In environmental samples, we

encounter a diverse mix of genetic material from multiple species and individuals, presenting unique challenges in the analysis compared to the analysis of genetic material of a single organism (Figure **??**). Additionally, in some environmental substrates, DNA from certain organisms can be preserved for long periods even after the organisms have died. This preserved DNA is referred to as ancient eDNA (Pedersen et al. 2015).

Introducing the element of time to the already complex eDNA scenario further intensifies its complexity. Time can be unkind to DNA, and the deeper we venture into the past, the more challenging it becomes to recover intact genetic material (Figure **??**). Ancient environmental DNA embodies the intertwining challenges of temporal degradation and environmental complexity. Consequently, the earliest studies in this field have emerged recently in the early 2000s, only two decades ago (Coolen and Overmann 1998; Willerslev et al. 2003).

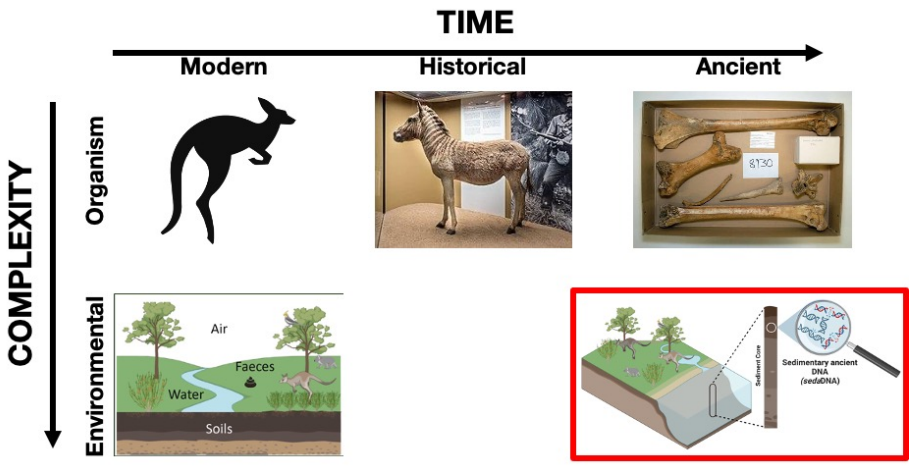


Figure 5.2: Ancient eDNA embodies the intertwining challenges of temporal degradation and environmental complexity.

Over the past decade, there has been a growing interest in researching ancient eDNA, particularly the DNA present in sedimentary archives (E. Capo, Barouillet, and Smol 2023).Sedimentary DNA can be well-preserved in freshwater and marine sediments, palaeo-soils, caves and permafrost. This preservation is facilitated by oxygen-depleted and cold sediments in aquatic environments, as well as the sheltered conditions provided by caves, which can offer relatively stable temperature and humidity (E. Capo, Barouillet, and Smol 2023). These preservation conditions allows for significant potential in reconstructing past environments and tracking ecological changes over time (Kjær et al. 2022b).

5.1 What can we find in sedimentary DNA records?

Sediments consist of inorganic and organic matter, including eDNA, that accumulate over time, creating temporally resolved archives (Picard et al. 2024). The vertical stratigraphy of these sediments allows for the reconstruction of temporal transects through the sediment core. In aquatic environments, the sedimentary DNA record comprises a mixture of genetic materials from organisms that live within the lake (autochthonous material) and those from the surrounding area and its catchment (allochthonous material) (Eric Capo et al. 2022b). In terrestrial enviroments like caves, the sources of DNA that form the sedimentary record are different. Caves can act as natural traps where organisms repeatedly fall in and perish, or as places of refuge where organisms seek shelter (Murchie et al. 2023). Consequently, the sedimentary records in caves can include genetic material from these organisms. Additionally, microorganisms can live within cave sediments, integrating their genetic material into the sedimentary DNA record, similar to aquatic environments.

The pool of DNA preserved within sediments can be categorised into two primary fractions (Eric Capo et al. 2022b) (Figure **??**):

1. **Ancient molecules:** This category comprises DNA found within dead cells or as extracellular DNA, either in its free from or attached to particles
2. **DNA within viable or living cells:** This includes DNA in cysts, spores, pollen, and eggs These cells may be actively growing within the sediments or in a dormant state, capable of reactivation under favourable environmental conditions

Therefore, (Eric Capo et al. 2021) defined the two distinct fractions of DNA found in sediment based on their degradation states as:

1. **Sedimentary DNA (sedDNA):** includes DNA that is relatively young and better preserved
2. **Sedimentary ancient DNA (sedaDNA):** comprises older DNA that tends to be more poorly preserved

5.2 Taphonomic processes of sedimentary DNA

The reliability of DNA records is influenced by taphonomic processes (Sand et al. 2024; Giguet-Covex et al. 2023). Taphonomic processes refer to the various mechanisms that regulate the production, transfer, incorporation, and preservation of sedimentary DNA. In this section, we will focus on aquatic environments (Figure **??**), but similar processes can also affect sediment records in caves.

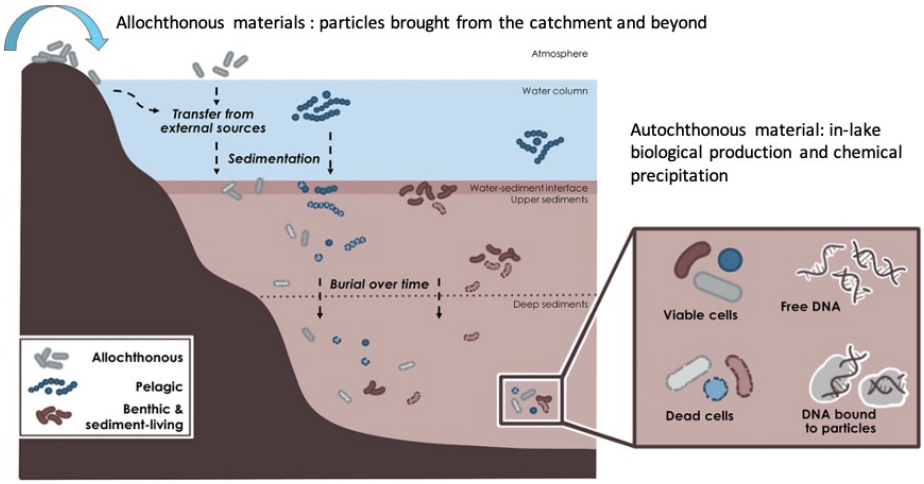


Figure 5.3: Composition of the DNA pool preserved in aquatic sediments. Modified from (Eric Capo et al. 2022b)

5.2.1 Origin of eDNA in sediments

Several factors come into play in determining how DNA is incorporated and buried within the sediments (Giguet-Covex et al. 2023). These include the abundance and spatial distributio of organisms, their ability to form cysts or spores, and their edibility (Eric Capo et al. 2021). For terrestrial species living in the surrounding area of aquatic environments, understanding material and DNA transport from the original habitat is essential to ensure reliable interpretation of the genomic records. For example, more terrestrial plant DNA is expected to be transported to lakes with high soil erosion and well developed hydrological connections (Garcés-Pastor et al. 2023).

5.2.2 Fate of eDNA in the environment

Upon release into the environment, DNA molecules can undergo various processes (Torti, Lever, and Jørgensen 2015).

- **Biotic degradation**: Extracellular DNA becomes susceptible to extracellular and cell-associated DNases, which are likely the most immediate cause of DNA degradation (Torti, Lever, and Jørgensen 2015). These enzymes are widespread in most environments and catalyse the hydrolysis of the phosphodiester bond between the phosphate and the deoxyribose moieties of DNA.
- **Natural transformation**: Living cells can take up extracellular DNA molecules from the environment, even if the DNA is extensively damaged, and integrate them into their own genomes (Overballe-Petersen et al. 2013).

- **Abiotic decay**: As you already saw in the ancient DNA lecture, DNA repair mechanisms cease to be active, and DNA will start accumulating lesions inflicted by various abiotic (physical and chemical) factors (Orlando et al. 2021).
- **Long-term preservation in the environment**: Despite the occurrence of the previous processes, extracellular DNA can still persist in the environment. Optimal DNA preservation conditions in aquatic environments include systems with cooler temperatures, lakes with minimal bioturbation, anoxic conditions at the sediment-water interface, and freshwater and neutral to slightly alkaline lakes (pH 7-9) (Jia et al. 2022). DNA can be protected from nucleases by binding to humic acids due to a negative surface charge, and therefore prolong DNA survival (Stotzky 2000). DNA is also protected from degradation upon adsorption onto mineral grains in the sediment matrix, which reduces DNA accessibility to nucleases. This adsorption process is crucial for preserving DNA molecules in sediments for millennia.

Let's delve deeper into this preservation process.

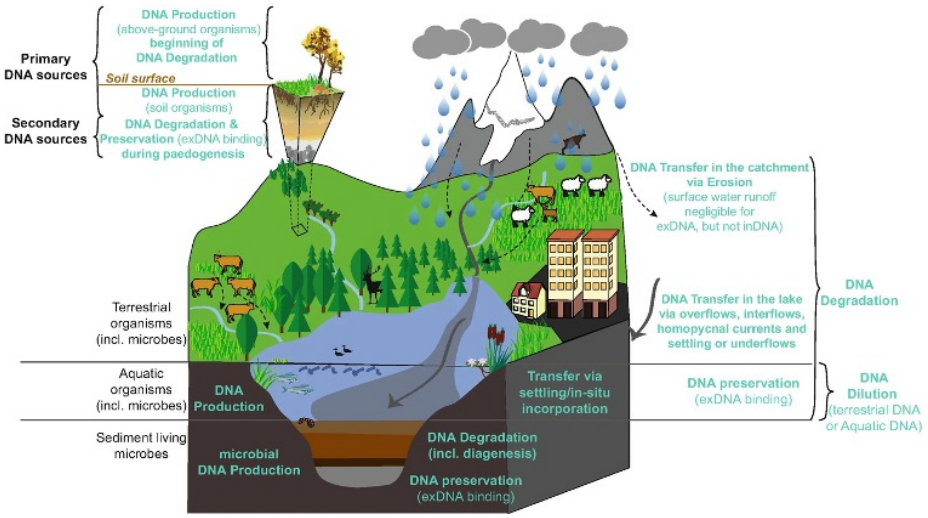


Figure 5.4: Taphonomic processes affecting sedimentary DNA signals from terrestrial, water and sedimentary environments (Giguet-Covex et al. 2023)

5.2.3 Mineralogical influence on preservation

Different minerals have varying adsorption capacities, yet research has demonstrated that specific clays appear to be the most effective at preserving eDNA compared to other common soil and sediment silicates (Lorenz, Aardema, and Krumbein 1981; Kjær et al. 2022b). For example, clay minerals such as montmorillonite (belonging to smectite clays) can absorb more than their own weight

in DNA, because of their relatively large negatively charged surface area (Pedersen et al. 2015). Compared with clays, sand has been found less effective in binding DNA, but adsorption increases with divalent cation concentrations (Ca2+ and Mg2+) (Lorenz and Wackernagel 1987). We can use this information to target layers in sediment records that have high clay content as hotspots for ancient eDNA (Kanbar et al. 2020).

How do they work? Let's take smectite clay as an example (Figure **??**). Smectite clays belongs to the 2:1 clay type, characterised by two tetrahedral layers surrounding an octahedral layer. In the case of smectites, the middle octahedral layer contains a variable cation content. When spaces become available, a charge is generated. This charge allows both organic and inorganic compounds to intercalate into this layer, effectively 'nano-confining' and shielding the DNA from the various factors that might otherwise cause its degradation over time, such as nucleases. These nucleases can themselves adsorb to sand and clay, potentially inhibiting their ability to hydrolyse extracellular DNA (Beall et al. 2009).

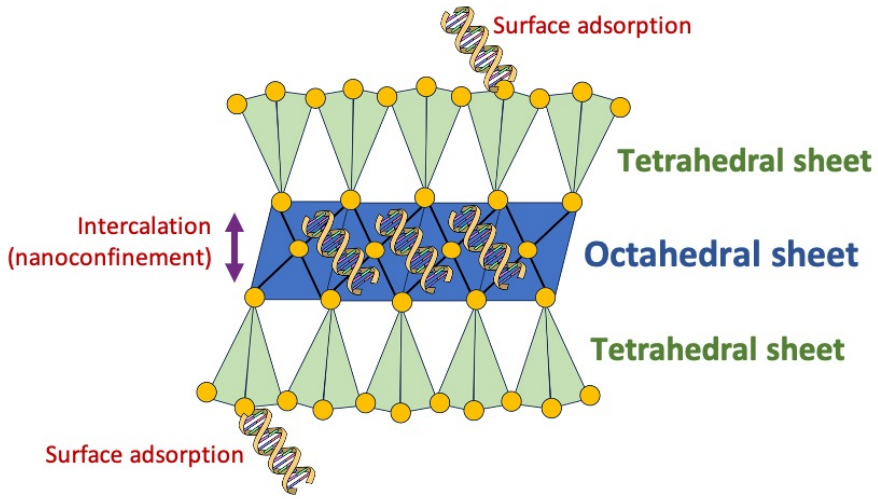


Figure 5.5: DNA binding to smectite clay can protect it from nucleases and degradation.

5.3 DNA analysis: laboratory processing

The reliability of DNA records is influenced not only by taphonomic processes but also methodological aspects. As with any aDNA study, all laboratory processes must be conducted in specialised aDNA facilities, adhering to strict aDNA protocols when handling samples, as covered in previous lectures.

Once the sediment sample has been collected, the first laboratory processing

step involves DNA extraction (Heintzman et al. 2023; Picard et al. 2024) (Figure **??**). This process involves isolating and purifying DNA from various biological samples and separating it from other cellular or environmental components.

DNA extraction methods can vary depending on the DNA source and the specific downstream applications. In the field of eDNA and therefore, sedimentary DNA, the choice of DNA extraction method can significantly impact the resulting DNA composition. Some methods may fail to recover a significant number of DNA molecules from a sediment sample, affecting the accuracy of the reconstruction of the genetic record (Eric Capo et al. 2021).

A wide range of methods have been used in sedimentary DNA research, ranging from commercial kits (e.g., Powersoil® and PowerMax® from Qiagen) to custom protocols optimised for recovering short DNA fragments, a characteristic feature of ancient DNA (Epp, Zimmermann, and Stoof-Leichsenring 2019). Studies comparing different extraction methods have emphasised the importance of selecting methods optimised for different sediment types and taxonomic targets (e.g., Slon et al. 2017; L. Armbrecht et al. 2020; Murchie et al. 2021; Pérez et al. 2022).

After extracting eDNA, various analytical methods have been used to analyse the DNA (Heintzman et al. 2023; Picard et al. 2024). These methods encompass the following approaches:

1. Techniques designed to detect or quantify specific target organisms using endpoint PCR assays, quantitative real-time qPCR, and specialised applications like droplet digital ddPCR.
2. DNA metabarcoding methods that involve PCR amplification of marker loci, often referred to as "barcodes," coupled with high-throughput sequencing.
3. Metagenomic strategies centred on untargeted shotgun sequencing of the entire pool of DNA retrieved from sediment.
4. Hybridization-based target enrichment methods for the recovery of DNA fragments of interest from the sediment metagenome. Hybridization capture involves the use of short RNA probes, often referred to as "baits," which are designed to complement specific DNA sequences of interest, such as taxonomic marker genes. These baits can then be employed to selectively extract the target DNA fragments from the DNA extract for subsequent sequencing.

Recovered DNA sequences are then taxonomically and/or functionally annotated using a suite of bioinformatic tools to answer palaeoecological questions.

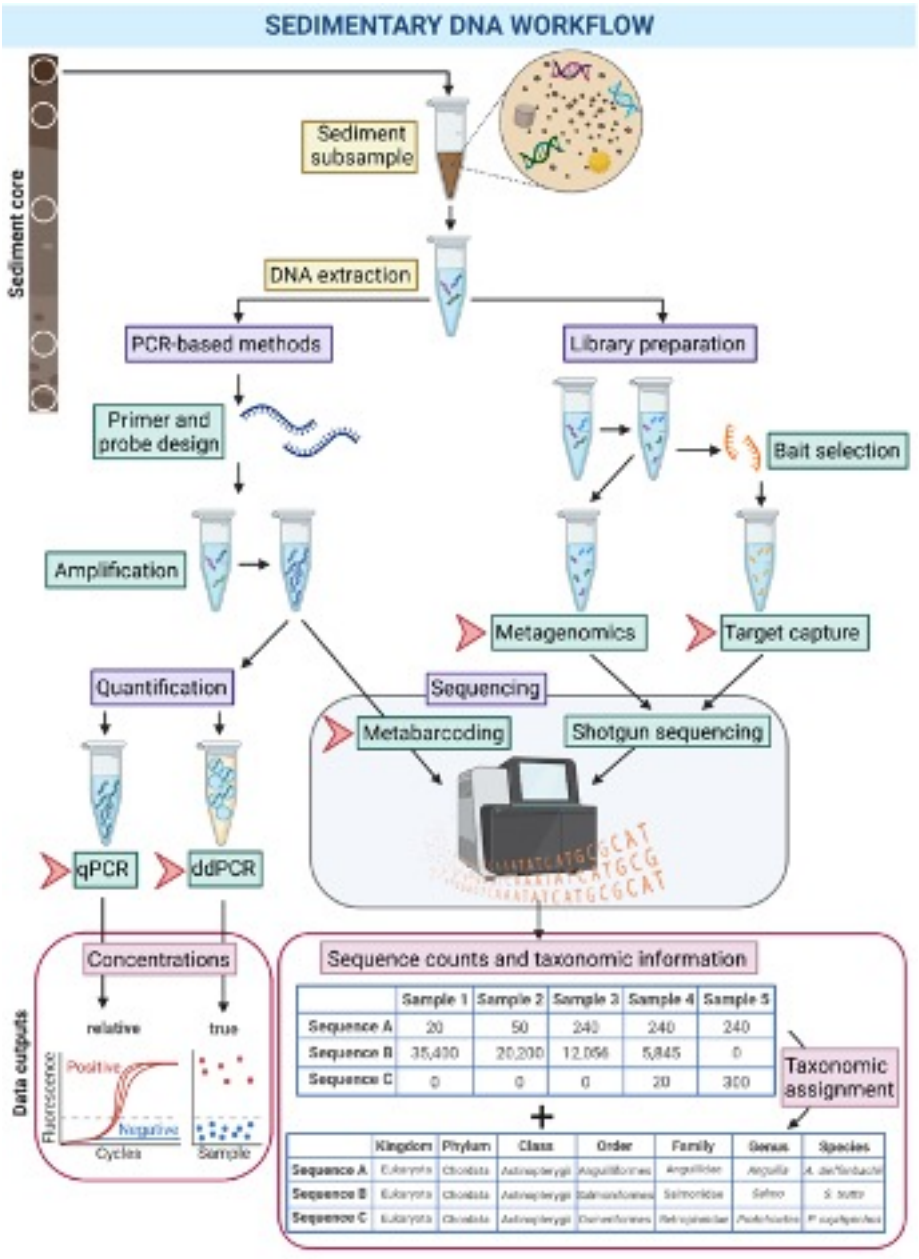


Figure 5.6: Sedimentary DNA workflow suggested by (Picard et al. 2024)

5.4 Precautions and considerations in ancient eDNA research

While the application of ancient eDNA may appear promising, it is, in fact, quite challenging. There are numerous precautions and considerations that must be taken into account when conducting this type of research to accurately reconstruct past biological communities.

5.4.1 Contamination control in field and laboratory sampling

Contamination control measures begin at the sample collection stage (Rawlence et al. 2014). Sampling must follow rigorous protocols, which include the use of protective gear such as gloves, facemasks, or full-body suits, and the utilization of sterilised equipment (Figure **??**).

Figure 5.7: Sample collection following aDNA sterility protocols. Left: Field sampling, (Rawlence et al. 2014) Journal of Quaternary Science. Right: Laboratory sub-sampling: (Crump 2021)

5.4.2 Tracking contaminants in field sampling

In coring procedures, a visible chemical (e.g., perfluoromethyldecalin, PFMD) or DNA-based tracer can be applied to the sediment core to assess the extent to which potential external contamination infiltrates the extracted core (Figure **??**). The tracer can be applied to the surface of the core before sampling, or it can be applied during core retrieval in the field. After applying the tracer, the core surface is removed, and the sample is collected (Epp, Zimmermann, and Stoof-Leichsenring 2019). The presence of the tracer DNA on both the removed

surface and the collected samples is then verified using PCR or qPCR (Bang-Andreasen et al. 2017).

5.4.3 Taxonomic assignment and community reconstruction

One crucial factor when creating lists of species from DNA sequences found in sediment is having good reference databases. If these databases are not comprehensive and meticulously maintained, you may miss the identification of certain species in your samples. Some databases may even contain errors, like contaminants or wrongly identified sequences, which could lead to incorrect results (Cribdon et al. 2020).

Another essential consideration is the analysis process, known as the bioinformatic pipeline. This includes the specific software tools and steps used to turn your DNA sequences into a list of species and their quantities. Setting the right parameters and thresholds in this pipeline is crucial for accurate results. So, to generate reliable species lists from sediment DNA data, you need both a solid reference database and a well-designed analysis pipeline (Cribdon et al. 2020).

A good example, though from a modern metagenomic study, is the widely reported metagenomic analysis of the New York subway, which initially suggested that pathogens like *Yersinia pestis* and *Bacillus anthracis* were part of the "normal subway microbiome" (Afshinnekoo et al. 2015). However, subsequent reanalysis using more appropriate methods did not detect these pathogens (Ackelsberg et al. 2015). Commonly used software pipelines can produce results that lack prima facie validity (e.g., reporting widespread distribution of notorious endemic species) but appropriate use of inclusion and exclusion sets can avoid this issue.

5.4.4 Spatial heterogeneity of sedaDNA

Spatial distribution patterns of eDNA can be strongly heterogenous within sediments even at a microscale. For instance, eDNA from animals may originate not only from microscopic bone and teeth fragments but also from extracellular DNA present in bodily fluids or as a result of tissue decomposition. This distribution can be uneven across different sections of a sedimentary profile.

A study conducted by Massillani and colleagues in 2021 (Figure **??**) illustrated the varying taxonomic compositions of mammalian mitochondrial DNA on a microscale within resin-impregnated archaeological sediment blocks collected from prehistoric sites in Europe, Asia, Africa, and North America (Massilani et al. (2022b)). Their findings highlighted that DNA may be concentrated in specific sources such as bone and coprolites within the sediments.

Researchers should consider this factor when designing studies based on sedaDNA, particularly during the sample collection phase. Neglecting this heterogeneity could potentially lead to false negative results, especially for rare

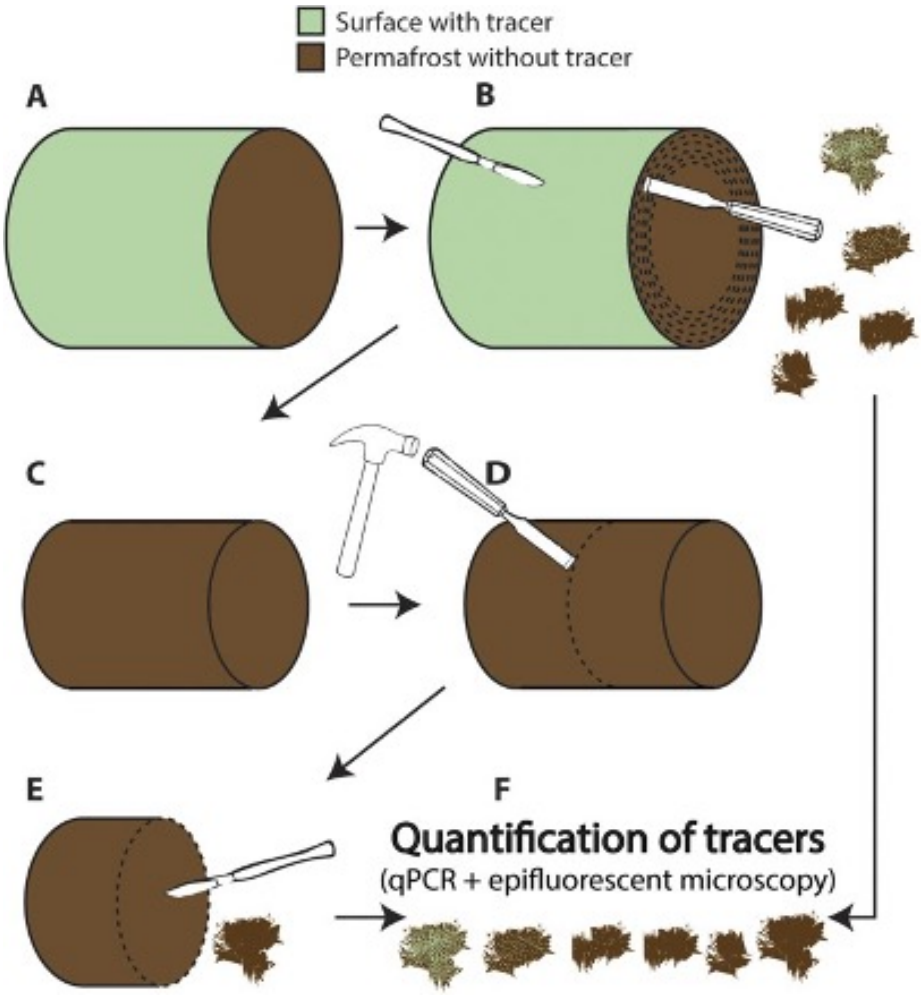


Figure 5.8: Example of a procedure for tracer quantification. (A) Outer sides of the permafrost core were covered with tracers. (B) The outer layers (dashed lines) were removed one by one using sterile tools and extreme caution to avoid cross contamination between layers. (C) Resulting inner core, smaller in diameter and free of tracers and potential microbial contamination. (D) The inner core was transversely split in two using sterile chisel. (E) From the newly exposed surface, soil was collected from the center of the inner core for quantification of tracer elements. (F) Quantification of tracers in all the collected layers and from the center of the permafrost core (quantitative PCR: qPCR). (Bang-Andreasen et al. 2017)

species. Modifying the number of sampling locations and increasing natural replicates per location in the study design could enhance the reliability of species detection.

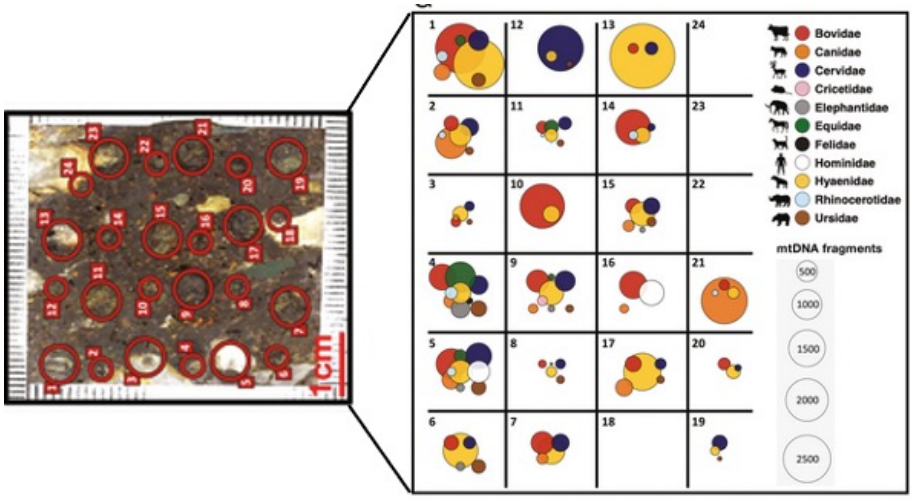


Figure 5.9: Microstratigraphic distribution of mitochondrial DNA of ancient mammalian taxa in impregnated sediment. (Massilani et al. 2022b)

5.4.5 Vertical migration of DNA

Another challenge associated with the sedaDNA approach, especially in non-frozen environments such as cave sediment records, is the potential risk of DNA moving vertically between different layers (DNA leaching) and contaminate lower (older) layers.

To investigate this issue, Haile and colleagues (2007) extracted aDNA from sediments up to 3300 years old at two cave sites on New Zealand's North Island (Figure **??**). These sites offered a valuable opportunity to study vertical DNA migration, as the sediment layers spanned both pre- and post-European periods. This allowed them to test for the presence of non-native species, such as sheep, in sediments deposited before European settlement, providing an indicator of DNA movement within the strata. Their findings revealed that sheep DNA was found alongside moa DNA, an indigenous species, in pre-European strata, suggesting that genetic material had migrated downward through the sediment (Figure **??**). However, the amount of sheep DNA reduced as sediment age increased. This research highlighted the potential of sedimentary DNA as a valuable resource for understanding past environments but also suggested that considerable caution should be taken when interpreting DNA profiles from sediment records.

Downward infiltration of water can transport DNA through the porous, granular sediments of caves and potentially soil profiles (Haile et al. 2007; Jenkins et

al. 2012). However, other studies indicate that this phenomenon is less likely in lake sediments or permanently frozen sediments. Once lake sediments are compacted, vertical pore water movement is minimal, leading to the immobilization of multivalent metals and organic compounds in the sediment matrix (Anderson-Carpenter et al. 2011).

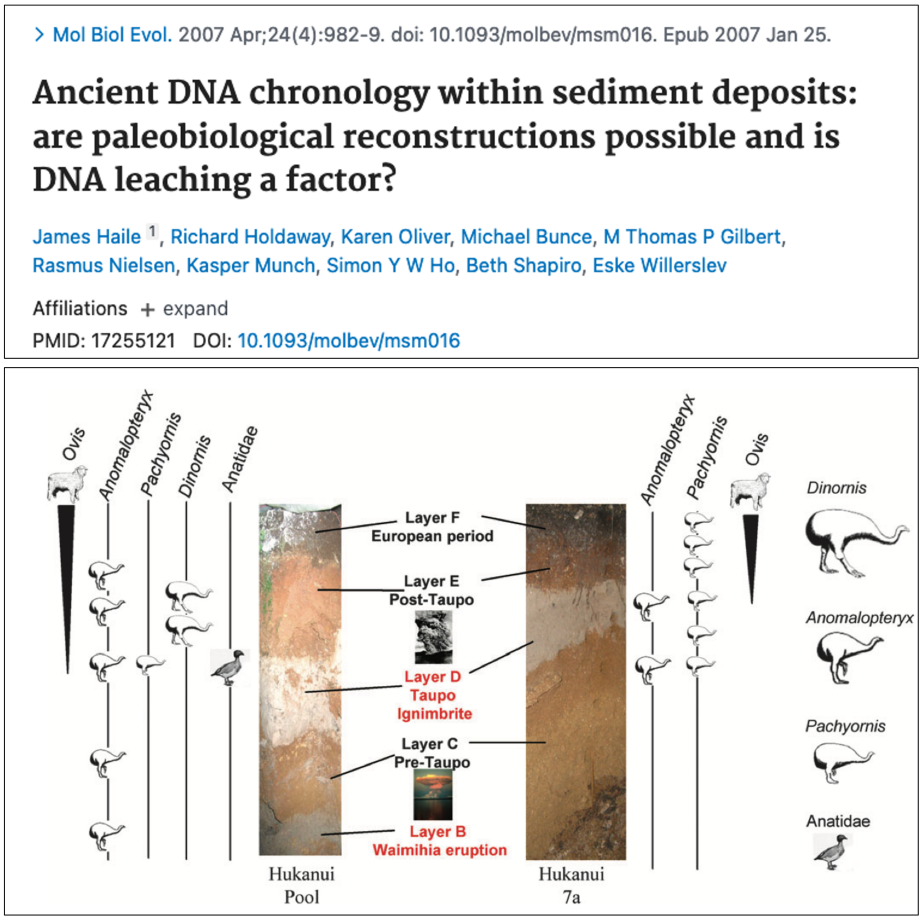


Figure 5.10: Stratigraphic sections in two cave sites in the North Island of New Zealand. The amount of moa and sheep DNA is shown in the figure across the sediment records. (Haile et al. 2007)

5.4.6 Potential applications

The use of ancient environmental DNA has enabled the reconstruction of palaeoenvironments, facilitating the detection of hominins and the study of biotic turnovers in ecosystems. This has also provided insights into ecosystem restoration targets.

5.4.7 Palaeoenvironmental reconstruction

Classical palaeoecology has explored the responses of biota to environmental change, but it has not fully captured the diversity of organisms, their co-occurrences, and interactions. SedaDNA offers an opportunity to achieve this, as *theoretically* allows the reconstruction of an ecosystem diversity across all domains of life.

A good example of this type of this application is the study published by (Kjær et al. 2022b). The Late Pliocene and Early Pleistocene epochs (3.6 to 0.8 million years ago) exhibited climates resembling future warming predictions. However, knowledge of the biological communities in the Arctic during this period remains limited due to the scarcity of fossils. In this study, the authors recovered ancient DNA from permafrost sediment samples in North Greenland, unveiling a diverse ecosystem that included an open boreal forest, various vegetation types, and a wide range of animals such as mastodons, reindeer, rodents, and geese. The presence of certain marine species (e.g., horseshow crab and green algae) suggests a warmer climate compared to the present. This use of ancient eDNA and traditional palaeoecological proxies (e.g., pollen) opened up new avenues for tracing the ecology and evolution of biological communities dating back *two million years* (a new record in the aDNA field!). The long-term preservation of this ancient eDNA is most likely attributed to its binding to mineral surfaces.

5.5 Insights provided by microbial shifts over time

Understanding how microbial diversity and function change over time and space is crucial for comprehending how ecosystems respond to global changes. Microorganisms, particularly prokaryotes, play essential roles in ecosystems by facilitating biogeochemical cycles and they are also the fastest responders to environmental changes. However, prokaryotes have been overlooked in long-term ancient eDNA studies due to the difficulty in determining whether their DNA signals originate from sediment-living organisms or from dead microorganisms deposited over time (Vuillemin et al. 2023). Recent advances in molecular genetic methods have enabled the identification of past microbial communities, including prokaryotes, and the tracing of their evolution and adaptation to environmental changes (Eric Capo et al. 2021). This information is crucial for predicting future ecosystem shifts. The following studies showcase the type of information that can be obtained from microbial ancient eDNA and their interpretations.

5.5.1 Phytoplankton changes in response to climate in Polar ecosystems

Polar ecosystems are highly vulnerable to the ongoing climate change. The melting ice sheets and changes in oceanography in marine ecosystems are impacting all levels of the food web, especially primary producers. Changes in sea temperature and atmospheric CO2 levels are linked to shifts in the composition and size structure of phytoplankton communities, making them valuable palaeoindicators of climate change and ecosystem responses. However, some diatoms with heavily silicified structures preserve better than others, potentially skewing fossil analyses. This highlights the importance of sedaDNA techniques in complementing microfossil and other palaeo-proxy analyses for detailed investigations of periods undergoing environmental change, such as glacial-to-interglacial transitions.

Different studies have utilised sedimentary DNA from the Arctic and Antarctic environments to describe changes in phytoplankton communities, aiming to better understand the past and present responses of marine ecosystems to climate change. For example, a study by Buchwald and colleagues focused on the subarctic western Bering Sea, analysing metagenomic shotgun and diatom *rbcL* amplicon sequencing data over glacial–interglacial cycles, including the Eemian interglacial period (Buchwald et al. 2024). They identified distinct plankton communities during these periods: the Holocene was dominated by picosized cyanobacteria and bacteria-feeding protists, the Eemian featured eukaryotic picosized chlorophytes and Triparmaceae, and the glacial period had microsized phototrophic protists, including sea ice-associated diatoms and diatom-feeding crustaceous zooplankton. This long-term record revealed a decrease in phytoplankton cell size with rising temperatures, similar to current changes in the warming Bering Sea. These shifts suggest that future warming may increase productivity but reduce carbon export, weakening the Bering Sea's role as a carbon sink.

On the other hand, Armbrecht and colleagues examined a sedimentary DNA record spanning ~1 million years from the Scotia Sea in Antarctica to understand changes in phytoplankton communities over time (L. Armbrecht et al. 2022) . They found records of diatom and chlorophyte sedaDNA dating back ~540 thousand years, reconstructing the oldest marine eukaryote sedaDNA record to date. When analysing the results, the authors concluded that warm phases were associated with high diatom abundance and a significant shift from sea-ice to open-ocean species around 14.5 thousand years ago, following Meltwater Pulse 1A.

Together, these studies illustrate the profound impact of climate change on marine ecosystems in both the Arctic and Antarctic regions and demonstrate the potential of sedaDNA tools to study long-term marine ecosystem shifts and palaeo-productivity across glacial-interglacial cycles.

5.5.2 Environmental microbiome associated with vegetational changes

In this study, researchers used shotgun metagenomics to analyse ancient eDNA from lake sediments in northern Siberia, aiming to understand ecosystem changes over the last 6700 years. They combined taxonomic and functional gene analysis to gain insights into how the genetic records of biological communities preserved in the sediment evolved over time. Additionally, they described changes in eco-physiological adaptations and ecosystem functions of microbial and vegetational communities in response to past climate variations.

The researchers found that 6700 years ago, the area was an open boreal forest, which gradually transitioned into tundra. The changes in the plant community reconstructed using sedimentary DNA matched perfectly with those described in previous studies using other palaeoecological proxies such as pollen, DNA metabarcoding, and hybridization capture (Schulte et al. 2021), further validating their observations. These shifts in palaeovegetation significantly impacted the environmental microbiome, with clear implications for ecosystem functioning. For example, there was an enrichment of certain bacteria and archaea that thrive in tundra conditions, particularly bacterial and archaeal ammonia oxidisers like Nitrospira, Nitrosopumilus, and Ca. Nitrosocosmicus.

This study demonstrated that microbial taxa, functions, and eco-physiological traits are valuable indicators of environmental conditions, highlighting their use as proxies in palaeoecosystem reconstruction.

5.6 Summary

1. Environmental DNA (eDNA) refers to genetic material shed by organisms into their surroundings, including water, soil, and sediments. It enables species identification without direct organism sampling, offering valuable insights into both current and historical ecosystems.

2. Environmental samples are inherently complex, containing a mix of genetic material from multiple species. This complexity is further amplified in ancient eDNA, where modern and degraded DNA are combined, making sample recovery and analysis more challenging.

3. The preservation of eDNA depends largely on environmental conditions. Cold, anoxic environments and clay-rich sediments help protect DNA for millennia. Taphonomic processes play a key role in the reliability of the DNA records.

4. Analysing ancient eDNA requires stringent laboratory protocols, with contamination control being critical during both field sampling and lab work to ensure the accuracy and integrity of the research.

5. Ancient eDNA provides a powerful tool for reconstructing past environ-

ments and tracking long-term ecological changes, such as climate-driven shifts in microbial, plant, and animal communities. This information is crucial for understanding ecological responses and guiding conservation efforts.

5.7 References

Part II

Useful Skills

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

Introduction to the Command Line (Bare Bones Bash)

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

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

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

Introduction to R

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

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

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

Introduction to Python

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

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

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

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

Introduction to Git and GitHub

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

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

Chapter 6

Introduction to the Command Line

Self guided: chapter environment setup

For this chapter's exercises, if not already performed, you will need to download the chapter's dataset, decompress the archive, and create and activate the conda environment.
Do this, use `wget` or right click and save to download this Zenodo archive: 10.5281/zenodo.13759270, and unpack.

```
tar xvf bare-bones-bash.tar.gz
cd bare-bones-bash/
```

You can then create the subsequently activate environment with

```
conda env create -f bare-bones-bash.yml
conda activate bare-bones-bash
```

6.1 Preamble

This chapter is a condensed version of both 'Basic' and 'Boosted' Bare Bones Bash courses that can be found in the Bare Bones Bash website (https://barebonesbash.github.io/). The material is displayed here under a CC-BY-SA 4.0 license.

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

Figure 6.1: Logo of BareBonesBash, a computer monitor with a command prompt dollar symbol, BBB and a bone on the screen.

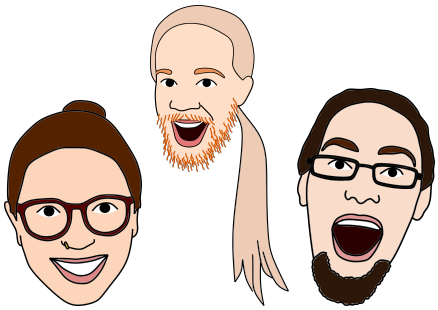


Figure 6.2: Cartoony pictures of the three authors of the BareBonesBash course. Designed by Zandra Fagernäs.

TL;DR: uncollapse me!

Below is a quick reference guide to the commands discussed in this tutorial (Table **??**). 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.

Table 6.1: Table with a quick summary of all the commands in this tutorial, some common arguments for them, and a short example command

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

6.2 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).

In this tutorial, you will learn:

- 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

6.3 The 5 commandments of Bare Bones Bash

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

Table 6.2: The five commandments of BareBonesBash

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 StackOverflow sooner or later.

6.4 What is a terminal?

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

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

`bash` (**b**ourne **a**gain **sh**ell) is one of the most popular languages used in the terminal.

6.5 Understanding the command prompt

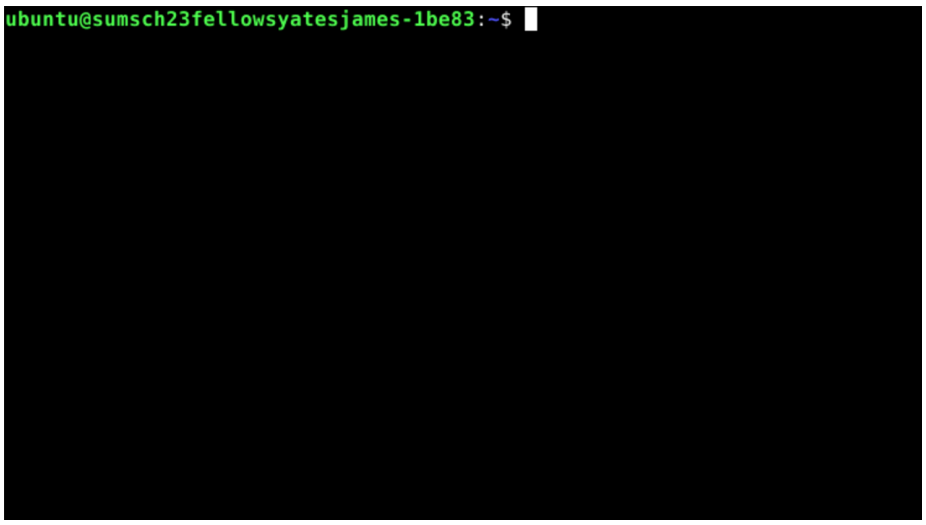


Figure 6.3: An example command prompt

After opening the terminal what we will normally see is a blank screen with a 'command prompt', like the one shown above. This typically consists of our 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 we know who and where we are. When copying a command we 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).

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

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

If we want to know what the shorthand means, (here comes our first command!) we can type in `pwd`, which stands for "**p**rint **w**orking **d**irectory".

Our "working directory" is whichever directory we are currently in.

```
pwd
```

```
/home/<YOUR_USERNAME>
```

> **Warning**
>
> In programming *documentation*, it is very common to use `<ALL_CAPS>` notation to indicate something that doesn't exist, or will vary depending on the user.
> Whenever you see such a notation (i.e., triangular brackets and all caps), you must *always* replace that whole section (including the <>) with whatever is on your system! You should never copy and paste this blindly!

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 our particular working directory.

> **Note**
>
> If you're not in your **home** directory (i.e., the output isn't exactly the same as the output above), please run the following command.
>
> ```
> cd $HOME
> ```
>
> You'll learn about this commands in a bit!

> **Question**
>
> Given the following command prompt: bare@bones_bash:~$, what does "bare" mean?

> **Answer**
>
> That's right, "bare" is the username that is log into the machine.

6.6 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 our specific home directory. That is the directory path we see in the output of the `pwd` command we just ran.
- Alternatively a **relative** path always begins from our working directory (i.e. our 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').

6.6.1 A real life analogy for paths

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

We see a friendly-looking metalhead (Figure **??**), and decide to ask them for directions!

Figure 6.4: A friendly-looking metalhead (actually James).

I'm Happy to help, but I only give directions in **absolute paths!**
From Leipzig Main Station, you should take Querstraße southward.
Continue straight and take Nürnberger 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 us to MPI-EVA, but we can only apply these directions **if** we start from Leipzig Main Station!
Examples of absolute paths:
`/home/<PATH>/<TO>/`
`/Leipzig_Main_Station/Querstraße/Nürnberger_Str/Str_18_Oktober/Deutscher_Platz/MPI-EVA`

Not sure how to get back to Leipzig Main Station to follow those directions, we decide to ask someone else for directions…

Lucky for us, a friendly looking local is passing by (Figure **??**)!

Figure 6.5: A friendly-looking local (actually Aida).

> 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 we happen to start at the position we were in when we first got the directions!
Examples of relative paths:
`./<PATH>/<TO>/my_file.txt`
`../Str_18_Oktober/Deutscher_Platz/MPI-EVA`

Question

What would be the relative path to the file "bones.txt" if you are in the folder "test" given the following absolute path "/home/sweet/home/lets/test/your/knowledge/to/the/bones.txt"

> **Answer**
>
> The relative path from the "test" folder to the "bones.txt" is: "./your/knowledge/to/the/bones.txt"

6.7 Basic commands

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

- **l**i**s**t directory contents

```
ls
```

Output should look like the following.

```
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.

- **m**a**k**e a **dir**ectory

```
##  this will be our working directory for the rest of this chapter! Do not move
mkdir barebonesbash
```

- **m**o**v**e (or rename) files and directories

```
mv barebonesbash BareBonesBash
```

- **c**hange **d**irectories

```
cd BareBonesBash
```

- Download (**w**ww **get**) a remote file to our computer

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

- **c**o**p**y a file or directory to a new location

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

- re**m**ove (delete) files

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

- Con**cat**enate 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
```

> **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
```

- **w**ord **c**ount the number of **l**ines (-l) in a file

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

```
15 Boosted-BBB-meta.tsv
```

> **Question**
>
> Which command can you use to print the last 6 lines of the file "Boosted-BBB-meta.tsv"

> **Answer**
>
> For that you will need to use to use tail and modify the parameter given to the *-n* flag like the following.
>
> ```
> tail -n 6 Boosted-BBB-meta.tsv
> ```

6.8 Datastreams, piping, and redirects

Each of the commands we 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:

6.8.1 Datastreams

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

- `stdin` : the **st**an**d**ard **in**put
- `stdout`: the **st**an**d**ard **out**put
- `stderr`: the **st**an**d**ard **err**or

Figure 6.6: 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 our terminal to let us know when something is not going according to plan (Figure **??**)!

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

Figure 6.7: 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.

6.8.2 Piping

A "pipe" (`|`) is a really useful feature of `bash` that lets us 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 **??**)! The `stderr` is still printed on our screen, so we can always know when things fail.

Example

```
head -n 10 Boosted-BBB-meta.tsv | tail -n 1
```

```
netsukeJapan    C       Artwork
```

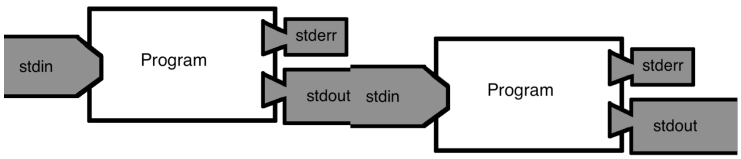


Figure 6.8: Diagram showing stdin going into the program, and two output streams from the program: stderr and stdout, with the stdout becoming the stdin of a second program.

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.

Question

How can we combine tail and head to print the 10th line starting from the end of the file "Boosted-BBB-meta.tsv"?

Answer

You will use the following command employing a | (pipe):

```
tail -n 10 Boosted-BBB-meta.tsv | head -n 1
```

6.8.3 Redirects

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

This way we 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 the program name!
- `stdout` can be redirected with `>`
  - An arrow pointing AWAY the program name!
- `stderr` can be redirected with `2>`
  - 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 our 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 6.9: Here we 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.

6.9 Help text

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

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

  ```
  whatis cat
  ```

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

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

  ```
  man cat
  ```

6.10 Variables

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

We can assign variables (tell the computer what do we 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 we open our terminal or log into a server, they ensure the system works as intended and should not be change unless we 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, we can store in a variable anything we 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.
```

> **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.
```

We 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 we want, we 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 we have seen how the same variable has been overwritten, this means that we can only access the last content that we 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.

6.11 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.

Question

What is the correct notation to assign a value to a variable in bash?

A) `$arr=2`

B) `arr$=2`

C) `arr=2`

Answer

The option C is the correct one. To assign a variable we need to put the chosen name followed by `=` and finally the value (number, string) that we want to be stored. When we want to call the variable we need to add `$` in from of the variable name.

6.12 Find

We can also ask our computer where we have put our files, in case we forgot. To do this we can use `find`! The `find` command has the following syntax.

```
##  Don't run! Fake example
find /<PATH>/<TO>/ -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:
    - **f**ile
    - **d**irectory
- **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 we have learnt about `find`.

For that we will download a messy folder from a collaborator, remember to check we 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 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 our collaborator they are marked with `JPG`. We first try to check the contents of the directory to find them quickly.

```
ls Boosted-BBB
```

```
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 we retrieve all the files? Thanks to our wonderful teachers we have learnt how to use find and can simply run the following.

```
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/weimanarer.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 https://barebonesbash.github.io/.

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

Question

How should you write a `find` comand to search for all files ending with .TXT in the folder Boosted-BBB?

Answer

The correct find command will be the following

```
find Boosted-BBB -type f -name '*.TXT'
```

As mentioned the first part is the folder `Boosted-BBB` followed by the type which is file and finally the name of what we are looking for, in our case all the files ending with .TXT

6.13 For loops

Until now we have seen how to run single commands on a file. But, what about when we 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.

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.

Question

How many times will the following for loop run the command `echo`?

```
for i in Greece Spain Britain Italy Slovenia; Do
  echo "I would love to go to $i"
done
```

Answer

The for loop will run the `echo` command 5 times, once for every element on the list `Greece Spain Britain Italy Slovenia`.

6.14 How to Google like a pro

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

Here are some quick tips to get you started:

- ALWAYS include the name of the language in your query
  - **BAD:** "How to cat" (Figure **??**)
  - **GOOD:** "How to cat bash" (Figure **??**)
- 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

All the cool hackers say:

- `string` and not `text`
- `float` and not `decimal`

Note: some of these terms can be language specific.

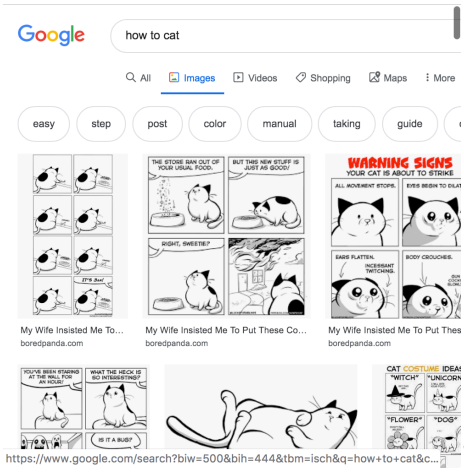


Figure 6.10: How to cat the wrong way (i.e., don't include the language in our Google search and get loads of cat pictures)

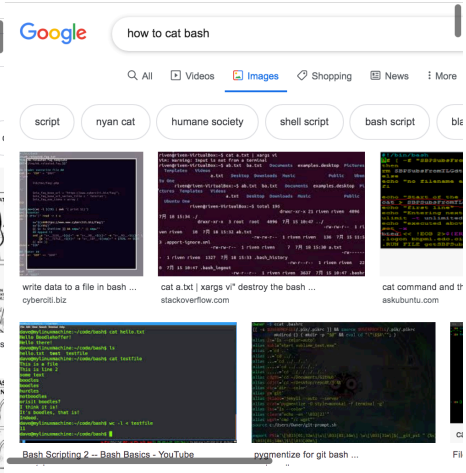


Figure 6.11: How to cat the BASH way (i.e., include the the language in our Google search and we get lots of terminal pictures! Much better, right? ...right?)

6.15 (Optional) clean-up

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

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

```
cd ~     ## We shouldn't delete a directory while we are still in it. (It is possible t
rm -r /<PATH>/<TO>/BareBonesBash*
```

We can also get out of the `conda` environment with

```
conda deactivate
```

To delete the conda environment

```
conda remove --name bare-bones-bash --all -y
```

6.16 Summary

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 (https://barebonesbash.github.io/).

Chapter 7

Introduction to R and the Tidyverse

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

> **Self guided: chapter environment setup**
>
> For this chapter's exercises, if not already performed, you will need to download the chapter's dataset, decompress the archive, and create and activate the conda environment.
> To do this, use `wget` or right click and save to download this Zenodo archive: 10.5281/zenodo.13758879, and unpack
>
> ```
> tar xvf r-tidyverse.tar.gz
> cd r-tidyverse/
> ```
>
> You can then create and subsequently activate the conda environment with
>
> ```
> conda env create -f r-tidyverse.yml
> conda activate r-tidyverse
> ```

README if you already have Rstudio installed and don't need conda

Open Rstudio, and check that you have the two following packages installed.

```
library(tidyverse)
library(palmerpenguins)
```

If one or neither are installed, please install as follows. Delete already-installed packages from the function as necessary.

```
install.packages(c("tidyverse", "palmerpenguins"))
```

README if you want to create the test datasets yourself from the palmerpenguins package

```
install.packages(c("tidyverse", "palmerpenguins"))

library(magrittr)
set.seed(5678)

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

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

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

7.1 R, RStudio, the tidyverse and penguins

This chapter introduces the statistical programming environment R and how to use it with the RStudio editor. It is structured as self-study material with examples and little exercises to be completed in one to four hours. A larger exercise at the end pulls the individual units together.

The didactic idea behind this tutorial is to get as fast as possible to tangible, useful output, namely data visualisation. So we will first learn about reading and plotting data, and only later go to some common operations like conditional queries, data structure transformation and joins. We will focus exclusively on tabular data and how to handle it with the packages in the tidyverse framework. The example data used here is an ecological dataset about penguins.

So here is what you need to know for the beginning:

- R (R Core Team 2023) is a fully featured programming language, but it excels as an environment for (statistical) data analysis (https://www.r-project.org)
- RStudio (RStudio Team 2020) is an integrated development environment (IDE) for R (and other languages) (https://www.rstudio.com/products/rstudio)
- The tidyverse (Wickham et al. 2019) 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 `tidyr`. `forcats` will be briefly mentioned, but `purrr` and `stringr` are left out.
- The `palmerpenguins` package (Horst, Hill, and Gorman 2020) provides a neat example dataset to learn data exploration and visualisation in R (https://allisonhorst.github.io/palmerpenguins)

7.2 Loading R Studio and preparing a project

Before we begin, we can load RStudio from within your `conda` environment, by running the following.

```
rstudio
```

> **Caution**
>
> It is *not* recommended to download and update Rstudio if asked to on loading while following this textbook or during the summer school. You do so at your own risk. We recommend pressing 'Remind later' or 'Ignore'.

The RStudio window should then open.

Open RStudio and create a new project by going to the top tool bar, and selecting `File -> New Project...`.

When asked, create the new directory in an 'Existing directory' and select the `r-tidyverse/` directory.

Once created, add new R script file so that you can copy the relevant code from this textbook into it to run them by pressing in the top tool bar `File -> New File -> New Rscript`.

7.3 Loading data into tibbles

7.3.1 Reading tabular 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 memory, both from our file system or the internet. It provides functions to read data in almost any (text) format.

```
readr::read_csv() # .csv files (comma-separated) -> see penguins.csv
readr::read_tsv() # .tsv files (tab-separated)
readr::read_delim() # tabular files with arbitrary separator
readr::read_fwf() # fixed width files (each column with a set number of tokens)
readr::read_lines() # files with any content per line for self-parsing
```

7.3.2 How does the interface of `read_csv` work?

We can learn more about any 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
    ...
)
```

When calling this - or any - function in R, we can either set the arguments explicitly by name or just by listing them in the correct order. That means `readr::read_csv(file = "path/to/file.csv")` and `readr::read_csv("path/to/file.csv")` are identical, because `file = ...` is the first argument of `readr::read_csv()`.

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

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

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

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

As a by-product of reading the file `readr` also 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 = "iccddcc" # this string encodes the desired types for each column
)
```

The `col_types` argument takes a string with a list of characters, where each character denotes one columns types. Possible types are `c` = character, `i` = integer, `d` = double, `l` = logical, etc. Remember that you can check `?readr::read_csv` for more.

`readr` finally returns an in-memory representation of the data in the file, 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 of it 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
```

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"

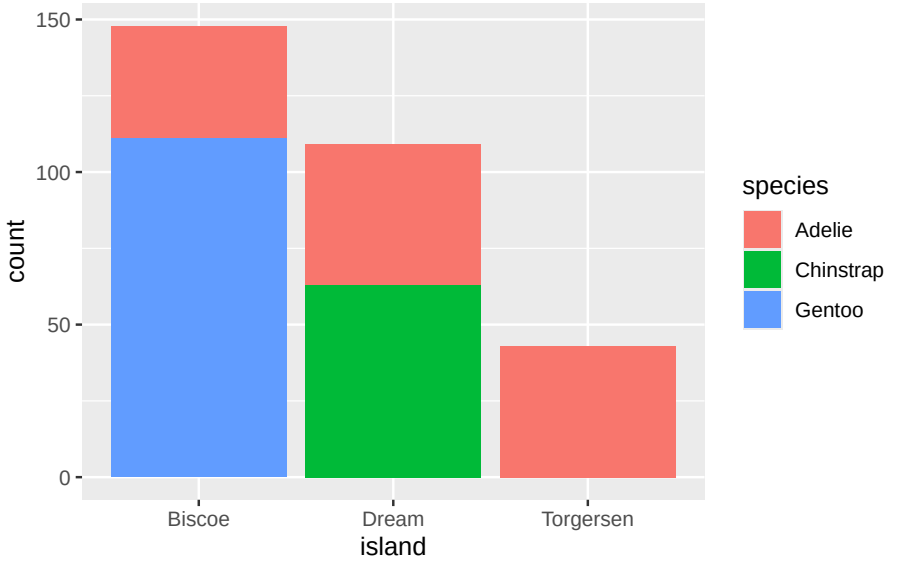
To understand and present data, we usually have to visualise it.

`ggplot2` is an R package that offers a slightly unusual, but powerful and logical interface for this task (Wickham 2016).

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 in the data argument
) + # The plot consists of individual functions linked with "+"
    geom_bar( # "geoms" define the plot layers we want to draw,
              # so in this case a bar-chart
        mapping = aes( # The aes() function maps variables to visual properties
            x = island,    # publication_year -> x-axis
            fill = species # community_type   -> fill colour
        )
    )
```



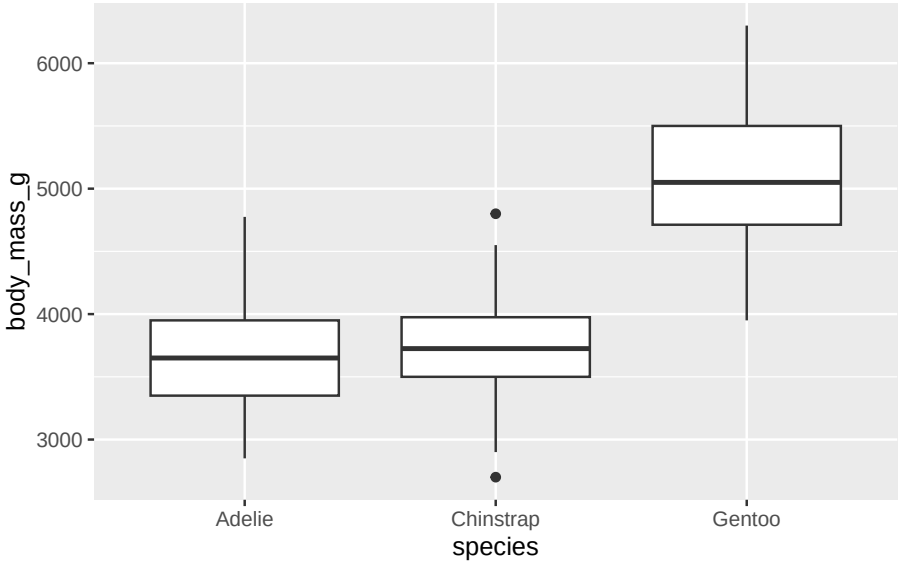
A `geom_*` combines data (here `peng`), a geometry (here vertical, stacked bars) and a statistical transformation (here counting the number of penguins per island and species). Each `geom` has different visual elements (e.g. an x- and a y-axis, shape and size of geometric elements, fill and border colour, …) to which we can *map* certain variables (columns) of our input dataset. The visual elements will then represent these variables in the plot. `ggplot2` features many `geoms`: A good overview is provided by this cheatsheet: https://rstudio.github.io/cheatsheets/html/data-visualization.html.

Beyond `geoms`, a ggplot2 plot can be further specified with (among others) `scales`, `facets` and `themes`.

7.4.2 `scales` control the exact behaviour of visual elements

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

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

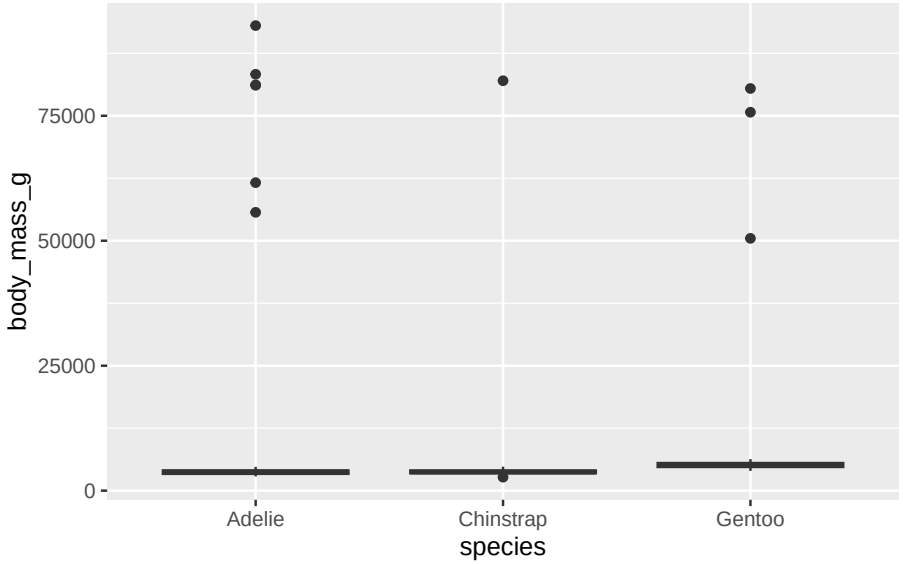


Let's assume we had some extreme outliers in this dataset. To simulate this, we replace some random weights with extreme values.

```
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".

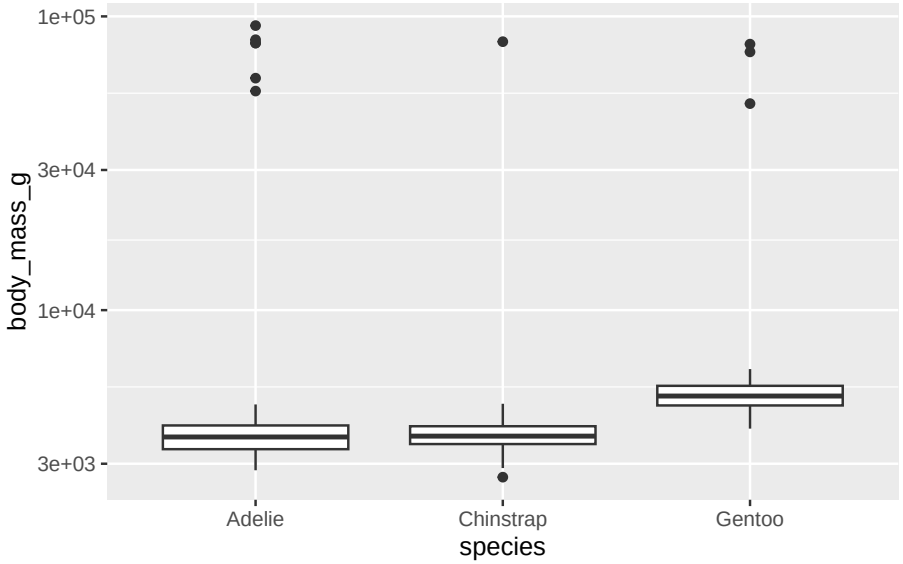
```
ggplot(peng_out) +
    geom_boxplot(aes(x = species, y = body_mass_g))
```



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 were valid observation we want to include in the plot.

To mitigate the visualisation 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
```

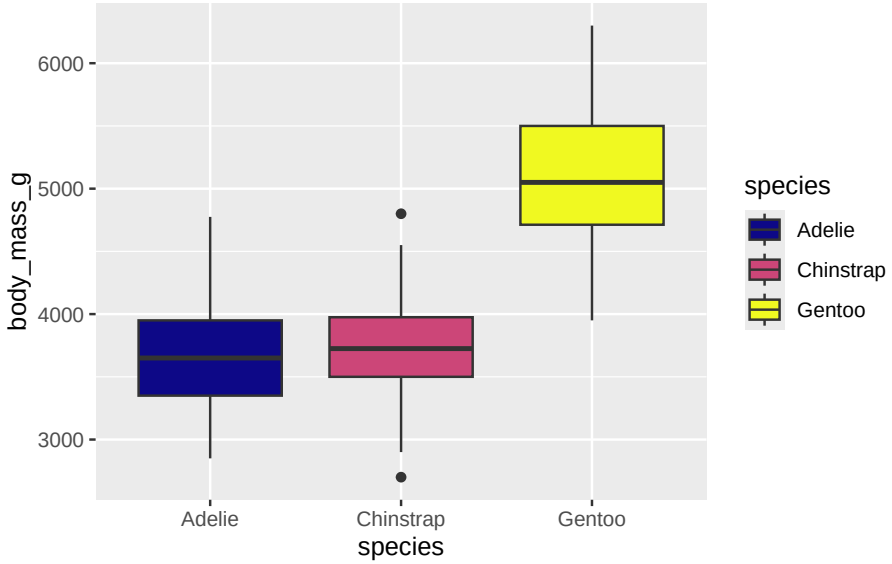


We will now go back to the normal dataset without the artificial outliers.

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")
```



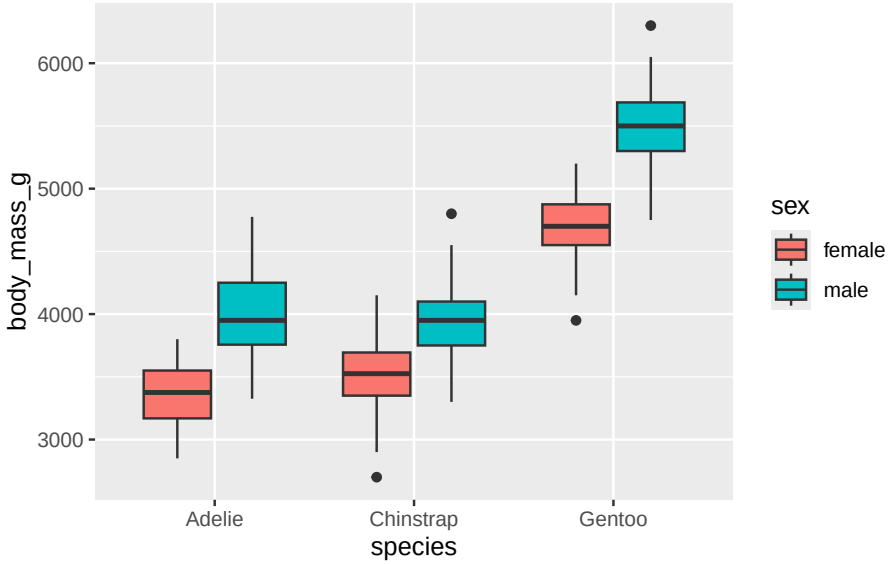
We use the `scale_*` function to select one of the visually appealing (and robust to colourblindness) viridis colour palettes (https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html).

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.

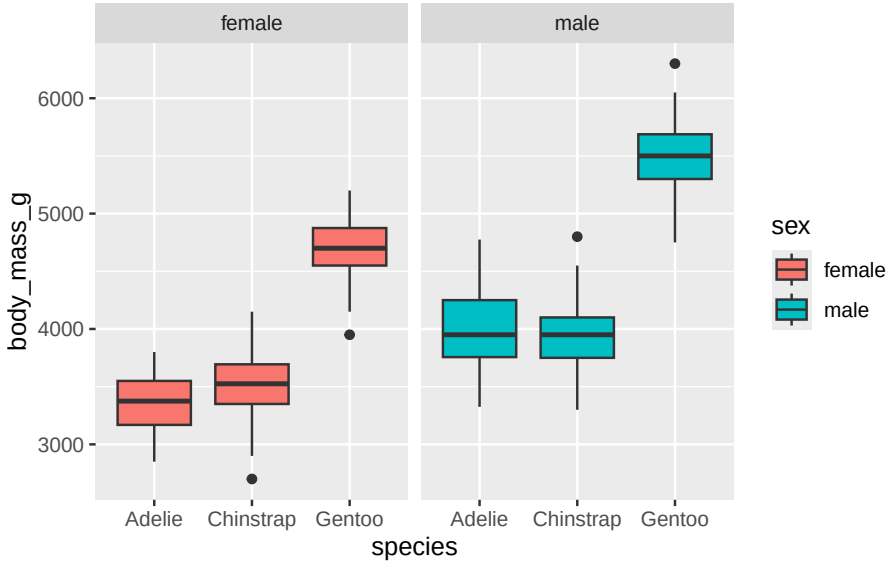
```
ggplot(peng) +
    geom_boxplot(aes(x = species, y = body_mass_g, fill = sex))
```



Note how mapping another variable to the fill colour automatically splits the dataset and how this is reflected in the number of boxplots per species.

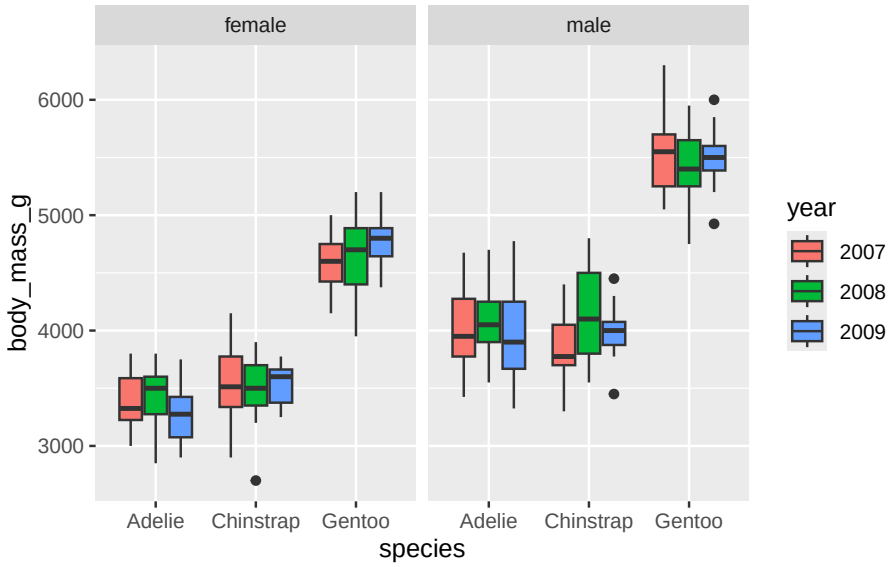
Another way to visualise more variables in one plot is to split the plot by categories into **facets**, so sub-plots per category. Here we split by sex, which is already mapped to the fill colour:

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



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)
```



This plot already visualises the relationship of four variables: species, body

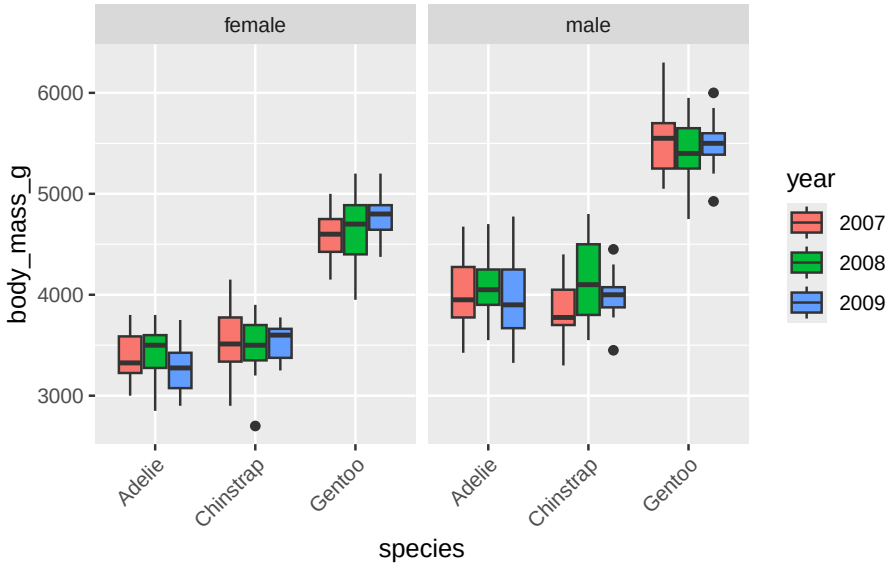
mass, sex and year of observation.

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 over-plotting.

```
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))
```



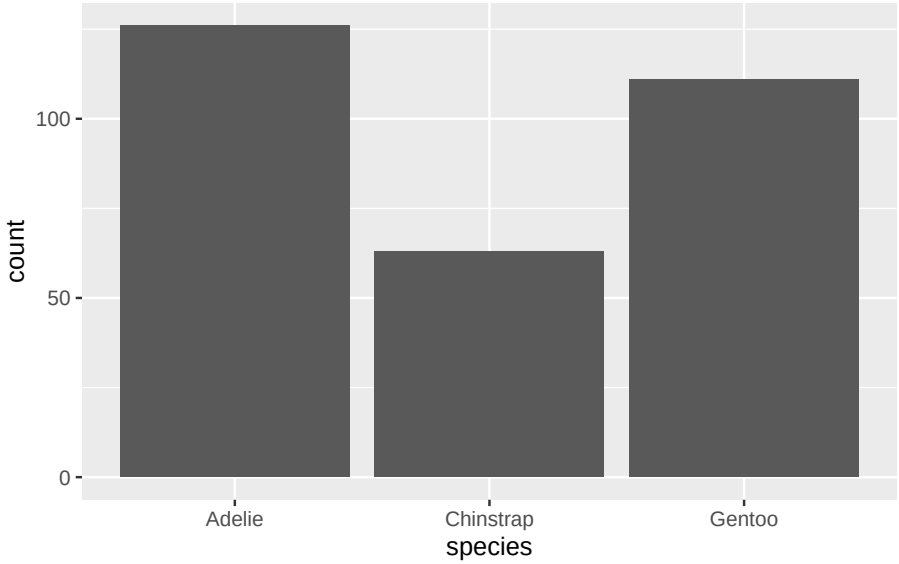
7.4.6 Ordering elements in a plot with `factors`

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 handling them.

Elements based on `character` columns are by default ordered alphabetically.

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

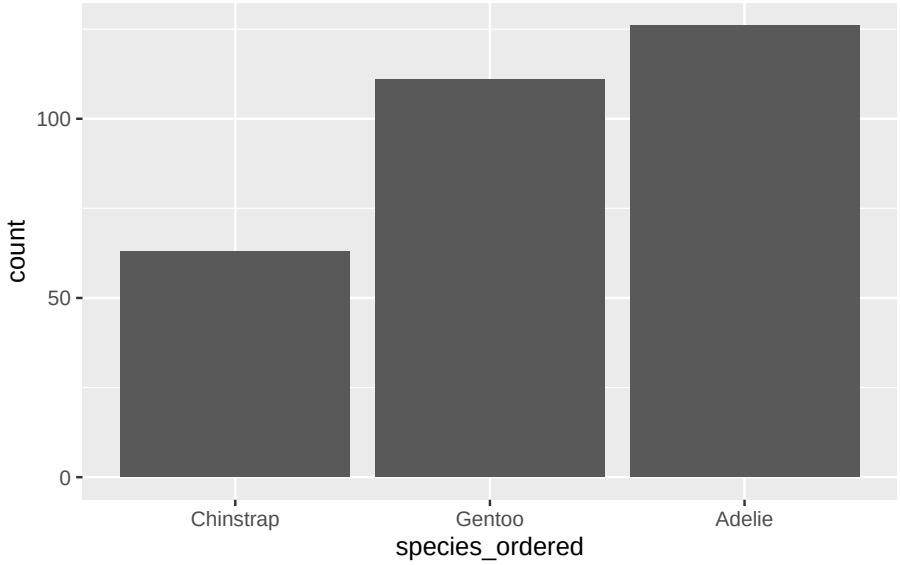


With `forcats::fct_reorder` we can transform an input vector to a `factor`, ordered by a summary statistic (even based on another vector).

```
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
```



7.4.7 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 always be accessed just by typing `mtcars` in the console.

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

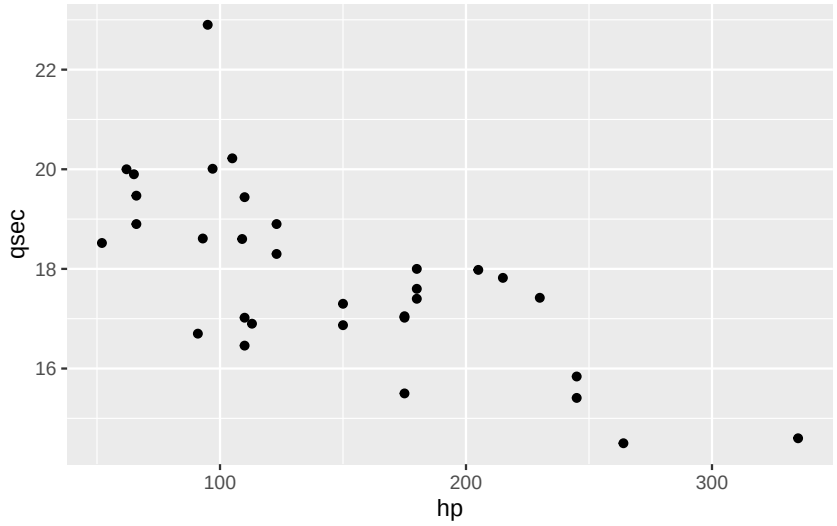
3. Integrate the *Number of cylinders* into your plot as an additional variable.

Possible solutions

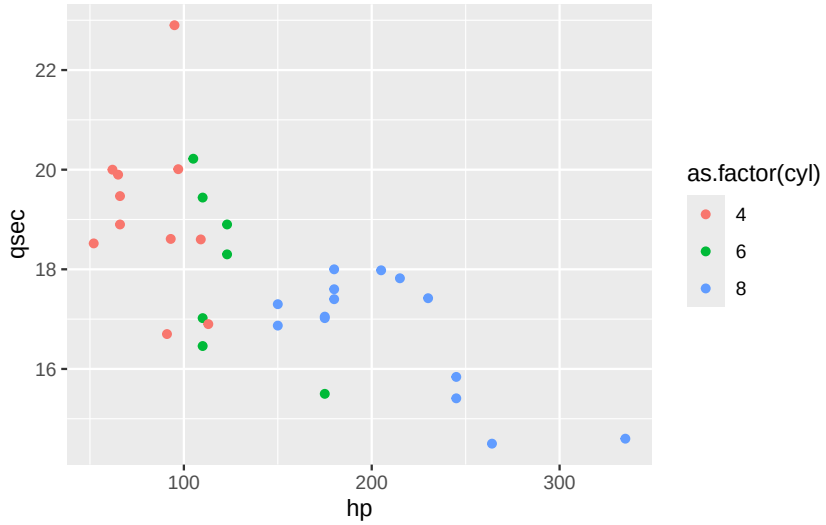
```
?mtcars
```

```
[, 1] mpg     Miles/(US) gallon
[, 2] cyl     Number of cylinders
[, 3] disp    Displacement (cu.in.)
[, 4] hp      Gross horsepower
[, 5] drat    Rear axle ratio
[, 6] wt      Weight (1000 lbs)
[, 7] qsec    1/4 mile time
[, 8] vs      Engine (0 = V-shaped, 1 = straight)
[, 9] am      Transmission (0 = automatic, 1 = manual)
[,10] gear    Number of forward gears
[,11] carb    Number of carburetors
```

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




```
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`

Among the most basic tabular data transformation operations is the conditional selection of columns and rows. The `dplyr` package includes powerful functions to subset data in tibbles.

`dplyr::select` allows to select columns:

```
dplyr::select(peng, id, flipper_length_mm) # select 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
```

```
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
```

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

```
# an alternative way to express OR with the match operator "%in%"
dplyr::filter(peng, year %in% c(2007, 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
```

```
# Adelie penguins heavier than 4kg
dplyr::filter(peng, species == "Adelie" & body_mass_g >= 4000)
```

```
# 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
```

Note how each function here takes `peng` as a first argument. This invites a more elegant syntax.

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
     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) of **%>%** as the first argument of the function appearing on the RHS (right-hand side) to enable sequences of function calls ("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 operator **%$%** is particularly useful to easily extract individual variables from a tibble.

```
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 a variety of basic operations in its **base** package. Many of them operate on **vectors**, so lists of values of one type. Individual columns are vectors.

```
# we extract a single variable as a vector of values
chinstraps_weights <- peng %>%
    dplyr::filter(species == "Chinstrap") %$%
    body_mass_g
chinstraps_weights
```

```
 [1] 3500 3900 3650 3525 3725 3950 3250 3750 4150 3700 3800 3775 3700 4050 4050
```

```
[16] 3300 3450 4400 3400 2900 3800 3300 4150 3400 3800 3700 4550 3200 4300 3350
[31] 4100 3600 3900 3850 4800 2700 4500 3950 3650 3550 3500 3675 4450 3400 4300
[46] 3675 3325 3950 3600 4050 3350 3450 3250 4050 3800 3525 3950 3650 3650 4000
[61] 3775 4100 3775
```

```
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
```

```
sd(chinstraps_weights) # standard deviation
```

```
[1] 391.1941
```

```
# quantiles for the given probabilities
quantile(chinstraps_weights, probs = c(0.25, 0.75))
```

```
 25%  75%
3500 3975
```

Many of these functions can ignore missing values (so `NA` values) with the option `na.rm = TRUE`.

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

These vector 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 %>%
    # group the tibble by the material column
    dplyr::group_by(species) %>%
    dplyr::summarise(
        # new col: min weight for each group
        min_weight = min(body_mass_g),
        # new col: median weight for each group
        median_weight = median(body_mass_g),
        # new col: max weight for each group
        max_weight = max(body_mass_g)
    )
```

```
# 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.

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

```
# 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
```

```
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 three 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>
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 (see `mpg`) cars for both *transmission types* (`am`).

Possible solutions

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

```
[1] 12
```

```
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
```

```
# make the care name an explicit column
mtcars2 <- tibble::rownames_to_column(mtcars, var = "car")

# Solution 1
mtcars2 %>%
    dplyr::group_by(am) %>%
    dplyr::arrange(mpg) %>%
    dplyr::slice_head(n = 1) %$%
    car
```

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

```
# Solution 1 only returns n = 1 result per group even if
# there are multiple cars with the same minimal mpg value.
# Solution 2 shows both, if this is desired.

# Solution 2
mtcars2 %>%
    dplyr::group_by(am) %>%
    dplyr::filter(mpg == min(mpg)) %$%
    car
```

```
[1] "Cadillac Fleetwood"  "Lincoln Continental" "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
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>   <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
```

7.6.2 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 as a modification of an existing column
        kg = body_mass_g / 1000
    )
```

```
# 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
```

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

```
peng %>%
    dplyr::transmute(
        # overwrite the id column with a modified version
        id = paste("Penguin Nr.", id), # paste() concatenates strings
        flipper_length_mm # select this column without modifying it
    )
```

```
# 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(
    # add a modified version of a column
    # note the . representing the LHS of the pipe
    flipper_length_cm = .$flipper_length_mm / 10,
    # add the columns after this particular other columns
    .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>
```

`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.

Here is a realistic sequence of operations that makes use of this feature:

```
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() %>%
    # mutate does not remove rows, unlike summarise, so we use select
    dplyr::select(id, species, sex, year, relation_to_mean) %>%
    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
```

7.6.3 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(
        # is weight below or above mean weight?
        test = body_mass_g >= 4200,
        yes  = "above mean",
        no   = "below mean"
    )
)
```

```
# 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
```

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

```
peng %>% dplyr::mutate(
    weight = dplyr::case_when(
        # the number of conditions is arbitrary
        body_mass_g >= 4200 ~ "above mean",
        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",
            "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
```

7.6.4 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 **??**). Both data

formats have useful applications and usually a given R function requires either, so we need to know how to convert between the two.

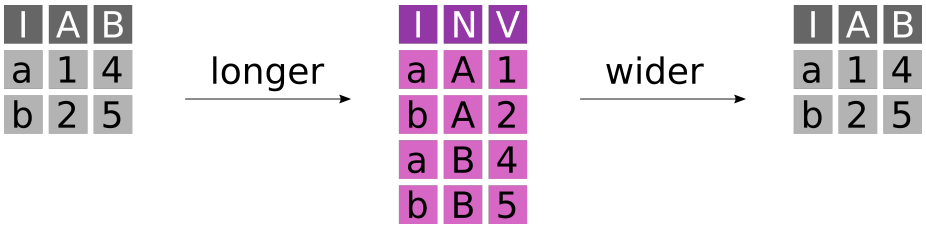


Figure 7.1: Graphical representation of converting a table from a wide to a long and back to a wide format.

- 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.

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
```

In this wide format information is spread over many columns. Based on what we learned previously we can not easily plot it with `ggplot2`. Although it is often more verbose and includes more duplication, in the tidyverse we generally prefer data in long, "tidy" format – well justified by Wickham (2014).

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

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

```
    values_to = "sales"
)
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
5 VW     2014    67
# i 3 more rows
```

Wide datasets are not always the wrong choice. They are well suitable for example for adjacency matrices to represent graphs, covariance matrices or other pairwise statistics. When the data gets big, then wide formats can be significantly more efficient (e.g. for spatial data).

To transform data from long to wide, we can use `tidyr::pivot_wider`

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

7.6.5 Exercise

1. Move the column `gear` to the first position of the `mtcars` dataset.
2. Make a new dataset `mtcars2` from `mtcars` with only the columns `gear` and `am_v`. `am_v` should be a new column 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`) in `mtcars2`. 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> <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::transmute(
        gear,
        am_v = dplyr::case_match(
            am,
            0 ~ "automatic",
            1 ~ "manual"
        )
    ) %>%
    tibble::as_tibble()
mtcars2
```

```
# A tibble: 32 x 2
   gear am_v
  <dbl> <chr>
1     4 manual
2     4 manual
3     4 manual
4     3 automatic
5     3 automatic
# i 27 more rows
```

```
mtcars2 %>%
    dplyr::group_by(am_v, gear) %>%
    # dplyr::tally() is identical to dplyr::summarise(n = dplyr::n())
    # -> it counts the number of entities in a group
    dplyr::tally() %>%
    tidyr::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. We can generally distinguish two kinds of joins:

1. Mutating joins add columns and rows of x and y:
    - **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.
2. Filtering joins remove observations from x based on their presence in y.
    - **Semi join**: Keep every observation in x that is in y.
    - **Anti join**: Keep every observation in x that is not in y.

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
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 **??**).

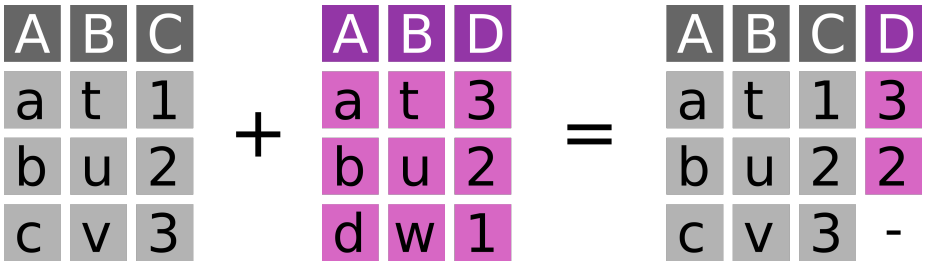


Figure 7.2: Graphical representation of a left join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to include the columns A B C D. As A and B have a one to one match of values, this remains the same in the joined table. The B column between the two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

```
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
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 **??**).

```
dplyr::right_join(
    x = peng, # 300 observations
    y = bills, # 300 observations
```

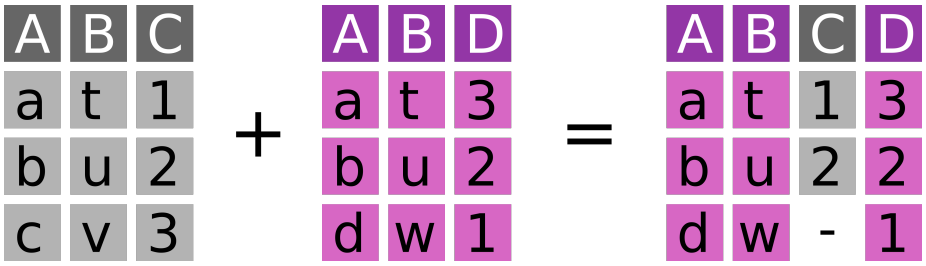


Figure 7.3: Graphical representation of a right join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have columns A B C D. As A and B have a one to one match of values, this remains the same in the joined table. The B column between the two have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

```
    by = "id"
) %>%
    # we arrange by id to highlight the missing observation in the peng dataset
    dplyr::arrange(id)
```

```
# 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     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 **??**).

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

```
# A tibble: 275 x 9
```

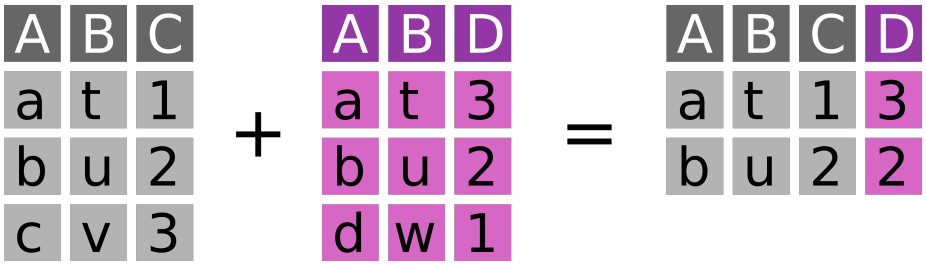


Figure 7.4: Graphical representation of an inner join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have columns A B C D. Only rows from both tables that have exact matches on columns A and B are retained. The third rows from both tables that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

```
     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
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 **??**).

```
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
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 320 more rows
```

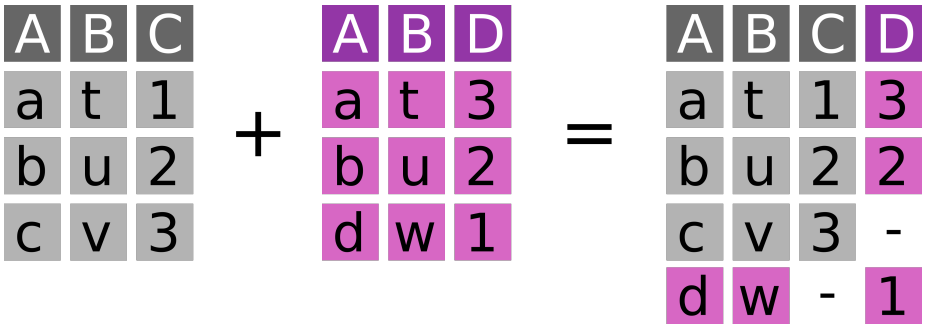


Figure 7.5: Graphical representation of a full join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have columns A B C D. All rows from both tables are retained, even though they do not share the same value in column B on both tables. The missing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

```
# 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 **??**).

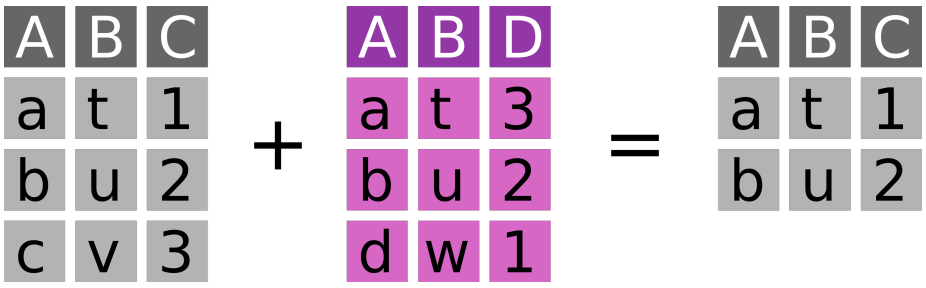


Figure 7.6: Graphical representation of a semi join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have only the columns A B C. Only columns A B and C are retained in the joined table. Row three of both tables are not included as the values in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

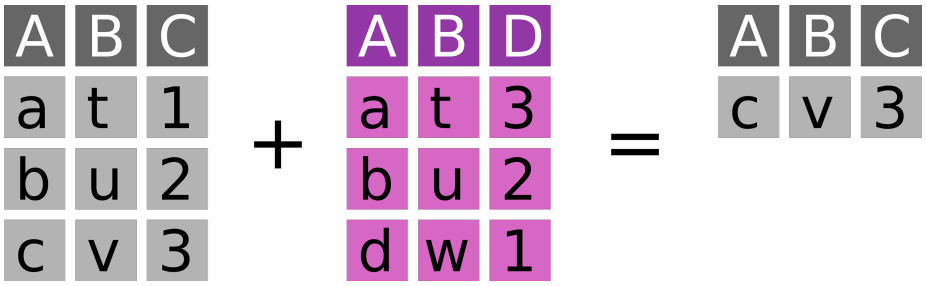


Figure 7.7: Graphical representation of an anti join operation. Two tables with a shared first column (A B C, and A B D respectively) are merged together to have only the columns A B C and only row three of the first table. Only row three is retained from the first table as this is the only row uniquely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

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> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <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> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <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
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 (Optional) Final exercise

In this final exercise we reiterate many of the concepts introduced above. We also leave penguins and cars behind and finally start working with a dataset relevant to the topic of this book: The environmental samples table of the AncientMetagenomeDir.

Here's the URL to the table for v24.06 of the AncientMetagenomeDir:

> "https://raw.githubusercontent.com/SPAAM-community/AncientMetagenomeDir/e29eb729e4b5d3
> environmental/samples/ancientmetagenome-environmental_samples.tsv"

To get going create a new R script where you load `magrittr` and `ggplot2` and create a variable for this URL:

```
library(magrittr)
library(ggplot2)
url_to_samples_table <- "https://raw.githubusercontent.com/SPAAM-community/AncientMetag
```

1: Load the samples table as a tibble in R, into a variable "samples". The `readr` package can read directly from URLs.

Solution

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

```
Rows: 702 Columns: 19
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr (13): project_name, publication_doi, site_name, geo_loc_name, study_prim...
dbl  (6): publication_year, latitude, longitude, depth, sample_age, sampling...

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.
```

A naive assumption about this dataset might be that there is a correlation of the variables `depth` and `sample_age`. Here is a definition of these variables taken from the meta-data specification:

- `depth`: "Depth of sample taken from top of sequence in centimeters. In case of ranges use midpoint"
- `sample_age`: "Age of the sample in year before present (BP 1950), to the closest century"

2: Of course we could only detect this for samples with both depth and age information. Filter the dataset to only include samples with it. And also remove samples without an archaeological site name (`(!= "Unknown")`).

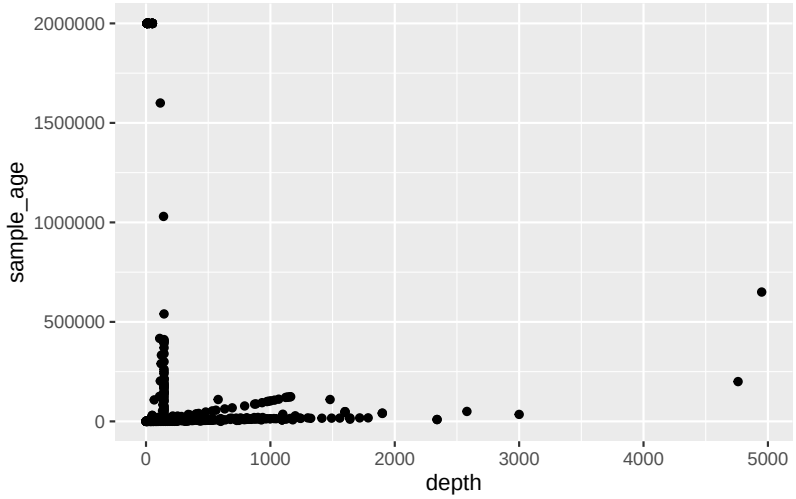
Solution

```
samples_with_depth_and_age <- samples %>%
    dplyr::filter(
        !is.na(depth) & !is.na(sample_age),
        site_name != "Unknown"
    )
```

3: Now plot `depth` against `sample_age` in a scatterplot to see if there is a potential signal.

Solution

```
samples_with_depth_and_age %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age))
```

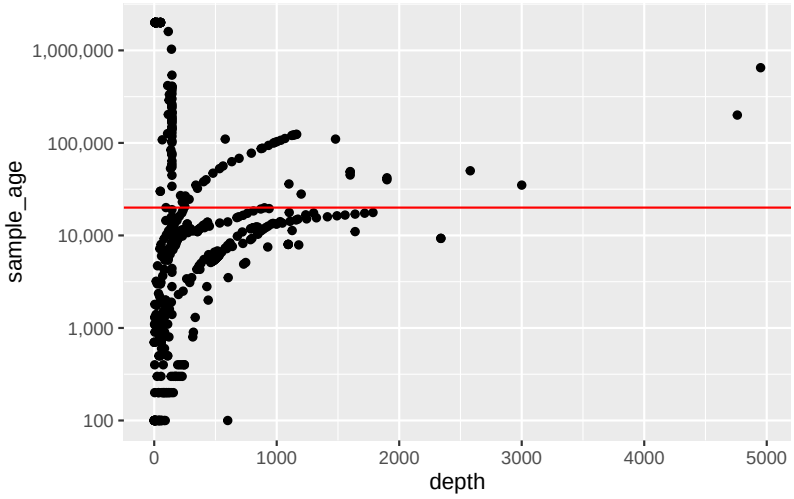


We can't see much here, because samples with very large ages dominate the y-scale.

4: Recreate this plot with a log-scaled axis.

Solution

```
samples_with_depth_and_age %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age)) +
    scale_y_log10(labels = scales::label_comma()) +
    geom_hline(yintercept = 20000, color = "red")
```



This is more interesting. There may be a signal for samples, specifically below a certain age, maybe 20000 years BP.

5: Filter the dataset to remove all samples that are older than this threshold. Store the result in a variable `samples_young`.

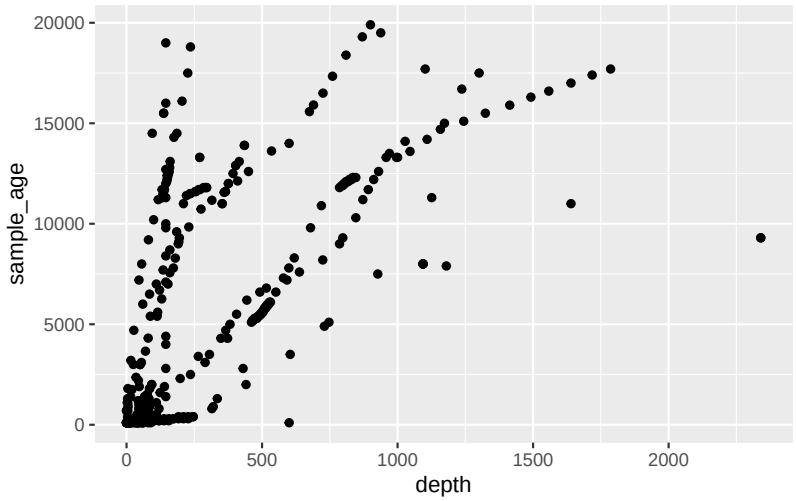
Solution

```
samples_young <- samples_with_depth_and_age %>%
    dplyr::filter(
        sample_age < 20000
    )
```

6: Recreate the plot from above. The log-scaling can be turned off now.

Solution

```
samples_young %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age))
```



With the old samples removed, there indeed seems to be some correlation. Pearson's correlation coefficient is not that strong though.

```
cor(samples_young$depth, samples_young$sample_age, method = "pearson")
```

```
[1] 0.5850941
```

Maybe what we see is mostly driven by individual sites. How many sites are there actually?

7: Determine the number of sites in the filtered dataset.

Solution

```
samples_young$site_name %>%
    unique() %>%
    length()
```

```
[1] 28
```

And how many samples are there per site?

8: Calculate the number of samples per site with `group_by` and `summarize`. Sort the result table by the number of samples with `arrange`.

Solution

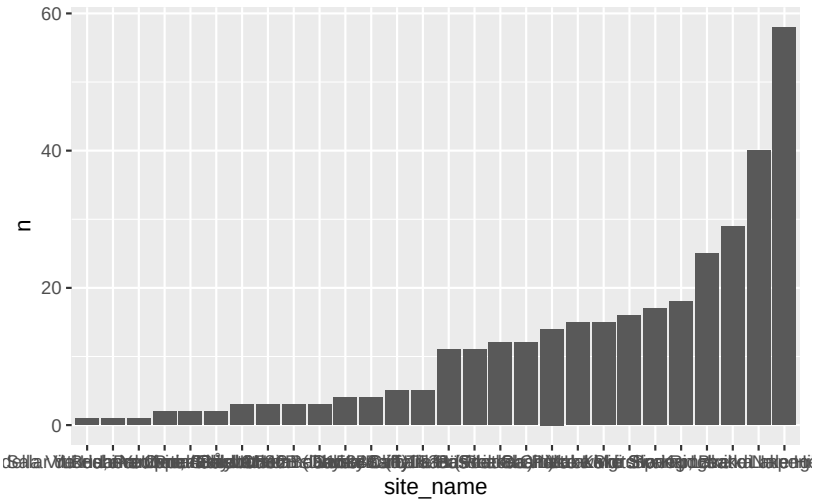
```
sample_count_per_site <- samples_young %>%
    dplyr::group_by(site_name) %>%
    dplyr::summarise(n = dplyr::n()) %>%
    dplyr::arrange(n)
```

9: Prepare a bar-plot that shows this information, with the sites on the x-axis and the number of samples per site on the y-axis. The bars should be ordered by the number of samples.

Solution

```
sample_count_per_site$site_name <- forcats::fct_reorder(
    sample_count_per_site$site_name,
    sample_count_per_site$n
)

sample_count_per_site %>%
    ggplot() +
    geom_bar(aes(x = site_name, y = n), stat = "identity")
```

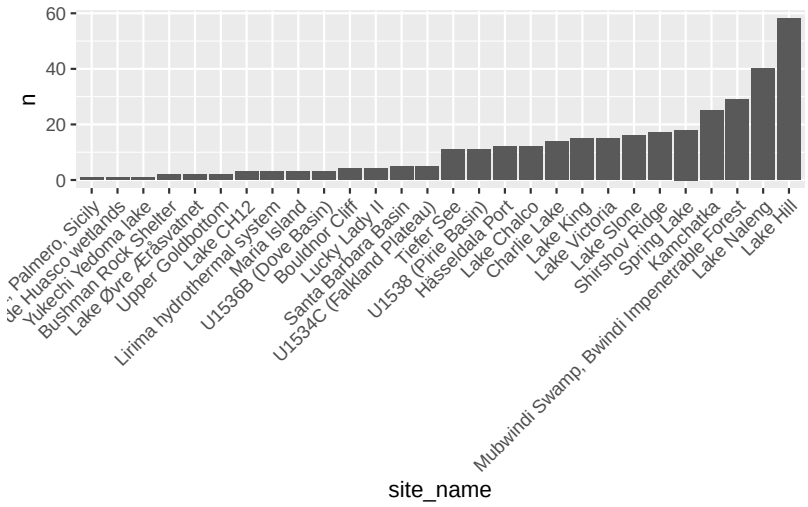


Oh no - the x-axis labels are not well readable in this version of the plot.

10: Create a version where they are slightly rotated.

Solution

```
sample_count_per_site %>%
    ggplot() +
    geom_bar(aes(x = site_name, y = n), stat = "identity") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```



What the oldest and youngest samples for each site?

11: Use `group_by`, `arrange` and `dplyr::slice(1, dplyr::n())` to get the oldest and youngest sample for each site in the filtered dataset.

Solution

```
sites_oldest_youngest <- samples_young %>%
    dplyr::group_by(site_name) %>%
    dplyr::arrange(sample_age) %>%
    dplyr::slice(1, dplyr::n()) %>%
    dplyr::ungroup()
```

The result is a bit hard to read because it includes all columns of the input table.

12: Select only the columns `site_name` and `sample_age` and show all rows with `print(n = Inf)`.

Solution

```
sites_oldest_youngest <- sites_oldest_youngest %>%
    dplyr::select(site_name, sample_age) %>%
    print(n = Inf)
```

```
# A tibble: 56 x 2
   site_name                                sample_age
   <chr>                                         <dbl>
 1 Bouldnor Cliff                                 8000
 2 Bouldnor Cliff                                 8000
 3 Bushman Rock Shelter                            500
 4 Bushman Rock Shelter                            500
 5 Charlie Lake                                  11300
 6 Charlie Lake                                  12800
 7 Hässeldala Port                               11000
 8 Hässeldala Port                               13900
 9 Kamchatka                                      1080
10 Kamchatka                                     19900
11 Lake CH12                                      1900
12 Lake CH12                                      6700
13 Lake Chalco                                     100
14 Lake Chalco                                   11500
15 Lake Hill                                      3400
16 Lake Hill                                     17500
17 Lake King                                       100
18 Lake King                                       400
19 Lake Naleng                                     100
20 Lake Naleng                                   17700
21 Lake Slone                                      200
22 Lake Slone                                     2000
23 Lake Victoria                                   100
24 Lake Victoria                                   400
25 Lake Øvre Æråsvatnet                          17700
26 Lake Øvre Æråsvatnet                          19500
27 Lirima hydrothermal system                      500
28 Lirima hydrothermal system                     1100
```

13: Further simplify this dataset to only one row per site (`group_by`) and add a column (`summarize`) that shows the distance between min and max age, so the age range per site.

Solution

```
sites_age_range <- sites_oldest_youngest %>%
    dplyr::group_by(site_name) %>%
    dplyr::summarise(age_range = max(sample_age) - min(sample_age)) %>%
    print(n = Inf)
```

```
# A tibble: 28 x 2
   site_name                                  age_range
   <chr>                                          <dbl>
 1 Bouldnor Cliff                                     0
 2 Bushman Rock Shelter                               0
 3 Charlie Lake                                    1500
 4 Hässeldala Port                                 2900
 5 Kamchatka                                      18820
 6 Lake CH12                                       4800
 7 Lake Chalco                                    11400
 8 Lake Hill                                      14100
 9 Lake King                                        300
10 Lake Naleng                                    17600
11 Lake Slone                                      1800
12 Lake Victoria                                    300
13 Lake Øvre Æråsvatnet                            1800
14 Lirima hydrothermal system                       600
15 Lucky Lady II                                   2200
16 Maria Island                                       0
17 Mubwindi Swamp, Bwindi Impenetrable Forest      2100
18 Piazza della Vittoria, Palmero, Sicily             0
19 Salar de Huasco wetlands                           0
20 Santa Barbara Basin                            10200
21 Shirshov Ridge                                 17000
22 Spring Lake                                    10400
23 Tiefer See                                     11200
24 U1534C (Falkland Plateau)                      18300
25 U1536B (Dove Basin)                             3900
26 U1538 (Pirie Basin)                            14400
27 Upper Goldbottom                                   0
28 Yukechi Yedoma lake                                0
```

So some sites have a huge age range of thousands of years, and others do not. This information is not really meaningful without the number of samples per site, though.

14: Join the sample count per site (as computed above) with the age range per site to get a table with both variables.

Solution

```
sites_joined <- dplyr::left_join(
    sites_age_range,
    sample_count_per_site,
    by = "site_name"
) %>%
    print(n = Inf)
```

```
# A tibble: 28 x 3
   site_name                                age_range     n
   <chr>                                        <dbl> <int>
 1 Bouldnor Cliff                                   0     4
 2 Bushman Rock Shelter                             0     2
 3 Charlie Lake                                  1500    14
 4 Hässeldala Port                               2900    12
 5 Kamchatka                                    18820    25
 6 Lake CH12                                     4800     3
 7 Lake Chalco                                  11400    12
 8 Lake Hill                                    14100    58
 9 Lake King                                      300    15
10 Lake Naleng                                  17600    40
11 Lake Slone                                    1800    16
12 Lake Victoria                                  300    15
13 Lake Øvre Æråsvatnet                          1800     2
14 Lirima hydrothermal system                     600     3
15 Lucky Lady II                                 2200     4
16 Maria Island                                     0     3
17 Mubwindi Swamp, Bwindi Impenetrable Forest    2100    29
18 Piazza della Vittoria, Palmero, Sicily           0     1
19 Salar de Huasco wetlands                         0     1
20 Santa Barbara Basin                          10200     5
21 Shirshov Ridge                               17000    17
22 Spring Lake                                  10400    18
23 Tiefer See                                   11200    11
24 U1534C (Falkland Plateau)                    18300     5
25 U1536B (Dove Basin)                           3900     3
26 U1538 (Pirie Basin)                          14400    11
27 Upper Goldbottom                                 0     2
28 Yukechi Yedoma lake                              0     1
```

15: Calculate the mean sampling interval by dividing the age range by the number of samples and add this information in a new column with mutate.

Solution

```
sites_joined %>%
    dplyr::mutate(
        sampling_interval = age_range / n
    ) %>%
    print(n = Inf)
```

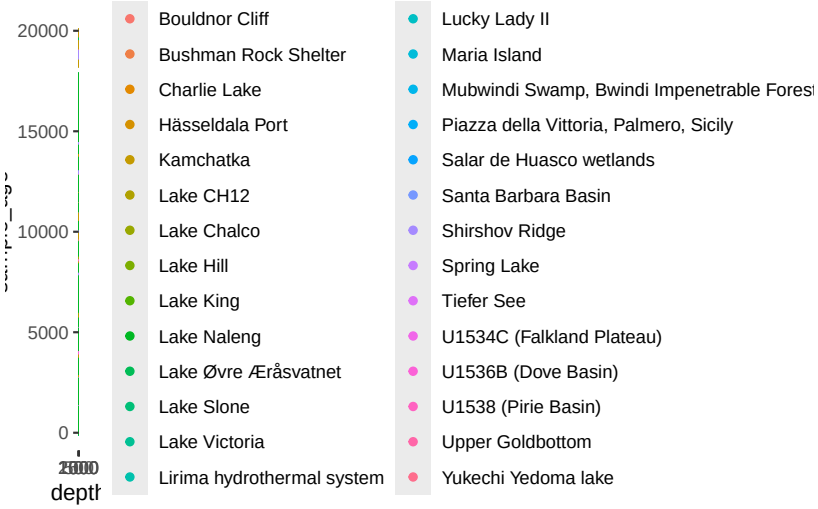
```
# A tibble: 28 x 4
   site_name                                age_range     n sampling_interval
   <chr>                                        <dbl> <int>             <dbl>
 1 Bouldnor Cliff                                   0     4                 0
 2 Bushman Rock Shelter                             0     2                 0
 3 Charlie Lake                                  1500    14               107.
 4 Hässeldala Port                               2900    12               242.
 5 Kamchatka                                    18820    25               753.
 6 Lake CH12                                     4800     3              1600
 7 Lake Chalco                                  11400    12               950
 8 Lake Hill                                    14100    58               243.
 9 Lake King                                      300    15                20
10 Lake Naleng                                  17600    40               440
11 Lake Slone                                    1800    16               112.
12 Lake Victoria                                  300    15                20
13 Lake Øvre Æråsvatnet                          1800     2               900
14 Lirima hydrothermal system                     600     3               200
15 Lucky Lady II                                 2200     4               550
16 Maria Island                                     0     3                 0
17 Mubwindi Swamp, Bwindi Impenetrable Forest    2100    29                72.4
18 Piazza della Vittoria, Palmero, Sicily           0     1                 0
19 Salar de Huasco wetlands                         0     1                 0
20 Santa Barbara Basin                          10200     5              2040
21 Shirshov Ridge                               17000    17              1000
22 Spring Lake                                  10400    18               578.
23 Tiefer See                                   11200    11              1018.
24 U1534C (Falkland Plateau)                    18300     5              3660
25 U1536B (Dove Basin)                           3900     3              1300
26 U1538 (Pirie Basin)                          14400    11              1309.
27 Upper Goldbottom                                 0     2                 0
28 Yukechi Yedoma lake                              0     1                 0
```

After this never-ending digression we can go back to the initial question: Is there a global relationship between sample_age and depth?

16: Take the `samples_young` dataset and recreate the simple scatter plot from above. But now map the `site_name` to the point colour.

Solution

```
samples_young %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age, color = site_name))
```

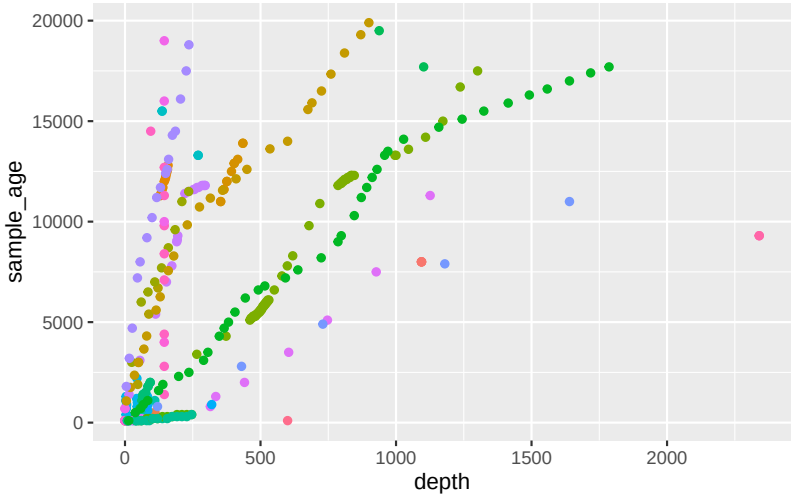


There are a lot of sites, so the legend for the colour space is annoyingly large.

17: Turn it off with `+ guides(color = guide_none())`.

Solution

```
samples_young %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age, color = site_name)) +
    guides(color = guide_none())
```

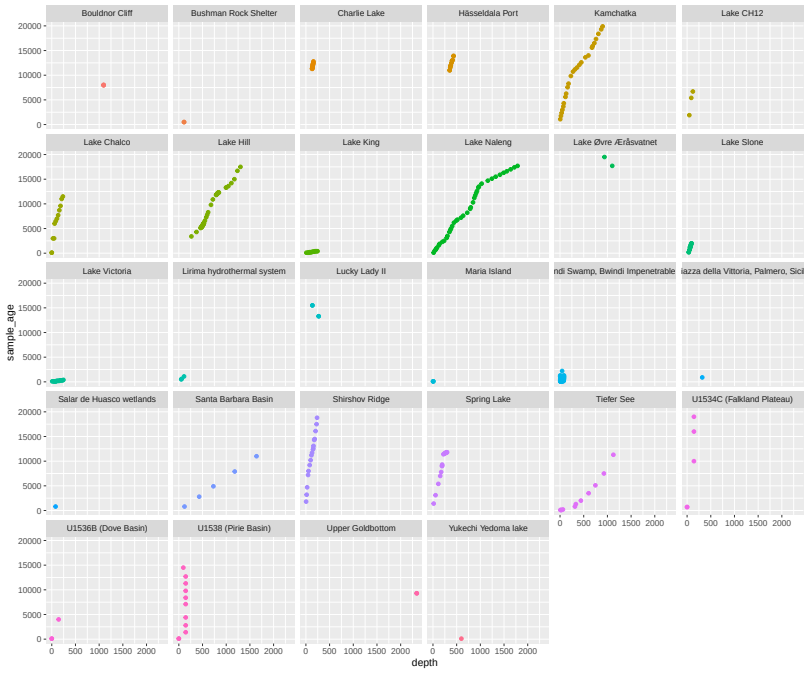


It might be helpful to look at the sites separately to make sense of this data.

18: Use faceting to split the plot into per-site subplots.

Solution

```
samples_young %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age, color = site_name)) +
    guides(color = guide_none()) +
    facet_wrap(~site_name)
```

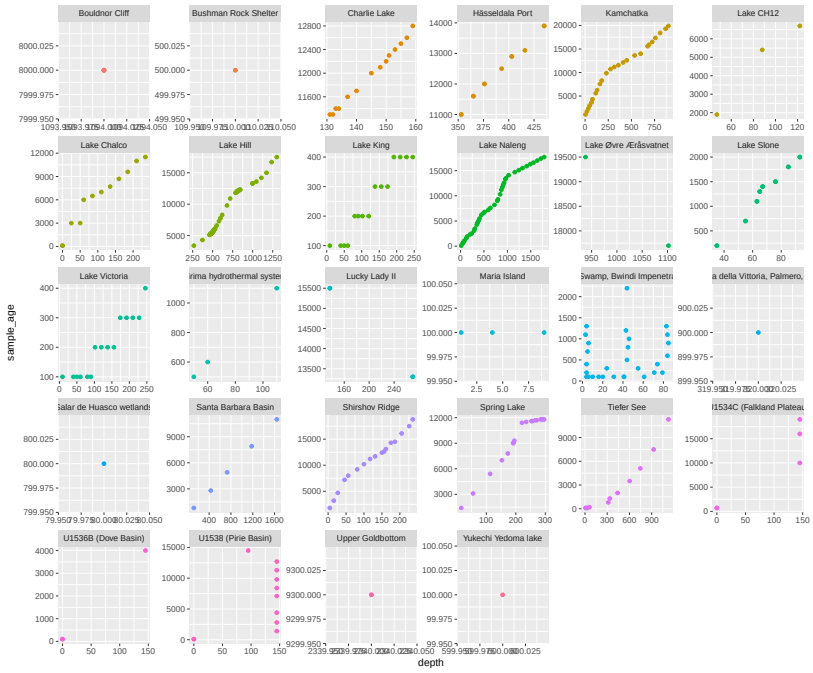


It is not exactly surprising that the sites operate on different scales regarding age and depth.

19: Add the `scales = "free"` option to `facet_wrap` to dynamically adjust the scaling of the subplots.

Solution

```
samples_young %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age, color = site_name)) +
    guides(color = guide_none()) +
    facet_wrap(~site_name, scales = "free")
```

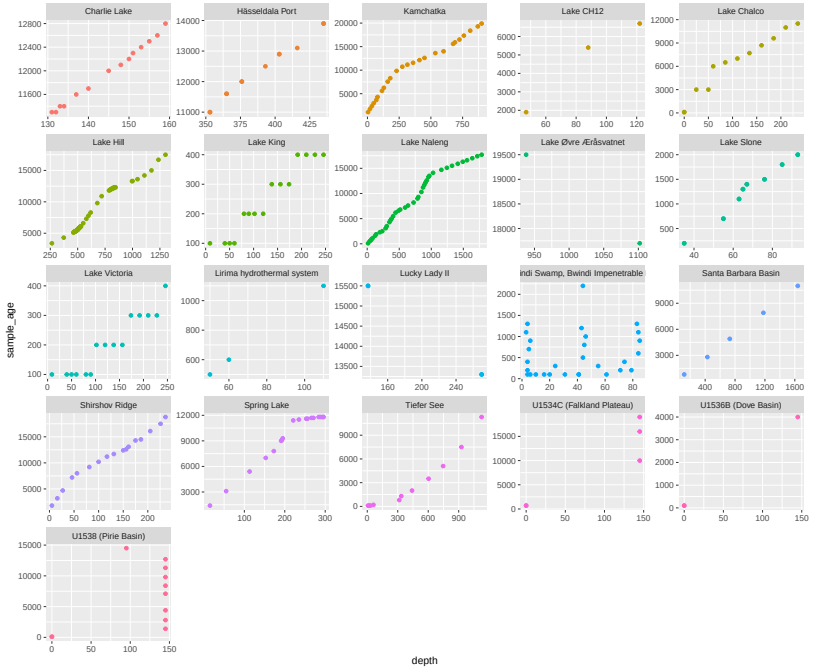


Some sites have too few samples to contribute meaningfully to our main question.

20: As a final exercise remove these "single-dot" sites and recreate the plot. There are many possible ways to do this. One way may be to filter by the standard deviation (sd) along the age or the depth axis.

Solution

```
samples_young %>%
    dplyr::group_by(site_name) %>%
    dplyr::filter(sd(depth) > 5) %>%
    dplyr::ungroup() %>%
    ggplot() +
    geom_point(aes(x = depth, y = sample_age, color = site_name)) +
    guides(color = guide_none()) +
    facet_wrap(~site_name, scales = "free")
```



Unsurprisingly this facetted plot visually confirms that `depth` and `sample_age` are often correlated for the sites in the environmental samples table of the AncientMetagenomeDir. It also shows a number of notable exceptions that clearly stand out in this plot.
We conclude the analysis at this point.

7.9 (Optional) clean-up

Let's clean up your working directory by removing all the data and output from this chapter.

When closing `rstudio`, say no to saving any additional files.

The command below will remove the `/<PATH>/<TO>/r-tidyverse` directory **as well as all of its contents**.

> 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!

```
rm -r /<PATH>/<TO>/r-tidyverse*
```

Once deleted you can move elsewhere (e.g. `cd ~`).

We can also get out of the `conda` environment with

```
conda deactivate
```

To delete the conda environment

```
conda remove --name r-tidyverse --all -y
```

7.10 References

Chapter 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.

> **Self guided: chapter environment setup**
>
> For this chapter's exercises, if not already performed, you will need to download the chapter's dataset, decompress the archive, and create and activate the conda environment.
> Do this, use `wget` or right click and save to download this Zenodo archive: 10.5281/zenodo.11394586, and unpack
>
> ```
> tar xvf python-pandas.tar.gz
> cd python-pandas/
> ```
>
> You can then create the subsequently activate environment with
>
> ```
> conda env create -f python-pandas.yml
> conda activate python-pandas
> ```

Over the last few years, *Python* has gained popularity thanks to the numerous libraries (packages with pre-written functions) in bioinformatics, statistical data analysis, and machine learning. 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 useful libraries for working on metagenomic

data. Existing libraries exist for many different file formats encountered in metagenomics, such as fasta, fastq, sam, bam, etc. Furthermore, python is fast and extremely useful for writing programs that can be easily called from the command line like many existing tools.

This tutorial/walkthrough will provide a short introduction to the popular libraries for data analysis `pandas` ((https://pandas.pydata.org/). This library has functions for reading and manipulating *tabular data* similar to the *`data.frame()`* in *R* together with some basic data plotting. This will set the base for learning Python and use it for data analysis.

There are many IDEs in which Python code can be written. For data analysis, Jupyter is powerful and popular which looks and functions similar to R markdown, where code is written in code blocks with space in text blocks for annotations. In this tutorial/walkthrough, we will use these notebooks for running and visualising Python code.

Learning objectives:

- Get familiar with the Python code syntax and use Jupyter Notebook for executing code
- Get a kickstart to utilising the endless possibilities of data analysis in Python that can be applied to our data

8.1 Working in a Jupyter environment

This tutorial/walkthrough is using a Jupyter Notebook (https://jupyter.org) for writing and executing Python code and for annotating.

Jupyter notebooks have two types of cells: *Markdown* and *Code*. The *Markdown cell* syntax is very similar to *R markdown*. The markdown cells are used for annotating code, which is important for sharing work with collaborators, reproducibility, and documentation.

Change the directory to the the working directory of this tutorial/walkthrough.

```
cd python-pandas_lecture/
```

To launch jupyter, run the following command in the terminal. This will open a browser window with jupyter running.

```
jupyter notebook
```

Jupyter Notebook should have a file structure with all the files from the working directory. Open the `student-notebook.ipynb` notebook by clicking on it. This notebook has exactly the same code as written in this book chapter and is only a support so that it is not necessary to copy and paste the code. It is of course also possible to copy the code from this chapter into a fresh notebook file by clicking on: `File` > `New` > `Notebook`.

> **Note** If the notebook is not there
>
> If you cannot find `student-notebook.ipynb`, it is possible the working directory is not correct. Make sure that `pwd` returns `/<path>/<to>/python-pandas/python-pandas_lecture`.

8.1.1 Creating and running cells

There are multiple ways of making a new cells in jupyter, such as typing the letter `b`, or using the cursor on the bottom of the page that says *`click to add cell`*. The cells can be assigned to `code` or `markdown` using the drop down menu at the top. Code cells are always in edit mode. Code can be run with pressing `Shift + Enter` or click on the botton. To make an markdown cell active, double-click on a markdown cell, it switches from display mode to edit mode. To leave the editing mode by running the cell.

> **Clear your code cells**
>
> Before starting it might be nice to clear the output of all code cells, by clicking on:
> `edit` > `Clear outputs of All Cells`

8.1.2 Markdown cell syntax

Here a few examples of the syntax for the Markdown cells are shown, such as making words **bold**, or *italics*. For a more comprehensive list with syntax check out this Jupyter Notebook cheat-sheet (https://www.ibm.com/docs/en/watson-studio-local/1.2.3?topic=notebooks-markdown-jupyter-cheatsheet).

List of *markdown cell* examples:

- `**bold**` : **bold**
- `_italics_` : *italics*

Code

- ‘inline code‘ : `inline code`

LaTeX maths

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

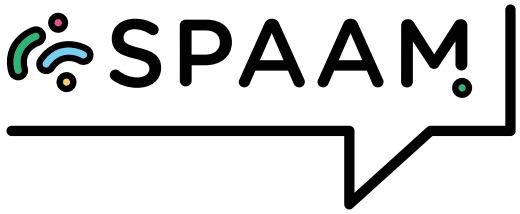
$$x = \frac{\pi}{42}$$

URL links

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

Images

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



> **All roads lead to Rome**
>
> In many cases, there are multiple syntaxes, or 'ways of doing things,' that will give the same results. For each section in this tutorial/walkthrough, one way is presented.

8.1.3 code cell syntax

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()**
>
> `print()` is a python function for printing lines in the terminal
> `print()` is the same as `echo` in bash

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

> **Expand to see output**
>
> ```
> Hello World from Python!
> ```

> **Running bash code in Jupyter**
>
> It is also possible to run bash commands in Jupyter, by adding a *!* at the start of the line.
>
> ```
> ! echo "Hello World from bash!"
> ```
>
> ```
> Hello World from bash!
> ```

Stings or numbers can be stored as a variable by using the = sign.

```
i = 0
```

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

To see what value a variable contains, the `print()` function can be used.

```
print(i)
```

> Expand to see output
>
> ```
> 0
> ```

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

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

> Expand to see output
>
> ```
> The number is, 0, Wow!
> ```

8.2 Pandas

8.2.1 Getting started

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

We can import the library like this.

```
import pandas as pd
```

> Why import as pd?
>
> We set `pandas` to the alias `pd` because we are lazy and do not want to write the full word too many times.

Now that `Pandas` is imported, we can check if it worked correctly, and check which version is running by runing `.__version__`.

```
pd.__version__
```

> Expand to see output
>
> '2.2.2'

8.2.2 Pandas data structures

The primary data structures in `Pandas` are the `Series` and the `DataFrame`. A Series is a one-dimensional array-like object containing a value of the same type and can be imagined as one column in a table Figure **??**. Each element in the `series` is associated with an index from 0 to the number of elements, but these can be changed to labels. A `DataFrame` is two-dimensional, and can change in size after it is created by adding and removing rows and columns, which can hold different types of data such as numbers and strings Figure **??**. The columns and rows are labelled. By default, rows are unnamed and are indexed similarly to a `series`.

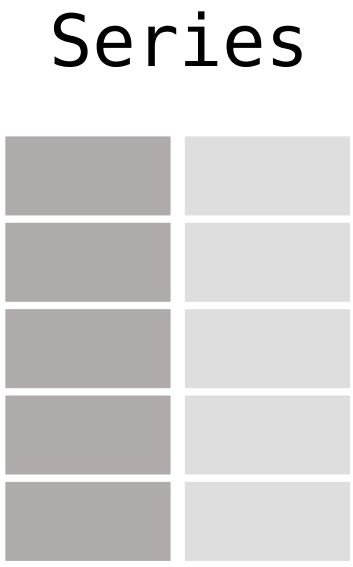


Figure 8.1: A single row or column (1-dimensional data) is a `Series`. The dark grey squares are the index or row names, and the light grey squares are the elements.

More details on pandas

For a more in detail pandas getting started tutorial click here (https://pandas.pydata.org/docs/getting_started/index.html#)

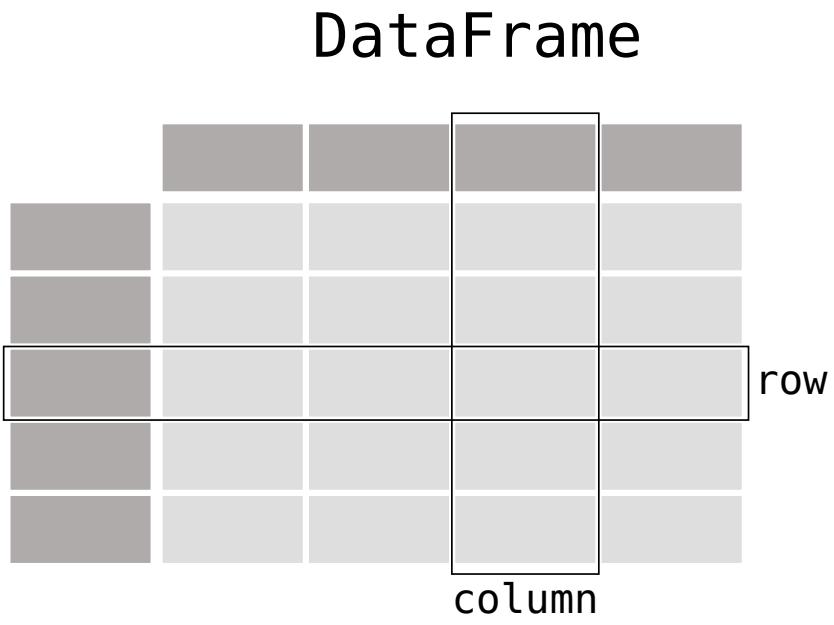


Figure 8.2: A dataframe with columns and rows. The dark grey squares are the index/row names and the column names. The light grey squares are the values.