Data Management and Digital Archaeology

Andreas Angourakis Tim Klingenberg

11 October, 2024

Table of contents

Course overview

040468 Datenmanagement und digitale Archäologie (ÜB, unterrichtet in Englisch) Übungen (3 CP)

Time slot / ***Zeitfenster***: Fr 14-16 Uhr c.t.

Place / ***Ort***: Raum 2

Course instructors / ***Kursleiter***: Andreas Angourakis / Tim Klingenberg

Support / ***Unterstützung***: Thomas Rose

Course schedule/*Kursplan*:

	Date	Topic
Getting started		
1	2024-10-18	Data and research data management
2	2024-10-25	Git and GitHub
Programming in R		
3	2024-11-08	Introduction to R
4	2024-11-15	Best practices in programming
Data Science in R		
5	2024-11-22	Data Science Workflow
6	2024-11-29	Count data and seriation
7	2024-12-06	Compositional data
Databases		
8	2024-12-13	Databases (I)
9	2024-12-20	Databases (II)
GIS		
10	2025-01-10	GIS (I)
11	2025-01-17	GIS (II)

Evaluation / *Kursbewertung*

(Attendance and final examination)

Acknowledgements

The conception of the course structure, as well as the short summaries, exercises, and images shown in each chapter, greatly benefited from Large Language Models used as companion writer and programmer. As such, we own greatly to the current richness of reference information freely available on Internet.

The models and services used are:

- ChatGPT (GPT-4o) by OpenAI for brainstorming, text and code writing suggestions, collection and articulation of references.
- WebChatGPT, a free browser extension that enhances ChatGPT by providing Internet access directly within the chat interface, used to aid Internet search.
- Leonardo.ai (user tokens) for generating purely aesthetic visual assets.

Part I

Getting started

1 Data and research data management

1.1 Introduction to Research Data

1.1.1 What is data? What is it for?

Data refers to factual information, often in quantitative or qualitative form, used as a basis for reasoning, discussion, or calculation. It serves as the foundational element for analysis and supports decision-making across diverse fields. In research, data helps to generate new insights, verify hypotheses, and contribute to the overall body of knowledge in a specific area.

1.1.2 Archaeological Data: Particularities

In archaeology, data encompasses various types such as field notes, artefacts, images, geospatial coordinates, and more. By systematically collecting, organizing, and analysing this data, researchers can reconstruct past human behaviours, understand environmental contexts, and explore cultural practices.

Archaeological data is unique due to its diverse formats and the complexity of its collection. It often includes material remains like pottery, bones, tools, and structures, as well as environmental data like pollen samples and soil types. Since archaeological data often comes from excavation sites, it is usually non-renewable; once excavated, a site cannot be restored to its original state.

Archaeologists rely heavily on careful documentation to preserve as much information as possible for future study. Furthermore, the context in which artefacts are found is crucial, as it helps interpret their use, significance, and the broader cultural setting.

1.1.3 Open Science

Open Science promotes transparency and collaboration in research by making methodologies, data, and findings accessible to the public and other researchers. In archaeology, this can involve sharing excavation reports, datasets, and analysis methods to facilitate wider understanding and scrutiny of findings.

1.1.4 FAIR Principles

FAIR stands for Findable, Accessible, Interoperable, and Reusable. These principles aim to enhance the usefulness of digital assets by ensuring they can be easily located, understood, and utilized by others. Applying FAIR principles in archaeology involves creating well-documented, open-access datasets that other researchers can readily use.

1.1.5 Open Source

Open Source refers to software and tools that are freely available for anyone to use, modify, and distribute. For archaeologists, open-source tools can offer affordable solutions for data analysis, visualization, and data management, promoting a more inclusive research environment.

1.1.6 Reproducibility

Reproducibility refers to the ability to replicate the results of a study using the original author's assets or following their methodology. It ensures that findings can be independently verified under similar conditions, reinforcing the reliability of scientific research (National Academies of Sciences et al. 2019).

In scientific research, reproducibility is crucial for confirming the validity of experimental findings and building upon existing knowledge. It enhances transparency and trustworthiness in scientific practices, promoting better peer review and collaboration "GRN · German Reproducibility Network" (n.d.).

Most research in archaeology does not necessarily involve controlled experiments and archaeological survey and excavation are destructive, thus unrepeatable. However, the reproducibility of data collection, processing and analysis is not a trivial concern.

Making research reproducible must be considered as a spectrum of practices, in which researchers should thrive for doing better, despite the challenges (Marwick 2017).

1.1.7 The lay of the land: Organizations, tools, data repositories and where to find them

Several organizations and platforms support Open Science and the use of FAIR and Open Source principles, providing archaeologists with access to valuable tools and datasets:

- Organizations:

  - OpenAIRE: OpenAIRE AMKE is a non-profit organization with a mission to promote open scholarship and improve discoverability, accessibility, shareability, reusability, reproducibility, and monitoring of data-driven research results, globally (Iatropoulou n.d.).
  - Go FAIR Initiative: Provides guidelines on implementing FAIR principles, with resources tailored for researchers in various fields, including archaeology ("FAIR Principles" n.d.).
  - The Turing Way: collaborative writing of an online handbook (Community 2022).
  - CAA: Computer Applications and Quantitative Methods in Archaeology (CAA) is an international organisation bringing together archaeologists, mathematicians, and computer scientists. CAA counts with National Chapters, including one for Germany.

- Data Repositories and Tools:

  - Zenodo: A repository where researchers can deposit datasets, software, and publications.

  - Open Science Foundation: OSF is a free, open platform to support your research and enable collaboration.
  - Archaeology Data Service (ADS): An archive for archaeological data from the UK, which provides access to various datasets, including excavation reports and geospatial data.

  - Open Context: A platform offering access to archaeological data from various global sources, adhering to FAIR principles [noauthor_open_nodate-1].
  - GitHub: GitHub is a developer platform that allows developers to create, store, manage and share their code through the use of Git software and a series of additional automated services.

- Open Source Tools:

  - Markdown: Markdown is a lightweight markup language designed for creating formatted text using a plain-text editor ("Markdown Guide" n.d.).

  - R and Python: Programming languages with extensive libraries for data analysis, which are widely used in archaeology for statistical analysis and modeling (R Core Team 2024, noauthor_welcome_2024). See more about these and other programming languages below.

  - RStudio: RStudio IDE is an integrated development environment for R, a programming language for statistical computing and graphics.
  - Quarto: An open-source scientific and technical publishing system.

  - Git: Git is a distributed version control system designed for tracking changes in source code during software development.
  - Zotero: Zotero is a free, easy-to-use tool to help collect, organize, annotate, cite, and share metadata on references.

  - QGIS: A free, open-source GIS tool useful for mapping and spatial analysis in archaeology ("Spatial Without Compromise · QGIS Web Site" n.d.).

By utilizing these resources, archaeologists can ensure that their research aligns with Open Science, FAIR, and Open Source principles, ultimately enhancing the transparency, accessibility, and longevity of their work.

1.2 Introduction to Programming for Research

1.2.1 What is Programming?

Programming is the process of designing and implementing instructions that a computer can follow to perform specific tasks. It involves writing code in various programming languages that communicate with the computer's hardware and software to solve problems, process data, and automate repetitive tasks. At its core, programming is about creating a sequence of steps, known as algorithms, which help achieve a desired outcome ("What Is Programming? And How To Get Started" 2024).

1.2.2 Importance of Learning Programming

For researchers, learning programming offers several significant advantages:

- *Efficiency and Automation*: Programming can help automate data collection, processing, and analysis, saving time and reducing human error. It also enables researchers to handle large datasets and complex calculations with ease.
- *Reproducibility*: Writing scripts to perform analysis allows other researchers to replicate experiments, thus ensuring results can be verified and reproduced, a fundamental aspect of scientific research.
- *Access to Powerful Tools*: With programming skills, researchers can access a wide range of tools for data visualization, statistical analysis, machine learning, and simulation. These tools can enhance the scope and quality of research projects.

1.2.3 Overview of Common Programming Languages

Several programming languages are popular in research due to their specific features and libraries("Introduction to Programming Languages" 2018):

- *Python*: Known for its readability and versatility, Python is widely used for data analysis, machine learning, and automation. It has extensive libraries such as NumPy, pandas, and matplotlib, which are particularly useful for data-intensive research.
- *R*: A language specifically developed for statistical computing and graphics, R is preferred in data science, bioinformatics, and fields requiring extensive statistical analysis. The Comprehensive R Archive Network (CRAN) offers thousands of packages that can handle a range of analytical tasks.
- *MATLAB*: Commonly used in engineering and scientific research, MATLAB excels at numerical computing, simulation, and algorithm development. It is particularly popular in fields like physics, engineering, and finance.
- *JavaScript*: While primarily a web development language, JavaScript is also used in research for developing interactive data visualizations and web-based applications.

1.2.4 Concept of Research Software: Tools and Scripts

Research software comprises tools, libraries, and scripts that aid in conducting and managing research activities. It can range from simple scripts for data cleaning and preprocessing to complex software packages for statistical analysis and simulation. Although any software used in research could be considered as research software, advance training focus on programming or scripting skills, not on graphical user interface operations (e.g., creating a plot in R or Python rather than in Microsoft Excel).

- *Tools*: Research software tools like Jupyter Notebooks, RStudio, PyCharm, Visual Studio Code, etc., offer integrated development environments (IDEs) tailored to a specific range of languages, enhancing productivity and facilitating code sharing and version control.
- *Scripts*: Scripts are text files encoding programs written to perform specific tasks. In research, scripts are often used for data cleaning, model training, and result visualization. Scripts can be used even to generate and format an entire dataset (have a peek on how the course schedule has been built with R code). These scripts can be shared among researchers to ensure that analyses are reproducible and consistent.

Learning programming for research allows scientists to leverage these tools and scripts effectively, making research more efficient, reproducible, and insightful.

1.2.5 Markdown: a language in between

As already mentioned, Markdown is a markup language with minimal syntax, designed for creating formatted text using a plain-text editor ("Markdown Guide" n.d.). For us, humans, Markdown is easily readable and editable as text. However, it is technically a programming language, as it holds machine-readable instructions (i.e. program) which, once compiled (or rendered) by the right software, produces the desired output in form of a formatted text. A advanced text editor, such as Microsoft Word, allow you to do exactly the same (and more), but only through a sequence of interactions with the graphic user interface (GUI).

Writing in Markdown or "writing Markdown code" is straightforward and will only require some getting use to, especially if you costumed to advanced text editors. For example, instead of marking a fragment of text in bold font through the usual steps, we would do it by typing ** (two asterisks) before and after the fragment.

To illustrate this, consider the rendering and source code of a markdown text:

"**Live** *as if* you were to ***die tomorrow***. **Learn** *as if* you were to ***live forever***." - Mahatma Gandhi, Source: BrainyQuote

```
"**Live** *as if* you were to **_die tomorrow_**. **Learn** as if you were to **_live forever
Mahatma Gandhi, Source: [BrainyQuote](https://www.brainyquote.com/citation/quotes/mahatma_ga
```

The simplicity of this language might feel unnecessary, or even annoying, but it is actually the trait that opens a great range of potential applications. Markdown is today indispensable for all practices related to open science and can offer a good entry point for other, more complex programming languages.

> **i** See also
>
> - Obregon (2024)
> - "What Is Markdown? Syntax, Examples, Usage, Best Practices" (2021)
> - "Functional Documentation with Markdown and Version Control" (2017)
> - "Basic Syntax | Markdown Guide" (n.d.)

1.2.6 Where to learn (and keep learning)

- Datacamp ("Learn R, Python & Data Science Online" n.d.)
- Udemy ("Online Courses - Learn Anything, On Your Schedule | Udemy" n.d.)

- Coursera ("Coursera | Degrees, Certificates, & Free Online Courses" n.d.)
- Courses and workshops by high education institutions (e.g„ RUB Research School)

Hands-on Practice

- Task 1: register yourself as user at:
  - GitHub
  - Zenodo
  - Zotero (optional)
- Task 2: installations:
  - Git. Follow general instructions here (de) or a video step-by-step tutorial like this) or this (de), among many available online.
  - GitHub Desktop
  - R
  - RStudio
  - QGIS
- Task 3: explore datasets at Open Context
- Task 4: try writing a short document in markdown at Markdown Live Preview.

2 Git and GitHub

2.1 Version-control and Git

2.1.1 Version control: general concept and its usefulness

Version control is a system that helps track and manage changes to files over time. It's widely used in software development, but its applications extend to any field where file management is essential, including document editing, research, and project management. At its core, version control provides a historical record of changes, allowing users to revert to previous versions, identify when and why changes were made, and work collaboratively without the risk of overwriting each other's work.

There are two main types of version control: centralized and distributed. Centralized version control systems, such as Subversion (SVN), store files in a central repository. Users check out files, make changes, and then commit them back to the central repository. While effective, centralized systems can be vulnerable if the central server fails. Distributed version control systems, like Git, address this by allowing every user to have a complete copy of the repository on their local machine. This setup enhances collaboration and provides redundancy, as users can work offline and synchronize changes with others when connected.

Git icon

2.1.2 Benefits of Version Control

- *Collaboration*: Version control systems make collaboration easier and more efficient by allowing multiple users to work on the same project simultaneously. With distributed systems like Git, branches can be created for different features or tasks, and changes can later be merged into the main project seamlessly. This enables teams to work independently and minimize conflicts.
- *Historical Tracking*: Version control systems keep a detailed history of all changes made to the files. This allows users to see who made changes, when they were made, and why. If an issue arises, it's possible to revert to a previous state without losing any data, making debugging easier.

- *Backup and Redundancy*: In distributed systems, each user's local copy serves as a backup of the entire project. This redundancy reduces the risk of data loss due to server failures or other issues and allows users to work offline and sync changes later.
- *Version Management*: Version control systems assign unique identifiers (usually called "commits") to each change. These identifiers allow users to switch between different versions of the project easily. It's also possible to create branches for experimental features and merge them with the main project once they're stable, facilitating smoother integration of new features.
- *Enhanced Workflow*: Many version control systems support automated processes such as Continuous Integration (CI) and Continuous Deployment (CD), which streamline development and testing. These systems can automatically test changes before they are merged, ensuring higher code quality and reducing the risk of introducing bugs.

Overall, version control systems are crucial tools in modern project management and development workflows. They enable collaboration, ensure data integrity, and improve productivity by providing a structured approach to managing changes in any type of project.

Get a short introduction on Git by watching the official Git Documentation videos here.

2.2 Git terminology

Here are some essential Git terms to know:

- **Repository**: A storage space for your project files and their history. Repositories can be local (on your computer) or remote (on platforms like GitHub).
- **Initialise**: configure a specific local directory as a working directory as a local repository by creating all necessary files for Git to work.
- **Add/Stage**: adds a change in the working directory to the staging area, telling Git to include updates to a particular file in the next commit. However, adding or staging doesn't really affect the repository since changes are not actually recorded until they are committed (see below).
- **Commit**: A snapshot of changes in the repository. Each commit has a unique ID, allowing you to track and revert changes as needed [1].
- **Branch**: A separate line of development. The default branch is usually called `main` or `master`. Branches allow you to work on features independently before merging them into the main project [2].

- **Merge**: The process of integrating changes from one branch into another. Typically, this involves merging a feature branch into the main branch.
- **Pull**: A command that fetches changes from a remote repository and merges them into your local branch, ensuring your local work is up-to-date with the remote [2].
- **Push**: Uploads your commits from the local branch to the remote repository, making your changes available to others.

Understanding these terms is crucial for effective Git usage and collaboration in any project.

> See also
>
> - "Git - Gitglossary Documentation" (n.d.)
> - "Git Definitions and Terminology Cheat Sheet" (n.d.)

> CHECK: Git software installation
>
> To verify if Git is installed on your machine, follow these steps:
>
> 1. **Open Command Prompt** (Windows 10 or 11)
>   - Press `Win + R`, type `cmd`, and hit Enter.
>   - Alternatively, you can search for "Command Prompt" in the Start menu and select it.
> 2. **Check for Git**
>   - In the Command Prompt window, type the following command and press Enter:
>
>      ```
>      git --version
>      ```
>
>   - If Git is installed, you will see the installed version, e.g., `git version 2.34.1`.
>   - If Git is not installed, you will receive an error message or see that the command is unrecognized. You can download the installer from git-scm.com and follow the installation instructions.

2.3 GitHub

2.3.1 What is GitHub?

GitHub is a cloud-based platform that enables developers to store, manage, and collaborate on code repositories. It builds on Git, a version control system, by adding collaborative features like pull requests, issue tracking, and discussions, which make it easier for teams to work together on software projects.

GitHub also offers hosting for open-source projects, allowing anyone to contribute or review code. With integrations for CI/CD, project management tools, and documentation, GitHub is a popular choice for developers worldwide to manage both personal and professional projects.

GitHub icon

CHECK: GitHub user and GitHub Desktop installation

Check GitHub Desktop Installation

To verify that GitHub Desktop is installed:

- On **Windows**: Go to the Start menu, search for "GitHub Desktop," and open the app. If it launches successfully, GitHub Desktop is installed.
- On **macOS**: Use Spotlight Search (`Cmd + Space`), type "GitHub Desktop," and press Enter. If the app opens, it is installed.

If you don't have GitHub Desktop, you can download it from desktop.github.com and follow the installation instructions [1][2].

Verify GitHub User

To check if you are signed in as a GitHub user:

- Open **GitHub Desktop**.
- Go to **File > Options** (on Windows) or **GitHub Desktop > Preferences** (on macOS).
- Under the **Accounts** tab, you should see your GitHub username and avatar if you are signed in. If not, you can sign in with your GitHub credentials here.

Bookmark your GitHub user profile page

In your Internet browser, make sure that your own GitHub user profile page is saved in Bookmarks for easy access later.

2.3.2 Working with GitHub

GitHub offers various workflows to manage repositories. Here are three common methods:

Local with GitHub Desktop

For those who prefer a graphical user interface (GUI):
Cloning a Repository

- Open GitHub Desktop.
- Go to File > Clone Repository.
- Select the repository and click "Clone."

Creating a New Branch

- Click on the "Current Branch" dropdown.
- Select "New Branch," name it, and click "Create Branch."

Making Changes

- Edit files in your editor.

Committing Changes

- Return to GitHub Desktop.
- Stage changed files by ticking the boxes.
- Write a summary of changes and click "Commit to new-branch."

Pushing Changes

- Click "Push origin" to upload your changes.

Remote with Web Browser

You can also work directly on GitHub.com:
Cloning a Repository

- Go to the repository page.

- Click the green “Code” button and copy the link.

Creating a New Branch

- Click the branch dropdown on the main page.
- Type a new branch name and click “Create branch.”

Making Changes

- Navigate to the file (and branch) you want to edit.
- Click the pencil icon to edit.
- Make your changes and scroll down to the “Commit changes” section.

Committing Changes

- Enter a commit message.
- Choose whether to “commit to directly to main” or “Commit to a new branch…”.

Pushing Changes
(No push is needed as changes are automatically saved to GitHub.)

Local with Console Commands (advanced users)

To work with Git via the command line:
Navigate to the directory to hold the local copy

```
cd path/to/local/directory
```

Cloning a Repository

```
git clone https://github.com/username/repository.git
```

Creating a New Branch

```
git checkout -b new-branch
```

Making Changes Edit files in your favorite text editor or IDE.
Committing Changes

```
git add .
git commit -m "Describe your changes"
```

Pushing Changes

```
git push origin new-branch
```

These workflows enable flexibility in how you manage your projects on GitHub.

2.3.3 Markdown (GitHub-flavoured)

When Markdown files (.md) are placed in a GitHub repository, they will be automatically rendered within GitHub web interface by default, while the raw code can still be seen and edited in Markdown.

There are some particularities about how Markdown files will be rendered in GitHub through Internet browsers. Consult GitHub Docs for knowing more about them.

2.3.4 How to organise repositories

When structuring your repositories, it's helpful to follow some common conventions for organizing files in subdirectories. This makes projects more readable and easier for others to navigate. Here are some commonly used subdirectories:

When software development is a significant part:
* `source/` or `src/`: Contains the main source code for the project.
* `documentation/`, `docs/`, or `doc/`: Stores documentation, such as guides or API references.
* `tests/`: Includes test scripts to ensure code functionality. * `bin/`: Holds executable scripts or binaries. * `config/`: Contains configuration files, like YAML or JSON.

When rendering an interface for users, such as an website, web app or video game:
* `assets`: to hold files and subdirectories with closed content and functionality files. * `assets/images/`, `assets/media/`, etc.: Holds all images or other media files generated externally (not by the repository's source code). * `assets/styles.css` or `assets/css/`: all CSS code for formatting HTML objects. * `assets/js/`: JavaScript source code enabling interactive functionalities (it would also apply for other programming languages in similar position). Source code of this kind might also be placed inside the source code folder, if present.

These folder structures are conventions and not strict rules. You can adapt or modify them based on your project's needs.

See also

- jimmy (2022)
- Zestyclose-Low-6403 (2023)
- danijar (2019)
- Suhail (2024)
- Cioara (2018)

2.3.5 Conventional files

- `README.md`: Provides an overview of the project, including what it does, how to set it up, and how to contribute. A few sections examples are:
  - General description
  - Authors and/or contributors
  - Acknowledgements
  - Funding
  - Installation or use instructions
  - Contributing
- `LICENSE`: Specifies the terms under which the content of the repository can be used, modified, and distributed. There are many licenses, varying in *permissiveness* and *type of content*. Generally, for projects involving both code and other kinds of content, we recommend CC0-1.0 or MIT. See https://choosealicense.com/ and GitHub Docs).
- `CITATION.cff`: human- and machine-readable citation information for software (and datasets). See example here.
- `.gitignore`: Lists files and directories that Git should ignore, such as build outputs and temporary files.
- `CHANGELOG.md`: Logs a chronological record of all notable changes made to the project, often following conventions like Conventional Commits.

- `references.bib`: a file containing references in BibTex format, which can be cited within the markdown files of the repository.

2.3.6 Version Tags and Releases on GitHub

To manage different versions of your project, GitHub allows you to create **tags** and **releases**:

1. **Create a Tag**:
    - Open your repository on GitHub and navigate to the **Releases** section.
    - Click **Draft a new release**.
    - In the **Tag version** field, type a version number (e.g., `v1.0.0`) to create a new tag (see more in the note below).
    - Specify the target branch or commit for this tag.
2. **Create a Release**:
    - After tagging, enter details such as the release **title** and **description**.
    - Optionally, add **release notes** to summarize changes or new features [1].
    - Click **Publish release** to make it public.

Releases are tied to tags and provide a stable reference for each version, making it easy for users to download specific versions of your project [2].

> **i** About versioning
>
> If unfamiliar with the logic behind versioning, consult the reference to Semantic Versioning which can also be found on the right of the "Create a new release" page in GitHub. Their summary states:
>
> > Given a version number MAJOR.MINOR.PATCH, increment the: 1. MAJOR version when you make incompatible API changes 2. MINOR version when you add functionality in a backward compatible manner 3. PATCH version when you make backward compatible bug fixes
>
> However, if your repository is not about creating software products and services, we can do well by simply obeying a few general conventions:
>
> - Add a PATCH version **discretionally** when correcting bugs, typos, tuning aesthetics, etc, or *refactoring* code (explained in Chapter 4).
> - Add a MINOR version when expanding code functionality or adding new content (text sections, images)
> - Add a new PATCH or MINOR version every time the repository reaches a natural stable point (i.e., there are no changes planned any time soon).

- Make sure that every new MAJOR version is released (GitHub) and published (Zenodo, see below).

See also

- "Creating GitHub Releases Automatically on Tags" (2024)
- "Automatically Generated Release Notes" (n.d.)
- Signell (2013)

2.3.7 Establishing a GitHub-Zenodo Connection

To link your GitHub repository with Zenodo and enable citation via DOI:

1. **Login to Zenodo**: Go to Zenodo and sign in or create an account.
2. **Authorize GitHub Access**:
   - Click on your profile in Zenodo and select **Linked accounts**.
   - Choose **Connect** next to GitHub.
   - You will be redirected to GitHub to authorize Zenodo's access. Approve the request to complete the connection.
3. **Select Repository for DOI Generation**:
   - In Zenodo, navigate to **GitHub** in the **Linked Accounts** section.
   - Enable DOI generation for the desired repository. Zenodo will automatically mint DOIs for any new release you publish.

This connection allows you to generate and manage DOIs for GitHub repositories, enhancing your project's citation and research accessibility.

See also

- "Referencing and Citing Content" (n.d.)
- "Zenodo - Research. Shared." (n.d.)
- "Created New Organization in GitHub and Zenodo Did Not Send a Request for Accessing It · Issue #1596 · Zenodo/Zenodo" (n.d.)
- "Module-5-Open-Research-Software-and-Open-Source/Content_development/Task_2.md at Master · OpenScienceMOOC/Module-5-Open-Research-Software-and-Open-

Source" (n.d.)

- "Issue a Doi with Zenodo" (n.d.)

Hands-on Practice

- Task 1: Create a profile repository with a README file following GitHub Docs: Quickstart for writing on GitHub.
- Task 2: Making others' repositories your own's
  1. Fork the Course Book repository.
  2. Clone your fork to a local directory.
  3. Modify or add a `.qmd` file in your local directory.
  4. Commit to you local repository (one or more times).
  5. Push the changed local repository to your remote repository (i.e. your fork).
  6. Create a pull request back to the original repository.
- Task 3: Create a personal project repository
  1. Create a GitHub repository under your user. Named it `rub-archwiss_` followed by your surname (no special characters). Initialise with the following properties:
     a. Public
     b. With a README file
     c. with .gitignore (template for R)
     d. With a license of you choice.
  2. Add all other conventional files mentioned above, even if they remain empty for now.
  3. Edit your README file and commit/push your changes.
- Task 4:
  - Set up the GitHub-Zenodo connection.
  - Publish your repository.

– Update README with the new Zenodo DOI (badge).

Part II

Programming in R

3 Introduction to R

3.1 Preparation

- Installing R and RStudio.
- Overview of RStudio interface.

3.2 R syntax and workflow

- Basic R syntax: variables and data types.
- Writing and executing R scripts.
- Arithmetic operations, logical operations in R.
- Algorithm structures: `if`, `else`, `while`, `function`
- Packages. Mention the most important packages used later, including `tidyverse` and `tesselle`.

3.3 Basic Data Structures in R

- Vectors, matrices, data frames, lists.
- Basic operations on data structures (indexing, subsetting, adding/removing elements).

3.4 Data Manipulation in R

- Importing data: reading data from CSV files, using canonical datasets (`iris`, `archdata::DartPoints`).

- Basic data manipulation using base R (`subset`, `merge`, `apply` functions).
- Introduction to `dplyr` package for data manipulation (filtering, selecting, mutating data).

3.5 Data Visualization

- Introduction to plots: histograms, bar plots, scatter plots.
- Creating plots in R with base R graphics.
- Creating multiple plot figures with `layout`.
- Creating plots in `ggplot2`.
- Creating multiple plot figures with `gridExtra::grid.arrange`.
- Base R graphics and `ggplot2`: comparative
- Saving plots: open and closing graphic devices in R.

3.6 (EXTRA)Interactive Visualizations

- Introduction to creating interactive visualizations.
- Example: Building an interactive plot using `plotly` and `knitr`.

Hands-on Practice

- Create a new project in RStudio, placing it at the root directory of your own repository (cloned local branch).
- Create a data.frame named "stone_tools_data" directly in R with the following characteristics (based on Carlson 2017, p. 26):
  - Set of six stone tools with inventory number
  - Recording of dimensions (length, breadth, thickness), material type, and whether the material is local or non-local.
  - Data per object:

* LN15:
    - Length: 18
    - Breadth: 9
    - Thickness: 3
    - Material type: chert
    - Material provenance: local
* LN17:
    - Length: 14
    - Breadth: 7
    - Thickness: 2
    - Material type: chert
    - Material provenance: local
* LN18:
    - Length: 21
    - Breadth: 10
    - Thickness: 3
    - Material type: obsidian
    - Material provenance: local
* LN21:
    - Length: 14
    - Breadth: 7
    - Thickness: 3

        - Material type: chert
        - Material provenance: non-local
    * LN23:
        - Length: 17
        - Breadth: 8
        - Thickness: 3
        - Material type: obsidian
        - Material provenance: local
    * LN24:
        - Length: 16
        - Breadth: 8
        - Thickness: 2
        - Material type: obsidian
        - Material provenance: non-local

- Check that the data and data types are coherent with the specifications. Save it as a CSV file and load it back as a new R object (e.g. "stone_tools_data2"). Compare.
- Create a plot showing the counts of objects made of chert and obsidian. Save it as a PNG file.
- Create a new variable ("type_and_provenance") that combines type and provenance and create a plot showing the counts in each category. Save it as a PNG file.
- Create a single figure displaying the variable distribution of the three dimensions measured. Save it as both a PNG and a SVG file.
- Create a plot displaying the relationship between length and breadth. Save it as a

PNG file.

- Create a plot displaying the relationship between length and breadth, this time marking (point type, colour) objects by their "type_and_provenance". Save it as both a PNG and a EPS file.
- (EXTRA) Create a figure to help explore the question: Do stone tools of different material and provenance tend to be of different size?
- Commit all changes and push to the remote using RStudio.
- Q&A and troubleshooting.

4 Best practices in programming

4.1 Code Organisation

- **Modular Programming**
  - Importance of modularity: breaking down code into functions and modules.
  - Example: Creating in-script custom functions.
  - Example: Creating and importing custom R scripts.
- **Code Structuring**
  - Structuring a data science project: folder organization, separating code, data, and outputs.
  - Example: Setting up a basic project structure in R and RStudio.

4.2 Writing Clean and Readable Code

- **Naming Conventions**
  - Using meaningful and consistent names for variables, functions, and files.
  - Example: Best practices in naming conventions in R (tidyverse style guide).
- **Commenting and Documentation**
  - Importance of comments and inline documentation.
  - Example: Writing using `roxygen2` in R for documenting functions.
- **Avoiding Magic Numbers and Hardcoding**

  - Use of constants and configuration files.
  - Example: Using constants in R.

4.3 Writing Efficient and Scalable Code

- **Vectorization**
  - Avoiding loops by using vectorized operations for efficiency.
  - Example: Implementing vectorized operations in R (base R, `dplyr`).
- **Memory Management**
  - Managing memory usage and avoiding memory leaks.
  - Example: Best practices for handling large datasets in R (using `data.table`).

4.4 (EXTRA)Testing and Validation

- **Writing Unit Tests**
  - Importance of testing: ensuring code correctness.
  - Example: Writing basic unit tests in Python (`unittest` or `pytest`) and R (`testthat`).
- **Data Validation**
  - Validating data inputs and outputs, ensuring data integrity.
  - Example: Implementing data validation checks in data processing scripts.

4.5 (EXTRA)Error Handling and Debugging

- **Error Handling Techniques**
  - Using tryCatch in R.

  - Writing meaningful error messages.
  - Example: Implementing error handling in a data processing script.

- **Debugging Tools**
  - Introduction to debugging tools: `browser` in R.
  - Example: Walkthrough of a debugging session in R.

4.6 (EXTRA)Code Reusability and Sharing

- **Creating Reusable Code**
  - Writing functions and libraries for reuse across projects.
  - Example: Creating a simple R package.

- **Sharing Code**
  - Sharing code with others: publishing packages, sharing notebooks.
  - Example: Publishing an R package on CRAN/GitHub.

Hands-on Practice

- **Refactoring Code (30min)**
  - First attempt: Refactoring a sample script to follow best practices (clean code, modularity, documentation).
  - Second attempt: try using a Large Language Model (LLM) to refactor.

- **Collaborative Exercise (40min)**
  - Simulating a collaborative workflow with Git: making and reviewing pull requests.
  - Groups of two or three
  - Re-use one of the repositories created in GitHub (Session 2) and populated by R code and output files (Session 3).

- Mutual reviews and change suggestions.
- Discussion, accepting/rejecting changes, and merge decision

- **Open discussion (10min)**
  - Addressing common challenges in applying best practices to real-world projects.

Part III

Data Science in R

5 Count data and seriation

5.1 Introduction to Count Data in Archaeology

- **Understanding Count Data**
  - Definition and significance of count data in archaeological contexts (e.g., artifact counts, feature frequencies).
  - Example: Overview of typical archaeological datasets involving count data.
- **Challenges in Analysing Count Data**
  - Issues with skewness, overdispersion, and zero inflation.
  - Example: Common problems encountered in archaeological count data analysis.

5.2 Basic Statistical Methods for Count Data

- **Poisson and Negative Binomial Distributions**
  - Introduction to Poisson distribution and its application to count data.
  - Example: Fitting a Poisson model using `glm` in R.
  - Introduction to the Negative Binomial distribution for overdispersed data.
  - Example: Fitting a Negative Binomial model using `MASS::glm.nb`.
- **Goodness-of-Fit Testing**
  - Assessing the fit of count data models.
  - Example: Performing a chi-square goodness-of-fit test in R.

5.3 Introduction to Seriation

- **What is Seriation?**
  - Overview of seriation techniques and their importance in archaeology for ordering artefacts or sites chronologically.
  - Example: Historical applications of seriation in archaeology.
- **Seriation Techniques**
  - Introduction to different seriation methods (e.g., frequency seriation, contextual seriation).
  - Example: Basic seriation using traditional methods.

5.4 Using the `tesselle` Package for Seriation

- **Introduction to `tesselle`**
  - Overview of the `tesselle` package and its tools for seriation.
  - Example: Installation and loading of `tesselle`.
- **Practical Seriation in R**
  - Performing seriation.
  - Example: Applying seriation to an archaeological dataset (e.g., pottery styles, stratigraphic data).
- **Visualizing Seriation Results**
  - Creating visual representations of seriation results.
  - Example: Plotting seriation outputs.

Hands-on Practice

- **Case Study: Seriation of Archaeological Artefacts**
  - Step-by-step walkthrough of a seriation analysis using count data.

  - Example: Seriation of pottery fragments or lithic tools using `tesselle`.

- **Q&A and Troubleshooting**
  - Addressing common issues in count data analysis and seriation in archaeological contexts.

6 Compositional data

6.1 Introduction to Compositional Data in Archaeology

- **Understanding Compositional Data**
  - Definition and examples of compositional data in archaeology (e.g., proportions of different materials, chemical compositions).
  - Example: Typical archaeological datasets that include compositional data.
- **Challenges in Analysing Compositional Data**
  - Issues with relative proportions and the "closed" nature of compositional data.
  - Example: Limitations of traditional statistical methods on compositional data.

6.2 Basic Concepts in Compositional Data Analysis

- **Overview of methods in Multivariate statistics**
- **Log-Ratio Transformations**
  - Introduction to log-ratio transformations (CLR, ILR, ALR) for compositional data.
  - Example: Applying a centred log-ratio (CLR) transformation in R (e.g. `nexus::transform_clr`).
- **Visualizing Compositional Data**
  - Techniques for visualizing compositional data (ternary plots, bar charts).
  - Example: Creating a ternary plot using the `isopleuros::ternary_plot` and `ggtern`.

6.3 Exploratory Data Analysis

- **Principal Component Analysis (PCA)**
  - Introduction to PCA tailored for compositional data.
  - Example: Performing PCA on compositional data using `tesselle::nexus::pca`.
  - Biplots and screeplots
  - Example: Plotting PCA results as a biplot and add visualisation aids using `tesselle::dimensio` functions.

- **Cluster Analysis**
  - Overview of clustering techniques for exploring groupings in compositional data.
  - Example: Applying hierarchical clustering on transformed compositional data using `tesselle`.
  - `ggplot2`: dendrograms with `ggraph`

- **Correspondence Analysis**
  - Correspondence Analysis for compositional data.
  - Example: Performing Correspondence Analysis using `tesselle::ca`.

Hands-on Practice

- **Case Study: Analysis of Compositional Data from Archaeological Sites**
  - Step-by-step walkthrough of an exploratory analysis of compositional data.
  - Example: Analysing chemical compositions of ceramics or soils using `tesselle`.

- **Q&A and Troubleshooting**
  - Addressing challenges in visualizing and analyzing compositional data in archaeological research.

Part IV

Databases

7 Databases (I)

7.1 Introduction to Databases in Archaeology

- **What is a Database?**
  - Definition and importance of databases in archaeology (data storage, retrieval, and management).
  - Example: Common uses of databases for managing archaeological data (e.g., artifacts, excavation records).
- **Types of Databases**
  - Overview of relational vs. non-relational databases.
  - Example: Advantages of relational databases for structured archaeological data.

7.2 Relational Database Concepts

- **Basic Concepts**
  - Introduction to tables, records, fields, primary and foreign keys.
  - Example: Organizing excavation data in relational tables (e.g., site locations, stratigraphy, finds).
- **Normalization**
  - Introduction to database normalization (avoiding redundancy, ensuring data integrity).
  - Example: Structuring artifact data into normalized tables (e.g., artifact type, material, condition).

- **Entity-Relationship (ER) Models**
  - Creating ER models to visualize relationships between archaeological datasets.
  - Example: Designing an ER diagram for an archaeological database (e.g., site, context, finds).

7.3 Introduction to SQL (Structured Query Language)

- **Basic SQL Commands**
  - Overview of SQL syntax and common commands (e.g., `CREATE`, `INSERT`, `SELECT`, `UPDATE`).
  - Example: Creating tables for archaeological data and inserting records.
- **Creating a Database with SQL**
  - Step-by-step process of building an archaeological database using SQL.
  - Example: Creating an artifact catalogue with fields such as artifact ID, type, material, and context.
- **Indexing and Keys**
  - Explanation of primary keys, foreign keys, and indexing for optimizing database performance.
  - Example: Defining keys to link excavation sites to finds in different tables.
- **Basic Querying with SQL**
  - Writing basic queries to retrieve data from a database.
  - Example: Using `SELECT` statements to extract information about finds or excavation layers.
- **Filtering and Sorting Data**
  - Using `WHERE`, `ORDER BY`, and `GROUP BY` clauses to filter and sort data.

  - Example: Querying artifacts by material type or sorting excavation records by date.

- **Joining Tables**
  - Using `JOIN` statements to combine related tables (e.g., linking artifact data with stratigraphy).
  - Example: Writing queries to retrieve artifacts based on their excavation context.

7.4 Database Tools for Archaeology

- **Accessing Databases from R**
  - Introduction to R packages (`DBI`, `RSQLite`, `RPostgres`) for interacting with databases.
  - Example: Connecting to an SQLite or PostgreSQL database from R for archaeological data analysis.

- **SQLite for small projects**
  - Introduction to SQLite as a lightweight database solution for archaeological projects.
  - Example: Creating a simple SQLite database for site data using R (RSQLite, e.g. https://cran.r-project.org/web/packages/RSQLite/vignettes/RSQLite.html) or Python.

- **PostgreSQL for larger projects**
  - Overview of PostgreSQL for larger archaeological datasets.
  - Example: Setting up a PostgreSQL database in R for managing excavation records.

- **Querying Databases in R**
  - Writing SQL queries in R and retrieving data for further analysis.
  - Example: Querying an excavation database from R and visualizing the results with `ggplot2`.

- **Data Wrangling and Cleaning**
  - Using R to clean and manipulate database data for analysis.
  - Example: Using `dplyr` in R to filter and transform queried data.

Hands-on Practice

- **Building a Small Archaeological Database**
  - Step-by-step walkthrough of creating a database schema for a hypothetical excavation project (using RSQLite).
  - Example: Creating tables for stratigraphy, finds, and context.
- **Inserting and Managing Data**
  - Practical examples of adding and managing archaeological data.
  - Example: Inserting excavation records into the database using SQL.
- **Q&A and Troubleshooting**
  - Addressing challenges in database design and SQL queries.

8 Databases (II)

8.1 Using GIS Databases for Archaeology

- **Introduction to Spatial Databases**
  - Overview of spatial databases and their use in archaeology (e.g., storing geospatial excavation data).
  - Example: Introduction to PostGIS for spatial data management in PostgreSQL.
- **Linking GIS Data to Databases**
  - Using databases to manage spatial data for archaeological analysis.
  - Example: Storing site coordinates and linking them to excavation records in a spatial database.
- **Querying Spatial Data**
  - Writing spatial queries to retrieve geospatial data from a database.
  - Example: Querying sites within a specific radius or extracting artifact distributions across layers.

Hands-on Practice

- **Querying and Analyzing Archaeological Data**
  - Walkthrough of writing SQL queries to extract meaningful insights from archaeological data.
  - Example: Retrieving and analyzing artifact distributions based on context and stratigraphy.
- **Combining Database and Spatial Data**
  - Practical examples of integrating database queries with GIS data for archaeo-

logical site analysis.

  - Example: Using a database to visualize the spatial distribution of artifacts across an excavation site.

- **Q&A and Troubleshooting**
  - Addressing challenges in querying and analyzing archaeological databases.

Part V

GIS

9 GIS (I)

(Basic concepts, installation and setting up, loading files)

Cool archive video about ARCInfo (1988): https://youtu.be/7xqNyUOIRCs?si=FulmlUVzaThGE9BU

(Dooley n.d.)

> Hands-on Practice

10 GIS (II)

(get a map image displaying data from shape and raster files)

> Hands-on Practice

References

"Automatically Generated Release Notes." n.d. *GitHub Docs*. Accessed October 11, 2024. https://docs.github.com/en/repositories/releasing-projects-on-github/automatically-generated-release-notes.

Baker, Monya. 2016. "1,500 Scientists Lift the Lid on Reproducibility." *Nature* 533 (7604): 452–54. https://doi.org/10.1038/533452a.

"Basic Syntax | Markdown Guide." n.d. Accessed October 11, 2024. https://www.markdownguide.org/basic-syntax/.

Cioara, Andrei. 2018. "How I Organize My GitHub Repositories." *Medium*. https://andreicioara.com/how-i-organize-my-github-repositories-ce877db2e8b6.

Community, The Turing Way. 2022. "The Turing Way: A Handbook for Reproducible, Ethical and Collaborative Research." Zenodo. https://doi.org/10.5281/ZENODO.3233853.

"Coursera | Degrees, Certificates, & Free Online Courses." n.d. Accessed October 8, 2024. https://www.coursera.org/.

"Created New Organization in GitHub and Zenodo Did Not Send a Request for Accessing It · Issue #1596 · Zenodo/Zenodo." n.d. *GitHub*. Accessed October 11, 2024. https://github.com/zenodo/zenodo/issues/1596.

"Creating GitHub Releases Automatically on Tags." 2024. https://jacobtomlinson.dev/posts/2024/creating-github-releases-automatically-on-tags/.

danijar. 2019. "Can I Arrange Repositories into Folders on Github?" Forum post. *Stack Overflow*. https://stackoverflow.com/q/11852982/6199967.

Dooley, Andy MacLachlan, Adam Dennett and Claire. n.d. *CASA0005 Geographic Information Systems and Science*. Accessed October 9, 2024. https://andrewmaclachlan.github.io/CASA0005repo/index.html.

"FAIR Principles." n.d. *GO FAIR*. Accessed October 7, 2024. https://www.go-fair.org/fair-principles/.

"Functional Documentation with Markdown and Version Control." 2017. *Leon Hassan*. https://blog.leonhassan.co.uk/functional-documentation-with-markdown-and-version-control/.

"Git - Gitglossary Documentation." n.d. Accessed October 11, 2024. https://git-scm.com/docs/gitglossary.

"Git Definitions and Terminology Cheat Sheet." n.d. Accessed October 11, 2024. https://www.pluralsight.com/resources/blog/cloud/git-terms-explained.

"GRN · German Reproducibility Network." n.d. Accessed October 7, 2024. https://reproducibilitynetwork.de/.

Iatropoulou, Katerina. n.d. "OpenAIRE." *OpenAIRE*. Accessed October 7, 2024. https://www.openaire.eu/.

"Introduction to Programming Languages." 2018. *GeeksforGeeks*. https://www.geeksforgeeks.org/introduction-to-programming-languages/.

"Issue a Doi with Zenodo." n.d. *Github for Collaborative Documentation*. Accessed October 11, 2024. https://cassgvp.github.io/github-for-collaborative-documentation/docs/tut/6-Zenodo-integration.html.

jimmy. 2022. "How to Organize GitHub Repositories." *Backrightup*. https://backrightup.com/blog/how-to-organize-github-repositories/.

"Learn R, Python & Data Science Online." n.d. Accessed October 8, 2024. https://www.datacamp.com.

"Markdown Guide." n.d. Accessed October 7, 2024. https://www.markdownguide.org/.

Marwick, Ben. 2017. "Open Science in Archaeology," January. https://doi.org/10.17605/OSF.IO/3D6XX.

"Module-5-Open-Research-Software-and-Open-Source/Content_development/Task_2.md at Master · OpenScienceMOOC/Module-5-Open-Research-Software-and-Open-Source." n.d. *GitHub*. Accessed October 11, 2024. https://github.com/OpenScienceMOOC/Module-5-Open-Research-Software-and-Open-Source/blob/master/content_development/Task_2.md.

National Academies of Sciences, Engineering, Policy and Global Affairs, Engineering Committee on Science, Board on Research Data Information, Division on Engineering and Physical and Sciences, Committee on Applied and Theoretical Statistics, Board on Mathematical Sciences Analytics, et al. 2019. "Understanding Reproducibility and Replicability." In *Reproducibility and Replicability in Science*. National Academies Press (US). https://www.ncbi.nlm.nih.gov/books/NBK547546/.

Obregon, Alexander. 2024. "What Is Markdown? Uses and Benefits Explained." *Medium*. https://medium.com/@AlexanderObregon/what-is-markdown-uses-and-benefits-explained-947300e1f955.

"Online Courses - Learn Anything, On Your Schedule | Udemy." n.d. Accessed October 8, 2024. https://www.udemy.com/.

R Core Team. 2024. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

"Referencing and Citing Content." n.d. *GitHub Docs*. Accessed October 11, 2024. https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content.

Signell, Rich. 2013. "How to Handle Releases of Markdown Document on Github." Forum post. *Stack Overflow*. https://stackoverflow.com/q/19727632/6199967.

"Spatial Without Compromise · QGIS Web Site." n.d. Accessed October 7, 2024. https://qgis.org/.

Suhail, Muhammad Ahmed. 2024. "Structuring and Organizing My Github: A Developer's Guide." *Medium*. https://medium.com/@muhammadahmedsuhail007/structuring-and-organizing-my-github-a-developers-guide-7353610f04fd.

"What Is Markdown? Syntax, Examples, Usage, Best Practices." 2021. https://www.knowledgehut.com/blog/web-development/what-is-markdown.

"What Is Programming? And How To Get Started." 2024. *Coursera*. https://www.coursera.

org/articles/what-is-programming.
"Zenodo - Research. Shared." n.d. Accessed October 11, 2024. https://help.zenodo.org/docs/profile/linking-accounts/.
Zestyclose-Low-6403. 2023. "How to Organize Repos Within an 'Organization'?" Reddit {Post}. *R/Github*. www.reddit.com/r/github/comments/188d324/how_to_organize_repos_within_an_org