NOAA Technical Memorandum NMFS-XXX-##

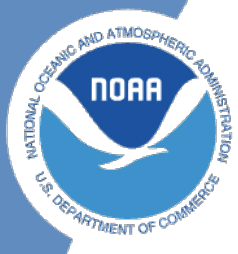
SignalFlowEEG Book

Ernie Pedapati

January 2023

U.S. DEPARTMENT OF COMMERCE

National Oceanic and Atmospheric Administration
National Marine Fisheries Service
Northwest Fisheries Science Center



NOAA FISHERIES

SignalFlowEEG Book

Ernie Pedapati[1]

1. Cincinnati Children's Hospital Medical Center, Neurobehavioral Research Laboratory

Table of contents

List of Figures

List of Tables

Overview

Overview

This EEG signal analysis program is designed to provide a comprehensive and user-friendly solution for processing and analyzing EEG data in the context of clinical trials and studies. It leverages the power of open-source libraries, such as MNE, to offer a robust and flexible framework for clinical EEG analysis.

Features

- **Data Preprocessing**: Perform standard EEG preprocessing tasks, including filtering, artifact detection and removal, and channel selection.
- **Event Detection and Segmentation**: Accurately identify and segment EEG events of interest, such as evoked responses, oscillatory activity, and sleep stages.
- **Signal Analysis**: Apply a wide range of signal processing techniques, including time-domain analysis, frequency-domain analysis, and time-frequency analysis.
- **Statistical Analysis**: Conduct statistical tests to assess the significance of EEG findings, including group comparisons and correlations with clinical outcomes.
- **Visualization**: Generate informative visualizations, such as topographic maps, time-series plots, and spectrograms, to facilitate data exploration and presentation.
- **Integration with Clinical Data**: Seamlessly integrate EEG data with other clinical measures, such as cognitive assessments, behavioral observations, and medical records, to enable multimodal analysis.
- **Scalable and Automated Workflows**: Develop and deploy scalable and automated workflows to handle large-scale clinical datasets efficiently.
- **Reproducibility and Collaboration**: Ensure the reproducibility of analyses and enable collaborative work through the use of version control and documented workflows.

Part I.

CORTICAL EXCITABILITY

1. Project Overview

1.1. Assessing Cortical Excitability Using Resting-State EEG

1.1.1. Introduction

Resting state EEG analysis can provide valuable insights into cortical excitability, which is often altered in neurodevelopmental disorders like Fragile X Syndrome (FXS). One powerful tool for assessing cortical excitability is the Fitting Oscillations & One-Over F (FOOOF) algorithm, which can decompose the resting state EEG power spectrum into its underlying oscillatory and aperiodic components.

In this analysis, we will use FOOOF to compare cortical excitability, as measured by the aperiodic component of the power spectrum, between a group of individuals with FXS and a matched control group. This will allow us to better understand the neurophysiological underpinnings of the altered sensory processing and hyperexcitability often observed in FXS.

1.1.2. Methods Overview

1. `p050_setup_environment`: Set up the analysis environment
2. `p100_load_data_catalog`: Organize datasets
3. `p150_preprocess_data`: Preprocess the data
4. `p200_extract_features`: Extract features
5. `p300_compare_features`: Compare features
6. `p400_visualize_results`: Visualize results

- Preprocess the resting state EEG data, including filtering, artifact rejection, and channel selection.
- Apply the FOOOF algorithm to the preprocessed, source localized data to extract the aperiodic and oscillatory components of the power spectrum for each participant.

- Compare the aperiodic component between the FXS and control groups using appropriate statistical tests, such as t-tests or non-parametric tests, depending on the distribution of the data.

2. Setting up the Analysis Environment

2.1. Cortical Excitability: Setting up the Analysis Environment

2.1.1. Overview

This tutorial will walk you through setting up the analysis environment for this project.

Tasks to Complete: 1. Install: Python 2. Install: MNE 3. Install: SignalFlowEeg 4. Install: FOOOF

We have outlined these steps in a single code file `p050_setup_environment`.

2.1.2. Install Python

There are many ways to install Python which makes the process even more challenging. Since we will be using MNE, I recommend for beginners installing Python per the MNE teams instructions.

For advanced users or those who want to learn how to setup a custom python environment, we have included our approach using `pyenv`. This guide is optimized for Mac/Unix and Windows Subsystem for Linux.

2.1.3. Setting up a Python Environment with `pyenv`

1. **Homebrew**: The easiest way to install `pyenv` is via Homebrew. Make sure to follow the instructions following install to have the `brew` command available.
2. **Pyenv**: Follow the instructions to install `pyenv` via Homebrew.
3. **Pyenv-Virtualenv**: We will be using `pyenv-virtualenv` to create isolated python environments. Install via Homebrew.

4. **Python**: Pyenv can install and manage multiple versions of Python. We are following recommendations by the MNE team to start with a conda distribution with Python 3.11. MNE has a preference for using conda environments.
5. **MNE**: MNE is a Python package for processing and analyzing MEG and EEG data. It is a powerful tool for analyzing electrophysiological data.
6. **SignalFlowEeg**: SignalFlowEeg is a Python package for processing and analyzing MEG and EEG data. It is a powerful tool for analyzing electrophysiological data.
7. **FOOOF**: FOOOF is a Python package for analyzing the power spectrum of electrophysiological data. It is a powerful tool for analyzing electrophysiological data.

2.1.4. Setting up SignalFlowEeg

Use the following command to install `signalfloweeg` directly from its GitHub repository. This command will clone the repository and install the package in editable mode, which means you can update the package by pulling changes from the repository.

```
pip install -e git+https://github.com/cincibrainlab/signalfloweeg_py.git#egg=signalfloweeg
```

Here's what each part of the command does: - `pip`: This specifies the `pip` executable within the Conda environment you created. - `install -e`: The `-e` flag installs the package in "editable" mode. - `git+https://github.com/cincibrainlab/signalfloweeg_py.git`: This is the Git URL of the `signalfloweeg` repository. - `#egg=signalfloweeg`: This tells `pip` the name of the package to install. - `--src /home/username/src/signalfloweeg_dev`: This specifies the source directory where the package will be cloned.

Import the Package:

In your Python script or Jupyter notebook, import `signalfloweeg`:

```
import signalfloweeg
```

Citations: [1] https://github.com/cincibrainlab/signalfloweeg_py.git [2] https://docs.python.org/3/using/w
[3] https://phoenixnap.com/kb/how-to-install-python-3-windows [4] https://stackoverflow.com/questio
to-install-python-any-version-in-windows-when-youve-no-admin-privileges [5]
https://builtin.com/software-engineering-perspectives/how-to-install-python-on-
windows

```
brew install pyenv
```

1. `p050_setup_environment`: Set up the analysis environment
2. `p100_load_data_catalog`: Retrieve file list and associated metadata

- Preprocess the resting state EEG data, including filtering, artifact rejection, and channel selection.
- Apply the FOOOF algorithm to the preprocessed, source localized data to extract the aperiodic and oscillatory components of the power spectrum for each participant.
- Compare the aperiodic component between the FXS and control groups using appropriate statistical tests, such as t-tests or non-parametric tests, depending on the distribution of the data.

3. Catalog Class

3.1. Organizing Data with Catalog Class

Effective dataset organization is critical to maintain a productive analysis pipeline. Imagine if you could call specific data files or data directories from a function call anywhere in your analysis code? What if you could store this "catalog" in a simple, human-editable file, which can be accessed from a local folder or retrieved from a web link, without a complex database systems.

The `Catalog` class in the `signalfloweeg` module does exactly this!

3.2. An Example Catalog File

Let's look at an example of a Catalog file named `example_datasets.yaml` which can be edited in a text editor:

```
# Catalog Name: Example
# Catalog Date: 1/3/2024
# Comments: assr = auditory steady state recording
# Template: "dataset label": "path" or "file"

demo_rest_state: "/srv/RAWDATA/exampledata/Resting/128_Rest_EyesOpen_D1004.set"
demo_auditory_chirp: "/srv/RAWDATA/exampledata/Chirp/128_Chirp_D0657_DIN8.set"
demo_auditory_assr: "/srv/RAWDATA/exampledata/SteadyState/"
```

You can make as many catalog files as you need - maybe one for each project or server. You could also use a single catalog file to store different stages or versions of a project.

3.2.1. What is the YAML text format?

YAML (YAML Ain't Markup Language) is a human-readable data format that is stored as a text file. It's a way to store and organize information in a structured and easy-to-understand way, similar to how you might organize information in a spreadsheet or a database.

3.3. How to use the Catalog Class

Let's say that you wanted to tally the number of epochs across two different auditory evoked datasets. Using the example above, you could do something like this:

```
from signalfloweeg.catalog import Catalog

# Initialize the catalog
catalog = Catalog("example_datasets.yaml")

# Get the location of a dataset
data_folder1 = catalog.get_location("demo_auditory_chirp")
data_folder2 = catalog.get_location("demo_auditory_assr")

# Now you can use these locations to load your data and perform your analysis.
```

The method calls will return the file system path to the dataset. You can then use this path within your analysis code.

3.4. Usefulness of Abstracting the Data Location

By using dataset labels, you can easily change the underlying location of the dataset without having to modifications to your code. This is useful when developing code across different systems or sharing catalogs with others.

3.5. Creating a Catalog File

Let's return to our example catalog file:

```
# Catalog Name: Example
# Catalog Date: 1/3/2024
# Comments: assr = auditory steady state recording
# Template: "dataset label": "path" or "file"

demo_rest_state: "/srv/RAWDATA/exampledata/Resting/128_Rest_EyesOpen_D1004.set"
demo_auditory_chirp: "/srv/RAWDATA/exampledata/Chirp/128_Chirp_D0657_DIN8.set"
demo_auditory_assr: "/srv/RAWDATA/exampledata/SteadyState/"
```

Let's break down the structure and usage of this template:

1. **Catalog Name and Date**: The first two lines provide metadata about the catalog, including the name and the date it was created. This information can be helpful for keeping track of different versions or iterations of your data catalog.
2. **Comments**: The third line allows you to add any relevant comments or notes about the datasets included in this catalog. In this example, the comment indicates that the "assr" datasets are related to auditory steady-state recordings.
3. **Template**: The template section is where you define the actual dataset information. Each dataset is represented by a key-value pair, where the "key" is the dataset label (e.g., `demo_rest_state`, `demo_auditory_chirp`, `demo_auditory_assr`) and the "value" is the file path or location of the dataset.
    - **Dataset Label**: The dataset label should be a descriptive and unique identifier for each dataset. These labels will be used to reference and access the datasets within your code.
    - **File Path or Location**: The value should be the full file path or location of the dataset. In this example, the paths are provided as absolute paths on a server (`/srv/RAWDATA/exampledata/...`), but they can also be relative paths or even file names if the datasets are located in a consistent directory structure.

To use this YAML template with the `Catalog` class, you would typically save the content as a text file (e.g., `example_catalog.yml`).

3.6. Loading the Catalog File

To load the catalog file, you would use the `Catalog` class as shown below:

```
from signalfloweeg.catalog import Catalog

catalog = Catalog(catalog_file='example_catalog.yml')
```

Or if you have stored the catalog file on a web server, you can load it from a URL:

```
catalog = Catalog(catalog_url='https://example.com/example_catalog.yml')
```

3.7. Exploring the Catalog

Once you have a `Catalog` instance, you can use its various methods to interact with the dataset information.

Here's a concise table summarizing the key functions of the `Catalog` class:

Function	Description
`load_catalog`	Loads the catalog from a local or web-hosted YAML file.
`get_location`	Retrieves the file or folder path for a specific dataset.
`get_dataset_type`	Determines whether a dataset is a file or a folder.
`get_associated_fdt`	Checks for an associated FDT file for a `.SET` dataset.
`get_filelist`	Retrieves a list of files for a dataset, with optional filtering.
`summarize_filelist`	Provides a visual summary of the files associated with a dataset.
`create_yaml_template`	Generates a YAML template file with sample dataset information.

This table provides a quick reference for the main functions available in the `Catalog` class, allowing you to easily identify the appropriate method for your data management needs.

3.8. Specific Method Details

3.8.1. Retrieving Dataset Locations

The `get_location` method allows you to retrieve the file or folder path associated with a specific dataset:

```
# Get the location of a dataset
dataset_location = catalog.get_location("demo_rest_state")
print(dataset_location)
```

3.8.2. Determining Dataset Types

The `get_dataset_type` method can be used to determine whether a dataset is a file or a folder:

```
# Check the type of a dataset
dataset_type = catalog.get_dataset_type("proj_ketamine")
print(dataset_type)
```

3.8.3. Checking for Associated FDT Files

If you have a dataset with a `.SET` extension, you can use the `get_associated_fdt_file` method to check if there is an associated FDT file:

```
# Check for an associated FDT file
fdt_file_path, fdt_file_present = catalog.get_associated_fdt_file("demo_rest_state")
if fdt_file_present:
    print(f"Associated FDT file: {fdt_file_path}")
else:
    print("No associated FDT file found.")
```

3.8.4. Retrieving File Lists

The `get_filelist` method allows you to retrieve a list of files associated with a specific dataset, optionally filtered by file extension, subfolder search, and filename regex:

```
# Retrieve the files for a dataset, filtering by extension
dataset_files = catalog.get_filelist("demo_rest_state", extension=".set")
for file_info in dataset_files:
    print(f"Folder path: {file_info['folder_path']}")
    print(f"File name: {file_info['file_name']}")
    print(f"Extension: {file_info['extension']}")
    print()
```

3.8.5. Summarizing File Lists

The `summarize_filelist` method provides a visual summary of the files associated with a dataset, including the total number of files, total size, and file type breakdown:

```
# Summarize the files for a dataset
catalog.summarize_filelist()
```

3.8.6. Creating a YAML Template

The `create_yaml_template` method allows you to generate a YAML template file with sample dataset names and file paths, which you can then customize with your own dataset information:

```
# Create a YAML template file
catalog.create_yaml_template()
```

3.9. Conclusion

The `Catalog` class provides a comprehensive set of tools for managing your EEG data files. By using this class, you can streamline your data loading process, ensure consistency across your analysis workflows, and improve the overall reproducibility of your research.

4. Reproducible Environment with Docker

4.1. "Operating System in a Box"

Using a Docker container is a compelling option for creating a reproducible analysis environment. Docker containers provide a self-contained, portable "operating system in a box" that can be deployed on any machine with Docker installed, whether it's your local computer or a remote server. By using a consistent Linux-based Docker container, your setup scripts can be used across different platforms, ensuring your analysis environment is consistent and reproducible.

Working within a container has become much easier with the introduction of Visual Studio Code. VS Code is a free and open-source editor that supports development in most of the popular programming languages including Python. It also has a built-in terminal and can be used to run Docker containers.

4.2. Graphical Desktop in Docker

Docker containers can run a full graphical desktop interface that can be easily accessed via a web browser. This is particularly useful for running Jupyter notebooks and other web-based applications. Several well-supported graphical containers are available for neuroscience, such as `signalflow-stacks`, `neurodesk`, and 'Jupyter Stacks'. More complex docker containers can run a full graphical desktop interface that can be easily accessed via a web browser. This is particularly useful for running Jupyter notebooks and other web-based applications.

Several well-supported graphical containers are available for neuroscience:

1. `signalflow-stacks` MATLAB, Python, and R with Jupyter notebook support.
2. `neurodesk` A web-based desktop for neuroscience.
3. 'Jupyter Stacks' A collection of Docker images for Jupyter notebook, R, and Python.

For more information on Docker, including installation instructions, visit the Docker documentation. Mac users can also refer to the Orbstack documentation.

4.3. Custom SignalFlowEeg Docker Container

We have created a custom Docker container to simplify the setup process and ensure a smooth workflow. This Docker container is built upon the `quay.io/jupyter/datascience-notebook` image, a widely-used base for data science and scientific computing. We've carefully curated and added the specific packages and libraries required for our signal analysis work, eliminating the need for you to worry about the intricate details of setting up your development environment.

In addition to the Python environment, we've also included several essential packages:

1. **MNE**: This powerful Python package for processing and analyzing MEG and EEG data is a crucial tool in our arsenal.
2. **SignalFlowEeg**: Our custom-developed Python package for signal analysis is pre-installed, allowing you to seamlessly integrate it into your workflow.
3. **FOOOF**: The FOOOF package, which enables us to analyze the power spectrum of our electrophysiological data, is another essential component we've included.
4. **Quarto**: To streamline the creation of dynamic reports and visualizations, we've set up Quarto, a versatile open-source tool, within the container.
5. **Visual Studio Code**: For a familiar and efficient coding experience, we've installed Visual Studio Code and configured it to run with the `--no-sandbox` flag, addressing any potential issues that may arise in a containerized environment.

By using this custom Docker container, you can focus on your research without the hassle of setting up a complex development environment from scratch. Everything is pre-installed and configured, ensuring a consistent and reproducible analysis environment.

To get started, simply follow the instructions in the `p040_run_docker_ubuntu22.sh` script. If you have any questions or encounter any issues, don't hesitate to reach out to the team. We're here to support you throughout this process and ensure you have a smooth and productive experience.

4.4. Example: Running a Jupyter Notebook in a Docker Container

4.4.1. Understanding Docker Run Command Options

Docker is a powerful platform for developing, shipping, and running applications inside containers. The `docker run` command below can create a new reproducible analysis workspace.

```
docker run \                                        # Run a new container
  -v /Users/ernie/Documents/GitHub/cbl_spectparm_fxsrest:/srv \  # Mount a volume from host
  -e DISPLAY="host.docker.internal:0" \             # Set DISPLAY environment variable for X11 fo
  -e GRANT_SUDO=yes \                               # Allow sudo command within the container
  --user root \                                     # Run as root user
  -it \                                             # Interactive terminal
  --rm \                                            # Remove container on exit
  --name dev-ubuntu \                               # Name the container 'dev-ubuntu'
  quay.io/jupyter/datascience-notebook:latest \  # Use the 'datascience-notebook' image fro
  /bin/bash                                         # Start a Bash shell
```

Let's break down the different parts of a complex `docker run` command to understand what each option does.

4.4.1.1. The Base Command

```
docker run
```

This is the base command that tells Docker to run a new container.

4.4.1.2. Volume Mounting

```
-v /Users/ernie/Documents/GitHub/cbl_spectparm_fxsrest:/srv
```

The `-v` flag mounts a volume. It maps a directory from the host (`/Users/ernie/Documents/GitHub/cbl_spectparm_fxsrest`) into the container (`/srv`). This allows for data to be persisted and shared between the host and the container. In this case, it is pointing to my analysis code repository.

4.4.1.3. Environment Variables

```
-e DISPLAY="host.docker.internal:0"
-e GRANT_SUDO=yes
```

The `-e` flag sets environment variables inside the container. Here, `DISPLAY` is set to `host.docker.internal:0`, which is typically used for X11 forwarding to allow GUI applications to display on the host (see below).

`GRANT_SUDO` is set to `yes` to allow the `sudo` command within the container. This is a specific option for the `quay.io/jupyter/datascience-notebook` image which allows the default user to be given admin privileges.

4.4.1.4. User Configuration

```
--user root
```

The `--user` option specifies which user the container should run as. In this case, it's set to `root`, to temporarly allow the container at start up to assign admin privledges to the default user as specified in the documentation.

4.4.1.5. Interactive Terminal

```
-it
```

The `-it` flags are shorthand for `--interactive` and `--tty`. This combination allows you to interact with the container via a terminal interface.

4.4.1.6. Container Cleanup

```
--rm
```

The `--rm` flag tells Docker to automatically remove the container when it exits. This is useful for not leaving behind any stopped containers, keeping your system clean.

4.4.1.7. Container Naming

```
--name dev-ubuntu
```

The `--name` option assigns a name to the container, making it easier to reference. In this case, the container is named `dev-ubuntu`.

4.4.1.8. Docker Image

```
quay.io/jupyter/datascience-notebook:latest
```

This specifies the Docker image to use for the container. The image is hosted on `quay.io`, and it's the `datascience-notebook` repository with the `latest` tag, indicating the most recent version.

4.4.1.9. Command Override

```
/bin/bash
```

Finally, `/bin/bash` is the command that will be run inside the container once it starts. This overrides the default command specified in the Docker image, and in this case, it will start a Bash shell.

4.4.1.10. Full Command Breakdown

Here's the full `docker run` command with each part explained:

```
docker run \                                    # Run a new container
  -v /Users/ernie/Documents/GitHub/cbl_spectparm_fxsrest:/srv \  # Mount a volume from host
  -e DISPLAY="host.docker.internal:0" \         # Set DISPLAY environment variable for X11 fo
  -e GRANT_SUDO=yes \                           # Allow sudo command within the container
  --user root \                                 # Run as root user
  -it \                                         # Interactive terminal
```

```
  --rm \                                          # Remove container on exit
  --name dev-ubuntu \                             # Name the container 'dev-ubuntu'
  quay.io/jupyter/datascience-notebook:latest \  # Use the 'datascience-notebook' image fro
  /bin/bash                                       # Start a Bash shell
```

Understanding each part of the `docker run` command is crucial for managing Docker containers effectively. With this knowledge, you can customize your Docker containers to fit your development needs perfectly.

4.5. Using an X-server on your Local Machine

A common use case for Docker is to run a graphical application with an X-server. This is done by running the Docker container with the `-e DISPLAY` flag. This flag tells the container to use the X-server on the host machine. This allows the container to display the graphical applications on the host machine.

If you have an Mac, you can use the XQuartz X11 server. XQuartz is a free X11 server that allows the Docker container to display the graphical interface on the Mac. Before troubleshooting, make sure your X11 preferences are set to allow connections from network clients and turn off "Authenticate connections". On my system I only had to make one addition to my `docker run` command to run GUI applications and figures from my container:

```
  -e DISPLAY="host.docker.internal:0" \
```

4.6. Frequently Asked Questions

4.6.1. How do I access my data in my container?

This is a common question. The best way to access your data in your container is to use a volume mount. A volume mount is a mechanism to allow a container to access a directory on the host machine. This is done by specifying a `-v` flag in the `docker run` command. Any data saved within the container on the mounted volume will be saved on the host machine even after the container is stopped.

4. Reproducible Environment with Docker

```
# Start a new Ubuntu 22.04 Docker container in interactive mode
# Local folder is mapped to a folder within the container
docker run -v /local_folder:/container_folder -it --name dev-ubuntu ubuntu:22.04 /bin/bash`
```

Part II.

MANUSCRIPT REPOSITORY

5. Manuscript Repository

5.1. Organizing a Comprehensive Manuscript Repository

When working on multiple scientific research projects, managing the various components of your manuscripts can quickly become a complex task. From the manuscript text itself to the associated code, datasets, figures, and submission materials, keeping everything organized and version-controlled is crucial for efficient collaboration and successful publication.

In this chapter, we'll explore a robust folder structure that organizes your manuscript repository around the individual research projects, making it easier to navigate and manage your scientific work.

5.2. The Manuscript-Centric Repository Structure

The proposed manuscript repository structure consists of a top-level folder, `Manuscript_Repository/`, which houses individual folders for each of your research manuscripts. This manuscript-centric approach ensures that all the necessary files and materials for a given project are kept together, providing a clear and intuitive organization.

Within each manuscript folder, you'll find the following subfolders:

1. **v[version number]/**:
    - This version-specific subfolder contains all the files related to a particular iteration of the manuscript, including the manuscript text, code, data, figures, tables, and submission materials.
    - The version number allows you to easily track and manage the evolution of your manuscript over time.
2. **Code/**:

- This folder stores the scripts, notebooks, and other code files associated with the research and analysis for the manuscript.

3. **Data/**:
    - The raw and processed data files used in the manuscript are organized in the `raw/` and `processed/` subfolders, respectively.
4. **Figures/**:
    - All the figures, images, and visualizations referenced in the manuscript are kept in this folder.
5. **Tables/**:
    - Tabular data, such as Excel or CSV files, that are included in the manuscript are stored here.
6. **Submission/**:
    - This folder contains any files related to the manuscript submission process, such as the cover letter, data usage agreements, or author statements.

By adopting this manuscript-centric folder structure, you can easily navigate and manage your research projects, ensuring that all the necessary components for a given manuscript are easily accessible and organized.

6. Suggested Optional Folders

These are optional folders that you can incorporate into your manuscript repository, allowing you to tailor the structure to your specific needs and research workflow. The number and organization of these folders are based on the collective experience of managing multiple scientific manuscripts.

For example, if you're not working on multiple manuscripts concurrently, you may not need the top-level "Manuscripts/" folder and can instead organize your files directly within the repository. Similarly, if you don't have a significant number of supporting materials, you may not require folders like "Ideas/", "Meetings/", or "Presentations/". The key is to tailor the folder structure to your specific requirements and standard names, ensuring that your manuscript repository remains well-organized, comprehensive, and easy to navigate.

1. **Ideas/**: This folder captures the initial research ideas, project proposals, and early brainstorming materials that eventually lead to the manuscript.
2. **Literature/**: This folder contains relevant research papers, literature reviews, and other sources that inform the manuscript.
3. **Manuscripts/**: The core of the repository, this folder houses the individual manuscript projects, each with its own version-specific subfolders.
4. **Meetings/**: This folder stores notes, agendas, and action items from meetings related to the manuscript development.
5. **Drafts/**: Earlier versions of the manuscript drafts are kept in this folder, allowing you to track the evolution of the writing.
6. **Reviews/**: Peer review comments, editor feedback, and response materials are organized in this folder.
7. **Presentations/**: Any presentations (e.g., conference talks, lab meetings) related to the research are stored here.
8. **Protocols/**: This folder holds detailed experimental protocols, data collection methods, and other documentation related to the research.

9. **Licenses/**: Data use agreements, material transfer forms, and other relevant licenses are kept in this folder.
10. **Templates/**: Standardized document templates, such as manuscript formatting and cover letter templates, are stored here.
11. **Archive/**: As the manuscript progresses through the publication process, older versions, rejected drafts, and other materials no longer actively used are moved to the archive folder.

6.1. Leveraging Git for Manuscript Versioning and Collaboration

As a researcher, you can utilize version control systems like Git to manage the evolution of your scientific manuscripts. Even if you're new to using code repositories, Git can provide valuable benefits for your manuscript workflow.

6.1.1. Understanding Git's Capabilities

Git is a distributed version control system that allows you to track changes to your files, collaborate with others, and maintain a comprehensive history of your project. Its features can be highly beneficial for managing your manuscript repository.

6.1.2. Getting Started with Git

To use Git for your manuscripts, you'll first need to set up a Git repository in your project folder. This is as simple as running the `git init` command in your manuscript directory.

6.1.3. Tracking Changes and Revisions

With Git, you can easily track changes to your manuscript files. Use `git add` to stage your changes, and `git commit` to save snapshots of your work with descriptive messages.

6.1.4. Collaborating with Git

Git facilitates collaboration by allowing you to share your repository with team members. They can then clone the repository, make changes, and push their updates, which Git will merge seamlessly.

6.1.5. Essential Git Commands

As you continue using Git, familiarize yourself with commands like `git status`, `git log`, `git pull`, and `git push` to streamline your manuscript versioning and collaboration workflows.

By incorporating Git into your manuscript management process, you can benefit from increased version control, improved collaboration, and better traceability of your research work. Don't be intimidated – the basic Git concepts are easy to learn and can have a significant positive impact on how you manage your scientific manuscripts.

6.2. Incorporating Git LFS

To effectively manage large files, such as data, figures, and other binary assets, the repository utilizes Git LFS (Large File Storage). This allows you to seamlessly track and version control these large files alongside the manuscript text and other supporting materials.

By initializing Git LFS in the repository and specifying the file types to be managed by it, you can ensure that your manuscript project remains lightweight and performant, while still maintaining a comprehensive version history and collaboration capabilities.

Git LFS is separate software from Git. Prior to running the commands below first visit Git LFS and follow the install instructions.

The process of incorporating Git LFS into your manuscript repository involves the following steps:

1. **Initialize Git LFS**: In the top-level `Manuscript_Repository/` folder, run the command `git lfs install` to set up the Git LFS extension in your local repository.

2. **Track Large File Types**: Decide which file types in your repository should be managed by Git LFS, such as `.csv`, `.rdata`, `.png`, `.tiff`, and `.eps`. Add these file types to the Git LFS tracking using the command `git lfs track "*.csv" "*.rdata" "*.png" "*.tiff" "*.eps". git lfs track "*.ai" "*.doc" "*.docx"`
3. **Commit the .gitattributes File**: Git LFS uses a `.gitattributes` file to track the file types you specified. Commit this file to your repository with `git add .gitattributes` and `git commit -m "Initialize Git LFS tracking"`.
4. **Add Large Files to the Repository**: As you add large files to your repository, Git LFS will automatically manage them. For example, when adding a large data file, use `git add Data/Experiment1/raw/participant_data.csv` and `git commit -m "Add raw experiment data"`.
5. **Manage Large File Storage**: Git LFS provides commands to manage the storage of large files, such as `git lfs env` to view the current usage, and `git lfs push` and `git lfs pull` to manually upload and download large files.
6. **Update the Folder Structure**: In your existing folder structure, replace the references to large files (e.g., in the `Data/`, `Figures/`, and `Tables/` folders) with placeholder files or symlinks that point to the actual large files managed by Git LFS.

By adopting this manuscript repository structure with Git LFS, you can streamline your research documentation, improve collaboration, and ensure the long-term preservation and accessibility of your scientific work.

Part III.

ODDS AND ENDS

7. Odds and Ends

7.1. Odds and Ends

7.2. Streamlining Analytical Projects with Organized File Names

Effectively managing files for complex analyses requires a thoughtful naming convention. A well-structured naming system helps bring order to potentially chaotic situations, enabling team members to locate, comprehend, and collaborate on different aspects of the project. Let's examine a powerful file naming pattern that can streamline your analytical workflows:

```
sec[section_number][section_name]_step[step_number][step_description].qmd
```

Now, let's analyze the components of this naming pattern:

7.3. The Components of an Efficient File Naming Convention

7.3.1. Section Identifier: sec

The 'sec' prefix serves as a constant reminder that the file belongs to a larger section within the project. It establishes the file's location in the hierarchy and prepares the groundwork for further identifying details.

7.3.2. Section Number: [section_number]

Following the 'sec' prefix, [section_number] represents the numerical identifier of the section. Typically zero-padded (e.g., 01, 02, 03), this number ensures correct sorting in file systems that arrange items lexicographically. Zero-padding becomes essential when there are more than nine sections, preventing the tenth section from appearing before the second one.

7.3.3. Section Name: [section_name]

After the section number, [section_name] provides a succinct yet descriptive label for the section. This could refer to a broader topic or a specific theme covered by the section. For instance, 'data_preparation' or 'model_evaluation' instantly conveys the subject matter of the section without opening the file.

7.3.4. Step Identifier: step

The 'step' prefix functions similarly to 'sec', indicating that the file pertains to a specific step within the section. It signals to the reader that the file focuses on a particular task or subtopic.

7.3.5. Step Number: [step_number]

Like [section_number], [step_number] is zero-padded to preserve proper sorting. This number denotes the file's sequence within the section, offering a clear arrangement of tasks or reading order.

7.3.6. Step Description: [step_description]

Lastly, [step_description] is a placeholder for the specific action or topic addressed in the step. This part of the file name should be as informative as possible to summarize the file's contents at a glance. Examples include 'import_datasets' or 'train_regression_model'.

7.4. Example Application

Suppose you're working on a project involving multiple sections, each with several steps. Here's how your files might appear using this naming convention:

```
sec01_data_collection_step01_download_data.qmd
sec01_data_collection_step02_clean_data.qmd
sec02_feature_engineering_step01_select_features.qmd
sec02_feature_engineering_step02_transform_variables.qmd
```

In this illustration, it's evident that the first two files belong to the data collection section and deal with downloading and cleaning data, respectively. The subsequent two files are part of the feature engineering section, focusing on selecting features and transforming variables.

7.5. Conclusion

Implementing a structured naming convention like `sec[section_number][section_name]_step[step_number][step_description].qmd` can greatly simplify the management of complex analysis projects. By employing this pattern, you ensure that your files remain organized and self-explanatory, facilitating a smoother and more productive analytical process.