modality XPLR Documentation

biomodal Ltd.

Feb 27, 2026

CONTENTS

1 User documentation contents **2**

1.1 Getting started . . . 2
1.1.1 Video: Install modality XPLR . . . 2
1.1.2 System requirements . . . 2
1.2 Installation of modality XPLR . . . 4
1.2.1 Install the latest version in a new virtual environment on macOS or Linux . . . 4
1.3 Authenticate modality XPLR . . . 8
1.4 Core Workflow . . . 9
1.4.1 Video: Running the modality XPLR Core Workflow . . . 10
1.4.2 Requirements and prerequisites . . . 10
1.4.3 Core Workflow configuration . . . 10
1.4.4 Running the Core Workflow . . . 10
1.5 Interpreting Core Workflow outputs . . . 11
1.5.1 Video: Reviewing Core Workflow results . . . 11
1.5.2 Output directory structure . . . 11
1.5.3 Reviewing results . . . 12
1.6 Advanced analysis workflow . . . 13
1.6.1 Download a test dataset . . . 14
1.6.2 Advanced configuration file setup . . . 14
1.6.3 Core Workflow overview: step-by-step . . . 15
1.6.4 Advanced customisation and extension . . . 30
1.6.5 Script completion and monitoring . . . 31
1.7 Toolkit . . . 32
1.8 Preparing for analysis . . . 32
1.8.1 Input file types . . . 32
1.8.2 Preparing annotation files . . . 35
1.8.3 Top up sequencing & aggregating samples . . . 37
1.8.4 Saving modality XPLR output files . . . 39
1.9 Handling Zarr stores . . . 39
1.9.1 Overview . . . 39
1.9.2 Subcommands . . . 39
1.9.3 Exporting Zarr stores to community-supported file types . . . 43
1.10 Prepare annotated regions . . . 45
1.10.1 CpG-informed segmentation for region generation . . . 45
1.10.2 Analysis examples . . . 47
1.11 Biological QC . . . 48
1.11.1 Quality control of methylation data . . . 48
1.11.2 Analysis examples . . . 50
1.12 Calling differentially methylated regions (DMRs) . . . 51
1.12.1 Method used for calling DMRs . . . 51
1.12.2 Typical DMR workflow . . . 52
1.12.3 Calling DMRs . . . 56
1.12.4 Outputs . . . 58
1.12.5 Analysis examples . . . 59

1.13 Plotting differentially methylated regions (DMRs) . . . 59
1.13.1 Inputs . . . 59
1.13.2 Command . . . 60
1.13.3 Outputs . . . 60
1.13.4 Analysis examples . . . 61
1.14 Generate genomic track plots . . . 61
1.14.1 Inputs . . . 62
1.14.2 Command . . . 66
1.14.3 Outputs . . . 67
1.14.4 Analysis examples . . . 68
1.15 Extracting methylation statistics . . . 68
1.15.1 Inputs . . . 69
1.15.2 Commands . . . 69
1.15.3 Outputs . . . 72
1.15.4 Analysis examples . . . 73
1.16 Audit trail . . . 74
1.16.1 Terminal feedback . . . 74
1.16.2 Logging and provenance tracking . . . 74
1.16.3 Setting verbosity levels . . . 75
1.17 Example workflow . . . 76
1.17.1 Overview of the workflow . . . 76
1.17.2 1. Assess methylation data quality . . . 77
1.17.3 2. Checking CpG context depth across the regions of interest . . . 78
1.17.4 3. Call differentially methylated regions (DMRs) in late stage disease . . . 83
1.17.5 4. Filtering DMRs to identify statistically significant changes . . . 87
1.17.6 5. Use biological priors to call DMRs in early stage disease . . . 87
1.17.7 6. Visualisation of key early-stage 5hmC DMRs . . . 89
1.17.8 7. Place methylation changes (DMRs) into genomic context . . . 90
1.17.9 8. Generate feature sets for building classifiers . . . 93
1.17.10 Conclusion . . . 94
1.18 FAQs and troubleshooting . . . 95
1.18.1 Frequently Asked Questions (FAQs) . . . 95
1.18.2 General questions . . . 95
1.18.3 Installation and setup . . . 96
1.18.4 Input files and formats . . . 96
1.18.5 Running Analyses . . . 98
1.18.6 Outputs and visualisation . . . 100
1.18.7 Error handling . . . 102
1.18.8 Debugging . . . 102
1.18.9 biomodal Software License . . . 102

2 How to use this guide **105**
2.1 Navigation + searching . . . 105
2.2 Support links . . . 105

3 Release notes **106**

Methylation data has diverse applications from basic science to translational research and test development. The modality XPLR analysis software is a fast and scalable command line interface (CLI) toolkit, for the analysis of methylation data that is derived from 5-base (duet multiomics solution +modC) and 6-base (duet multiomics solution evoC) genomes. The software leverages multi-core, out-of-memory processing to enable efficient computation on a standard laptop or HPC; allowing users to work on large datasets with ease.

The modality XPLR analysis software includes tools for exploratory analyses (e.g. computation and plotting of methylation statistics and differentially methylated regions (DMRs)), as well as supporting downstream analysis (e.g. feature extraction and export functions). Designed for efficiency and ease of use, it enables you to rapidly transition from raw data to actionable insights and publication-ready visualisations.

The core data file used by modality XPLR is the Zarr store, which is an array format that includes methylation and coordinate data, produced by the (duet Software).

The software also allows export to community-supported file types for analysis with other informatics tools.

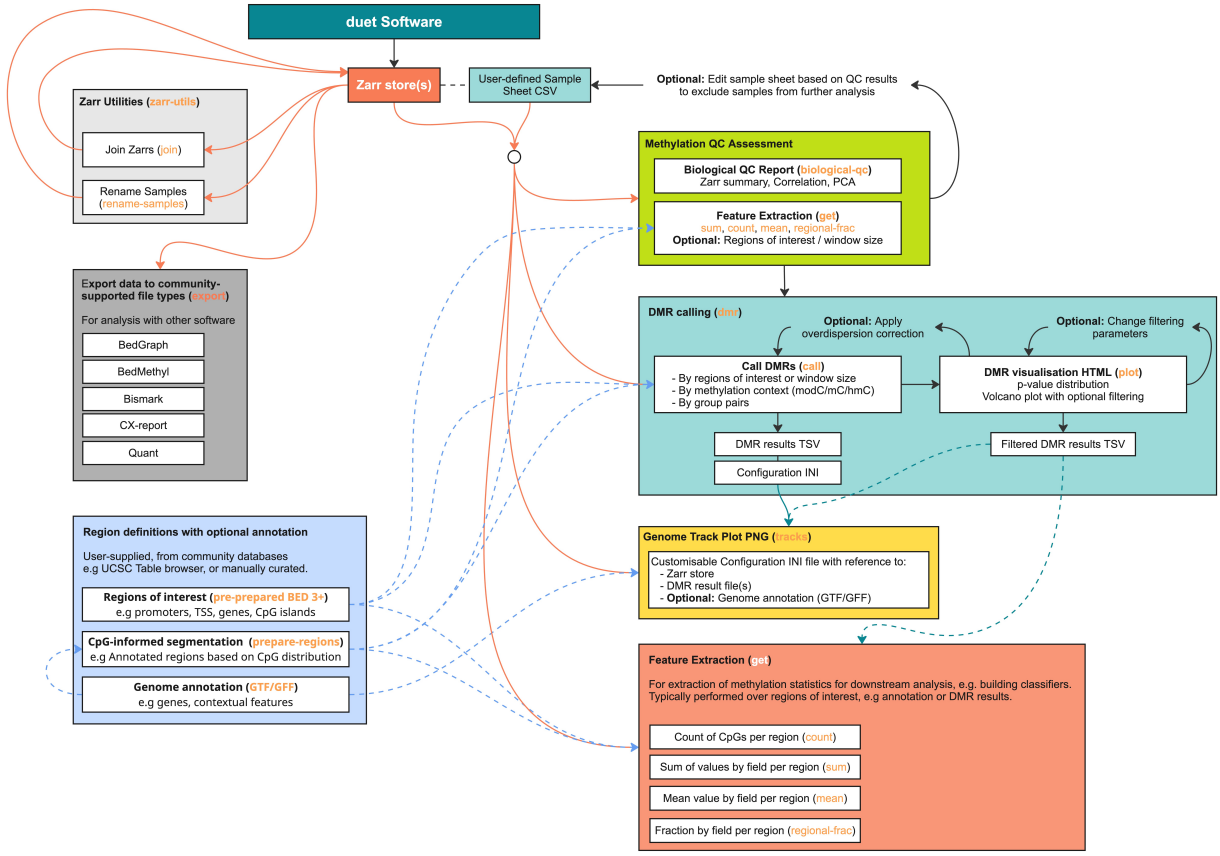

Figure 1: Schematic of the modality XPLR analysis software and example workflows. The Zarr store is generated by the **duet Software**, and can be used as input to modality XPLR tools.

CHAPTER

ONE

USER DOCUMENTATION CONTENTS

1.1 Getting started

1.1.1 Video: Install modality XPLR

1.1.2 System requirements

Before installing modality XPLR, review the following system requirements to ensure compatibility and optimal performance.

Note: Internet access is required for installation and upgrading.

Note: HPC users with restricted internet access may need to speak to their system administrator and request that the biomodal PyPI repository URL, passed to `--extra-index-url` during `pip` installation, is whitelisted `https://europe-python.pkg.dev/prj-biomodal-modality/modality-pypi/simple` see *Install modality XPLR on a system with restricted internet access* for instructions on offline installation.

Recommended operating systems

modality XPLR is compatible with the following operating systems. We recommend using the latest version of each operating system for the best experience:

- **macOS**
- **Linux (Ubuntu)**

Note: Installation on Windows is supported via Windows Subsystem for Linux (WSL). Once WSL is installed, you can follow the Linux installation instructions for modality XPLR.

Recommended hardware requirements

Processor: 4 CPU cores or equivalent 64-bit processor.

Memory (RAM): 16 GB.

Storage: At least 500 MB of available disk space for installation; additional space is required for local storage of Zarr store data to be analysed.

Python programming language

The Python programming language is required to run modality XPLR. Currently, modality XPLR supports Python versions 3.10 and 3.11 and it is recommended to have the latest supported python version installed.

Non-python dependencies

Essential

It is essential to have gzip installed on your system. The latest versions of macOS and Linux have these natively installed however, if you do not have these packages, they can be installed using `brew install gzip` for Mac and `sudo apt-get install gzip` for Linux (sudo may not be needed for some users).

- gzip: For compression of genomic data.

Optional

The following non-python dependencies are optional but may provide performance enhancements. See installation section for further information.

- bgzip: For block compression of genomic data.
- tabix: For quick index look up of block compressed genomic data.

Installation of WSL on Windows

Installation of modality XPLR on Windows is supported via Windows Subsystem for Linux (WSL). Once WSL is installed, you can follow the Linux installation instructions for modality XPLR.

1. Start a PowerShell session by pressing the Start key, typing "PowerShell", and clicking the PowerShell icon.
2. In the PowerShell window, run the following command to enable the WSL feature. Refer to Microsoft's WSL guide for more details:

```
wsl --install
```

Note: WSL can only be installed if you have virtualisation enabled on the BIOS on your computer. The command `wsl --install` requires administrator privileges to be run. You may need to ask IT to run this command for you.

---

3. Launch WSL. You will be prompted to create a username and password. This user can run `sudo` commands. If WSL is not launching, you may need to restart your computer first.

```
wsl
```

4. Once launched, change to the WSL home directory, rather than operating in the main Windows home directory:

```
cd $HOME
```

5. Install Python 3.11. modality XPLR is not compatible with Python 3.12 (which is the default in Ubuntu 24), so install python3.11. Alternatively, install an older version of Ubuntu.

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.11 python3.11-venv
python3.11 --version
```

6. Install necessary dependencies. modality XPLR needs to build some packages to install, so install some build tools.

```
sudo apt install gcc g++ python3.11-dev
```

7. Create and activate a virtual environment for modality XPLR:

```
python3.11 -m venv modality_env
source modality_env/bin/activate
which python
```

8. Upgrade pip

```
pip install --upgrade pip
```

9. Proceed to the *Perform a new install of modality XPLR* step below, following the Linux instructions where applicable.

Important: By default, Windows allocates half the system memory to the WSL session - if your laptop has 16GB RAM, WSL gets 8GB by default. This can be altered by following Microsoft's configuration guide.

1.2 Installation of modality XPLR

Important: It is recommended that setup and installation are performed by your system administrator to meet all local policies. i.e., the person responsible for managing your organisation's IT systems.

1.2.1 Install the latest version in a new virtual environment on macOS or Linux

This section describes how to set up a new virtual environment and install modality XPLR.

Check your version of python

1. Open the Terminal and type `python` followed by the Tab key. This will list versions of python that are currently installed on your computer.
2. If `python3.11` is *not* listed, you can install it on:
    - **macOS:** using homebrew (see https://brew.sh/). With homebrew installed, run `brew install python@3.11`.
    - **Linux:** using apt install, `apt install python3.11`
3. Proceed once `python3.11` is installed and listed according to **Step 1** above.

Important: On Linux you may need to run `apt get update` before running the install command. In some cases, you may require sudo (admin) privileges which would change the command to `sudo apt get update` and then `sudo apt install python3.11`. We recommend that you consult with your system administrator before using `sudo`.

Note: modality XPLR does not currently support python versions >3.11.

Create a new virtual environment for modality XPLR

The instructions below are for both macOS and Linux. Where platform-specific commands are needed (e.g., installing Python), they are provided side-by-side.

It is recommended to always install modality XPLR in a dedicated virtual environment, to specify the version of python to use, and to allow clean installation and uninstallation.

In this example we'll create a new environment called `modality_env` with python3.11, in a dedicated directory called `biomodal-modality`. Run each line individually in your terminal.

```
mkdir biomodal-modality
cd biomodal-modality
python3.11 -m venv modality_env
```

When you want to install, use, or uninstall modality XPLR, first activate the virtual environment by specifying its name.

```
source modality_env/bin/activate
```

Note: This activation command is the same for macOS and Linux.

The terminal will then show the name of the virtual environment in parentheses, before the user/computer name, indicating that the virtual environment is active. To deactivate the virtual environment, enter:

```
deactivate
```

Perform a new install of modality XPLR

These instructions describe how to install modality XPLR when there is no version of modality XPLR currently installed. In this example, we will install modality XPLR in the virtual environment called `modality_env`.

Important: When installing modality XPLR, it is reccommended to always provide a new, clean virtual environment. This can help to avoid version conflicts with modality dependencies that are not removed from the environment when running `pip uninstall modality`.

1. Activate your new virtual environment:

```
cd biomodal-modality
source modality_env/bin/activate
```

2. Install the latest version of modality XPLR using `pip`:

```
pip install --extra-index-url https://europe-python.pkg.dev/prj-biomodal-modality/modality-
↪pypi/simple modality
```

Note: To install a specific version of modality XPLR, specify it explicitly (e.g., `modality==1.0.0`).

3. To comply with modality XPLR's licensing requirements, we must distribute the source of the following dependencies together with modality XPLR by downloading the `tar` files.

```
VM_PYPATH=$(pip show modality | awk -F': ' '/^Location:/ {print $2}')
pip download -d $VM_PYPATH --no-deps --no-binary :all: crc32c==2.7.1
wget https://github.com/scikit-learn/scikit-learn/archive/refs/tags/1.2.0.tar.gz -O $VM_
↪PYPATH/scikit-learn-1.2.0.tar.gz
```

(continues on next page)

(continued from previous page)

```
wget https://github.com/scipy/scipy/archive/refs/tags/v1.11.4.tar.gz -O $VM_PYPATH/scipy-1.
↪11.4.tar.gz
```

4. Check the currently installed version of modality XPLR at any time by running:

```
modality --version
```

Important: After a new install of modality XPLR, checking the version can take a few minutes to complete. Wait for the terminal to return the software version.

Install modality XPLR on a system with restricted internet access

Note: Requires at least 1 machine with non-restricted internet access to download modality XPLR and it's package dependency `.whl` files and remote transfer these to the target machine with restricted internet access.

If you encounter an error while attempting to install modality XPLR using `pip` as described in step 2 of *Perform a new install of modality XPLR*, this may be due to restricted internet access on your system. In such cases, we recommend contacting your system administrator to whitelist the biomodal PyPI repository URL `https://europe-python.pkg.dev/prj-biomodal-modality/modality-pypi/simple`

After the URL has been whitelisted, please try the installation again.

Alternatively, you can perform an offline installation by downloading the modality XPLR package `.whl` files on a machine with internet access and transferring them to the target machine. Follow the steps below for offline installation.

1. Create a new folder and download the modality package and its dependencies to this folder

```
mkdir modality_offline_download
pip download --extra-index-url https://europe-python.pkg.dev/prj-biomodal-modality/modality-
↪pypi/simple modality==1.0.0 -d modality_offline_download
```

2. To comply with modality XPLR's licensing requirements, we must distribute the source of the following dependencies together with modality XPLR by downloading the `tar` files.

```
VM_PYPATH=$(pip show modality | awk -F': ' '/^Location:/ {print $2}')
pip download -d modality_offline_download --no-deps --no-binary :all: crc32c==2.7.1
wget https://github.com/scikit-learn/scikit-learn/archive/refs/tags/1.2.0.tar.gz -O␣
↪modality_offline_download/scikit-learn-1.2.0.tar.gz
wget https://github.com/scipy/scipy/archive/refs/tags/v1.11.4.tar.gz -O modality_offline_
↪download/scipy-1.11.4.tar.gz
```

3. Transfer the modality XPLR package and its dependencies to the target machine using `rsync`

```
rsync -av --checksum modality_offline_download username@remote_host:/path/to/remote/
↪destination/
```

4. Connect to the remote machine according to your organisation's protocol

```
ssh username@remote_host
```

5. Change directory to where the modality XPLR package and its dependencies were transferred

```
cd /path/to/remote/destination/
```

6. Activate your virtual environment according to the steps followed in *Create a new virtual environment for modality XPLR*. Note that this step expects that you are in the folder `biomodal-modality` created earlier when setting up the virtual environment.

```
source modality_env/bin/activate
```

7. Run the offline installation of modality XPLR in your activated virtual environment pointing to path of the downloaded files.

```
pip install --no-index --find-links=modality_offline_download modality
```

Optional: Install non-python dependencies, tabix and bgzip

Refer back to *Non-python dependencies* for more information on these optional dependencies.

Note: The installation of non-python dependencies may bring performance gains and is managed with conda through the htslib library.

This section is not required to run modality XPLR but will provide you with additional performance gains for particular workflows where `bgzip` (block compression) and `tabix` (quick index look up) can be leveraged. Tabix indexes enable fast retrieval of specific genomic regions from large genomic data files. To be indexable by Tabix, files must be both BGZF-compressed and sorted by genomic coordinates. This allows tools to access regions of interest without loading the entire file into memory - particularly useful for processing BED-format data efficiently.

Therefore, we recommend the installation of the htslib library using `conda`. To check if conda is already installed, run the command below. If conda is installed, proceed to installation of htslib.

```
conda --version
```

The following code describes how to install conda. Run each line individually.

On macOS:

```
mkdir -p ~/miniconda3
curl https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-arm64.sh -o ~/miniconda3/
↪miniconda.sh
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm ~/miniconda3/miniconda.sh
source ~/miniconda3/bin/activate
```

On Linux:

```
mkdir -p ~/miniconda3
curl https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -o ~/miniconda3/
↪miniconda.sh
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm ~/miniconda3/miniconda.sh
source ~/miniconda3/bin/activate
```

The following code describes how to install htslib with conda. Run each line individually.

```
conda init --all
conda activate base
conda install -c bioconda htslib
```

This will install the non-python dependencies in the base environment of conda. Therefore, you will need to activate the conda base environment as well as your modality XPLR environment before running modality XPLR commands.

Upgrade the existing version of modality XPLR

When a new version of modality XPLR is available, you can upgrade the software by adding `--upgrade` to the end of the install prompt. Ensure that your modality XPLR environment is activated when installing, upgrading, or uninstalling modality XPLR.

If you encounter any issues during the upgrade process, try installing modality XPLR in a new virtual environment.

```
pip install --extra-index-url https://europe-python.pkg.dev/prj-biomodal-modality/modality-
↪pypi/simple modality --upgrade
```

Check the currently installed version of modality XPLR at any time by running:

```
modality --version
```

Uninstall modality XPLR

To uninstall a version of modality XPLR, run the following command. Checking the version afterwards will confirm that modality XPLR has been uninstalled, if it cannot be found in the virtual environment:

```
pip uninstall modality
```

1.3 Authenticate modality XPLR

Upon first use of running any `modality` command, you will be prompted to authenticate your installation with an e-mail address and password. Authentication is only required once per machine.

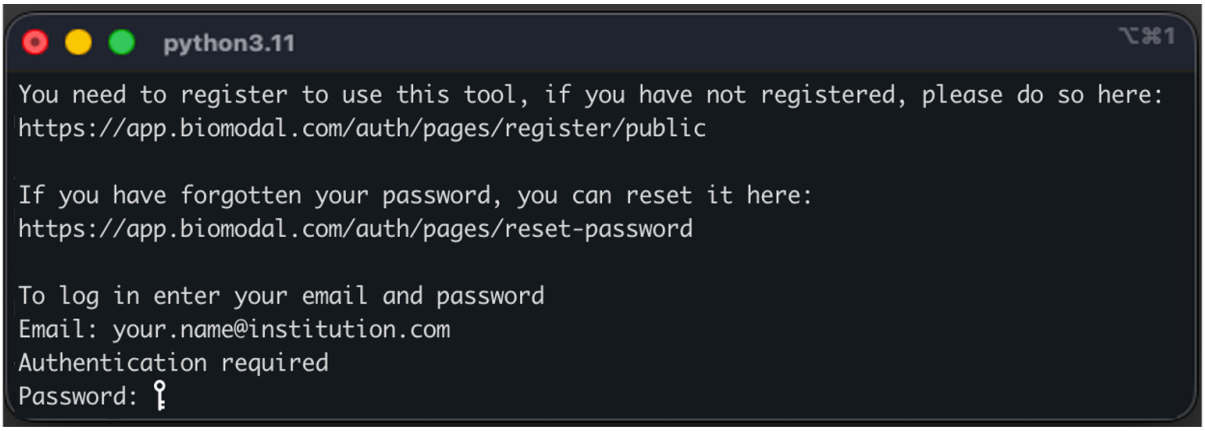

Figure 1.1: modality XPLR authentication prompt.

If you or your institution is already a user of biomodal **duet Software** for pipeline processing, login to modality XPLR using your existing biomodal credentials.

If you are not already a biomodal user, you can self-register for access to modality XPLR. Please use your institution e-mail address when registering.

If you have forgotten your password, you can reset it.

Note: Self-registered users will have access to modality XPLR only. For access to **duet Software**, please contact your biomodal Sales or Support representative.

1.4 Core Workflow

The modality XPLR Core Workflow is a simple-to-use analysis tool that leverages bash scripting to automate key modality XPLR commands on biomodal data contained in the *Zarr store*. The Core Workflow is well-suited for exploration of pilot study data, or for new users who want rapid insights from their data through a standardised, low-code approach.

For experienced users who want a deep dive into the Core Workflow script, proceed directly to the *Advanced analysis workflow* documentation.

This section describes how to set up and run the modality XPLR Core Workflow. This workflow is designed for Human, biomodal 6-base datasets (including `5mC` and `5hmC`), that are aligned to genome reference `hg38`. The workflow is pre-configured to analyse gene bodies and promoter regions only. See the *Pre-prepared BED files* section in the main documentation for more information on how the region files were generated.

- **Gene bodies:** `gencode.v44.human.genes.annotation.bed.gz`
- **Promoter regions:** `gencode.v44.human.promoters.annotation.bed.gz`

The Core Workflow performs the following analyses and outputs:

Biological QC

1. `biological-qc` to generate an HTML report summarising the dataset and sample information, with Pearson correlation and PCA plots for genome-wide `5mC` and `5hmC` data. See *Biological QC* for more information.
2. `get mean` to generate an HTML report (violin and correlation plots) and a `.tsv` data file, showing the mean CpG coverage in gene bodies and promoter regions. Context depth filter is **not applied** to this command. See *Extracting methylation statistics* for more information on this and other `get` commands.
3. `get regional-frac` to generate an HTML report (violin and correlation plots) and a `.tsv` data file, showing the fraction of `5mC` and `5hmC` (over `total_c`) in gene bodies and promoter regions, with an optional minimum depth filter **applied**.
4. `get count` to generate a `.tsv` data file containing the total number of contexts (`total_c`) in each gene body and promoter region. Context depth filter is **not applied** to this command, as it is not applicable to the `count` command.

Biological insight

1. `dmr call` to identify differentially methylated regions (DMRs) in gene bodies and promoter regions, for both `5mC` and `5hmC` contexts, across all specified groups with an optional context depth filter **applied**. This command generates separate `.bed` files for each methylation context and group pair. See *Calling differentially methylated regions (DMRs)* for more information.
2. `dmr plot` to generate an HTML report for each DMR results file, showing summary information and unfiltered results as a volcano plot and p-value distributions. See *Plotting differentially methylated regions (DMRs)* for more information.
3. `dmr plot` **with filtering** to generate an HTML report for each DMR results file, showing summary information and filtered results (q-value < 0.05) as a volcano plot and p-value distributions. When filtering is applied, a new DMR results `.bed` file is generated containing only the filtered DMRs for each region type and methylation context.

Each DMR analysis is also accompanied by an `.ini` configuration file that can be used for visualising DMR result with `modality tracks`, see *Generate genomic track plots* for more information.

The Core Workflow is well-suited for studies involving:

- Disease vs control comparisons
- Treatment vs untreated samples
- Time-course experiments
- Multi-group studies with covariates

1.4.1 Video: Running the modality XPLR Core Workflow

1.4.2 Requirements and prerequisites

Software requirements:

- modality XPLR (version 1.0.0 or later is recommended, see *Installation of modality XPLR*)
- Bash shell (version 4.0 or later)
- Standard Unix utilities (awk, sed, find, etc.)

Data input and format requirements:

- **Zarr store(s)**: Generated by the upstream biomodal duet software, containing methylation data. Zarr stores must be readable by modality XPLR (generated by biomodal duet software v1.2 or later).
- **Sample metadata**: `.csv` or `.tsv` file with sample information, group names and optional covariates (also referred to as the sample sheet). The metadata file (*Sample sheet*) must be formatted as described in the main documentation.

1.4.3 Core Workflow configuration

For ease-of-use, the Core Workflow can be configured and downloaded using the *Interactive Core Workflow generator* tool below. This generator allows you to specify input parameters, and then download a complete `.zip` folder containing the workflow script, configuration file, and pre-prepared gene and promoter regions files.

Core Workflow folder contents:

```
modality-core-workflow_YYYYMMDD_HHMMSS/
├── core-workflow-v1.3.sh                              # The Core Workflow script
├── config.txt                                         # Configuration file for modality␣
↪XPLR commands
├── Regions/                                           # Directory containing pre-prepared␣
↪region files
│   ├── gencode.v44.human.genes.annotation.bed.gz
│   └── gencode.v44.human.promoters.annotation.bed.gz
├── README.txt                                         # Usage instructions
└── Output/                                            # Output directory(created during␣
↪execution)
```

Interactive Core Workflow generator

1.4.4 Running the Core Workflow

To run the Core Workflow, follow these steps:

1. Download the Core Workflow .zip folder to your working location and extract it.
2. In the terminal, run the following lines.

```
# Change to the Core Workflow directory
cd /path/to/modality-core-workflow_YYYYMMDD_HHMMSS

# Make the script executable
chmod +x core-workflow-v1.3.sh

# Run the Core Workflow script
./core-workflow-v1.3.sh
```

3. Monitor the output logs for progress and any potential errors.

Advanced Core Workflow usage

The `config.txt` file contains all the parameters for running the Core Workflow script. You should not need to modify this file, as it is pre-configured by the *Interactive Core Workflow generator*. Advanced users can modify the `config.txt` file to change parameters such as the Zarr store path, metadata file, group column, covariates, and minimum depth filter. In addition, the Output directory path can be changed to a different location if required. This may be useful if you want to re-run the analysis with different parameters (e.g. context depth filter, or covariates), without needing to download the Core Workflow folder again.

It is also possible to provide additional region files in the `Regions/` directory, or to change the region files used by the Core Workflow. Refer to the main documentation for the required format of the *Region files*. Apart from the Biological QC HTML report (which operates on Zarr store(s)), the Core Workflow will perform all analysis steps on each region file found in the `Regions/` directory.

For advanced users who are familiar with bash scripting, a description of the Core Workflow code is provided in the *Advanced analysis workflow* section, which can be used to customise the workflow further.

1.5 Interpreting Core Workflow outputs

The Core Workflow generates a comprehensive set of outputs organised into structured directories. Understanding this organisation is crucial for interpreting results and sharing findings. Each analysis produces timestamped outputs and provenance metadata for traceability and reproducibility of the results.

The workflow organises outputs by creating a subdirectory within both `Results/` and `Reports/` for each region file processed. This means that all analysis results (feature extraction, DMR calling, and DMR visualisation) related to a specific region file (e.g., genes, promoters) are grouped together in their respective subdirectories. Similarly, all HTML reports related to a specific region file are organised in matching subdirectories within the `Reports/` directory. The Biological QC results, which are not related to a specific region file input, are stored in a `BioQC_` timestamped directory in `Results/` and the QC HTML report is placed directly in the main `Reports/` directory.

1.5.1 Video: Reviewing Core Workflow results

1.5.2 Output directory structure

A `modality_cli.log` file is generated in the working directory and contains logging information relating to the modality XPLR CLI commands only (not the Core Workflow bash outputs). This file is generated automatically by modality XPLR and is not directly controlled by the Core Workflow script. Other outputs are described below.

```
Output/
├── Results/                                          # All analysis outputs
│   ├── BioQC_YYYYMMDD_HHMMSS/                        # Biological QC results
│   │   ├── biological_qc_report_*.html               # Biological QC HTML report
│   │   └── modality_metadata_YYYYMMD_HHMMSS.json     # Provenance metadata file
│   ├── <region_file_name_1>/                         # Results for first region file␣
↪(e.g. genes)
│   │   ├── Extract_YYYYMMDD_HHMMSS/                  # Feature extraction results
│   │   │   ├── Extract_*.tsv.gz                      # Feature extraction results file␣
↪(compressed)
│   │   │   ├── Extract_*.html                        # Violin and Correlation plots␣
↪(mean and regional-frac operations only)
│   │   │   └── modality_metadata_YYYYMMD_HHMMSS.json # Provenance metadata file
│   │   ├── DMR_YYYYMMDD_HHMMSS/                      # DMR analysis results
│   │   │   ├── DMR_*.bed                             # DMR result files
│   │   │   ├── DMR_*.ini                             # Track configuration files␣
↪(track plots are not included in this script)
│   │   │   └── modality_metadata_YYYYMMD_HHMMSS.json # Provenance metadata file
```

(continues on next page)

(continued from previous page)

```
│   │   └── DMR_Report_YYYYMMDD_HHMMSS/                    # DMR visualisation reports
│   │       ├── DMR_Report_*.html                          # Unfiltered DMR visualisation␣
↪report
│   │       ├── DMR_Report_*.max_q_0_05_*.html             # Filtered DMR visualisation␣
↪report
│   │       ├── DMR_*_qval_0_05_filtered_*.bed.gz          # Filtered DMR results file␣
↪(compressed)
│   │       └── modality_metadata_YYYYMMD_HHMMSS.json      # Provenance metadata file
│   ├── <region_file_name_2>/                              # Results for second region file␣
↪(e.g. promoters)
│   │   └── ... (same structure as above)
│   └── <region_file_name_N>/                              # Results for Nth region file (e.
↪g. if you add additional region files to the Regions/ directory)
│       └── ... (same structure as above)
└── Reports/                                               # Organised HTML reports
    ├── biological_qc_report*.html                         # Biological QC HTML report
    ├── <region_file_name_1>/                              # Reports for first region file␣
↪(e.g. genes)
    │   ├── Extract_*mean*.html                            # Mean CpG coverage violin and␣
↪correlation plots
    │   ├── Extract_*regional-frac*.html                   # 5mC and 5hmC fraction violin␣
↪and correlation plots
    │   ├── DMR_Report_*.html                              # Unfiltered DMR visualisation␣
↪reports
    │   └── DMR_Report_*_max_q_*.html                      # Filtered DMR visualisation␣
↪reports
    ├── <region_file_name_2>/                              # Reports for second region file␣
↪(e.g. promoters)
    │   └── ... (same structure as above)
    └── <region_file_name_N>/                              # Reports for Nth region file (e.
↪g. if you add additional region files to the Regions/ directory)
        └── ... (same structure as above)
```

1.5.3 Reviewing results

1. **Biological QC**
   - Check sample clustering in the Biological QC HTML report PCA analysis (located in the main `Reports/` directory). Use the colour-based grouping options directly on the report to visualise sample relationships. Groups are defined by the metadata (sample sheet) file.
   - Verify expected grouping patterns. Does the data make biological sense for your experimental design?
   - Identify any outlier samples.
   - Assess overall sample mean CpG coverage on the Biological QC report (Sample Information table).
   - Review the region-specific feature extraction plots (violin plots and correlation heatmaps) for mean CpG coverage, and 5mC and 5hmC fraction in the region-specific subdirectories within `Reports/`. These visualizations provide insights into feature distributions and sample correlations.
2. **Biological insights**
   - Navigate to the region-specific subdirectories within `Reports/` (e.g., `Reports/gencode.v44.human.genes.annotation/`).
   - Check the unfiltered DMR HTML reports for each region type and methylation context. Check the volcano plots and p-value distributions.

- Focus on significant DMRs by reviewing the filtered DMR HTML reports (files with `max_q_` in the name, indicating q-value < 0.05 filtering).
- If needed, review the DMR results files (`.bed`) for specific regions of interest, or to filter or sort based on other metrics.
- Note the region annotations in the DMR results files, which can be used to gain further biological insights into the DMRs.

3. **Validate findings**
   - Check for consistency with known biology
   - Consider experimental design and potential confounders
   - Plan follow-up validation experiments or to increase sample size if necessary

Common result patterns and interpretation

Healthy DMR patterns:

- Detection of significant DMRs between experimental groups
- Consistent p-value distributions (uniform with some enrichment near zero)
- Biologically meaningful effect sizes
- Concordant patterns between 5mC and 5hmC where expected

Potential issues to watch for:

- **Poor group separation**: May indicate experimental design issues
- **Skewed p-value distributions**: May benefit from *Overdispersion correction* or other statistical adjustments
- **Very large effect sizes**: May indicate technical artifacts
- **Inconsistent patterns**: Could suggest batch effects or confounders

Biological interpretation guidelines:

- **Promoter hypermethylation**: Often associated with deactivated regions
- **5hmC enrichment**: Often associated with active demethylation (transition to active regions), or a marker of active gene bodies.
- **Concordant changes**: 5mC loss with 5hmC gain may suggest active demethylation

1.6 Advanced analysis workflow

This section provides a more detailed description of the Core Workflow bash script, which may be beneficial to users who want to modify the script directly or build their own workflows.

The direct download link to the Core Workflow script is provided below, which provides the `.sh` file only.

Download: `core-workflow-v1.3.sh` - Core Workflow script assembled from all code sections below.

1.6.1 Download a test dataset

You can download a test folder containing mouse ES-E14 data to run the Core Workflow. The outputs are not intended to reveal biological insight, as the samples are technical replicates. However, it can be used to test that the script runs to completion.

The `E14-example` test folder contains:

- A sample metadata file `E14_chromosome_19.tsv`
- A zarr store `E14_chromosome_19.zarrz` containing methylation data
- A directory `Regions` containing `gencode.vM25.mouse.promoters.annotation.bed`
- A `config.txt` file with example parameters

Download: `E14-example.zip`

Note: The `E14-example.zip` does not contain the `core-workflow-v1.3.sh` script. This will need to be downloaded separately and run from within the `E14-example` directory after extraction.

1.6.2 Advanced configuration file setup

The Core Workflow uses a simple `config.txt` file for all settings. This design separates configuration from code, making it easy to run the same pipeline with different datasets or parameters. If you are new to modality XPLR or have not yet run the Core Workflow, we suggest using the *Interactive Core Workflow generator* to get started.

Configuration file:

This file must be present in your working directory before running the pipeline, with the name `config.txt`. Alternatively use the *Advanced interactive configuration file generator* below to create this file with additional options. The aim of the `config.txt` is to pre-define all parameters required by the workflow script. If you modify the Core Workflow script to include additional options or settings, these can be added to the `config.txt` as variables. For the Core Workflow, the `config.txt` file should contain the following parameters (parameter values are examples):

```
Zarr=/path/to/data.zarrz
Metadata=/path/to/metadata.csv
Group_Column=label
Condition_Order=condition1,condition2
Covariates=age,batch,sex
QC_Region=chr1
Regions_Directory=/path/to/regions
Main_Output=/path/to/output_dir
Depth_Filter=5
Overdispersion=False
```

Configuration parameters explained:

- **Zarr**: Path to zarr store(s). Use space separation for multiple paths
- **Metadata**: Path to `.csv` or `.tsv` sample sheet file containing sample metadata
- **Group_Column**: Column name in the metadata file, for grouping samples (e.g., “treatment”, “condition”)
- **Covariates**: Space-separated list of covariate columns (optional) (e.g., “age batch”, “sex race” etc.)
- **QC_Region**: Region specification for Biological QC analysis (optional) (e.g., “chr1” or “chr1:1000-2000”)
- **Regions_directory**: Directory containing BED/TSV files in BED3+ format, with genomic regions and annotations. See *Region files* for details
- **Main_Output**: Base directory where all results will be written
- **Depth_Filter**: Minimum depth filter for analyses (leave it blank to disable filtering)

- **Overdispersion**: Apply overdispersion correction for DMR calling ("True" or "False", optional)

For a description of *Sample sheet* and *Region files* formats, see the linked modality XPLR documentation sections.

The `config.txt` can be generated using the *Advanced interactive configuration file generator* below or created manually in a text editor.

Advanced interactive configuration file generator

1.6.3 Core Workflow overview: step-by-step

The Core Workflow script is organised into distinct sections that can be modified independently:

1. **Setup and Initialisation**: Script configuration and logging setup
2. **Input Validation**: Comprehensive checks of all inputs before execution
3. **Analysis Steps**: Four sequential analysis stages, designed to deliver **Biological QC** and **Biological Insight** analyses.
4. **Output Management**: Organised collection and reporting of results

Script setup and configuration

The script begins with essential setup including logging configuration and error handling.

Note: This code block includes instructions for enabling logging to a file, which may be useful for debugging and tracking script execution. To enable, uncomment (remove #) from the code lines as directed.

```
#!/usr/bin/env bash

################################################################################
# modality XPLR Core Workflow Pipeline
#
# This script runs a comprehensive analysis pipeline including:
# - Biological QC
# - Feature extraction
# - DMR calling and visualisation
#
# Input requirements:
# - config.txt file in current directory with required parameters
# - Zarr store(s) containing methylation data
# - Sample metadata CSV or TSV file
# - Directory with region BED/TSV files
#
# This script is intended for running on modality XPLR version 1.0.0 or later.
#
################################################################################

#-------------------------------------------------------------------------------
# CONFIGURATION
#-------------------------------------------------------------------------------
# Set script behaviour
set -euo pipefail

# Record start time for performance tracking
start_time=$(date +%s)

# Log file setup - uncomment to enable logging to file
```

(continues on next page)

(continued from previous page)

```
# LOG_FILE="./modality_pipeline_$(date +%Y%m%d_%H%M%S).log"
# exec > >(tee -a "$LOG_FILE") 2>&1
```

Helper functions

These logging functions provide consistent feedback throughout the pipeline execution.

```
#------------------------------------------------------------------------------
# HELPER FUNCTIONS
#------------------------------------------------------------------------------
# Logging functions
log_info() {
  echo "[INFO] $(date '+%Y-%m-%d %H:%M:%S') - $1"
}

log_warning() {
  echo "[WARNING] $(date '+%Y-%m-%d %H:%M:%S') - $1" >&2
}

log_error() {
  echo "[ERROR] $(date '+%Y-%m-%d %H:%M:%S') - $1" >&2
}

log_debug() {
  if [[ "${DEBUG:-false}" == "true" ]]; then
    echo "[DEBUG] $(date '+%Y-%m-%d %H:%M:%S') - $1"
  fi
}
```

Utility functions

These utility functions handle common tasks like directory validation and report collection.

```
 # Helper to trim whitespace
trim() {
  awk '{$1=$1};1'
}

# Function to check if a header exists in metadata
header_exists() {
  local col="$1"
  local headers=("${@:2}")

  for h in "${headers[@]}"; do
    [[ "$h" == "$col" ]] && return 0
  done
  return 1
}

# Function to create directories with validation
create_directory() {
  local dir="$1"

  if [[ ! -d "$dir" ]]; then
```

(continues on next page)

(continued from previous page)

```
    mkdir -p "$dir" 2>/dev/null || { log_error "Cannot create directory $dir"; exit 1; }
  fi
  if [[ ! -w "$dir" ]]; then
    log_error "No write access to directory $dir"
    exit 1
  fi
  log_info "Directory ready: $dir"
}

# Function to copy reports to destination
copy_reports() {
  local source_dir="$1"
  local pattern="$2"
  local dest_dir="$3"
  local description="$4"

  log_info "Copying $description HTML reports to $dest_dir..."
  local files=$(find "$source_dir" -type f -name "$pattern")

  if [[ -z "$files" ]]; then
    log_error "No $description HTML reports found in $source_dir"
    return 1
  fi

  local count=0
  while IFS= read -r file; do
    cp "$file" "$dest_dir/"
    ((count++))
  done <<< "$files"

  log_info "Copied $count $description HTML report(s)"
  return 0
}
```

Initialisation and configuration loading

This section initialises the script and loads the configuration file. If you want to change the name of the input configuration file from the default `config.txt`, you can do so by changing the `CONFIG_FILE` variable in the script. Similarly, if you want to control the verbosity of workflow logging, you can do so by changing the `DEBUG` variable in the script to `true`.

```
#------------------------------------------------------------------------------
# INITIALISATION
#------------------------------------------------------------------------------
printf "\n========================================\n"
printf "STEP 0 - CONFIGURATION & INITIALISATION\n"
printf "========================================\n"
log_info "Starting modality XPLR Core Workflow Pipeline"

# Default values for variables that could be customised
CONFIG_FILE="./config.txt"
DEBUG=false  # Set to true to enable debug output
```

The next part will read the configuration file and set values for all required parameters. If you wish to change existing parameters values, you can do so by editing the configuration (`config.txt`) file in your working directory. If you want to add new variables to be used in downstream modality XPLR commands, ensure they are defined here and then prompted to assign a value when reading in the configuration file. For any variables that should be hard-coded, the value can be set

directly in the script and excluded from the configuration file.

```
#------------------------------------------------------------------------------
# LOAD CONFIGURATION
#------------------------------------------------------------------------------
log_info "Reading configuration from $CONFIG_FILE"

# Validate config file exists
if [[ ! -f "$CONFIG_FILE" ]]; then
  log_error "config.txt not found in the current working directory: $(pwd)"
  log_error "Please ensure config.txt is present before running this script."
  exit 1
fi

# Read config.txt variables
ZARR=""                  # Path(s) to zarr store(s), comma-separated
METADATA=""              # Path to sample metadata CSV or TSV file
GROUP_COLUMN=""          # Column name in metadata for grouping samples
COVARIATES=""            # Space-separated list of covariate column names (optional)
CONDITION_ORDER=""       # Comma-separated order of conditions for DMR analysis (optional)
QC_REGION=""             # Region for Biological QC analysis (optional)
REGIONS_DIR=""           # Directory containing region BED/TSV files
MAIN_OUTPUT=""           # Base output directory
DEPTH_FILTER=""          # Minimum depth filter for analyses, e.g. 10 (optional)
OVERDISPERSION=""        # Apply overdispersion correction: "True" or "False" (optional)

# Parse config file
while IFS='=' read -r key value; do
  key=$(echo "$key" | trim)
  value=$(echo "$value" | trim)
  case "$key" in
    Zarr) ZARR="$value" ;;
    Metadata) METADATA="$value" ;;
    Group_Column) GROUP_COLUMN="$value" ;;
    Covariates) COVARIATES="$value" ;;
    Condition_Order) CONDITION_ORDER="$value" ;;
    QC_Region) QC_REGION="$value" ;;
    Regions_Directory) REGIONS_DIR="$value" ;;
    Main_Output) MAIN_OUTPUT="$value" ;;
    Depth_Filter) DEPTH_FILTER="$value" ;;
    Overdispersion) OVERDISPERSION="$value" ;;
    # You could add additional parameters here as needed
  esac
done < "$CONFIG_FILE"

# Log parsed configuration
printf "\n\n>>Provided inputs\n"
log_info "ZARR path(s): $ZARR"
log_info "Metadata file: $METADATA"
log_info "Group column: $GROUP_COLUMN"
log_info "Covariates: $COVARIATES"
log_info "Condition order: $CONDITION_ORDER"
log_info "QC region: $QC_REGION"
log_info "Regions directory: $REGIONS_DIR"
log_info "Main output directory: $MAIN_OUTPUT"
log_info "Depth filter: $DEPTH_FILTER"
log_info "Overdispersion correction: $OVERDISPERSION"
```

Input validation

This section validates all inputs and ensures they exist and are accessible. Use this section to add any custom validation checks for your specific workflow requirements, file paths, and access permissions, according to the variables defined in the `config.txt` file.

```
#-------------------------------------------------------------------------------
# VALIDATE INPUTS
#-------------------------------------------------------------------------------
printf "\n\n>>Validation & directory creation\n"
log_info "Validating inputs..."

# Parse and validate zarr paths
if [[ -n "$ZARR" ]]; then
  # Split comma-separated zarr paths into array
  IFS=',' read -r -a ZARR_RAW_ARRAY <<< "$ZARR"

  # Process each zarr path to remove quotes and trim whitespace
  ZARR_ARRAY=()
  for zarr_raw in "${ZARR_RAW_ARRAY[@]}"; do
    # Trim leading/trailing whitespace
    zarr_clean=$(echo "$zarr_raw" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')

    # Remove surrounding single or double quotes if present
    zarr_clean=$(echo "$zarr_clean" | sed 's/^["'"'"']\(.*\)["'"'"']$/\1/')

    # Add to processed array if not empty
    if [[ -n "$zarr_clean" ]]; then
      ZARR_ARRAY+=("$zarr_clean")
    fi
  done

  # Log the number of zarr paths identified and their strings
  log_info "Identified ${#ZARR_ARRAY[@]} zarr path(s):"
  for i in "${!ZARR_ARRAY[@]}"; do
    log_info "  [$((i+1))] ${ZARR_ARRAY[i]}"
  done

  # Validate each zarr path
  for zarr in "${ZARR_ARRAY[@]}"; do
    if [[ ! -s "$zarr" ]]; then
      log_error "Zarr file '$zarr' doesn't contain any data or is not readable"
      exit 1
    fi
  done
else
  log_error "No zarr paths specified in config.txt"
  exit 1
fi

# Process and validate metadata file path
if [[ -n "$METADATA" ]]; then
  # Trim leading/trailing whitespace
  METADATA=$(echo "$METADATA" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')

  # Remove surrounding single or double quotes if present
  METADATA=$(echo "$METADATA" | sed 's/^["'"'"']\(.*\)["'"'"']$/\1/')
```

(continues on next page)

(continued from previous page)

```
  log_info "Processed metadata file path: $METADATA"
else
  log_error "No metadata file specified in config.txt"
  exit 1
fi

# Process and validate main output directory path
if [[ -n "$MAIN_OUTPUT" ]]; then
  # Trim leading/trailing whitespace
  MAIN_OUTPUT=$(echo "$MAIN_OUTPUT" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')

  # Remove surrounding single or double quotes if present
  MAIN_OUTPUT=$(echo "$MAIN_OUTPUT" | sed 's/^["'"'"']\(.*\)["'"'"']$/\1/')

  log_info "Processed main output directory: $MAIN_OUTPUT"
else
  log_error "No main output directory specified in config.txt"
  exit 1
fi

# Process and validate regions directory path
if [[ -n "$REGIONS_DIR" ]]; then
  # Trim leading/trailing whitespace
  REGIONS_DIR=$(echo "$REGIONS_DIR" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')

  # Remove surrounding single or double quotes if present
  REGIONS_DIR=$(echo "$REGIONS_DIR" | sed 's/^["'"'"']\(.*\)["'"'"']$/\1/')

  log_info "Processed regions directory: $REGIONS_DIR"
else
  log_error "No regions directory specified in config.txt"
  exit 1
fi

# Check metadata file
log_info "Validating metadata file headers..."
if [[ ! -r "$METADATA" ]]; then
  log_error "Metadata file not found or not readable: $METADATA"
  exit 1
fi

# Check group column and covariates in metadata
HEADER_LINE=$(head -n 1 "$METADATA")

# Remove any line ending characters (Windows \r\n, Mac \r, Unix \n)
HEADER_LINE=$(echo "$HEADER_LINE" | tr -d '\r\n')

# Detect delimiter: use tab if present, otherwise comma
if [[ "$HEADER_LINE" == *$'\t'* ]]; then
  DELIM=$'\t'
else
  DELIM=','
fi

# Split header into array using the detected delimiter
IFS="$DELIM" read -r -a HEADERS_RAW <<< "$HEADER_LINE"
```

(continues on next page)

(continued from previous page)

```
# Trim whitespace from each header individually
HEADERS=()
for header in "${HEADERS_RAW[@]}"; do
  # Trim leading and trailing whitespace from each header
  header_clean=$(echo "$header" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')
  if [[ -n "$header_clean" ]]; then
    HEADERS+=("$header_clean")
  fi
done

if [[ -n "$GROUP_COLUMN" ]]; then
  if ! header_exists "$GROUP_COLUMN" "${HEADERS[@]}"; then
    log_error "Group column '$GROUP_COLUMN' not found in metadata file header. Please␣
↪review the metadata file and provide a valid group column."
    exit 1
  fi
fi

if [[ -n "$COVARIATES" ]]; then
  # Convert space-separated list to array
  read -r -a COVS <<< "$COVARIATES"
  for cov in "${COVS[@]}"; do
    cov=$(echo "$cov" | trim)
    if ! header_exists "$cov" "${HEADERS[@]}"; then
      log_error "Covariate column '$cov' not found in metadata file header"
      exit 1
    fi
  done
fi

log_info "Metadata validation passed"

# Check regions directory
if [[ ! -d "$REGIONS_DIR" ]]; then
  log_error "Regions directory not found: $REGIONS_DIR"
  exit 1
fi
```

Output directory setup

This section creates and defines output sub-directories in `MAIN_OUTPUT` that can be used to organise the results.

By default, all results for each modality XPLR CLI operation (including provenance metadata) will be written to the `Results` sub-directory. Then, all HTML reports and visualisations will be *copied* to the `Reports` sub-directory for quick access. All modality XPLR outputs are timestamped, however you may wish to modify this section to add additional sub-directories for specific analyses, or to change the output directory structure. The `temp` sub-directory is used to store intermediate files, such as the merged regions file, and can optionally be deleted after the workflow completes. By default, no delete command is included.

```
#------------------------------------------------------------------------------
# SETUP OUTPUT DIRECTORIES
#------------------------------------------------------------------------------
log_info "Creating output directories under $MAIN_OUTPUT..."

# Create output directories
```

(continues on next page)

(continued from previous page)

```
for sub in "" "Results" "Reports"; do
  dir="$MAIN_OUTPUT"
  [[ -n "$sub" ]] && dir="$MAIN_OUTPUT/$sub"
  create_directory "$dir"
done

# Define output subdirectories for easier reference
RESULTS_DIR="$MAIN_OUTPUT/Results"
REPORTS_DIR="$MAIN_OUTPUT/Reports"

# Create subdirectories for each region file found in the regions directory, to organise␣
↪outputs by region file input.
log_info "Creating region-specific subdirectories..."
for bedfile in "$REGIONS_DIR"/*.bed "$REGIONS_DIR"/*.bed.gz "$REGIONS_DIR"/*.tsv "$REGIONS_
↪DIR"/*.tsv.gz; do
  # Skip if no files match pattern
  [[ -e "$bedfile" ]] || continue
  [[ "${METADATA##*.}" == "tsv" && $(realpath "$bedfile") == $(realpath "$METADATA") ]] &&␣
↪continue

  # Extract region file name without path and extension
  region_name=$(basename "$bedfile")
  region_name="${region_name%%.bed.gz}"
  region_name="${region_name%%.bed}"
  region_name="${region_name%%.tsv.gz}"
  region_name="${region_name%%.tsv}"

  # Create region-specific subdirectories in both Results and Reports
  region_results_dir="$RESULTS_DIR/$region_name"
  region_reports_dir="$REPORTS_DIR/$region_name"
  create_directory "$region_results_dir"
  create_directory "$region_reports_dir"
  log_info "Created region directories: $region_results_dir and $region_reports_dir"
done
```

Step 1: Biological QC HTML report

The first analysis step performs biological quality control to provide Zarr store summaries, and to identify sample relationships.

What this script block does:

- Generates correlation matrices between samples
- Creates PCA plots to visualise sample clustering (grouping options are available on the HTML report)
- Calculates mean CpG coverage for each sample

Key parameters:

- Uses all zarr stores specified in configuration (`config.txt`)
- Allows sample metadata to be used for dynamic colour-based grouping, when available

---

Note: The Core Workflow does not include the `--aggregate-by-group` option, which could be included to aggregate sample data for technical replicates prior to QC.

```
#------------------------------------------------------------------------------
# STEP 1: BIOLOGICAL QC
#------------------------------------------------------------------------------
printf "\n\n========================================\n"
printf "STEP 1: RUNNING BIOLOGICAL QC ANALYSES\n"
printf "========================================\n"

# Build region argument if QC_REGION is set
REGION_ARG=""
[[ -n "$QC_REGION" ]] && REGION_ARG="--region $QC_REGION"

# Run QC without aggregation
log_info "Running biological QC without aggregation..."
modality -v biological-qc \
  --zarr-path "${ZARR_ARRAY[@]}" \
  --sample-sheet "$METADATA" \
  $REGION_ARG \
  --order-by-group "$GROUP_COLUMN" \
  --output-dir "$RESULTS_DIR"

# Copy QC reports to Reports directory
copy_reports "$RESULTS_DIR" "biological_qc_report*.html" "$REPORTS_DIR" "Biological QC" ||␣
↪exit 1
```

Expected outputs:

- `biological_qc_report.html`: HTML report with sample-wise plots and statistics.
- Provenence metadata `.json` files, stored in each Results/BioQC_YYYYMMDD_HHMMSS sub-directory that is generated by each modality XPLR CLI command.

Interpreting QC results:

- **Correlation heatmaps**: Correlation coefficients close to 1 indicate high similarity of 5mC or 5hmC levels between samples
- **PCA plots**: Separation and/or clustering between experimental groups indicates the % variability that is explained by the first two principal components, for 5mC or 5hmC.
- **Coverage statistics**: The mean CpG coverage per sample is an indicator of the usable depth for methylation analysis. See the next step for feature extraction (region-based) coverage statistics.
- **Outlier detection**: Samples that don't cluster with their expected group may need investigation. Removing a sample from the metadata (sample sheet) file will exclude it from future analysis, without needing to modify the original Zarr file.

Step 2: Region-based Biological QC

This step used the `modality get` tool to extract methylation features (statistics) from user-specified genomic regions, for additional quality control and analysis. The regions can be defined in BED3+ format (`.bed` or ``.tsv`) files, that are accessible in the `REGIONS_DIR`, defined in `config.txt`. For the Core Workflow script to run correctly, region files must be present. *Pre-prepared BED files* are provided to get started, or custom files following the same format can be used instead. Use the correct region files for the reference genome used for alignment in the **duet Software**.

Tip: The `Annotation` (e.g. gene, or promoter) and `Name` (e.g. associated gene name) columns should be completed for meaningful interpretation. Regions without an `Annotation` value will be skipped.

The pipeline performs multiple types of statistics extraction to provide comprehensive regional methylation data. Each statistic type is processed separately by modality XPLR, and the results are stored in sub-directories named according to the extraction type (e.g. `Extract_count`, `Extract_mean`, `Extract_regional_frac`).

Note: Only `modality get mean` and `modality get regional-frac` commands support plotting (`correlation` or `violin`), which are useful for visualising the distribution of methylation levels across samples. If no HTML report is generated, then the results will only be written to `MAIN_OUTPUT/Results/Extract_*` sub-directories as `.tsv.gz` files. HTML reports will be copied to the `MAIN_OUTPUT/Reports` sub-directory for easy access.

What this script block does:

- **Count extraction**: Returns the total number of CpG contexts (`total_c`) for each region.
- **Mean calculation**: Average coverage of CpG contexts (`total_c`) for each region, with violin and correlation plots for individual samples.
- **Regional fraction**: Methylation fractions for 5mC and 5hmC at the pre-specified minimum``DEPTH_FILTER`` value, with violin and correlation plots for individual samples.
- Processes each region file in the `REGIONS_DIR` independently. Therefore, expect individual results for each region file and extraction type.

```
#------------------------------------------------------------------------------
# STEP 2: FEATURE EXTRACTION
#------------------------------------------------------------------------------
printf "\n\n==================================================\n"
printf "STEP 2: RUNNING FEATURE EXTRACTION ON REGION FILES\n"
printf "==================================================\n"

# Process each region file for feature extraction
for bedfile in "$REGIONS_DIR"/*.bed "$REGIONS_DIR"/*.bed.gz "$REGIONS_DIR"/*.tsv "$REGIONS_
↪DIR"/*.tsv.gz; do
  # Skip if no files match pattern
  [[ -e "$bedfile" ]] || continue
  [[ "${METADATA##*.}" == "tsv" && $(realpath "$bedfile") == $(realpath "$METADATA") ]] &&␣
↪continue

  log_info "Processing region file: $bedfile"

  # Extract region file name without path and extension
  region_name=$(basename "$bedfile")
  region_name="${region_name%%.bed.gz}"
  region_name="${region_name%%.bed}"
  region_name="${region_name%%.tsv.gz}"
  region_name="${region_name%%.tsv}"

  # Set region-specific output directories
  region_output_dir="$RESULTS_DIR/$region_name"
  region_reports_dir="$REPORTS_DIR/$region_name"

  # 1. Extract count for total_c
  log_info "Running feature extraction: count total_c..."
  modality -v get count \
    --zarr-path "${ZARR_ARRAY[@]}" \
    --sample-sheet "$METADATA" \
    --bedfile "$bedfile" \
    --fields total_c \
    --output-dir "$region_output_dir"

  # 2. Extract mean for total_c with plots generated automatically
  log_info "Running feature extraction: mean total_c (plots generated automatically)..."
  modality -v get mean \
```

(continues on next page)

(continued from previous page)

```
    --zarr-path "${ZARR_ARRAY[@]}" \
    --sample-sheet "$METADATA" \
    --bedfile "$bedfile" \
    --fields total_c \
    --order-by-group "$GROUP_COLUMN" \
    --output-dir "$region_output_dir"

  # Copy mean HTML reports immediately
  for extract_dir in "$region_output_dir"/Extract_*; do
    if [[ -d "$extract_dir" ]]; then
      find "$extract_dir" -type f -name "*mean_extract_report*.html" -exec cp {} "$region_
↪reports_dir/" \; 2>/dev/null || true
    fi
  done

  # 3. Extract regional-frac for mc and hmc with depth filter (plots generated␣
↪automatically)
  log_info "Running feature extraction: regional-frac for mc and hmc with minimum depth
↪$DEPTH_FILTER (plots generated automatically)..."
  # Build the optional argument if DEPTH_FILTER is set
  MCV_ARG=""
  [[ -n "$DEPTH_FILTER" ]] && MCV_ARG="--min-coverage $DEPTH_FILTER"
  # --min-coverage 30, if DEPTH_FILTER=30 in config.txt

  modality -v get regional-frac \
    --zarr-path "${ZARR_ARRAY[@]}" \
    --sample-sheet "$METADATA" \
    --bedfile "$bedfile" \
    --fields mc hmc \
    $MCV_ARG \
    --order-by-group "$GROUP_COLUMN" \
    --output-dir "$region_output_dir"

  # Copy regional-frac HTML reports immediately
  for extract_dir in "$region_output_dir"/Extract_*; do
    if [[ -d "$extract_dir" ]]; then
      find "$extract_dir" -type f -name "*regional-frac_extract_report*.html" -exec cp {} "
↪$region_reports_dir/" \; 2>/dev/null || true
    fi
  done

  log_info "Feature extraction completed for $region_name"
done

log_info "Feature extraction HTML reports copied to region-specific directories"
```

Expected outputs:

- `Extract_*` directories: Organised by statistic type and region file
- `*.tsv.gz` files: Compressed quantitative results for each extracted statistic, by region and sample
- `*.html` reports: Interactive violin and correlation plots showing methylation statistics for `mean` and `regional-frac` extraction types.

Note: The optional regional dendrogram heatmap is not included in the Core Workflow script by default but can be added by modifying the `modality get mean` and/or `modality get regional-frac` command(s) to include `--heatmap`. Note that the heatmap will only be drawn if there are 200 regions in the region file, so it is not appropriate for use with

all promoters and gene bodies. Instead, add a custom subset region file to the Regions directory before running the Core Workflow.

Interpreting region-based Biological QC statistics:

- **Region-specific CpG coverage**: Should show adequate and consistent coverage across samples and may be used to inform the `DEPTH_FILTER` value for future analyses.
- **Methylation fraction distributions**: Violin and correlation plots reveal methylation heterogeneity.
- **Outlier samples**: Samples with unusual distributions may need investigation.

Customising statistics extraction:

You can modify this section of the script to:

- Add different field types (see *Zarr store* and *Extracting methylation statistics* sections in the main modality XPLR documentation).
- Adjust minimum coverage thresholds for regional fraction extraction.
- Include additional statistical summaries (`count`, `mean`, `sum`, `regional-frac`).
- Add an optional `--heatmap` parameter to generate regional dendrogram heatmaps for `mean` and/or `regional-frac` extraction types with 200 regions.

Step 3: DMR calling

This step identifies differentially methylated regions using statistical modelling, processing each region file individually. DMR calling is the core analytical component that identifies regions with significant methylation differences between two experimental groups.

What this script block does:

- Processes each `.bed` and `.tsv` file in the regions directory separately
- Performs statistical testing for differential methylation on each region file (see *Calling differentially methylated regions (DMRs)*)
- Analyses both 5mC and 5hmC modifications separately
- Analyses all available condition pairs from the metadata, preserving the sample sheet order for use with the `--condition-order` parameter. **Ensure the control group is the first one listed in the metadata file**; or analyses only the conditions specified in the `config.txt`, with the left-most condition name as the control group.
- Applies multiple testing correction (FDR)
- Incorporates covariates if specified

Note: The DMR calling does not enable `--overdispersion` for all analyses by default. For the Core Workflow, overdispersion correction is controlled by the `Overdispersion` parameter in the configuration file and can be set to "True" or "False". When set to "True", overdispersion correction is applied to all DMR analyses. The recommendation is to review DMR results prior to applying overdispersion correction, as this can lead to more conservative results. However, overdispersion correction may be useful when calling DMRs on large regions, such as gene bodies. For more information on when to use overdispersion, see *Overdispersion correction*.

Key analysis parameters:

- **Methylation contexts**: Analyses both `num_mc` and `num_hmc` by default
- **Depth filtering**: Should be applied according to the sequencing depth and methylation context coverage, and specified in the `config.txt` file
- **Covariate adjustment**: Controls for confounding variables when specified in the `config.txt` file

```
#------------------------------------------------------------------------------
# STEP 3: DMR CALLING
#------------------------------------------------------------------------------
printf "\n\n============================\n"
printf "STEP 3: RUNNING DMR CALLING\n"
printf "============================\n"

# Build covariate / depth_filter / condition_order / overdispersion flags if present
COV_FLAGS=""
DEPTH_ARG=""
CONDITION_ARG=""
OVERDISPERSION_ARG=""

[[ -n "$DEPTH_FILTER" ]] && DEPTH_ARG="--filter-context-depth $DEPTH_FILTER"
[[ -n "$CONDITION_ORDER" ]] && CONDITION_ARG="--condition-order $CONDITION_ORDER"
[[ "$OVERDISPERSION" == "True" ]] && OVERDISPERSION_ARG="--overdispersion"

if [[ -n "$COVARIATES" ]]; then
  # Use space-separated covariates directly for --covariates flag
  COV_FLAGS="--covariates $COVARIATES"
fi

# Run DMR calling on each region file individually
for bedfile in "$REGIONS_DIR"/*.bed "$REGIONS_DIR"/*.bed.gz "$REGIONS_DIR"/*.tsv "$REGIONS_
↪DIR"/*.tsv.gz; do
  # Skip if no files match pattern
  [[ -e "$bedfile" ]] || continue
  [[ "${METADATA##*.}" == "tsv" && $(realpath "$bedfile") == $(realpath "$METADATA") ]] &&␣
↪continue

  log_info "Running DMR calling on region file: $bedfile"

  # Extract region file name without path and extension
  region_name=$(basename "$bedfile")
  region_name="${region_name%%.bed.gz}"
  region_name="${region_name%%.bed}"
  region_name="${region_name%%.tsv.gz}"
  region_name="${region_name%%.tsv}"

  # Set region-specific output directory
  region_output_dir="$RESULTS_DIR/$region_name"

  # Run DMR calling
  # --covariates cov1 cov2, if COVARIATES="cov1 cov2" in config.txt
  # --filter-context-depth 15, if DEPTH_FILTER=15 in config.txt
  # --condition-order condition1,condition2, if CONDITION_ORDER="condition1,condition2" in␣
↪config.txt
  # --overdispersion is applied if OVERDISPERSION="True" in config.txt

  modality -v dmr call \
    --zarr-path "${ZARR_ARRAY[@]}" \
    --sample-sheet "$METADATA" \
    --condition-array-name "$GROUP_COLUMN" \
    --methylation-contexts num_mc num_hmc \
    --bedfile "$bedfile" \
    $COV_FLAGS \
    $DEPTH_ARG \
```

(continues on next page)

(continued from previous page)

```
    $CONDITION_ARG \
    $OVERDISPERSION_ARG \
    --output-dir "$region_output_dir"

  log_info "DMR calling completed for $bedfile"
done

log_info "All DMR calling completed"
```

Expected outputs:

- `DMR_*` directory: All DMR results (for all group pairs and contexts) are stored in a single directory with a timestamp
- `*.bed` files: DMR results in BED3+ format for each condition pair and methylation context
- `*.ini` files: Configuration files for genomic visualisation with `modality tracks` (not included as part of the Core Workflow). One `.ini` file is generated for each DMR `.bed` result file

Step 4: DMR visualisation

The final step generates visualisations for DMR results. This includes both unfiltered and filtered plots to help identify the most significant regions.

What this script block does:

- Creates HTML reports for each DMR result file (condition pair) and methylation context (5mC and 5hmC)
- HTML reports contain a volcano plot (modification difference vs -log10 q-value) and a p-value distribution histogram
- HTML reports are generated for both unfiltered DMR results, and filtered DMR results based on a q-value threshold of 0.05
- When filtering is applied, a new filtered, compressed DMR results file (`.bed.gz`) is also generated

---

Important: The filtering of DMR results is optional and can be removed by omitting the `--filter-qvalue` parameter in the `modality dmr plot` command below. Alternatively, the q-value threshold can be modified, or the additional `--filter-mod-difference` parameter can be added to filter based on the magnitude of methylation difference.

It is recommended to review the DMR results prior to applying filtering to determine the most appropriate thresholds, or to assess whether the analysis would benefit from overdispersion correction (if not already applied).

```
#------------------------------------------------------------------------------
# STEP 4: DMR VISUALISATION
#------------------------------------------------------------------------------
printf "\n\n======================================\n"
printf "STEP 4: GENERATING DMR VISUALISATIONS\n"
printf "======================================\n"

# Loop through each region directory
for region_dir in "$RESULTS_DIR"/*; do
  [[ -d "$region_dir" ]] || continue

  region_name=$(basename "$region_dir")
  # Skip if this is not a region directory (e.g., if it's a BioQC directory)
  [[ "$region_name" == "BioQC_"* ]] && continue

  log_info "Processing DMR visualisation for region: $region_name"

  # Set region-specific reports directory
```

(continues on next page)

(continued from previous page)

```
  region_reports_dir="$REPORTS_DIR/$region_name"

  # Loop through each DMR results directory within this region
  for dmr_dir in "$region_dir"/DMR_*; do
    [[ -d "$dmr_dir" ]] || continue

    # For each .bed file in the DMR directory
    for dmrresults in "$dmr_dir"/*.bed; do
      [[ -e "$dmrresults" ]] || continue

      log_info "Generating DMR plots for $dmrresults..."

      # Generate default plots - output to the same region directory
      modality -v dmr plot \
        --dmr-results "$dmrresults" \
        --output-dir "$region_dir"

      # Copy unfiltered DMR reports immediately
      for dmr_report_dir in "$region_dir"/DMR_Report_*; do
        if [[ -d "$dmr_report_dir" ]]; then
          find "$dmr_report_dir" -type f -name "*.html" -not -name "*max_q_*" -exec cp {} "
↪$region_reports_dir/" \; 2>/dev/null || true
        fi
      done

      # Generate filtered plots (q-value < 0.05) - output to the same region directory
      modality dmr plot \
        --dmr-results "$dmrresults" \
        --filter-qvalue 0.05 \
        --output-dir "$region_dir"

      # Copy filtered DMR reports immediately
      for dmr_report_dir in "$region_dir"/DMR_Report_*; do
        if [[ -d "$dmr_report_dir" ]]; then
          find "$dmr_report_dir" -type f -name "*max_q_*.html" -exec cp {} "$region_reports_
↪dir/" \; 2>/dev/null || true
        fi
      done

      log_info "DMR plots generated for $dmrresults"
    done
  done

  log_info "DMR visualisation completed for region: $region_name"
done

log_info "DMR HTML reports copied to region-specific directories"
```

Interpreting DMR visualisations:

Volcano plots:

- **Y-axis**: -log10(q-value) - higher values indicate more significant DMRs
- **X-axis**: Methylation modification difference - magnitude of effect
- **Colour coding**: Applied if multiple `Annotation` values are present in the supplied region files (`REGIONS_DIR`) e.g. promoter, enhancer, etc.
- **Significant regions**: Upper left/right quadrants (high significance, large effect)

- **Hyper- vs Hypo-methylation**: negative modification difference (hypo) vs positive modification difference (hyper) on the X-axis

p-value histograms:

- **Uniform distribution**: Indicates good statistical modelling
- **Enrichment near zero**: Suggests presence of true differential methylation, but over-representation may indicate the analysis would benefit from overdispersion correction. See *Overdispersion correction* for more information.
- **Unusual patterns**: May indicate technical issues or model violations

Quality assessment:

- **Volcano plot shape**: Expect to see a V-shape with most data points clustered near the base and centre. Significant hits may appear as outliers on one or both sides of the x-axis, with high values on the y-axis. The data distribution may or may not by symmetrical, depending on the extent of hyper- and hypo-methylation
- **Effect size distribution**: Biologically meaningful differences (typically `modification difference` > 0.1-0.2)
- **Annotation patterns**: Different region (`Annotation`) types may show distinct methylation patterns

1.6.4 Advanced customisation and extension

The Core Workflow is designed to be easily adapted for different research needs. Here are common modifications and extensions:

Customising analysis parameters

Adjusting coverage thresholds:

```
# In the config.txt or script
DEPTH_FILTER=15  # Increase for higher quality regions

# Or modify specific commands directly using the --min-coverage option
modality get regional-frac \
  --zarr-path "${ZARR_ARRAY[@]}" \
  --bedfile "$bedfile" \
  --fields mc hmc \
  --min-coverage 15 \  # Custom minimum coverage
  --output-dir "$RESULTS_DIR"
```

Adding different methylation contexts:

```
# Include modC analysis instead of 5mC and 5hmC.
modality dmr call \
  --methylation-contexts modc \
  # ... other parameters
```

Add overdispersion correction to DMR calling:

```
# Add additional DMR calling options
modality dmr call \
  --overdispersion \
  # ... other parameters
```

Performance optimisation tips

For large datasets:

- Use subset regions for initial testing
- Implement parallel processing for independent steps
- Monitor disk space and memory usage
- Consider using HPC environments for large-scale analysis

For production use:

- Add comprehensive logging and monitoring
- Implement checkpointing for long-running analyses
- Add email notifications for completion/failure
- Include resource usage reporting

1.6.5 Script completion and monitoring

The script includes basic completion reporting and performance tracking:

```
#------------------------------------------------------------------------------
# COMPLETION
#------------------------------------------------------------------------------
log_info "Pipeline completed successfully. Reports are available in $REPORTS_DIR and region-
↪specific subdirectories"

# Calculate and report duration
end_time=$(date +%s)
duration=$((end_time - start_time))
log_info "Script completed in $duration seconds"

# Exit cleanly
exit 0
```

Monitoring features:

- **Execution time tracking**: Reports total pipeline duration
- **Success confirmation**: Clear indication of successful completion
- **Output location reporting**: Directs users to results
- **Clean exit handling**: Proper script termination

Monitoring execution:

The script provides real-time logging output. You can also redirect output to a log file using the code below, or code it into the script as described in *Script setup and configuration*.

```
./core-workflow-v1.3.sh 2>&1 | tee workflow.log
```

Expected runtime:

Runtime varies significantly based on:

- **Dataset size**: Number of samples and genomic regions
- **Analysis complexity**: Number of methylation contexts and covariates
- **System resources**: Available CPU and memory
- **Storage speed**: Local vs network storage performance

Typical runtimes:

- Small dataset (10 samples, 1000 regions): 10-30 minutes
- Medium dataset (50 samples, 5000 regions): 1-3 hours
- Large dataset (100+ samples, 10000+ regions): 3-8 hours

This modular workflow provides a comprehensive foundation for methylation analysis using the modality XPLR toolkit. By understanding each component, you can adapt it to meet your specific research requirements while maintaining robust error handling and organised outputs.

Warning: **Version compatibility**: This pipeline is designed for modality XPLR version 1.0.0 and later. Some features may not be available in earlier versions. Check your modality XPLR version with `modality --version` before running.

Tip: **Getting help**: If you encounter issues, first try enabling debug mode (`DEBUG=true`) for detailed logging. The modular design allows you to run individual sections for troubleshooting specific steps.

1.7 Toolkit

1.8 Preparing for analysis

1.8.1 Input file types

modality XPLR offers a variety of analysis workflows, that leverage common input files and parameters.

The main input files are described in this section, and include:

- Zarr store files, `.zarrz`
- Sample sheet, `.csv` or `.tsv`
- Region files in BED3+ format, `.bed` or `.tsv`

Input files from external sources may be required, such as annotation references (`.gtf` or `.gff`). Descriptions of these file types and formats are also provided below.

Zarr store

The Zarr store is a compressed file format (`.zarrz`) used to store large datasets in a distributed manner. It is designed for efficient storage and retrieval of large arrays, making it suitable for high-throughput sequencing data such as methylation data. The Zarr format allows for chunked storage, enabling efficient access to subsets of data without loading the entire dataset into memory. This is particularly useful for large datasets that may not fit into memory all at once.

The Zarr store is a directory structure that contains metadata and data files. The metadata files describe the structure and properties of the dataset, while the data files contain the actual data. The Zarr store can be accessed using various programming languages and tools, including Python, R, and command-line interfaces.

In the context of modality XPLR, the Zarr store contains methylation count data for one of the cytosine contexts: CpG, CHG, or CHH. By default, the upstream duet pipeline generates CpG context data, but it can be configured to produce CHG or CHH context data as well. Each Zarr store contains data for only one context type at a time. For each site in the selected context, the Zarr store records the number of modified cytosines (5-methylcytosine and 5-hydroxymethylcytosine for duet evoC, modified cytosines for duet +modC) and unmodified cytosines, as identified in base-resolution sequencing data, along with total coverage and related counts (see table below for a full list).

For biomodal workflows, the Zarr store containing methylation data is produced by the upstream duet software. You will need the path to each Zarr store for your dataset to run the modality XPLR software. For simplified management, Zarr stores can be combined. See the section for *Handling Zarr stores* and the `modality zarr-utils join` operation for more information.

A list of count data variables contained in the Zarr store is provided below:

Data variable	Description
`c`	Unmodified C's.
`mc`	5-Methylcytosine (5mC) present in 6-base Zarr
`hmc`	5-Hydroxymethylcytosine (5hmC) present in 6-base Zarr.
`modc`	Combined 5mC and 5hmC signal (modC).
`total_c`	Total number of modified and unmodified C's (`mc` + `hmc` + `c`).
`total`	Sum of `total_c` + `n` + `other` (where `n` is an unresolved base).
`other`	SNPs in CpGs, i.e. where the CpG base is not a `c` or `n`.

Sample sheet

The sample sheet is a `.csv` or `.tsv` file that contains metadata about the samples in the Zarr store. The sample sheet can be created using any text editor. It must be saved as a `.csv` or `.tsv` file with the header row included. The sample sheet must include a column header, `sample_ids`, listing one or more sample IDs that match the sample IDs in the Zarr store. The sample sheet can also include additional columns with metadata, such as grouping information (e.g., sample type, condition), or covariates (e.g., age, smoker status, batch information). The analysis operations can handle both continuous and categorical metadata.

Formatting the sample sheet

modality XPLR expects UTF-8 encoding for sample sheet files but is fully compatible with ASCII characters. Do not use non-ASCII (Unicode) characters. Avoid using special characters, spaces, or commas in the sample sheet. We recommend using underscores (_) instead of spaces in the sample IDs and column headers. If you edit the sample sheet in a text editor, ensure it is saved as plain `.csv` or `.tsv` and that no extra spaces or formatting are introduced.

- Column Headers
  - Column headers must not contain spaces or commas.
  - Use only ASCII letters, numbers, and underscores (_) in column headers.
  - Each column header must be unique.
  - The first column must be named `sample_id` (case-sensitive) and contain unique sample identifiers.
  - Additional columns can be used for grouping (e.g., disease_stage) or covariates (e.g., age, smoker_status).
- Sample IDs
  - Sample IDs must not contain spaces or commas.
  - Use only ASCII letters, numbers, and underscores (_) in sample IDs.
  - Each sample ID must be unique within the sample sheet.
  - Sample IDs must exactly match those used in the Zarr store(s).

Important: Sample sheet metadata can be used for grouping samples in the analysis. Ensure that one of your columns can be used to group samples to use the grouping functions for Biological QC, DMR calling and Feature Extraction.

The sample sheet should be formatted as follows:

e.g., Sample IDs to be grouped by `disease_stage`, with the covariate `smoker_status` included in the analysis.

sample_id	disease_stage	smoker_status
sample1	control	no
sample2	I	yes
sample3	I	no
sample4	IV	yes
sample5	IV	no

Define a single region

Several of the tools in the modality XPLR software allow you to specify a single region of interest for analysis. This can be done by specifying the region in the command line using the format `chr` or `chr:<start>-<end>`. The start and end positions are optional, and indices are zero-based and half-open.

For example, to specify the region on chromosome 1 from position 1000 to 2000, use the following format: `chr1:1000-2000`. This will restrict the analysis to a specific region of the genome, which can be useful for focusing on regions of interest or reducing the size of the dataset during analysis. The region can be specified in the command line using the `--region` option. For example:

```
modality dmr call \
   --zarr-path path/to/data.zarrz \
   --methylation-contexts mc hmc \
   --region chr1:1000-2000 \
   --output-dir path/to/output
```

Note: If using `--region` in combination with `--window-size`, the specified region will be divided into windows of the specified size. Similarly, you can use a bedfile format to specify individual window sizes over a larger region of interest. See the **Region files** section below.

Region files

The user-supplied region file is a BED3+ format (`.bed` or `.tsv`) file that contains genomic regions of interest for analysis. It should include columns for the chromosome, start position, and end position of each region. The region file is optional but can be used to restrict the analysis to specific regions of the genome. The region file should be formatted as follows:

e.g., Region file with regions of interest.

chromosome	start	end
chr1	1000	2000
chr1	3000	4000
chr2	5000	1000
chr2	7000	3000

Annotations are optional but can be included in the analysis to provide additional context and information about the genomic regions of interest. Annotations can include information such as gene names, gene symbols, gene biotypes, and other relevant features. The annotations must be included in the region file and should be formatted as follows:

e.g., Region file with annotations.

chromosome	start	end	annotation	name
chr4	25250935	25251935	Promoter	KRAS
chr5	112706517	112707517	Promoter	APC
chr17	7687537	7688537	Promoter	TP53

Note: A region file can contain multiple different annotation classes, such as promoters, enhancers, or other genomic features.

1.8.2 Preparing annotation files

There are two types of annotation files that are used in the modality XPLR software:

- **Annotated region files**: Used to define genomic regions for analysis. E.g. as input to the `modality dmr call` and `modality get` tools.
- **Genome annotation GTF/GFF3 files**: Used to annotate new region definitions with `modality prepare-regions` (GFF3) or to place analysis outputs into a broader biological context with the `modality tracks` (GTF) tool.

You can download the human hg38 GENCODE v44 annotation files for the human (hg38) genome from biomodal, or prepare your own annotation files from public databases as described below.

To access biomodal demo data resouces, you need to install gcloud CLI and authenticate. See the download instructions here then use the following command:

```
gcloud storage cp gs://biomodal-data/annotation/gencode.v44.basic.annotation.gff3.gz .

gcloud storage cp gs://biomodal-data/annotation/gencode.v44.annotation.gtf.gz .
```

Alternatively, annotation files can be downloaded from e.g., the UCSC genome browser. The UCSC has a powerful Table Browser tool, that can be used to download annotation files for different reference genomes, either as complete files or as subsets of the genome. The output file type can be specified (`.gtf` or `.bed`) according to the utility you require.

The v44 human annotation above, was generated using the following steps:

1. Go to the UCSC Table Browser.
2. Select the following options:
   - clade: Mammal
   - genome: Human
   - assembly: hg38
   - group: Genes and Gene Predictions
   - track: Choose GENCODE v44
   - table: e.g., refGene, knownGene, ensGene, ncbiRefSeqGene
   - region: genome
   - output format: e.g. GTF or BED
3. Click on the "get output" button to download the annotation data.

   You are also able to perform additional customisation of the output e.g., of the track header or the rows in your BED output or select using the default settings.

   When you are ready, click on the "get output" button again and save the file on your local machine.
4. Note the file path for use in `modality` commands.

Pre-prepared BED files

The modality XPLR software includes a set of pre-prepared BED files for common genomic regions of interest. These `.bed.gz` files were prepared using:

- GENCODE v44 annotation for the human genome (hg38)
- GENCODE vM25 annotation for the mouse genome (mm10) - without "chr" prefix
- GENCODE vM25 annotation for the mouse genome (mm10p6)
- GENCODE M33 annotation for the mouse genome (mm39) - without "chr" prefix

The available region files are:

- Genes
- Promoters
- TSS
- CpG Islands
- CpG Shores
- CpG Shelves

These files can be used as input to the `modality dmr call` and `modality get` tools to restrict the analysis to specific regions of the genome. Contact **support@biomodal.com** or use the links below to obtain these files.

`Download human hg38 Annotated BED Files.zip`

`Download mouse mm10 Annotated BED Files.zip`

`Download mouse mm10p6 Annotated BED Files.zip`

`Download mouse mm39 Annotated BED Files.zip`

How were these files created?

- **Genes**: Read-through transcripts were removed from the GENCODE annotation file (i.e., transcripts with exons spanning >1 gene). This is because:
  - The genes involved are found on the same chromosome region on the same strand, typically adjacent to one another.
  - Their role is unclear
  - It's common to find them spanning up to 2 genes, but can span >2
  - They are ubiquitous
- **Promoters**: Promoters were defined as the 1kb region upstream of the gene `start` position and are 1000bp in length.
  - This is considered a good estimate and not a golden rule for all genes, as promoter distances from the 5'UTR will differ.
- **TSS**: Transcription start sites (TSS) were defined as the 200bp region upstream of the gene `start` position and are 400bp in length.

Note: Whilst genes may have multiple transcripts, we have used the GENCODE main gene definition to define a single TSS and promoter per gene for these files.

CpG shores and shelves files were created from the list of CpG islands in the GENCODE annotation file, using the following definitions:

- **CpG Shores**: CpG shores were defined as the 2kb region upstream and downstream of the CpG island.
- **CpG Shelves**: CpG shelves were defined as the 2kb region upstream and downstream of the CpG shores.

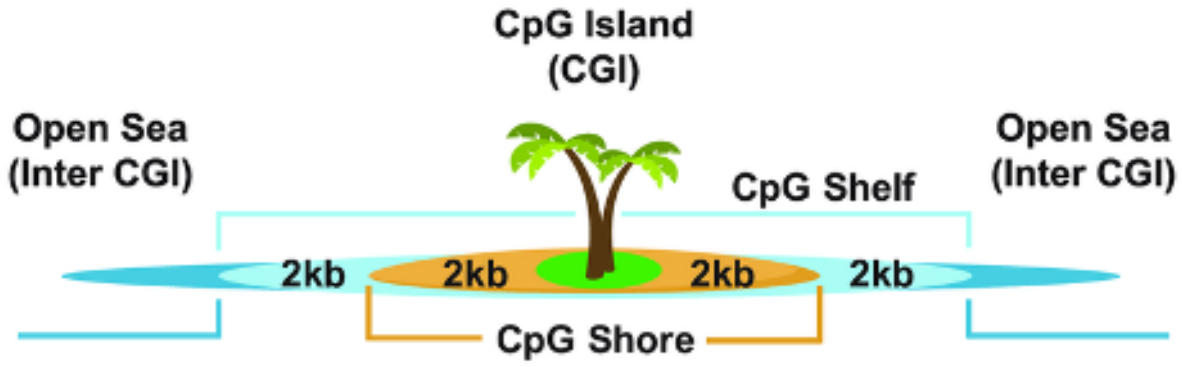

Figure 1.2: Definitions of CpG Islands, Shores and Shelves in pre-prepared BED files.

1.8.3 Top up sequencing & aggregating samples

When samples are sequenced across multiple sequencing runs (top-up sequencing), you can aggregate the data from these runs **during** analysis. modality XPLR handles multi-run data through analysis-time aggregation, where samples are combined at the point of running analysis.

Analysis-time aggregation workflow

Sample aggregation is performed during analysis execution and requires:

1. **Pre-joining Zarr stores**: For analyses with top-up sequenced samples in separate runs, we reccommend to join the Zarr stores before analysis. Use the `modality zarr-utils join` command to merge multiple Zarr stores. The sample IDs must be unique, with technical replicates differentiated with e.g. `_rep1`, `_rep2`. See *Handling Zarr stores* for detailed instructions. Once joined, samples are handled during analysis using the methods described below.

Note: If samples need to be renamed prior to joining, see *Renaming samples* for detailed instructions.

2. **Sample sheet preparation**: Create a sample sheet with a column that groups matched samples from different runs using a common identifier (e.g., the sample ID without the `_rep*` suffix for technical replicates).
3. **Analysis-specific handling**: Use the appropriate parameter for your chosen analysis tool:

Operation	Analysis tool.	Function
–aggregate by-group <label>	`modality biological-` `modality get`	Performs aggregation by summing count values in the Zarr store by a common metadata <label>, before analysis. Result files (BED3+) are reduced and plots show aggregated results.
–condition array-name <label>	`modality dmr call`	**Does not perform aggregation**. Defines DMR groups by metadata <label> prior to DMR calling. Technical replicates are considered as separate samples during DMR calling, even if they are assigned the same group label (`--condition-array-name`). This approach captures more sample variability.
–order-by-group <label>	`modality biological-` `modality get`	**Does not perform aggregation**. Use this feature to order samples by metadata <label>, to enhance visual order in plots. Result (BED3+) files are not ordered or reduced by metadata <label>.

Example workflow for top-up sequencing:

If you have sample "Patient_001" sequenced in two different runs (Run_A and Run_B), your sample sheet might look like:

sample_id	patient_id	condition
Patient_001_Run_A	Patient_001	control
Patient_001_Run_B	Patient_001	control
Patient_002_Run_A	Patient_002	treatment
Patient_002_Run_B	Patient_002	treatment

For `modality biological-qc` or `modality get`, use `--aggregate-by-group patient_id` to combine data from both runs for each patient. For `modality dmr call`, use `--condition-array-name condition` to group samples by treatment condition prior to DMR calling.

Important: When aggregating samples from multiple runs, ensure that:

- Sample identifiers in the Zarr stores match those listed in the sample sheet.
- The aggregation column in your sample sheet correctly groups samples that should be combined under a common ID.
- Run-level metadata is preserved in the sample sheet for quality control purposes.

Note: Aggregation is performed by summing count values for samples within the same group, making it suitable for combining technical replicates or top-up sequencing data.

Merging CpG strands

By default, modality XPLR merges CpG strands for analysis by summing the counts on the two strands. This is true for `export`, `biological-qc`, `dmr call` and `get`. Each entrypoint allows a `--disable-strand-merging` flag to disable strand merging if desired.

The modality `tracks` entrypoint is configured by the `.ini` file that is generated during DMR calling (`dmr call`), which contains a setting to enable or disable strand merging. This is because the `tracks` feature plots a methylation trace directly from the Zarr store data, and separate DMR modification difference and magnitude tracks that are based on DMR results files. Therefore, the `dmr call` setting for strand merging is carried over to the `.ini` so that tracks are drawn on matched data.

Here's an example of how strand merging works for 5mC (the same logic applies to 5hmC and modC):

If we had a CpG at reference position 10 on the + strand, and the corresponding CpG at position 11 on the - strand, the total number of C's may look like:

Position	Strand	mC	Total C
10	+	8	20
11	-	5	15

If strands are merged (default), the counts for the two strands are summed.

So the merged CpG will have `8 + 5 = 13` mC, and `20 + 15 = 35` Total C, so its methylation fraction is `13 / 35 = 0.37`.

If the `--disable-strand-merging` flag is used, strand merging is set to `False` and the counts for the two strands are kept separate.

The CpG at position 10 has a methylation fraction `8 / 20 = 0.4`, and the CpG at position 11 has a methylation fraction `5 / 15 = 0.33`.

Note: The + strand marks the reference position of the merged CpG, so in this example the merged CpG will be at position 10.

1.8.4 Saving modality XPLR output files

modality XPLR defaults to saving output files in the current working directory. To avoid outputs being overwritten, the output files or sub-folders created during analysis are time-stamped. It is recommended to specify an output directory for each analysis using the command option `--output-dir`, where applicable.

1.9 Handling Zarr stores

The `modality zarr-utils` command group provides tools for managing and manipulating Zarr stores used in methylation analysis. These utilities help you to inspect & rename samples as well as join Zarr stores from different runs for downstream analysis.

1.9.1 Overview

Zarr stores are the primary data format for modality XPLR workflows. The `zarr-utils` entrypoint offers subcommands to:

- Inspect sample IDs in a Zarr store.
- Rename sample IDs (with safety checks).
- Generate template mapping files for renaming.
- Join Zarr stores from different runs.

These utilities are especially useful when integrating data from multiple sources, standardising sample identifiers, or preparing data for collaborative projects.

1.9.2 Subcommands

The subcommands are:

- `rename-samples` which allows you to view and modify sample IDs in a Zarr store.
- `join` which allows you to combine multiple Zarr stores into a single store.

Renaming samples

This command lets you list and rename sample IDs in your Zarr store. The sample renaming can either be done directly by passing a command to define current and new values, or by passing a `.csv` or `.json` file for batch processing. You can generate a template `.csv` or `.json` file with your samples pre-populated for easier renaming of multiple samples at the same time.

Important: After renaming samples in a Zarr store, you must manually update the metadata (sample sheet) file to reflect the new sample IDs.

Inputs

`modality zarr-utils rename-samples` requires the following inputs:

- **Zarr store**: Path to a Zarr store, with samples to be renamed.
- **Mapping**: Either a `.csv` or `.json` file containing the mapping of current sample IDs to new sample IDs, or an inline mapping string (e.g., `old1:new1,old2:new2`).

Command and outputs

The `rename-samples` tool is invoked by typing `modality zarr-utils rename-samples` into the CLI terminal. To display a list of permitted arguments in the terminal, enter:

```
modality zarr-utils rename-samples --help
```

Typical usage examples:

```
modality zarr-utils rename-samples \
   --zarr-path path/to/data.zarrz \
   --mapping sample_mapping.csv \
   --output-path path/to/renamed_data.zarrz

# To overwrite the original store (use with caution)
modality zarr-utils rename-samples \
   --zarr-path path/to/data.zarrz \
   --mapping sample_mapping.csv \
   --overwrite
```

The following table describes the tool options:

OPTION	Require	Description
`-z`, `--zarr-p`	Yes	Path to the Zarr store to inspect or modify.
`-mp`, `--mappin`	No	Mapping of current to new sample IDs. Accepts a `.csv`, `.json` or inline mapping (e.g., "old1:new1,old2:new2"). Each sample being renamed is specified in the list.
`-otp`, `--output`	No	Path to save the modified Zarr store. If not provided, the original store will be overwritten (requires –overwrite).
`-ow`, `--overwr`	No	Allow overwriting the original Zarr store. Default: FALSE.
`-ls`, `--list-s`	No	Display current sample IDs and exit without making changes. Default: FALSE.
`-gt`, `--genera`	No	Generate a template mapping file (`.csv` or `.json`) for batch renaming and reducing entry of multiple sample IDs in the command. This option expects a file path, including the file name and extension (e.g., `sample_mapping.csv` or `sample_mapping.json`). The headers are set as `current_id` and `new_id` and must **not** be edited.

Analysis examples

Example 1: List sample IDs in a Zarr store

```
modality zarr-utils rename-samples \
   --zarr-path path/to/data.zarrz
   --list-samples
```

Example 2: Generate a template mapping file for renaming

```
modality zarr-utils rename-samples \
   --zarr-path path/to/data.zarrz \
   --generate-template sample_mapping.csv
```

Edit the generated template file to specify new sample IDs, then use it with `--mapping`.

Example 3: Rename sample IDs using a mapping file

```
modality zarr-utils rename-samples \
   --zarr-path path/to/data.zarrz \
   --mapping sample_mapping.csv \
   --output-path path/to/renamed_data.zarrz
```

Example 4: Rename and overwrite the original store (use with caution)

```
modality zarr-utils rename-samples \
   --zarr-path path/to/data.zarrz \
   --mapping sample_mapping.csv \
   --overwrite
```

Important: Safety and Best Practices:

- By default, the tool will not overwrite the original Zarr store unless `--overwrite` is explicitly set.
- If you do not specify `--output-path`, you must use `--overwrite` to confirm you want to modify the original file.
- The tool checks for conflicts (e.g., new sample IDs that already exist) and will raise an error if detected.
- Always review the list of sample IDs before and after renaming to ensure correctness.

Note: For more details on Zarr store structure and sample sheet requirements, see the *Zarr store* section above.

Combining Zarr store files

The modality XPLR software allows you to combine multiple Zarr stores into a single Zarr store for analysis, using the `modality zarr-utils join` command.

Important:

- Joining 5-base (duet +modC) and 6-base (duet evoC) Zarr stores is not supported.
- Caution should be taken when joining Zarr stores between duet pipeline versions where there could be significant changes impacting methylation quantification.

Joining Zarr stores using `modality zarr-utils join` is ideal for:

- Organising data from multiple sequencing runs.
- Avoiding entering multiple Zarr paths in the command line.

Where applicable, run-level information for each sample should be recorded in the metadata `--sample-sheet` for downstream analyses, such as the `modality dmr call` workflow.

Inputs

`modality zarr-utils join` requires the following inputs:

- **Zarr store(s)**: Paths to each Zarr store to be combined.
- **Output directory and filename**: Full path to the new compressed output file `.zarrz` containing the combined data.

Tip: To specify all Zarr stores in a single directory, you can use a wildcard pattern in a single path argument. E.g. `path/to/directory/*.zarrz`.

Command and outputs

The combine tool is invoked by typing `modality zarr-utils join` into the CLI terminal. To display a list of permitted arguments in the terminal, enter:

```
modality zarr-utils join --help
```

Typical usage example:

```
modality zarr-utils join \
   --zarr-path path/to/data1.zarrz path/to/data2.zarrz \
   --output-path path/to/new/combined.zarrz \
```

The following table describes the tool options:

OPTION	Required	Description
`-z, --zarr-path`	Yes	Two or more Zarr files to be combined. e.g., `path/to/file1.zarrz path/to/file2.zarrz` or `path/to/directory/*.zarrz` to combine all Zarr files in a sinlge directory.
`-otp, --output-pa`	Yes	Full path to an output combined zipped `.zarrz` file. e.g., `path/to/directory/combined.zarrz`

1.9.3 Exporting Zarr stores to community-supported file types

The `modality export` tool allows you to export methylation data stored in Zarr format to widely used file formats. This functionality is essential for integrating data with other bioinformatics tools, sharing data with collaborators, or preparing data for publication. The supported `export` formats are:

File type	CLI option	Description	IGV compatible
BedGraph	`bedgraph`	Export data to BedGraph format.	Yes
BedMethyl	`bedmethyl`	Export data to BedMethyl format.	Yes
Bismark	`bismark`	Export data to Bismark format.	Yes
CX-report	`cxreport`	Export data to CX-report format.	No
Quant	`quant`	Export data to quant file containing methylation counts on a per CpG basis.	No

The tool provides flexibility in selecting specific samples, regions, and data fields for export, ensuring compatibility with downstream workflows.

Inputs

`modality export` requires you to provide an export/file type e.g., `bismark` or `bedgraph` and has 1 mandatory input.

- **Zarr store**: Path to a Zarr store to be exported

This will however export all of your samples to the selected file type, if you want to export only a subset of your samples, you can use the `--sample-ids` option to specify a list of sample IDs to export. You can list the sample IDs in your Zarr store using the `zarr-utils` command, for more details, see *Example 1: List sample IDs in a Zarr store* in Renaming samples.

Command

The `export` tool is invoked by typing `modality export` into the CLI terminal. To display a list of permitted options and export formats, enter:

```
modality export --help
```

To display a list of permitted arguments for a specific export format, enter:

```
modality export <export-format> --help

# e.g., for Bismark
modality export bismark --help

# e.g., for BedGraph
modality export bedgraph --help

# e.g., for BedMethyl
modality export bedmethyl --help

# e.g., for CX-report
modality export cxreport --help

# e.g., for Quant
modality export quant --help
```

Typical usage example:

```
modality export bismark \
   --zarr-path path/to/data.zarrz \
   --tag genome \
   --modification-count-column modc \
   --denominator-column total_c \
   --skip-rows-zero-counts \
   --output-dir path/to/output
```

The following table describes the tool arguments for the export options:

OPTION	Re-quire	Description
-z, --zarr-path	Yes	Path to the Zarr file to be exported. e.g., path/to/data.zarrz
-s, --sample-ids	No	List of sample ID strings to include in the export. If not specified, all samples are included. One export file is produced per sample. e.g., sample1 sample2
-l, --tag	No	Tag string to append to the output file name. The output file name will use the format: <sample_id>.<tag>.<count_column>_<report>.<file_extension>. e.g., genome
-mcc, --modification-	No	Specify the column used for methylation counts. See *Zarr store* for a list of available data columns. Default: modc. e.g., modc
-dc, --denominator-c	No	Specify the column used to calculate the methylation fraction and coverage. See *Zarr store* for a list of available data columns. Default: total_c. e.g., total_c
-ac, --additional-c	No	Specify additional columns to include in the export. See *Zarr store* for a list of available data columns. e.g., total_c
-mcv, --min-coverage	No	Integer to specify the minimum context sequencing depth for filtering. Filters the positions if the mean coverage across all samples in each position is lower than the desired filter value. e.g., 10
-dsm, --disable-stra	No	Flag to disable strand merging to retain separate CpG sites on opposite strands (default: False, strands are merged). e.g., --disable-strand-merging
-x, --skip-rows-ze	No	Flag to turn ON, skip rows with zero counts. e.g., --skip-rows-zero-counts
-t, --tabix	No	Compress output files using bgzip and create Tabix index. This option is only available for BedGraph and BedMethyl formats. Requires installation of non-python dependencies (see installation section). e.g., --tabix
-otd, --output-dir	No	Directory to save the exported files. Default: current working directory. e.g., path/to/output
-r, --region	No	Restrict export to a specific genome region. Use the format chr or chr:<start>-<end>. e.g., chr1:1000-2000

For more information on strand merging, see the *Merging CpG strands* section.

Outputs

The `modality export` tool generates files in the specified format, saved in the output directory. The file names include the sample ID, tag, and data type. For example:

- **Bismark**: `sample1.genome.modc_bismark.bed.gz`
- **CX-report**: `sample1.genome.modc_cxreport.txt.gz`
- **BedGraph**: `sample1.genome.modc_bedgraph.gz`
- **BedMethyl**: `sample1.genome.modc_bedmethyl.gz`
- **Quant**: `sample1.genome.modc_quant.tsv.gz`

If `--tabix` is specified, an additional `.gz.tbi` output will be created per sample.

An additional provenance file is created in the output directory, named `modality_metadata_<YYYYMMDD>_<HHMMSS>.json`, which contains metadata about the export process, including the command used, input files, and parameters.

Example 1: Export bismark coverage report

```
modality export bismark \
   --zarr-path path/to/data.zarrz \
   --tag genome \
   --modification-count-column modc \
   --denominator-column total_c \
   --skip-rows-zero-counts \
   --tabix \
   --output-dir path/to/output
```

Example 2: Export bedgraph for specific region

```
modality export bedgraph \
   --zarr-path path/to/data.zarrz \
   --region chr1:1000-2000 \
   --output-dir path/to/output
```

Example 3: Export quant format for selected samples

```
modality export quant \
   --zarr-path path/to/data.zarrz \
   --sample-ids sample1 sample2 \
   --tag genome \
   --output-dir path/to/output
```

1.10 Prepare annotated regions

1.10.1 CpG-informed segmentation for region generation

The `modality prepare-regions` command generates and optionally annotates genomic regions from a Zarr dataset, using a CpG-informed segmentation approach. The bedfile output can then be used in downstream workflows such as `modality get` and `modality dmr call`, as with using *Pre-prepared BED files*. Instead of using fixed window sizes, CpG-informed segmentation defines regions based on the distribution of CpG sites in the genome. CpG-informed segmentation works by identifying clusters of CpG sites and segmenting the genome between them. New segment definitions are

based on user-controlled settings e.g., maximum region size, minimum number of CpG contexts per region, and a threshold for splitting continuous CpG positions into separate segments. A new segment starts when the gap between consecutive positions exceeds the provided value.

Inputs

`modality prepare-regions` requires the following input:

- **Zarr store(s)**: One or more paths to Zarr files containing methylation data.

The following inputs are **optional** inputs used for region generation and annotation. In the case of annotation files, one or multiple types may be provided.

Region generation

- CpG segmented region max size: Maximum size (in base pairs) for CpG-segmented regions
- CpG segmented split segment threshold: Threshold for splitting contiguous CpG positions
- CpG min contexts: Minimum number of CpG contexts required per region.

Region Annotation

- GFF3 annotation file: Path to a GFF3 annotation file used to annotate generated regions.
- CpG island annotation file: Path to a CpG island annotation file used for region annotation, e.g. downloaded from UCSC.
- Minimum overlap: Minimum number of bases or fraction of bases required to overlap for successful annotation.

Downloading annotation files

It is reccommended the GFF3 and CpG Island annotation files are downloaded from UCSC, according to your required parameters. See *Preparing annotation files* for more details.

Command

OPTION	Re-quire	Description
`-z, --zarr-path`	No	A Zarr file containing methylation data. e.g., `path/to/your/data.zarrz`
`-gff3, --gff3-annotation-fil`	No	Path to a GFF3 annotation file used to annotate generated regions.
`-cpg, --cpg-island-annotati`	No	Path to a CpG island annotation file (BED) used for region annotation.
`-cpgmxs, --cpg-segmented-regic`	No	Maximum region size (in base pairs) for CpG-segmented regions.
`-cpgmxd, --cpg-segmented-split`	No	Threshold for splitting contiguous CpG positions into separate segments. A new segment starts when the gap between consecutive positions exceeds this value.
`-cpgminc, --cpg-segmented-min-c`	No	Minimum number of CpG contexts required per region.
`-mo, --min-overlap`	No	Minimum number of bases or fraction of bases required to overlap for successful annotation.
`-otd, --output-dir`	No	Output directory for generated files. Defaults to the current directory.
`-at, --annotate`	No	Select one or more annotation(s) to include in the output. Use 'all' to include all annotations i.e., promoter, 5utr, 3utr, exon, gene and cpg_clusters (i.e. cpg_island, cpg_shore, cpg_shelve annotations).
`--help`	No	Show the help message and exit.

1.10.2 Analysis examples

Example 1 - Generate regions and annotate with GFF3

```
modality prepare-regions \
   --zarr-path path/to/data.zarrz \
   --gff3-annotation-file path/to/annotation.gff3 \
   --cpg-segmented-region-max-size 5000 \
   --cpg-segmented-split-segment-threshold 500 \
   --cpg-segmented-min-contexts 5 \
   --min-overlap 1 \
   --annotate promoter 5utr 3utr exon gene \
   --output-dir path/to/output
```

Example 2 - Generate regions and annotate with GFF3 & CpG islands

```
modality prepare-regions \
   --zarr-path path/to/data.zarrz \
   --gff3-annotation-file path/to/annotation.gff3 \
   --cpg-island-annotation-file path/to/cpg_islands.bed \
   --cpg-segmented-region-max-size 5000 \
   --cpg-segmented-split-segment-threshold 500 \
   --cpg-segmented-min-contexts 5 \
   --min-overlap 1 \
   --annotate all \
   --output-dir path/to/output
```

Example 3 - Generate regions without annotations

```
modality prepare-regions \
   --zarr-path path/to/data.zarrz \
   --cpg-segmented-region-max-size 5000 \
   --cpg-segmented-split-segment-threshold 500 \
   --cpg-segmented-min-contexts 5 \
   --min-overlap 1 \
   --output-dir path/to/output
```

Outputs

The modality prepare-regions command produces a BED-formatted file containing the generated genomic regions. These regions are derived using a CpG-based segmentation approach and are annotated using the provided GFF3, CpG island. Each row in the output file represents a single genomic region and includes the following columns:

- **Chromosome** - Chromomsome of the region
- **Start** - Start position of the region
- **End** - End position of the region
- **Name** - Name of the gene (if applicable e.g., not applicable for CpG islands, shelves and shores)
- **Annotation** - Type of feature e.g., promoter, exon etc.
- **Overlap** - Number of bases overlapping with the annotation.
- **Overlap_fraction** - Fraction of the region overlapping with the annotation.

Note: `Name`, `Annotation`, `Overlap` and `Overlap_fraction` will be delimited by a semicolon `;` if multiple region annotations are identified.

1.11 Biological QC

1.11.1 Quality control of methylation data

The `biological-qc` tool is used to perform biological quality control (QC) on methylation datasets stored in Zarr format. This command allows you to ask whether the data makes "biological sense" in the context of a broader experimental design (sample types, replicates, treatment or condition etc.).

Refer to the additional QC metrics generated by the upstream **duet software** for sequencing-level assessment of individual sample quality.

This command takes input Zarr store(s) and generates an HTML report containing a summary of Zarr and Sample Information, with key visualisations, including **Pearson correlation heatmaps** and **Principal Component Analysis (PCA) scatterplots**.

The report provides insights into:

- **Sample correlations**: Identifying relationships between samples to detect outliers or unexpected patterns, or to confirm expected relationships.
- **Data structure**: Using PCA to visualise the variance in the dataset and assess grouping or clustering of samples.

These outputs allow validation of data integrity according to experimental design, prior to downstream analyses such as calling differentially methylated regions (DMRs).

Inputs

`modality biological-qc` requires the following inputs:

- **Zarr store(s)**: Paths to each Zarr store to include in the analysis.
- **Sample sheet (recommended)**: Path to the sample sheet file containing metadata about the samples. Providing this file will allow colour-based grouping controls for the PCA plot on the HTML report.
- **Output directory (recommended)**: Path to the output directory where the HTML report will be saved.

Command

The `biological-qc` tool is invoked by typing `modality biological-qc` into the CLI terminal. To display a list of permitted arguments in the terminal, enter:

```
modality biological-qc --help
```

Typical usage examples:

```
modality biological-qc \
   --zarr-path path/to/data.zarrz \
   --output-dir path/to/output/directory
```

Using shortcuts:

```
modality biological-qc \
   -z path/to/data.zarrz \
   -otd path/to/output/directory
```

The following table describes the tool options and shortcuts:

OPTION	Re-quire	Description
`-z, --zarr-pat`	Yes	One or more paths to Zarr files. e.g., `path/to/your/data.zarrz`
`-r, --region`	No	String to restrict analysis to a specific genome region. Use the format `chr` or `chr:<start>-<end>`. The start and end positions are optional, and indices are zero-based and half-open. e.g., `chr1` or `chr1:1000-2000`
`-ss, --sample-s`	No	Path to a sample sheet file containing a column with the header `sample_ids`, listing sample IDs that match the sample IDs in the Zarr store. e.g., `path/to/your/metadata.csv` e.g., `path/to/your/metadata.tsv`
`-ag, --aggregat group`	No	Optional group name for aggregation. Reduces the dataset by summing count values for samples within each group. This affects all downstream analysis (correlation, PCA, etc.). Use this for merging technical replicates or combining samples by condition. Requires `--sample-sheet`. Cannot be used with `--order-by-group`.
`-og, --order-by`	No	Column name in the sample sheet to use for ordering samples in plots. Samples will be sorted by this group for visual organisation only. Does **not** perform aggregation - all individual samples remain in the analysis. Requires `--sample-sheet`. Cannot be used with `--aggregate-by-group`.
`-dsm, --disable-`	No	Flag to disable strand merging to retain separate CpG sites on opposite strands (default: False, strands are merged). e.g., `--disable-strand-merging`
`-otd, --output-d`	No	Option to specify a path to an output directory. If not specified, the default output path is to the current working directory. e.g., `path/to/output/directory`

For more information on strand merging, see the *Merging CpG strands* section.

Outputs

The `modality biological-qc` tool generates an **HTML report** and a provenance metadata file, `modality_metadata_<YYYYMMDD>_<HHMMSS>.json`, in the output directory.

The HTML report contains the following sections:

1. **Dataset Information**

Summary of the dataset, including the reference genome, 5- or 6-base data type (**duet +modC** or **duet evoC**), CG/CHG/CHH context, number of samples and number of contexts.

2. **Sample Information**

Sample metadata, including Sample IDs and mean CpG coverage that are read from the Zarr store. If a `--sample-sheet` metadata file is provided, this table shall also include sample group and covariate information that is read from the sample sheet.

Note: The number of contexts will depend on whether strands are merged (default) or not. Merging strands will show approximately half the number of contexts. The mean CpG coverage also depends on whether strands are merged or not. After merging, the CpG depth will closely match the NGS sequencing depth. If strands are not merged, the average CpG depth will be approximately half the NGS depth, due to a CpG-to-genome-wide coverage ratio of ~0.45-0.50.

---

3. **Pearson Correlation Heatmap**

Visualises the pairwise correlation between samples, helping to identify outliers or unexpected relationships. Use the `--order-by-group` option to control the order of samples on the axes based on a specified column in the sample sheet.

4. **Principal Component Analysis (PCA) Plots**

Displays the variance in the dataset, showing how samples cluster based on their methylation profiles. Controls to colour the data points by sample group or covariate are provided in the HTML report, allowing you to explore the data in more detail.

Important: Aggregation vs Ordering:

- Use `--aggregate-by-group` to reduce samples by summing within groups before all analyses. This creates group-level data (e.g., 3 groups instead of 10 samples) and is useful for technical replicates or combining samples by condition.
- Use `--order-by-group` to keep all individual samples but organise them by group in visualisations. This does not change the data, only how it's displayed.

These two options are mutually exclusive - you cannot use both simultaneously. In visualisations, the group order on the axis will follow the list order of the sample sheet.

1.11.2 Analysis examples

Example 1: Sample-wise QC on a single Zarr dataset

```
modality biological-qc \
   --zarr-path path/to/dataset.zarrz \
   --sample-sheet path/to/metadata.csv \
   --order-by-group label \
   --output-dir path/to/output
```

Example 2: Ordered sample-wise QC without aggregation

This example shows how to order samples by a grouping variable without aggregating:

```
modality biological-qc \
   --zarr-path path/to/dataset1.zarrz path/to/dataset2.zarrz \
   --sample-sheet path/to/metadata.csv \
   --order-by-group condition \
   --output-dir path/to/output
```

Example 3: Aggregated QC on large datasets from multiple Zarr stores

This example shows how to aggregate technical replicates:

```
modality biological-qc \
   --zarr-path path/to/dataset1.zarrz path/to/dataset2.zarrz \
   --sample-sheet path/to/metadata.csv \
   --aggregate-by-group replicate-group \
   --output-dir path/to/output
```

1.12 Calling differentially methylated regions (DMRs)

Differentially Methylated Regions (DMRs) are genomic regions that exhibit significant differences in DNA methylation levels between at least two different groups. A `region` represents the methylation status of summed CpGs within specified genomic coordinates, for each sample. Identifying DMRs is crucial for understanding the role of DNA methylation in various biological processes, including development, disease, and environmental or treatment responses.

1.12.1 Method used for calling DMRs

This tool can be used to identify the following types of DMRs, according to the biomodal chemistry used for sample preparation:

- **modC DMRs**: DMRs identified using 5-base (**duet +modC**)or 6-base (**duet evoC**) biomodal methylation data, where 5mC and 5hmC signal is combined to a single modC readout.
- **5mC DMRs**: DMRs identified using 6-base biomodal methylation data, where 5mC signal is used to identify DMRs.
- **5hmC DMRs**: DMRs identified using 6-base biomodal methylation data, where 5hmC signal is used to identify DMRs.

Important: The `modality dmr call` tool is compatible with Zarr stores that contain either 5-base or 6-base methylation data. See *Zarr store* for more details on the available fields in the Zarr store.

There are 6 steps in the method for calling DMRs:

1. **Data preparation**
   - Input data consists of methylation counts (e.g., 5mC, 5hmC, modC) stored in a Zarr store, with sample metadata provided in a `--sample-sheet`.
   - Optionally, regions of interest can be defined using a BED file or by specifying a `--window-size` for genome-wide tiling.
2. **Summing counts**
   - For each region (window or BED-defined), modality XPLR sums the counts of methylated and unmethylated cytosines within each region or window.
   - This produces, for each region and sample, the number of modified (e.g., `modc`) and total cytosines (`total_c`).
3. **Grouping samples**
   - Samples are grouped according to the experimental design (e.g., disease vs. control), as specified by `--condition-array-name`, which matches a column in the sample sheet.
4. **Statistical modeling**
   - For each region, modality XPLR fits a logistic regression model to test for differential methylation between groups.
   - The model compares the proportion of methylated cytosines between groups, adjusting for `--covariates` if specified and present in the provided sample sheet.
   - The null hypothesis is that there is no difference in methylation between groups.
5. **Wald test**
   - A Wald test is performed on the group coefficient from the logistic regression.
   - This yields a test statistic and a p-value for each region, indicating the likelihood that observed differences are due to chance.
6. **Multiple testing correction**
   - p-values indicate the probability of observing the data (or something more extreme) under the null hypothesis to test the likelihood that what you are observing is not occurring by random chance.

- p-values are adjusted for multiple testing using the Benjamini-Hochberg False Discovery Rate (FDR) method, resulting in a q-value (`dmr_qvalue`) for each region.
- q-values are adjusted p-values that represent the minimum FDR at which a test may be called significant.
- Multiple testing correction is necessary when many statistical tests are performed simultaneously, to control for false positives.
- See *Benjamini-Hochberg correction* for more details on the Benjamini-Hochberg correction.

The output of this analysis is a tab-delimited DMR result file adhering to the BED3+ format, for each group pair, and each methylation context that is specified in the command. In addition, the tool generates an additional `.ini` file per DMR result file, containing the parameters used for the analysis. The `.ini` file records the essential information required to view your DMRs visually in genomic context, including with genetic annotations using the plotting functionality `modality tracks`, see *Generate genomic track plots* for more details.

1.12.2 Typical DMR workflow

The Differentially Methylated Region (DMR) workflow analyses the methylation data to identify regions of the genome that are differentially methylated between experimental groups. The workflow is designed to be flexible and can be applied to a variety of experimental designs, including time-course experiments, treatment vs. control comparisons, and disease vs. normal comparisons.

DMRs can be used to identify hyper-methylated (increase in methylation compared to control) and hypo-methylated (loss of methylation compared to control) regions of the genome, which can provide insights into the underlying biological processes and mechanisms involved in the experimental conditions being studied. The direction of methylation change (hyper- or hypo-methylation) is described as the `mod-difference`.

The DMR workflow includes tools for visualising the data with key statistical metrics for hypotheses to be tested or patterns of methylation to be revealed between different experimental groups. The result data can then be combined with genome annotation tracks to provide biological context to the results.

A typical DMR workflow uses three `modality` tools:

1. `modality dmr call` identifies the statistically significant DMR(s)
2. `modality dmr plot` supports result filtering, QC, and visualisation of hypo- and hyper-methylated regions.
3. `modality tracks` for region-specific visualisation of DMRs, with annotated genome context.

This example demonstrates a complete workflow for identifying and visualising DMRs.

1. **Call DMRs**:

```
modality dmr call \
   --sample-sheet path/to/metadata.csv \
   --zarr-path path/to/data.zarrz \
   --condition-array-name disease_stage \
   --methylation-contexts mc hmc \
   --window-size 1000 \
   --output-dir path/to/output
```

2. **Plot DMR results**:

```
modality dmr plot \
   --dmr-results path/to/output/dmr_results.bed \
   --filter-qvalue 0.05 \
   --filter-mod-difference 0.1 \
   --output-dir path/to/output
```

3. **Generate genomic track plots**:

```
modality tracks \
   --tracks-file path/to/output/dmr_results.ini \
   --region chr1:100000-200000 \
   --output-dir path/to/output
```

The above three sections describe how to call DMRs, how to plot them and how to interpret them for insight into patterns and trends in DNA methylation.

Benjamini-Hochberg correction

The Benjamini-Hochberg (BH) procedure is a statistical method used to control the false discovery rate (FDR) when performing multiple hypothesis tests. One effect of BH correction, is that multiple DMRs may share the same q-value, particularly when the number of tests is large and the p-values are not uniformly distributed. This is because the BH procedure adjusts p-values based on their rank among all tests, and when many tests yield similar p-values, they may be assigned the same adjusted q-value.

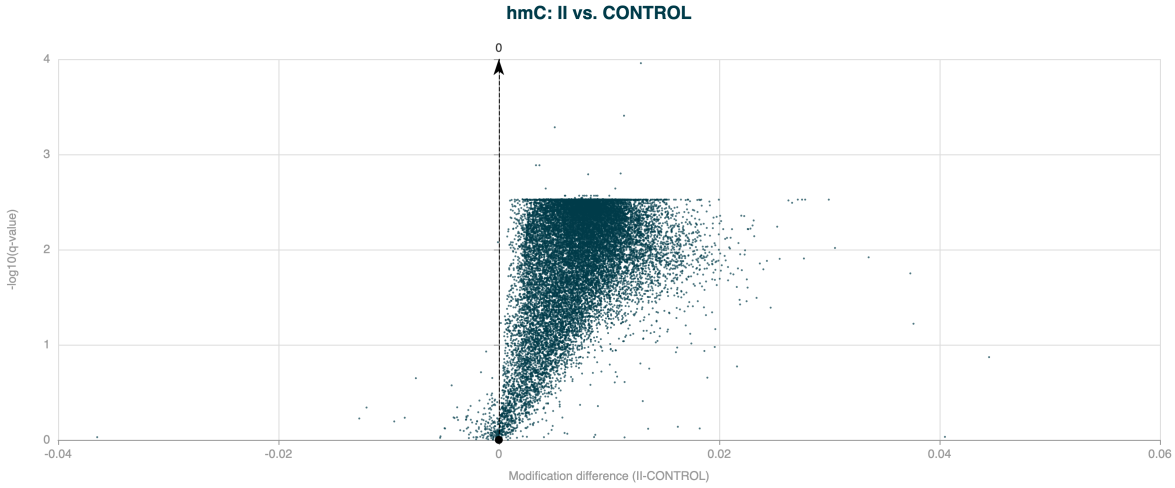

Figure 1.3: Volcano plot for 5hmC DMRs on gene bodies between Healthy Control and Stage II CRC cfDNA, showing many DMRs with similar q-values (plateau) after Benjamini-Hochberg correction.

Here's a breakdown of how the BH correction works and why multiple DMRs can have the same q-value:

Suppose we have `m` tests with raw p-values `p1, p2, ... , pm`:

1. First step of the BH is to sort them in ascending order, so that `p_sorted_1<p_sorted_2<...<p_sorted_m`.
2. Each sorted p-value gets a rank `i`. The smallest p-value has rank `1`, the highest has rank `m`.
3. The raw q-values are computed as `q_i = (p_sorted_i * m / rank_i)`.
4. Crucially, the BH procedure forces that q-values are always increased with rank: to do that, replace each q-value with the minimum of itself and all q-values ranked below it.

Here's how that might look in the data:

Rank	Test	p-value	Raw q-value	Adjusted q-value
1	A	0.0015	0.00750	0.00700
2	D	0.0028	0.00700	0.00700
3	E	0.0060	0.01000	0.01000
4	C	0.0250	0.03125	0.03125
5	B	0.0350	0.03500	0.03500

Notice how A has the lowest p-value, but the second lowest raw q-value? Therefore, the final step insures that the adjusted q-value of A is not higher than that of D by setting it to D.

This is not a bug, nor it is a problem - it's what the BH correction does, which remains one of the most commonly used and most robust corrections available. This effect most likely arises when there are a lot of very small p-values. Therefore, it's not surprising to observe this effect for gene-wide (i.e. very large region) p-values. Overdispersion correction can help mitigate this effect, but may not remove it completely. See *Overdispersion correction* for more details.

Overdispersion correction

When calling DMRs, the `modality dmr call` tool can apply an overdispersion correction to the statistical model.

When calling DMRs, the `modality dmr call` tool uses a logistic regression model to compare methylation levels between groups, assuming that methylation counts follow a binomial distribution (i.e., the variance is determined by the mean). The aim is to test the null hypothesis i.e., that there is no difference in `5mC`, `5hmC`, or `modC` methylation between the groups.

The comparison results in a p-value for any difference. A p-value tells you the likelihood that the difference being observed is due to random noise. The smaller the p-value, the more likely the difference is a genuine effect, and the null hypothesis can be rejected.

p-values hold values between 0 and 1 and when plotted, you would expect to see a flat distribution across most values but with a higher value close to 0. The peak close to 0 should represent a slight increase in frequency above the rest of the p-value distribution.

In real-world methylation data, the observed variance is often greater than expected under a simple binomial model due to biological and technical variability. This is called **overdispersion**, and can lead to increased levels of false positives, seen as a much higher peak close to 0 than expected. In the example below, DMR calling has been applied to human gene bodies, which are prone to overdispersion due to the large region size.

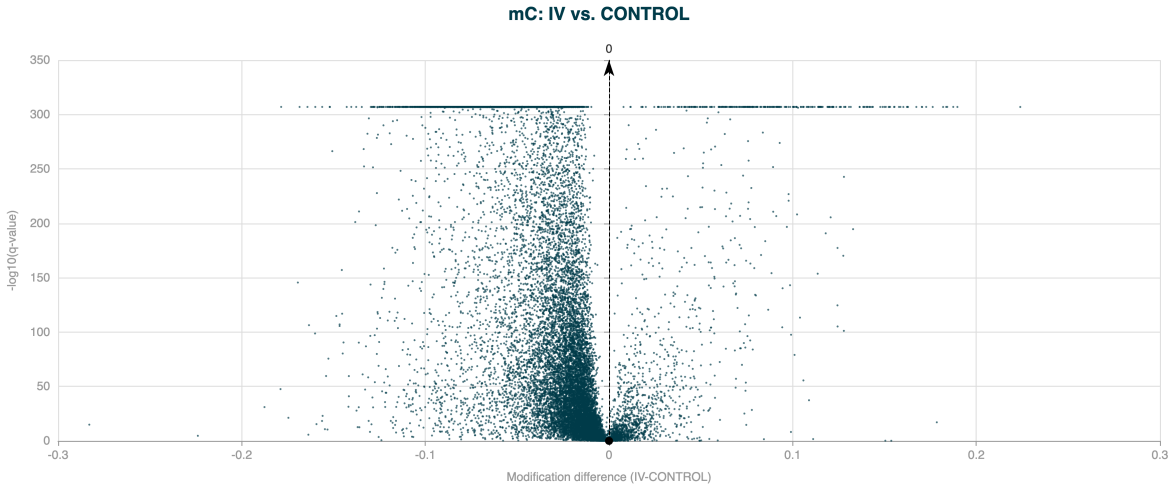

Figure 1.4: Volcano plot for 5hmC DMR results for human gene bodies, without overdispersion correction applied.

When `--overdispersion` is enabled, the model estimates an additional dispersion parameter () for each region, following the approach of McCullagh and Nelder (1989, Generalised Linear Models, Chapter 4.5). This parameter adjusts the standard errors of the regression coefficients. The Wald test statistic and p-values are then computed using these adjusted standard errors, providing more accurate significance estimates. In the example below, the DMR calling has been applied to the same human gene bodies, but with overdispersion correction applied.

Without overdispersion correction, the model can underestimate the true variance, leading to inflated test statistics and an excess of false positives (i.e., regions incorrectly identified as significant). Overdispersion correction ensures that statistical inference is robust, especially when analysing large regions, heterogeneous samples, or data with technical noise.

You can diagnose the need for this option by inspecting the p-value histogram from your DMR results: a very sharp peak near zero suggests overdispersion is present and correction may be needed.

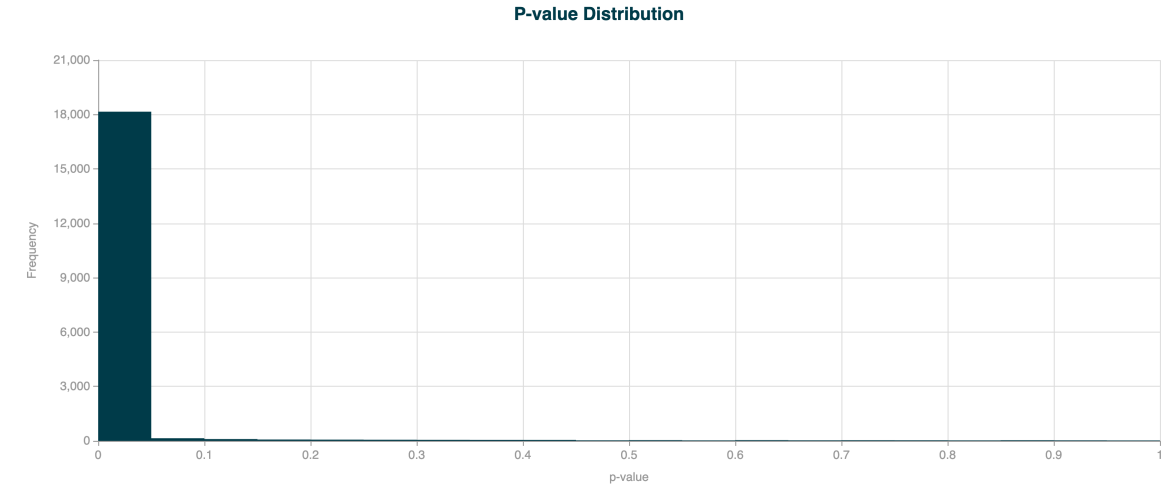

Figure 1.5: p-value distribution for 5hmC DMR results for human gene bodies, without overdispersion correction applied.

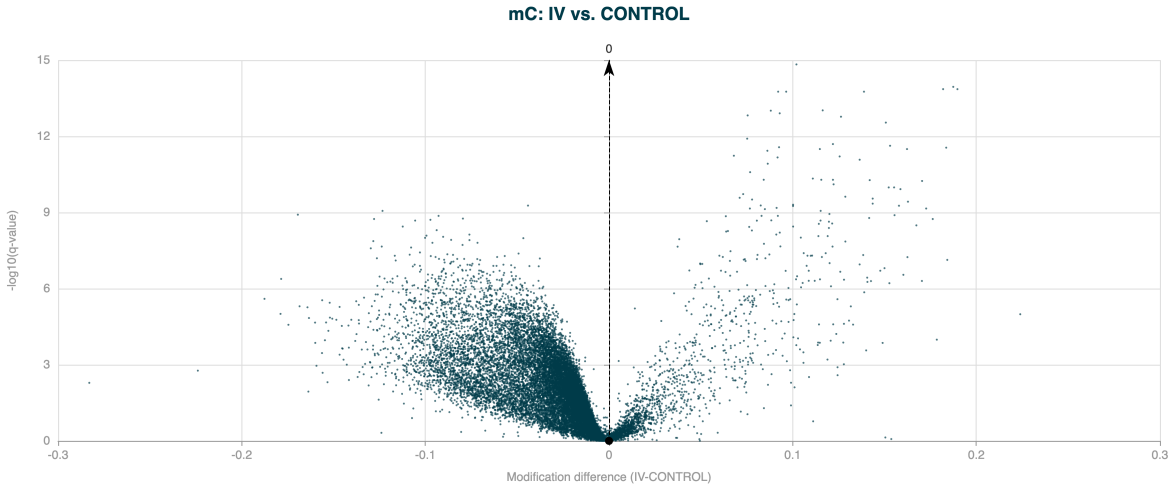

Figure 1.6: Volcano plot for 5hmC DMR results for human gene bodies, with overdispersion correction applied.

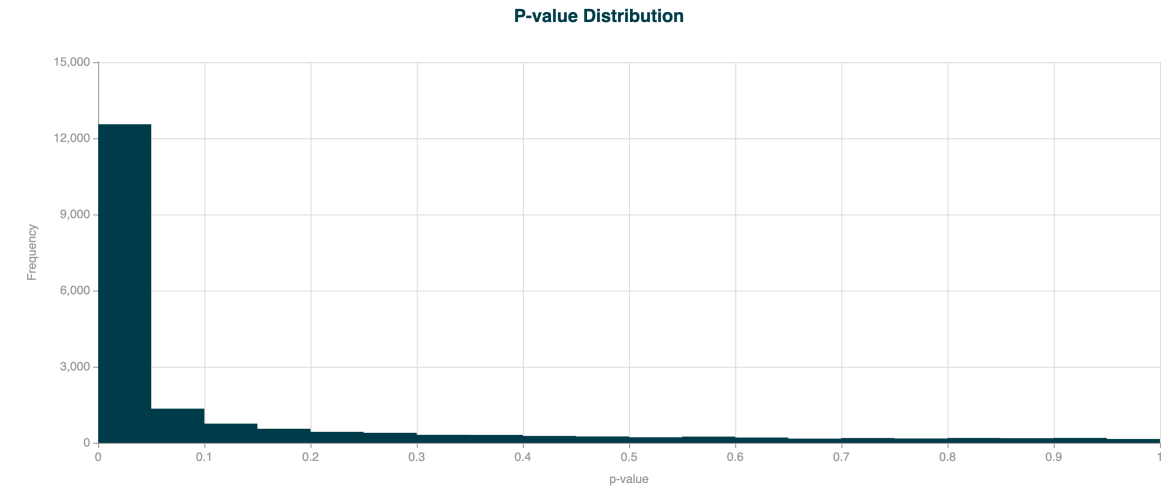

Figure 1.7: p-value distribution for 5hmC DMR results for human gene bodies, with overdispersion correction applied.

Sample size requirements for DMR analysis

We have implemented automatic sample size validation to prevent unreliable statistical inference in DMR calling. Logistic regression requires sufficient samples per group to estimate within-group variability - with too few samples (such as one sample per group), the model perfectly fits the limited data points but produces unreliable coefficients and artificially small standard errors, leading to meaningless p-values and increased false positive rates. Each group must have more samples than the number of model parameters (1 + number of optional covariates like age or batch) to ensure valid statistical inference - for example, with no additional covariates, each group needs at least 2 samples, increasing to 3 samples per group when adding one covariate. When insufficient samples are detected, the algorithm automatically skips the statistical test and returns conservative p-values of 1 and test statistics of 0 for all regions, effectively preventing false positives, and logs a warning message to inform users of the sample size limitation.

Users with insufficient samples can still examine the biological effect sizes by looking at the methylation differences between groups (mod_difference column) and fold changes (mod_fold_change column) in the results, which remain meaningful descriptive statistics even when statistical significance cannot be reliably assessed.

1.12.3 Calling DMRs

`modality dmr call` identifies regions where differentiated methylation occurs between different experimental groups. Examples of these groups are disease and normal, treated or un-treated or time-course experiments. A DMR is a defined genomic region where summed CpG sites show statistically significant differences in methylation between groups. Analyses may be scanning the whole genome, divided into fixed windows, or focussed on specific annotated regions provided in BED format.

Inputs

`modality dmr call` requires the following inputs:

- **Zarr store(s)**: Paths to each Zarr store to include in the analysis.
- **Sample sheet**: Path to the sample sheet file containing metadata about the sample groups, or covariates. Covariates can be continuous (integers or floats) or categorical (strings).
- **Output directory (optional)**: Path to the output directory where the result files will be saved.

Command

DMR calling is invoked by typing `modality dmr call` into the CLI terminal. To display a list of permitted arguments in the terminal, enter:

```
modality dmr call --help
```

Typical usage example:

```
modality dmr call \
   --sample-sheet path/to/metadata.csv \
   --zarr-path path/to/your/data.zarrz \
   --condition-array-name disease_stage \
   --methylation-contexts mc hmc \
   --bedfile path/to/regions.bed \
   --output-dir path/to/output/directory
```

The following table describes the tool options:

OPTION	Require	Description
-ss, --sample	Yes	Path to a sample sheet file containing a column with the header sample_ids, listing sample IDs that match the sample IDs in the Zarr store, and a column defining the experimental groups specified with the --condition-array-name option. Additional --covariates columns can be specified in the sample sheet but are not required. e.g., path/to/your/metadata.csv e.g., path/to/your/metadata.tsv
-z, --zarr-p	Yes	One or more paths to Zarr files separated by spaces. e.g., path/to/your/data.zarrz
-cn, --condit	Yes	String for grouping samples by a variable in the sample sheet, for DMR calling. The --condition-array-name value must exactly match the sample sheet column header. e.g., disease_stage.
-mc, --methyl	Yes	String to specify the methylation contexts to be analysed. Multiple contexts can be specified by separating them with spaces. The available contexts are: mC, hmC, and modc. Default value is modc e.g., mc hmc
-w, --window	No	Integer to specify the window size for DMR analysis. This option is mutually exclusive with the --bedfile option. e.g., 1000
-v, --covari	No	String to specify the covariates to be included in the analysis. Multiple covariates can be specified, which must match the column headers in the --sample-sheet metadata file. Use double quotes if the covariate name contains spaces. e.g., covariate1 "covariate two"
-b, --bedfil	No	Path to a BED file containing regions of interest for DMR analysis. Any columns included past Chromosome Start and End in your DMR results file. e.g., path/to/regions.tsv
-mcv, --min-co	No	Integer to specify the minimum context sequencing depth for filtering. Filters the positions if the mean coverage across all samples in each position is lower than the desired filter value. Applied before grouping by -cn. e.g., 10
-nc, --num-co	No	Integer to filter out regions with fewer than the specified number of contexts. Only regions with at least this many contexts will be included in the results. Default is 0 (no filtering). e.g., 5
-r, --region	No	String to restrict analysis to a specific genome region. Use the format chr or chr:<start>-<end>. The start and end positions are optional, and indices are zero-based and half-open. e.g., chr1 or chr1:1000-2000
-dsm, --disabl	No	Flag to disable strand merging to retain separate CpG sites on opposite strands (default: False, strands are merged). e.g., --disable-strand-merging
-od, --overdi	No	Include this flag to turn ON the overdispersion model. e.g., --overdispersion
-co, --condit	No	This feature is optional but recommended. If provided, the first condition in the list will be used as the reference (control) condition for DMR analysis. All other conditions will be compared to this condition, and then compared to each other. If not provided, sample order will match the --sample-sheet. e.g., HEALTHY I IV
-otd, --output	No	Option to specify a path to an output directory. If not specified, the default output path is to the current working directory. e.g., path/to/output/directory
-c, --compre	No	Option to specify whether to compress the DMR output file using gzip compression. Include this flag to turn ON compression.

For more information on strand merging, see the *Merging CpG strands* section.

Important: The --window-size and --bedfile options are mutually exclusive. You must specify either a window size for genome-wide tiling or provide a BED file with regions of interest for DMR analysis. If neither option is provided,

the tool will default to a window size of 1000 base pairs.

1.12.4 Outputs

The `modality dmr call` tool generates the following outputs in the specified output directory:

1. **DMR result files**: BED files containing the identified DMRs for each group pair and methylation context specified. The DMRs are annotated with the following columns:

Column	Description
`Chromoso`	Contig name.
`Start`	Start position.
`End`	End position.
`num_contex`	Number of contexts in the window or region.
`mean_cov`	Mean coverage of the contexts in the window or region.
`mean_mod`	Mean methylation level for group 1 (e.g., control).
`mean_mod`	Mean methylation level for group 2 (e.g., condition).
`mod_fold`	Difference in methylation level between groups 1 and 2, where mean(group 2) / mean(group 1). **Note:** Specifying the `--condition-order` option will determine the positive (hyper-) and negative (hypo-) methylation status of the DMR.
`mod_diff`	Difference in methylation level between groups 1 and 2, where mean(group 2) - mean(group 1). **Note:** Specifying the `--condition-order` option will determine the positive (hyper-) and negative (hypo-) methylation status of the DMR.
`test_sta`	The Wald test statistic associated with the p-value.
`dmr_pval`	The p-value of significance of the difference in methylation levels between the two groups.
`dmr_qval`	The FDR-corrected p-value.
`Annotati`	This is an **optional** column that would be carried over from your BED file if provided see `--bedfile` option for more details. There is no limit to how many columns are carried over.
`Name`	This is an **optional** column that would be carried over from your BED file if provided see `--bedfile` option for more details. There is no limit to how many columns are carried over.

The **DMR result files** are named according to the following convention:

- `dmr_<group1>_<group2>_<context>_window_<window_size>bp.bed`, or;
- `dmr_<group1>_<group2>_<context>_bedfile.bed` if a BED file is provided.

A DMR result file in BED3+ format will be generated per pairwise condition, per modification. For example, for a 6-base DMR analysis that includes both methylation contexts (mc and hmc), with a DMR sample sheet that contains three conditions (e.g. healthy, cancer_stage1, and cancer_stage4), six result BED files will be generated in the output directory.

Tip: Consider the use of `--output-dir` for efficient management of the output files.

2. `.ini` **files**:

`.ini` file(s) contain the parameters used for the analysis, including the paths to input files. The matched `.ini` files use the same naming convention as the DMR result files. There will be one `.ini` file per DMR result file.

The content of the `.ini` file can be changed, to control which plots are included when using `modality tracks`. This allows editing of figure titles, axis titles, and optimising the order of the plots for clear communication of your findings. Navigate to the *1. Auto-generated .ini file* section for more details.

3. **Provenance metadata**:

`JSON sidecar` files containing metadata about the analysis, including the command used, timestamp, version, and system information. The provenance metadata files are named according to the following convention:

`modality_metadata_<YYYYMMDD>_<HHMMSS>.json`.

1.12.5 Analysis examples

Example 1: DMR analysis on a single Zarr dataset for specific regions of interest

```
modality dmr call \
   --sample-sheet path/to/metadata.csv \
   --zarr-path path/to/your/data.zarrz \
   --condition-array-name disease_stage \
   --methylation-contexts mc hmc \
   --bedfile path/to/regions.bed \
   --output-dir path/to/output/directory
```

Example 2: DMR analysis on a single Zarr dataset for genome-wide screening by window size

```
modality dmr call \
   --sample-sheet path/to/metadata.csv \
   --zarr-path path/to/your/data.zarrz \
   --condition-array-name disease_stage \
   --methylation-contexts mc hmc \
   --window-size 1000 \
   --output-dir path/to/output/directory
```

1.13 Plotting differentially methylated regions (DMRs)

The `modality dmr plot` tool generates visualisations to aid the interpretation of DMR analysis.

The visualisations are **volcano plots** and **histograms of p-values** for each DMR result file, providing insights into the significance and magnitude of methylation differences between groups. These visualisations are helpful for result filtering, identifying significant DMRs and assessing the overall quality of the analysis.

The outputs of `dmr plot` provide insights into:

- The relationship between methylation differences and statistical significance: **Volcano Plots**
- The distribution of p-values across all tested regions, as a QC for DMR calling: **p-value Histograms**

Note: For different experimental reasons, there can be a very high frequency of p-values close to '0', if this is the case, refer to *Overdispersion correction* for additional details regarding the `--overdispersion` option available in `modality dmr call`.

1.13.1 Inputs

`modality dmr plot` requires the following inputs:

- **DMR result file(s)**: Paths to the output(s) of the `modality dmr call` tool, in `.bed` format. These files contain the DMRs identified in the analysis.
- **Output directory (recommended)**: Path to the output directory where the result files will be saved.

1.13.2 Command

The DMR visualisation tool is invoked by typing `modality dmr plot` into the CLI terminal. To display a list of permitted arguments in the terminal, enter:

```
modality dmr plot --help
```

Typical usage example:

```
modality dmr plot \
   --dmr-results path/to/dmr_results.bed \
   --filter-qvalue 0.05 \
   --filter-mod-difference 0.1 \
   --output-dir path/to/output/directory
```

The following table describes the tool options:

OPTION	Required	Description
`-dr`, `--dmr-results`	Yes	One or more paths to DMR result files. These files are generated by the `modality dmr call` tool. e.g., `path/to/dmr_results.tsv`
`-fq`, `--filter-qvalue`	No	Float to filter DMRs by q-value. Only DMRs with q-values the specified value will be included in the plots and filtered output. e.g., `0.05`
`-fm`, `--filter-mod-di`	No	Float to filter DMRs by modification difference. Only DMRs with `absolute` modification differences the specified value will be included in the plots and filtered output. e.g., `0.1`
`-otd`, `--output-dir`	No	Directory to save the generated plots. If not specified, the default output path is the current working directory. e.g., `path/to/output/directory`

1.13.3 Outputs

The `modality dmr plot` tool generates an HTML report per DMR results file input, in the specified output directory. Each HTML report contains the following sections:

1. **DMR summary table**

- Confirms the two Group names included in the DMR analysis, and which is the designated Control group and Test group, for calculating the modification difference, and directionality of hypo- and hyper-methylation.
- Shows the total number of regions (or windows) tested (after depth filtering).
- Shows the mean context depth, of all regions (or windows) passing q-value and/or mod-difference filter, if applied.
- Shows the mean number of contexts in all regions (or windows) passing q-value and/or mod-difference filter, if applied.
- If filtering was applied, the table will also show the number (and %) of regions (or windows) passing q-value and/or mod-difference filter.

2. **Volcano plot**

- Visualise the relationship between methylation differences (x-axis) and log transformed statistical significance $-\log_{10}$(q-value) (y-axis).
- DMRs are included based on the specified q-value and modification difference thresholds.
- If provided with a `--bedfile` with multiple values in the `Annotation` column, the volcano plot data points will be colour-coded and an Annotation legend will be shown on the plot. Note that input bedfile annotations are carried over to the DMR results file used for plotting.

- The volcano plot is interactive, allowing you to hover over points to see details about individual DMRs.
- The plot can be manipulated with embedded tools, such as zooming and panning, to focus on specific regions of interest.
- Use the chart tools to save the image as `.png`.

3. **p-value histogram**

- Shows the distribution of p-values across all tested regions.
- Provides an overview of the statistical significance of the results.
- Helps to assess the overall quality of the DMR calling process, namely the proportion of significant DMRs.
- If provided with a `--bedfile` with multiple values in the `Annotation` column, a stacked histogram will be shown together with a legend on the plot. Note that input bedfile annotations are carried over to the DMR results file used for plotting.

The output files are named according to the following convention:

- HTML report: `<dmr_result_file>_volcano.html`

1.13.4 Analysis examples

Example 1: Plotting DMR results with filters

```
modality dmr plot \
   --dmr-results path/to/dmr_results.bed \
   --filter-qvalue 0.05 \
   --filter-mod-difference 0.1 \
   --output-dir path/to/output
```

Example 2: Plotting multiple DMR result files

```
modality dmr plot \
   --dmr-results path/to/dmr1.bed path/to/dmr2.bed \
   --output-dir path/to/output
```

1.14 Generate genomic track plots

The `modality tracks` tool generates genomic track plots for visualising methylation data, differentially methylated regions (DMRs), and gene annotations. This tool uses an `.ini` configuration file to define the tracks to be plotted, allowing you to customise the visualisation according to their needs. The `.ini` file is viewed, modified or created in a standard text editor.

The outputs of `modality tracks` provide insights into:

- **Methylation traces**: Visualising methylation levels across genomic regions for individual samples or groups.
- **Methylation differences**: Highlighting differences in methylation levels between groups, specifically hyper- and hypo-methylation.
- **DMR tracks**: Displaying the location and significance of DMRs.
- **Annotations**: Adding biological context to the visualisation using GTF/GFF files.

These visualisations are essential for interpreting methylation data and understanding its biological relevance.

The `modality tracks` command will generate a single track plot (`.png` or `.pdf`) containing the individual tracks that are specified in the `.ini` file. The tracks in the track plot will appear in the order they are listed in the `.ini` file.

1.14.1 Inputs

`modality tracks` requires the following inputs:

- **ini file**: Path to the `.ini` file to be included in the analysis.
- **Output directory (optional)**: Path to the output directory where the result files will be saved.

This section details the data held in the `.ini` file, how it is structured, and which fields are editable. Instructions for creating a specific new `.ini` file are provided and these can be saved for future use.

1. Auto-generated `.ini` file

An `.ini` file is automatically generated by the `modality dmr call` tool when DMRs are called and saved in the output directory `--output-dir` specified in the command.

The `.ini` file is divided into sections. A section is indicated by square brackets `[ ]`, and is descriptive if the track type. The use of mc/hmc/modC is dependent on the type of DMR analysis that was run. The main sections are:

Section	The section purpose
`[DEFAULT]`	Settings for each of the tracks e.g. size,title
`[mc/hmc/modc methylation trace]`	Methylation trace track for 5mC, 5hmC, or modC
`[mc/hmc/modc methylation diff]`	Methylation difference track for 5mC, 5hmC, or modC
`[dmr]`	DMR bar track
`[spacer]`	Adds vertical space between tracks
`[gene]`	Annotation track
`[x-axis]`	x-axis configuration

Each of these sections has a set of components which are populated in the auto-generated `.ini` file.

An example `.ini` file, after running `modality dmr call` for `hmc` DMRs between two conditions, CONTROL and I (specified by the column header, `label`, in the sample sheet `metadata` file:)

```
    [DEFAULT]
width = 20
figure_title = CONTROL vs I - hmc DMRs
disable_strand_merging = False
min_coverage = 0

[mc methylation trace]
type = zarr_methylation_trace
zarr = path/to/data.zarrz
metadata = path/to/metadata.csv
ymin = 0
ymax = 1
height = 3
title = hmc fraction
modification_type = hmc
group = label
group_trace = True
group_points = True
group_names = CONTROL; I

[spacer]

[mc methylation diff]
type = methylation_diff
zarr = path/to/data.zarrz
```

(continues on next page)

(continued from previous page)

```
metadata = path/to/metadata.csv
height = 3
title = hmc difference
modification_type = hmc
ymin = -0.6
ymax = 0.6
group = label
group_names = CONTROL; I

[spacer]

[dmr]
type = dmr_bar
title = DMR magnitude
file = path/to/DMR_hmc_CONTROL__I.bed
dmr_header_present = True
y_min = -0.5
y_max = 0.5
height = 2

[spacer]

;[gene]
;type = gtf_gff
;file = path/to/gencode.gff3.gz
;file_type = gff3
;title = gencode
;height = 2

[x-axis]
```

Note: The `[gene]` section is commented out with a semicolon at the start of the line, meaning that the gene annotation track will not be included in the plot. To include the gene annotation track, remove the semicolon at the start of the line and ensure that the file path and type are correct. You will need to ensure that the GTF/GFF file is available and correctly formatted for the track to be built successfully.

Note: Users can choose whether they want to use a GFF3 or GTF file for annotation. The default expects a GFF3 but to pass a GTF file instead, change from `;file_type = gff3` to `;file_type = gtf` and exclude the semicolon ";".

Each section has its own components, the next table explains the purpose of the component, which are required, default settings and where input is needed.

Details of [DEFAULT]

Compo-nent	Setting	Purpose
width	default: 20	Width of the figure in inches
figure_tit	default: None	Title for the **entire** figure
disable_st	default: matches DMR call setting	Boolean flag to disable CpG strand merging across all methylation tracks
min_covera	default: 0	Minimum coverage filter: removes positions with coverage < min_coverage from tracks. Note: fractions are calculated with built-in min_coverage=1, so 0-coverage positions are not plotted regardless of this setting.

Details of [mc/hmc/modc methylation trace]

Component	Setting	Purpose
type	required	Must be zarr_methylation_trace
zarr	required	Path to the Zarr store
metadata	required	Path to the sample metadata CSV or TSV
ymin	default: 0	Minimum value for y-axis
ymax	default: 1	Maximum value for y-axis
height	default: 3	Height of the track in inches
title	default: None	Title for the track
modification_type	required	mc, hmc or modc
group	default: None	Metadata column to group samples
group_trace	default: True	Plot group traces (True/False)
group_points	default: True	Plot individual sample points (True/False)
group_names	default: None	Semicolon-separated group names, e.g., I; II

Details of [mc/hmc/modc methylation diff]

Component	Setting	Purpose
type	required	Must be methylation_diff
zarr	required	Path to the Zarr store
metadata	required	Path to the sample metadata CSV or TSV
ymin	default: -0.6	Minimum value for y-axis
ymax	default: 0.6	Maximum value for y-axis
height	default: 3	Height of the track in inches
title	default: None	Title for the track
modification_type	required	mc, hmc or modc
group	default: None	Metadata column to group samples
group_names	default: None	Semicolon-separated group names, e.g., I; II

Details of [dmr]

Component	Setting	Purpose
type	required	Must be dmr_bar
title	default: None	Title for the DMR bar track
file	required	Path to the DMR BED file
dmr_header_present	default: TRUE	Whether the BED file has a header
ymin	default: -0.5	Minimum value for y-axis
ymax	default: 0.5	Maximum value for y-axis
height	default: 2	Height of the track in inches

Details of [gene]

Important: To enable the gene annotation track, remove the semicolon at the start of only the following lines in the `.ini` file, including in front of the `[gene]` section header. Ensure that the GTF/GFF file is available and correctly formatted for the track to be built successfully.

Component	Setting	Purpose
`type`	required	Must be `gtf_gff`
`title`	default: None	Title for the annotation track
`file`	required	Path to the GTF/GFF annotation file
`file type`	required	`gtf` or `gff`
`height`	default: 2	Height of the track in inches

Details of [spacer]

There are no parameters for this section, when between sections it adds a vertical space between the tracks.

Details of [x-axis]

This is used to configure the x-axis and typically remains empty to deliver the default settings.

2. Customising `.ini` files

The default `modality tracks` visual is auto-generated but it can be customised. Table A shows default settings and ranges for components that can be used and Table B focusses on modifications of the `.ini` file. Changes are made by opening the `.ini` file in a text editor, updating with the changes and saving the file.

Important: Remember to save the updated `.ini` file and its path.

Table A: Default ranges

Component	Default	Range/Notes
width	20	>0
figure_title	None	Any string
disable_strand_merging	False	True/False
min_coverage	0	>=0, track filtering (see Note)
ymin, y_min	0/-0.5/-0.6	Any float, typically -1 to 1
ymax, y_max	1/0.5/0.6	Any float, typically -1 to 1
height	2 or 3	>0
group_trace	True	True/False
group_points	True	True/False
group_names	None	Semicolon-separated, e.g., `I; II`
dmr_header_present	True	True/False

Note: Understanding min_coverage in tracks:

There are two distinct coverage thresholds in the methylation plotting pipeline:

1. **Fraction calculation threshold** (always `min_coverage=1`): When methylation fractions are computed (e.g., mc / total_c), positions with coverage < 1 are automatically set to NaN to avoid division by zero. This happens internally and cannot be changed.
2. **Track filtering threshold** (default `min_coverage=0`): This is the parameter you set in the `.ini` file. It filters positions AFTER fractions are calculated:

- `min_coverage=0` (default): No filtering - all positions are kept, including those with NaN fractions from 0 coverage (these appear as gaps in plots)
- `min_coverage=N`: Removes positions where coverage < N before plotting

Example: If you set `min_coverage=10`, positions with coverage 0-9 will be removed from the plot. Positions with coverage 10 will be displayed with their methylation fraction values.

Table B: modifying the ``.ini`` file

Modification	How to modify
Add tracks (plots)	Copy and paste the section to duplicate the type of plot.
Remove tracks (plots)	Delete the section from the `.ini` file, or comment out the section by including `;` before each line to be ignored.
Change axis limits	Adjust `ymin`, `ymax`,``y_min``, y_max`` for each section.
Change grouping	Set `group` and `group_names` to match the metadata columns and values.
Add gene annotation	Remove `;`, e.g. change `;[gene]` to `[gene]` and set `file` to valid GTF/GFF file.
Customise titles	Edit `figure_title` and `title` fields to meaningful experimental titles for clarity.
Adjust panel order	Rearrange the complete sections in the `.ini` file.
Use p-value	Set `use_qvalues = False` in the `[dmr]` section to use raw p-values instead of q-values.

3. Creating an `.ini` file from scratch

When neither the default nor modification options for `.ini` file deliver the format of results required, a specific `.ini` file can be built. Using the details above as guidance for required components, here are the steps to building an `.ini` file:

1. Open a plain text editor such as TextEdit or a code editor such as Visual Studio Code.
2. Define the sections remembering they need to be enclosed in square brackets [Section name].
3. Add the key-value pairs e.g. title=hmc difference. The key is the 'title' and the value is 'hmc difference'. The key-value pairs are therefore the components and values required to create the plot required.
4. Save the file with an `.ini` extension (e.g., config.ini) and note the path for use with `modality tracks`.

Important: There are best practices when working with `.ini` files:

- Always check that the file paths are correct and accessible.
- Use consistent group names with the sample sheet.
- Consider the y-axis scale when comparing mC and hmC DMRs.
- Save custom `.ini` files to be able to re-use the format.

1.14.2 Command

The `tracks` visualisation tool is invoked by typing `modality tracks` into the CLI terminal. To display a list of permitted arguments in the terminal, enter:

```
modality tracks --help
```

Typical usage example:

```
modality tracks \
   --tracks-file path/to/config.ini \
```

(continues on next page)

(continued from previous page)

```
  --region chr1:100000-200000 \
  --output-dir path/to/output
```

The following table describes the tool options:

OPTION	Re-quirec	Description
`-tf`, `--tracks-file`	Yes	Path to the track configuration file (.ini) that defines the tracks to be plotted. The `.ini` file is created by the `modality dmr call` tool or can be created manually. e.g., `path/to/config.ini`
`-r`, `--region`	Yes	Genomic region to plot. Use the format `chr` or `chr:<start>-<end>`. The start and end positions are optional, and indices are zero-based and half-open. e.g., `chr1:100000-200000`
`-otd`, `--output-dir`	No	Path to save the generated plot. The output file format is determined by the file extension (e.g., .png, .pdf). e.g., `path/to/output/track_plot.png` or `path/to/output/track_plot.pdf`
`-if`, `--image-format`	No	Format for the output image file. Options: 'png', 'jpg', 'pdf', 'svg'. Default: 'png'. e.g., `png` or `pdf`
`-p`, `--padding`	No	Integer to specify the number of base pairs to extend the region on either side. Effectively, to zoom out. e.g., `1000`
`-dsm`, `--disable-strar`	No	Flag to disable strand merging to retain separate CpG sites on opposite strands (default: False, strands are merged). e.g., `--disable-strand-merging`

For more information on strand merging, see the *Merging CpG strands* section.

1.14.3 Outputs

The `modality tracks` tool generates a genomic track plot in the specified output file format (e.g., `.png` or `.pdf`). The plot includes the following tracks, as defined in the configuration file:

1. **Methylation trace**:
   - Visualise methylation levels across the specified genomic region.
   - Grouped by sample condition, as defined in the metadata file.
2. **Modification difference**:
   - Highlight differences in methylation levels between groups.
   - Smoothed and raw differences are displayed.

The solid line in the methylation trace and methylation difference tracks provides a smoothed representation of the data for improved readability. Smoothing is performed using a coverage-weighted Gaussian kernel-based approach. The smoothing scale is adaptive based on the length and density of the region that is being plotted.

3. **DMR track**:
   - Display the location and significance of DMRs.
   - Positive and negative DMRs are colour-coded.
   - Uses q-values by default.
4. **Annotation**:
   - Add biological context using GTF/GFF files.
   - Display gene names, exons, and other features.

The output file is named according to the specified path and file extension, e.g., `track_plot.png`.

1.14.4 Analysis examples

Example 1: Generating a track plot for a specific region

```
modality tracks \
   --tracks-file path/to/config.ini \
   --region chr1:100000-200000 \
   --output-dir path/to/output
```

Example 2: Adding padding to the genomic region

```
modality tracks \
   --tracks-file path/to/config.ini \
   --region chr1:100000-200000 \
   --padding 1000 \
   --output-dir path/to/output
```

1.15 Extracting methylation statistics

The `modality get` command is designed to extract methylation statistic summaries for genomic regions or windows, from methylation data stored in Zarr format, from the duet software. This analysis can be used for creating datasets for classification algorithms, modelling gene expression, and examining potential coverage or methylation biases. This tool supports four key statistical summaries under the `modality get` command:

Subcommand	Description	Example Use Case
`count`	Counts the number of CpGs for selected `--fields` in each region.	Identify the number of CpG contexts in each region or window, to allow filtering or preparation of region lists ahead of running other analyses.
`sum`	Computes the sum of values for selected `--fields` in each region.	Summarises total modifications across a region.
`mean`	Computes the mean value for selected `--fields` in each region.	Assess mean coverage across a region.
`regional-frac`	Computes the fraction for selected `--fields` over `total_c` (denominator) in each region.	Calculate the fraction of a field over `total_c` for methylation fraction analysis.

The four subcommands provide a specific type of statistical analysis, enabling you to explore methylation patterns across genomic regions of interest. The `modality get` command serves two main purposes:

1. Extraction of methylation statistics into a `tsv.gz` file, enabling feature-set generation and downstream analysis.
2. Visualise methylation statistics in HTML reports, to explore the distribution, correlation and hierarchical clustering of methylation across regions.

A typical Feature Extraction workflow requires:

- Specified data input.
- Selection of a statistical operation and methylation context(s) of interest.

- Specified genomic region(s) of interest or fixed window size.

Which in turn provides:

- Summarised methylation data as `tsv.gz`.
- HTML reports with interactive plots that can be saved as `.png`, for `mean` and `regional-frac` subcommands.

The next three sections explain how to extract the DNA methylation statistics, create and interpret HTML visualisations for insight into patterns in DNA methylation.

1.15.1 Inputs

The `modality get` command requires the following inputs:

1. **Zarr path**: Path to the Zarr file(s) containing methylation data.
2. **Fields**: Type(s) of cytosine to extract. Allowed `--fields` vary by subcommand (`count`, `sum`, `mean`, and `regional-frac`).
3. **Region definition**: Either a BED3+ file defining multiple genomic regions of interest, a single specified region, or a window size for genome-wide analysis.

Note: **Field values**: See *Zarr store* section for more information about available field values.

Optional input files

Optional file inputs are determined by the analysis being carried out:

- A text file in BED3+ format, listing the specific region(s) of interest.
- A text file in CSV or TSV format, containing sample group or condition metadata.

1.15.2 Commands

The `modality get` command is invoked by typing `modality get` into the CLI terminal. The command format is `modality get <subcommand> <OPTIONS>`.

Subcommands

To display a list of permitted arguments for a specific subcommand, type:

```
modality get <subcommand> --help

# e.g., for the `mean` subcommand
modality get mean --help

# e.g., for the `count` subcommand
modality get count --help

# e.g., for the `sum` subcommand
modality get sum --help

# e.g., for the `regional-frac` subcommand
modality get regional-frac --help
```

Example usage:

This illustration shows the subcommand with instructions to:

- Bring in specific Zarr store and .bed files
- Identify the modifications of interest (`--fields`)
- Visualise the results in violin, pearson correlation and regional heatmap plots
- Save the output files (Data `.tsv.gz` and Visualisations `.html`) to a specific output folder

```
modality get regional-frac \
   --zarr-path path/to/data.zarrz \
   --bedfile path/to/regions.bed \
   --fields mc hmc \
   --heatmap \
   --output-dir path/to/output
```

Common options

The following table describes the common options available. Note that not all options are available for all subcommands:

OPTION	Re-quire	Description
`-z`, `--zarr-pat`	Yes	Path to the Zarr file(s) to be analysed. e.g., `path/to/data.zarrz`
`-f`, `--fields`	Yes	Type(s) of cytosine to extract. Note that `mc` is synonymous to `num_mc` in the Zarr store. Allowed fields vary by subcommand. e.g., `modc`, `mc`, `hmc`, `total_c`
`-mcv`, `--min-cove`	No	Integer to specify the minimum context sequencing depth for filtering. Filters the positions if the mean coverage across all samples in each position is lower than the desired filter value. Applied before aggregation with `--aggregate-by-group`. e.g., `10`
`-nc`, `--num-cont`	No	Integer to filter out regions with fewer than the specified number of contexts. Only regions with at least this many contexts will be included in the results. Default is 0 (no filtering). e.g., `5`
`-b`, `--bedfile`	No	Path to a BED file defining genomic regions of interest. Mutually exclusive with `--window-size`. e.g., `path/to/regions.bed`
`-w`, `--window-s`	No	Integer to specify the window size for analysis. Mutually exclusive with `--bedfile`. e.g., `1000`
`-r`, `--region`	No	String to restrict analysis to a specific genome region. Use the format `chr` or `chr:<start>-<end>`. The start and end positions are optional, and indices are zero-based and half-open. e.g., `chr1` or `chr1:1000-2000`
`-ss`, `--sample-s`	No	Path to a CSV file containing sample metadata. e.g., `path/to/your/metadata.csv` e.g., `path/to/your/metadata.tsv`
`-ag`, `--aggregat group`	No	Optional group name for aggregation. Reduces the dataset by summing count values for samples within each group. This affects all downstream analysis (correlation, PCA, etc.). Use this for merging technical replicates or combining samples by condition. Requires `--sample-sheet`. Cannot be used with `--order-by-group`.
`-og`, `--order-by`	No	Column name in the sample sheet to use for ordering samples in plots. Samples will be sorted by this group for visual organisation only. Does **not** perform aggregation - all individual samples remain in the analysis. Requires `--sample-sheet`. Cannot be used with `--aggregate-by-group`.
`-hm`, `--heatmap`	No	Flag to include a regional heatmap in the HTML report. Only available for `regional-frac` and `mean`. Drawing of the heatmap is limited to 200 regions. Default: FALSE.
`-otd`, `--output-d`	No	Directory to save the output files. Default: current working directory. e.g., `path/to/output`
`-cs`, `--case-sen`	No	Flag to turn ON case-sensitivity for `--aggregate-by-group` and `--order-by-group` labels in the `--sample-sheet`. This option will preserve letter casing for label values. Default: FALSE. e.g., `--case-sensitive`
`-t`, `--tabix`	No	Compress output files using bgzip and create Tabix index. Requires installation of non-python dependencies (see *Non-python dependencies*). e.g., `--tabix`
`-dsm`, `--disable-`	No	Flag to disable strand merging to retain separate CpG sites on opposite strands (default: False, strands are merged). e.g., `--disable-strand-merging`

For more information on strand merging, see the *Merging CpG strands* section.

HTML report visualisations

Important: The HTML report is generated with `get mean` and `get regional-frac` commands, containing violin and Pearson correlation plots for each modification type `5mC`, `5hmC`, `5modC`. If specified, a regional heatmap is also included for `regional-frac` and `mean`.

Plotting is not supported for `count` and `sum` subcommands, as these operations do not produce meaningful visualisations.

HTML files include multiple plots, with tools for performing basic manipulations such as saving the image in PNG format. A description of supported plots is provided below:

- **Violin**: A violin plot visually represents the density of data across a range of values. It will show DNA modification distribution across the DNA region. The violin also includes box and whisker plots.
- **Correlation**: A Pearson correlation matrix plot which visualises the relationship between two or more variables. It does not show causality. The deeper the colour the higher the correlation.
- **Regional heatmap**: A heatmap visualising the correlation and heirarchical clustering of methylation profiles across genomic regions and samples.

Optimising plots

1. **Plotting subsets of samples**

When working with a large number of samples, it may be useful to restrict the plotting to specific samples or group the analysis output based on sample labels. This may be for speed or where some samples have a specific biological element that is of interest.

The following options apply to plotting functions:

- `--aggregate-by-group`: Aggregates samples based on a common ID or group name column in the `--sample-sheet` where group labels are listed e.g. control or condition 1, treated or untreated, healthy or disease stage 1, etc. This action will reduce output results files and plots.
- `--order-by-group`: Orders samples based on a specific column in the `--sample-sheet` where group labels are listed. This action will not aggregate samples but will order them on the plot axes for easier visual interpretation. Result files are not reduced.
- `--case-sensitive`: This option allows for case-sensitive group labels, negating the default behaviour of converting group labels to uppercase in plots (see below).

2. **Group labels**

For plotting, the group label values are converted to uppercase by default. For example, `Healthy` and `healthy` would be assigned to the same group, `HEALTHY`. To override this behaviour, use the `--case-sensitive` option to report `Healthy` and `healthy` as separate groups.

1.15.3 Outputs

According to the `modality get` subcommand and analysis performed, the software shall return the following result files to the output directory. If an `--output-dir` path is not provided, the result files shall be saved in the current working directory:

1. **Compressed TSV result file(s)**:

- Contains the extracted statistics for each genomic region.
- Columns include genomic coordinates, extracted values, and additional metadata in BED format.
- The file name is based on the subcommand used, e.g., `<methylation-context>_<subcommand>.tsv.gz`.

2. **HTML report**:

- Visualise the extracted statistics.

- Saved in the specified output directory as HTML files.

3. **Provenance metadata**:

- JSON sidecar files containing metadata about the analysis, including the command used, timestamp, version, and system information.
- The provenance metadata files are named according to the following convention: `modality_metadata_<YYYYMMDD>_<HHMMSS>.json`.
- The metadata includes the command used, timestamp, version, and system information.

1.15.4 Analysis examples

Example 1: Counting CpGs in genomic regions

This operation counts the number of CpGs in each region with a specific `--fields` modification status. It can also be used in conjunction with a minimum coverage setting.

```
modality get count \
   --zarr-path path/to/data.zarrz \
   --bedfile path/to/regions.bed \
   --fields total_c \
   --output-dir path/to/output
```

Example 2: Summing methylation counts across regions

This operation sums the total number of the selected `--fields` modifications across each region.

```
modality get sum \
   --zarr-path path/to/data.zarrz \
   --bedfile path/to/regions.bed \
   --fields mc \
   --output-dir path/to/output
```

Example 3: Calculating mean context value in each region

This operation can be used to assess the average coverage per CpG. For example, average number of CpGs covered at a specific depth in sequencing (`total`), or with a cytosine basecall (`total_c`).

```
modality get mean \
   --zarr-path path/to/data.zarrz \
   --bedfile path/to/regions.bed \
   --fields total_c \
   --output-dir path/to/output
```

Example 4: Computing regional methylation fraction over total_c

This operation is suitable for methylation fraction analysis and will calculate the proportion of a `--fields` out of the total number of unmodified and modified Cs (`total_c`), for each genomic region or window.

```
modality get regional-frac \
   --zarr-path path/to/data.zarrz \
   --bedfile path/to/regions.bed \
   --fields mc hmc \
   --heatmap \
   --output-dir path/to/output
```

1.16 Audit trail

1.16.1 Terminal feedback

modality XPLR provides immediate feedback in the terminal when commands are executed, showing:

- The command being run with all specified parameters
- Progress indicators for long-running operations
- Confirmation of output file creation
- Summary information upon successful completion

This feedback helps to confirm the command was interpreted correctly and analysis progression is being tracked.

Below is an example of the terminal output and provenance feedback generated during a typical `modality biological-qc` run:

```
2025-06-19 12:15:19  Provenance information:
2025-06-19 12:15:19  cmd='path/to/env/bin/modality biological-qc --zarr-path path/to/data.
↪zarrz'
timestamp=2025-06-19T12:15:17.997570
version=1.0.0
user=user_name

2025-06-19 12:15:19  Running biological QC analysis:
2025-06-19 12:15:19  Zarr path(s): ['path/to/data.zarrz']
2025-06-19 12:15:19  Region: None
2025-06-19 12:15:19  Sample Sheet: None
2025-06-19 12:15:19  Group Name: None
2025-06-19 12:15:19  Output Dir: path/to/output

2025-06-19 12:16:27 | WARNING | path/to/env/lib/python3.x/site-packages/dask/core.py:127:␣
↪RuntimeWarning: invalid value encountered in divide
return func(*(_execute_task(a, cache) for a in args))

2025-06-19 12:16:49 | WARNING | path/to/env/lib/python3.x/site-packages/dask/core.py:127:␣
↪RuntimeWarning: invalid value encountered in divide
return func(*(_execute_task(a, cache) for a in args))

2025-06-19 12:16:55  Biological QC report generated: path/to/output/BioQC_YYYYMMDD_HHMMSS/
↪biological_qc_report_<n>_samples_YYYYMMDD_HHMMSS.html
2025-06-19 12:16:55  Wrote provenance sidecar file to path/to/output/BioQC_YYYYMMDD_HHMMSS/
↪modality_metadata_YYYYMMDD_HHMMSS.json
```

1.16.2 Logging and provenance tracking

Logs for every CLI command are written to a file named `modality_cli.log` in the current working directory for further inspection. Logging provides a record of events during the execution of modality XPLR commands. It helps to:

- Debug issues by reviewing detailed logs.
- Monitor the progress of long-running commands.
- Maintain a history of executed commands for reproducibility.

Provenance tracking ensures that every analysis is traceable by recording:

- The command executed.
- The timestamp of execution.

- The working directory and user.
- The version of modality XPLR used.
- System information (e.g., OS, Python version).
- A list of output files generated.

This metadata is critical for reproducibility and for understanding the context of results.

Process

- **Provenance metadata**: Captured automatically for every command.
- **Output registration**: All output files are registered and tracked.
- **Sidecar file**: Provenance metadata is saved in a JSON sidecar file alongside outputs.
- **File comments**: Metadata can be embedded as comments in output files (e.g. BED files).

For example, **provenance metadata** includes:

- **Command**: modality dmr –sample-sheet metadata.tsv –zarr-path file.zarr
- **Timestamp**: 2025-05-20T12:00:00
- **modality XPLR version**: 1.0.0
- **User**: first_name.last_name
- **System info**: { “os”: “macOS”, “python_version”: “3.11” }
- **Outputs**: [“path/to/output/dmr_results.bed”]

Provenance data is written to a JSON sidecar file in the output directory. A timestamp is included in the filename. e.g. `modality_metadata_<YYYYMMDD>_<HHMMSS>.json`

For CLI commands that produce BED format result files, the provenance metadata can be embedded as comment header lines in the output file.

1.16.3 Setting verbosity levels

How to set verbosity levels

The verbosity level in `modality` commands controls the amount of information displayed in the terminal during execution. This feature is useful for debugging, monitoring progress, or reducing unnecessary output.

modality XPLR supports three verbosity levels:

1. **Default (no flag)**: Displays warnings and errors and command feedback only.
2. **Information**: Use `--verbose` or `-v` to display informational messages, warnings, and errors.
3. **Debug**: Use `-vv` to display detailed debugging information, along with informational messages, warnings, and errors.

To set the verbosity level, use the `--verbose`, `-v` (Information) or `-vv` (Debug) flags when running a `modality` command. The flag must be placed after the command and before any other options.

Each additional `v` increases the verbosity level. For example:

```
modality -v biological-qc \
   --zarr-path path/to/data.zarrz \
   --output-dir path/to/output/directory
```

```
modality -vv get regional-frac \
   --zarr-path path/to/data.zarrz \
   --fields mc hmc \
   --bedfile path/to/regions.bed \
```

This tool is particularly useful for monitoring long-running commands such as `modality dmr call`. Setting the verbosity level to **Information** will display progress bars, helping you to understand how far along the process is and if any issues arise. Below is an example of the terminal output when running `modality -v dmr call`:

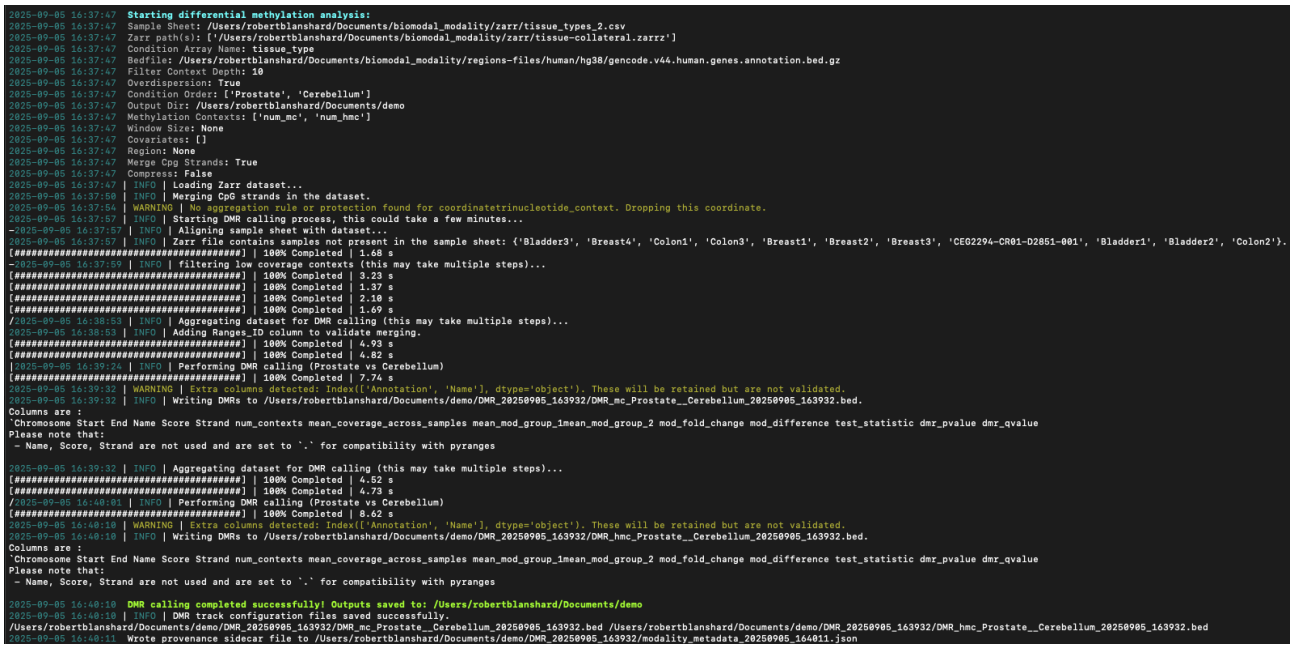

Figure 1.8: Terminal output showing progress bars for `modality -v dmr call` command.

1.17 Example workflow

modality XPLR workflows utilise data generated by duet software, to run statistical analysis and explore methylation patterns across the 6-base genome. A genome annotation can be used to examine the context of differences in methylation, providing mechanistic insight into the dynamic nature of methylation changes. modality XPLR consists of modular steps allowing an analysis flow to be built to address specific biological questions.

In this worked example, we are using a colorectal cancer (CRC) dataset of liquid biopsy samples (cfDNA) to demonstrate a typical discovery workflow. We will use the workflow to determine the methylation status of promoters at different stages of disease and investigate if discrimination of 5mC and 5hmC could provide new biomarkers for earlier stage disease.

1.17.1 Overview of the workflow

The workflow steps:

- Summarise Zarrs, confirm the depth of CpG coverage, and ask whether methylation patterns make biological sense, according to the experimental design, using `modality biological-qc`.
- Assess depth and coverage in specific regions of interest, e.g. gene promoters using `modality get mean`.
- Identify 5mC DMRs in promoter regions between Healthy Controls and Stage IV disease cfDNA using `modality dmr call`.
- View DMR results in a volcano plot, apply filters and identify regions with significant hypo-methylation in Stage IV cfDNA compared to Healthy Controls, using `modality dmr plot`.

- Using significantly hypo-methylated 5mC DMRs from the previous step as a biological prior, identify 5hmC DMRs between Healthy Controls and Stage I disease cfDNA using `modality dmr call`.
- View 5hmC DMRs in a volcano plot to identify promoter regions with significant hyper-methylation in Stage I cfDNA compared to Healthy Controls, using `modality dmr plot`, as a potential biomarker for early stage disease.
- Bring the analyses together to visualise the methylation changes in biological context, by aligning the results with genome annotations using `modality tracks`.
- Generate a set of 5mC and 5hmC features that can be used to build a model for early disease classification using `modality get regional-frac` for `mc` and `hmc`.

Note: When creating the workflow of modality XPLR features this example shows the iterative process, for example running DMR analysis for Controls vs Stage IV samples, and then using the DMR results `.bed` output as an input `--bedfile` for DMR analysis of Controls vs Stage I samples.

Let's examine each of these steps in this worked example, with a note to cover why the step is included, what inputs are needed, the commands used, and the outputs available for immediate results or for the next step of the workflow.

1.17.2 1. Assess methylation data quality

The first step is to ensure there is enough coverage for meaningful onwards statistical analysis, and assess whether expected methylation patterns are present, based on the experimental question. This is done with `modality biological-qc`

Insights gained from running Biological QC:

What does the analysis tell us?	Using which visual?
Confirmation that mean CpG coverage is sufficient for downstream analyses.	Summary table
Whether methylation levels of similar or different samples follow anticipated correlations e.g. replicates are similar, different disease stage samples show different levels of correlation.	Pearson correlation matrix
Samples cluster with expected 'like' samples e.g. controls, stage of disease and the feature on axis is significant in the analysis e.g. level of 5mC or 5hmC etc. and identification of outliers.	PCA scatterplot

Inputs

- **Zarr store(s)**: Paths to each Zarr store to include in the analysis.
- **Sample sheet (recommended)**: Path to the sample sheet file containing metadata about the samples.
- **Output directory (recommended)**: Path to the output directory where the HTML report will be saved.

Example code

When `modality biological-qc` is invoked, all three analyses above run automatically and are presented in a single HTML report.

```
modality biological-qc \
        --zarr-path path/to/data.zarrz \
        --sample-sheet path/to/sample_sheet.csv \
        --aggregate-by-group "label" \
        --disable-strand-merging \
```

(continues on next page)

(continued from previous page)

```
    --region chr1 \
    --output-dir path/to/output/directory
```

Example outputs including plots

Sample ID	Mean CpG Coverage	Unnamed: 0	experiment	age	sex_original	label	sex	cfdna	tumour_fraction	disease_status	sex_mismatch	filter
CEG1407-SY01-D894-007	14.71	6	1	68	FEMALE	CONTROL	FEMALE	175.5	0.0082	HEALTHY CONTROLS	False	PASS
CEG1407-SY01-D894-008	15.26	7	1	68	FEMALE	CONTROL	FEMALE	196.8	0.01113	HEALTHY CONTROLS	False	PASS
CEG1437-SY01-D978-021	17.86	13	3	67	FEMALE	CONTROL	MALE	40.37	0.01128	HEALTHY CONTROLS	True	PASS

Figure 1.9: Snapshot of the Sample Information table from the Biological QC HTML report, showing Sample IDs and the corresponding mean CpG coverage.

1.17.3 2. Checking CpG context depth across the regions of interest

In step 1, the summary table reports the general sequence coverage across all the samples. Let's look at how adding `modality get` analysis at this stage helps to check the coverage is sufficient at the specified promoter regions of interest.

Note: A prep-prepared bedfile of annotated promoter regions is used to focus on the regions of interest, see *Pre-prepared BED files*.

What is being analysed?	Subcommand	Output
The mean coverage of CpGs per region of interest.	`modality get mean`	A violin plot of coverage per region of interest
The number of CpGs per region of interest, with coverage 10	`modality get count`	A table of counts per region of interest (TSV file)

Inputs

- **Zarr path**: Path to the Zarr file(s) containing methylation data.
- **Fields**: Type(s) of cytosine to extract, which in this example is total_c (`mc` + `hmc` + `c`).
- **Region definition**: Path to a BED file defining the promoter sites as the regions of interest.
- **Minimum coverage**: When applicable, the level of coverage at a CpG site needed to be included in the analysis.
- **Output directory**: Path to the output directory where the results of the analysis will be saved.

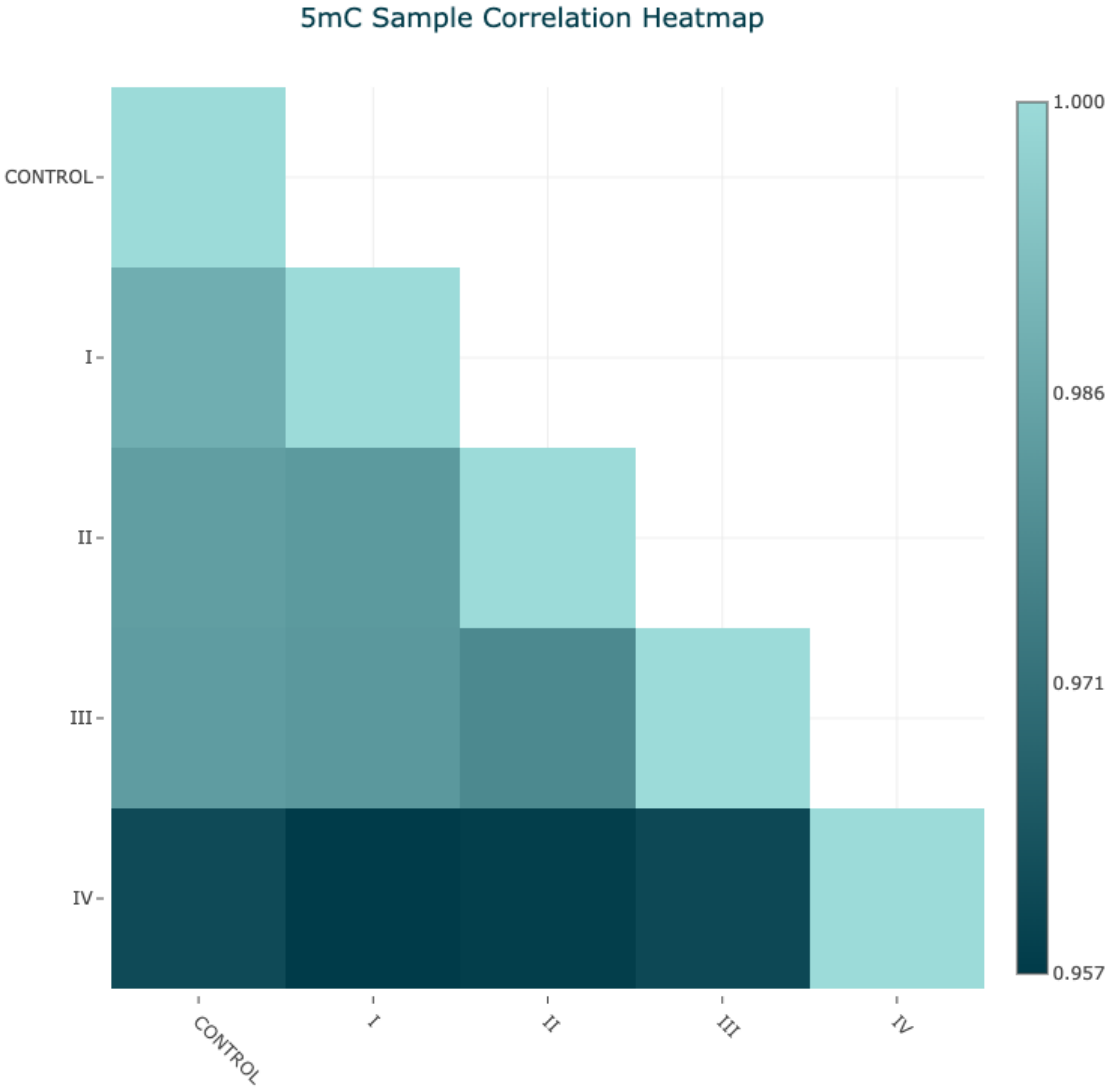

Figure 1.10: 5mC Pearson correlation matrix for aggregated samples

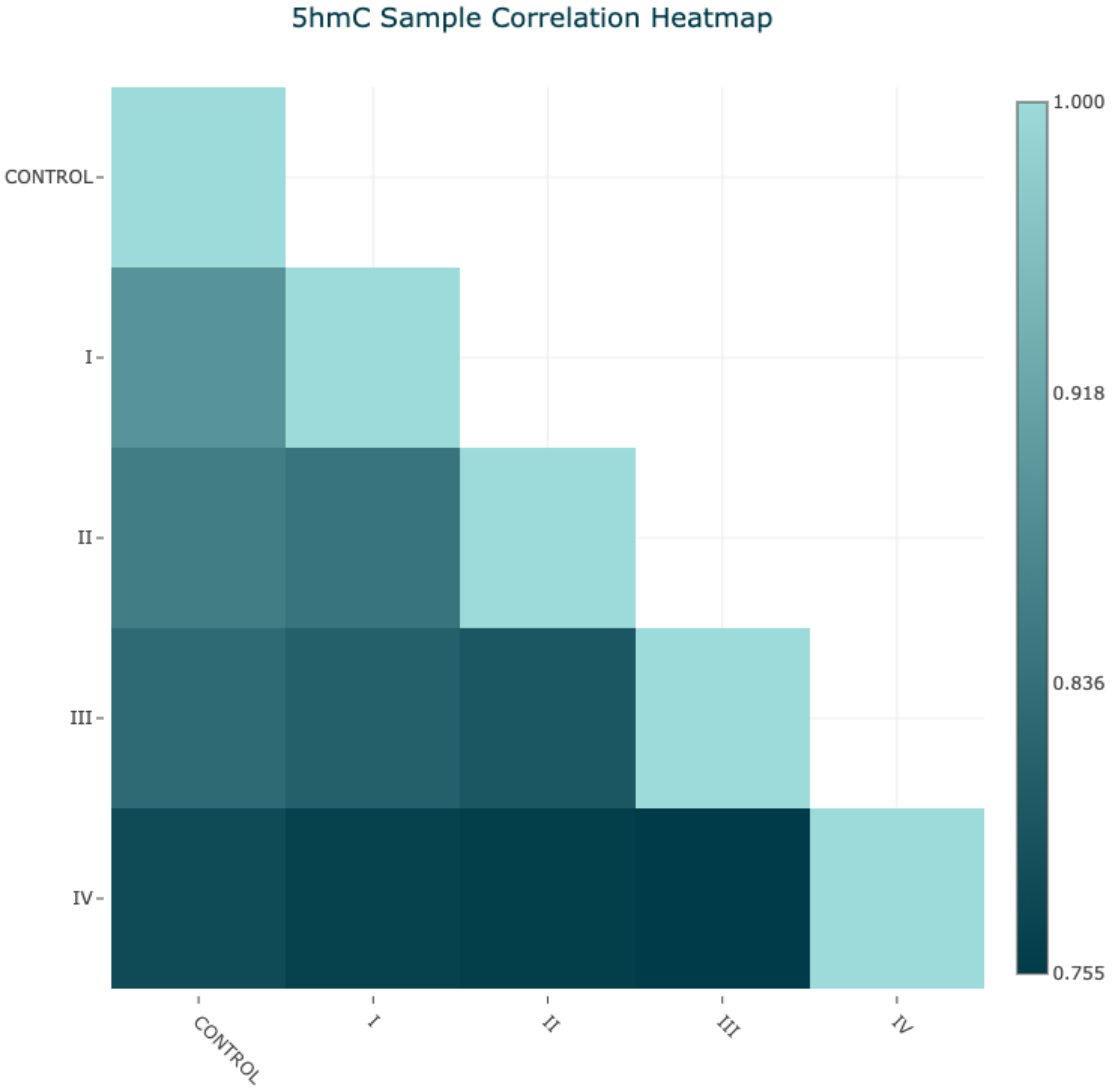

Figure 1.11: 5hmC Pearson correlation matrix for aggregated samples

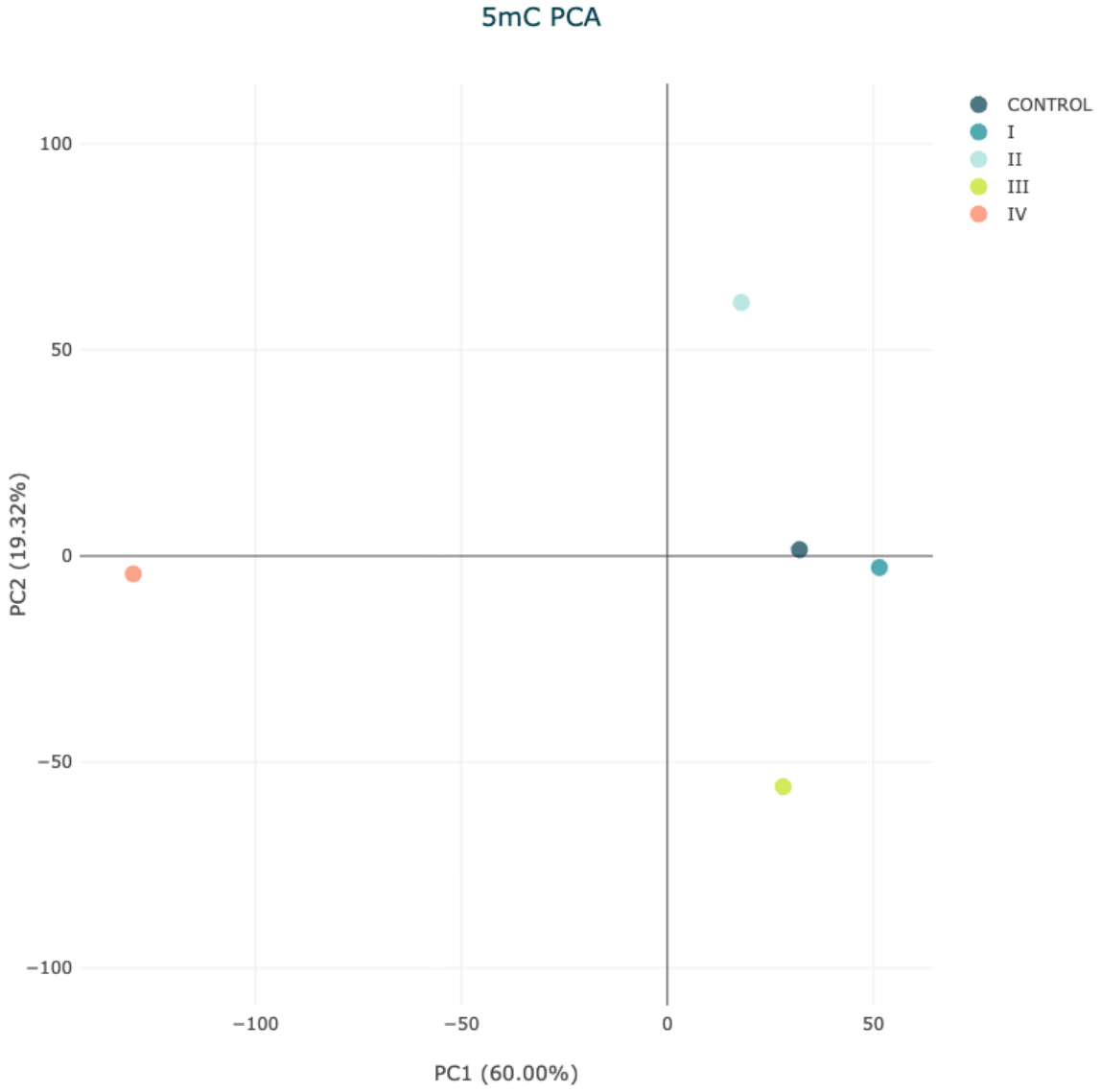

Figure 1.12: 5mC PCA plot showing aggregated samples by disease stage

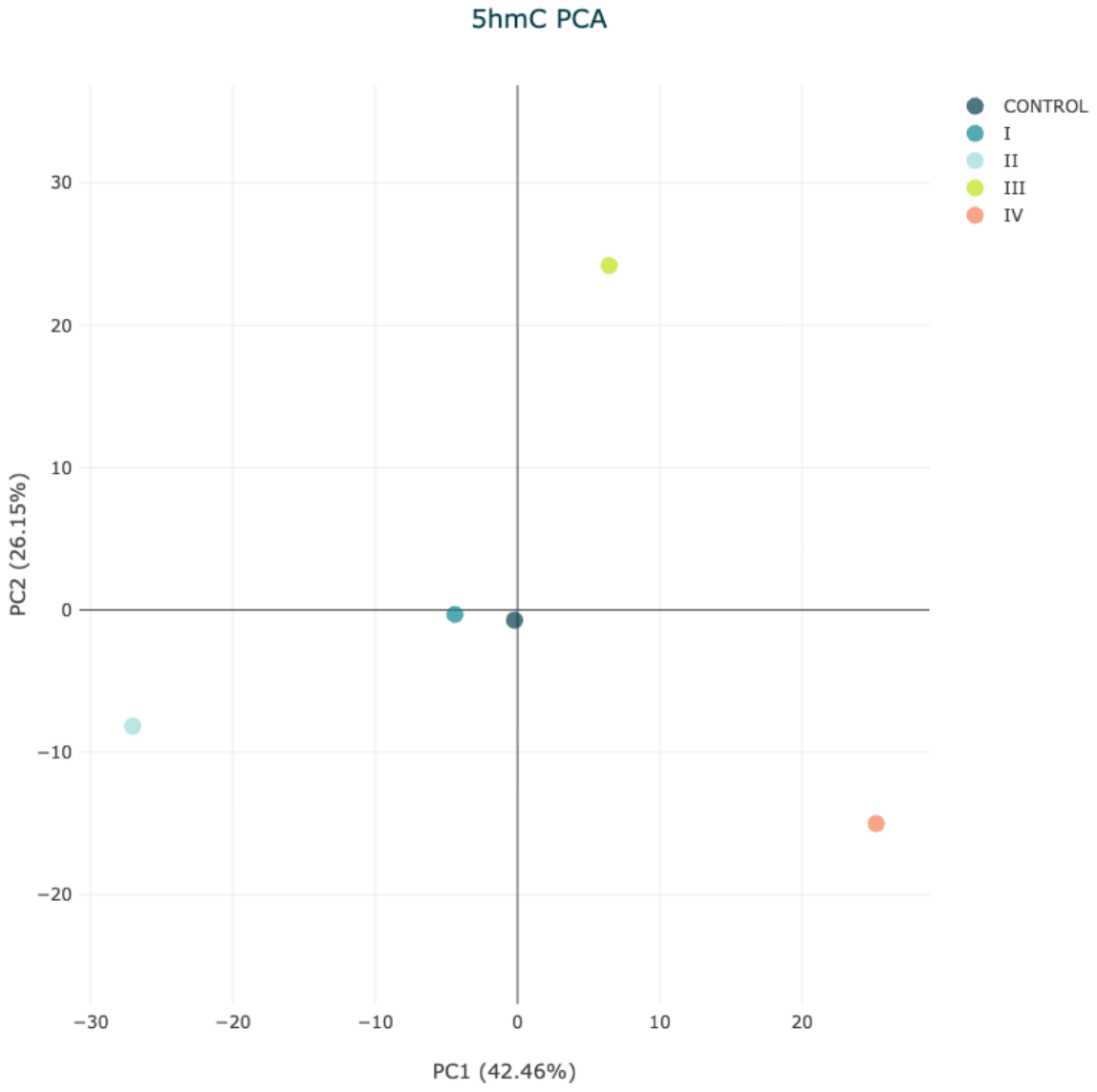

Figure 1.13: 5hmC PCA plot showing aggregated samples by disease stage

Example code and outputs

First, to view the coverage distribution of CpGs at promoter regions, we can use the `modality get mean` command. The violin plot is included in the output HTML report.

```
modality get mean \
        --zarr-path path/to/data.zarrz \
        --fields total_c \
        --bedfile path/to/gencode.v44.human.promoters.annotation.tsv \
        --disable-strand-merging \
        --output-dir path/to/output/directory
```

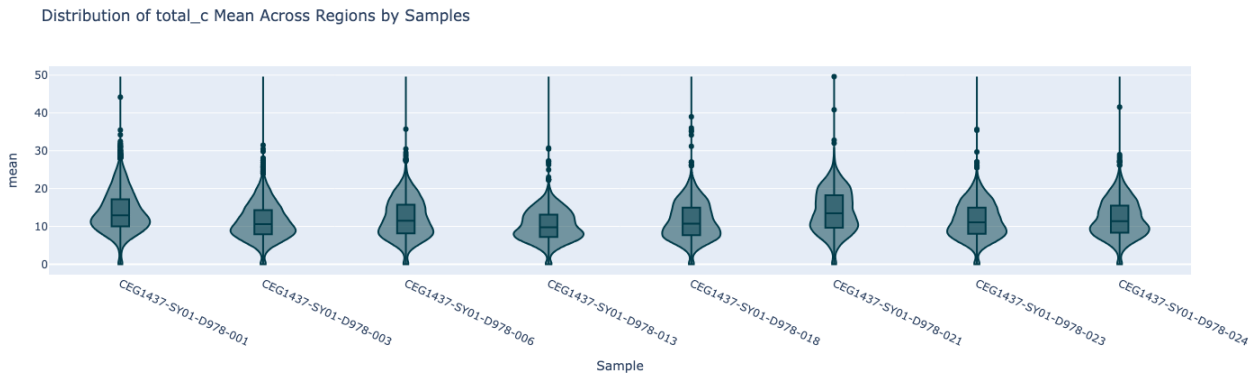

Figure 1.14: Violin plot showing the mean coverage of total_c per promoter region, for 8 samples.

Next, we may want to filter the promoter regions to only include those with sufficient coverage, for example, where the mean coverage is greater than 10. This can be done by first using the `modality get count` command, and then filtering the output `.tsv` to update the list of promoters for downstream processing.

Alternatively, minimum coverage settings can be applied directly in modality XPLR functions such as `modality dmr call` or `modality get regional-frac`. Note that the filter value should be selected according to whether strands are merged (default) or not (`--disable-strand-merging`) in the analysis. If strand merging is disabled, as shown here, the expected CpG coverage will be approximately half the NGS depth.

```
modality get count \
--zarr-path path/to/data.zarrz \
--fields total_c \
--bedfile path/to/gencode.v44.human.promoters.annotation.tsv \
--min-coverage 5 \
--disable-strand-merging \
--output-dir .path/to/output/directory
```

1.17.4 3. Call differentially methylated regions (DMRs) in late stage disease

With sufficient coverage confirmed of the regions of interest, the next step is to call DMRs in late stage disease to identify promoter regions where demethylation (hypo-methylation) is observed compared to Healthy Controls.

What is being analysed?	Output
To look for biomarkers of early disease, we first establish our biological priors by looking for changes in promoter regions between Healthy Controls and late-stage (Stage IV) cfDNA. We are looking to identify changes in 5mC.	5mC DMR `.bed` result file for the comparison of Healthy Controls and Stage IV cfDNA samples. `.ini` files and provenance metadata files.

Chromosome	Start	End	sample1	sample2	sample3	sample4	sample5	sample6	sample7	sample8	Annotation	Name
chr1	64418	65418	79	79	79	79	79	79	79	79	Promoter	OR4F5
chr1	451677	452677	67	67	67	67	67	67	67	67	Promoter	OR4F29
chr1	686653	687653	167	167	167	167	167	167	167	167	Promoter	OR4F16
chr1	922922	923922	121	121	121	121	121	121	121	121	Promoter	SAMD11
chr1	959308	960308	62	62	62	62	62	62	62	62	Promoter	NOC2L
chr1	959583	960583	41	41	41	41	41	41	41	41	Promoter	KLHL17
chr1	965481	966481	9	9	9	9	9	9	9	9	Promoter	PLEKHN1
chr1	982116	983116	193	193	193	193	193	193	193	193	Promoter	PERM1
chr1	1000137	1001137	25	25	25	25	25	25	25	25	Promoter	ISG15
chr1	1000171	1001171	229	229	229	229	229	229	229	229	Promoter	HES4
chr1	1019119	1020119	203	203	203	203	203	203	203	203	Promoter	AGRN
chr1	1074305	1075305	162	162	162	162	162	162	162	162	Promoter	RNF223
chr1	1116360	1117360	224	224	224	224	224	224	224	224	Promoter	C1orf159
chr1	1172879	1173879	90	90	90	90	90	90	90	90	Promoter	TTLL10
chr1	1206591	1207591	223	223	223	223	223	223	223	223	Promoter	TNFRSF18
chr1	1214152	1215152	35	35	35	35	35	35	35	35	Promoter	TNFRSF4
chr1	1231236	1232236	43	43	43	43	43	43	43	43	Promoter	B3GALT6
chr1	1232030	1233030	94	94	94	94	94	94	94	94	Promoter	SDF4
chr1	1246721	1247721	21	21	21	21	21	21	21	21	Promoter	C1QTNF12
chr1	1273863	1274863	13	13	13	13	13	13	13	13	Promoter	UBE2J2
chr1	1279435	1280435	102	102	102	102	102	102	102	102	Promoter	SCNN1D

Figure 1.15: Snapshot of the output `tsv.gz` for 8 samples, showing the number of CpGs per promoter region with coverage 5.

Inputs

- **Sample sheet**: The sample sheet is a `.csv` or `.tsv` file that contains metadata about the samples in the Zarr store.
- **Zarr path**: Path to the Zarr file(s) containing methylation data.
- **Region definition**: Path to a BED file defining the promoter sites as the regions of interest.
- **Methylation context**: Type(s) of cytosine to extract, which in this example is num_mc.
- **Output directory (optional)**: Path to the output directory where the results of the analysis will be saved.

Tip: Before starting analysis care should be taken to prepare the sample sheet to ensure the correct data is used. Note the column header name(s) that refer to the sample group or condition, and covariates (if applicable).

Example code

```
modality dmr call \
        --sample-sheet path/to/metadata.csv \
        --condition-array-name label \
        --zarr-path path/to/data.zarrz \
        --methylation-contexts mc \
        --bedfile path/to/promoters.bed \
        --condition-order "CONTROL" "IV" \
        --output-dir path/to/output/directory \
        --min-coverage 10 \
        --overdispersion
```

Note: `--overdispersion` was used in case the false positive occurrence was too high. This may not be necessary once the data has been examined. See *Overdispersion correction* in the user guide for more details.

Example outputs including plots

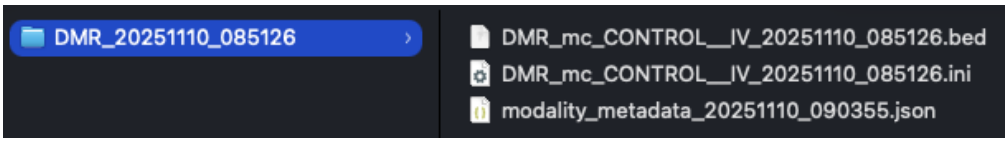

Figure 1.16: The output folder structure of the `modality dmr call` command. A timestamped sub-directory is created, containing the DMR results bedfile, provenance metadata file, and configuration `.ini` file.

Note the path of the DMR results bedfile, for use in `modality dmr plot` in the next step of the analysis.

What is being analysed?	Using which visual?
Finding samples where methylation differences are significant being either areas of hypo- or hyper-methylation	Volcano plot

Note: In this worked example the hypo-methylation regions were of interest as an indicator of where demethylation may have occurred in late stage disease. This allows us to ask whether we could detect increases of 5hmC in the same regions, at an earlier stage of disease.

Inputs

- **DMR results**: Path to the DMR results bedfile.
- **Output directory**: Path to the output directory where the results of the analysis will be saved.

Example code

```
modality dmr plot \
        --dmr-results path/to/DMR_mc_CONTROL__IV_20251110_085126.bed \
        --output-dir path/to/output/directory
```

Example outputs including plots

In differential methylation analysis, a volcano plot is a powerful visualisation tool that helps identify regions with both statistically significant and biologically meaningful differences in methylation between two groups. In the below plot we have DMRs called on 5mC comparing the two groups `control` and `Stage IV` in cfDNA data. The axes of the volcano plot represent:

- X-axis: Modification difference (e.g., methylation difference between two groups, often called $\Delta\beta$ or Δmethylation).
- Y-axis: Log10-transformed q-value (adjusted p-value) although note that when hovering over the data points you will see key information regarding the data point including the non-transformed q-values which is more commonly interpreted.

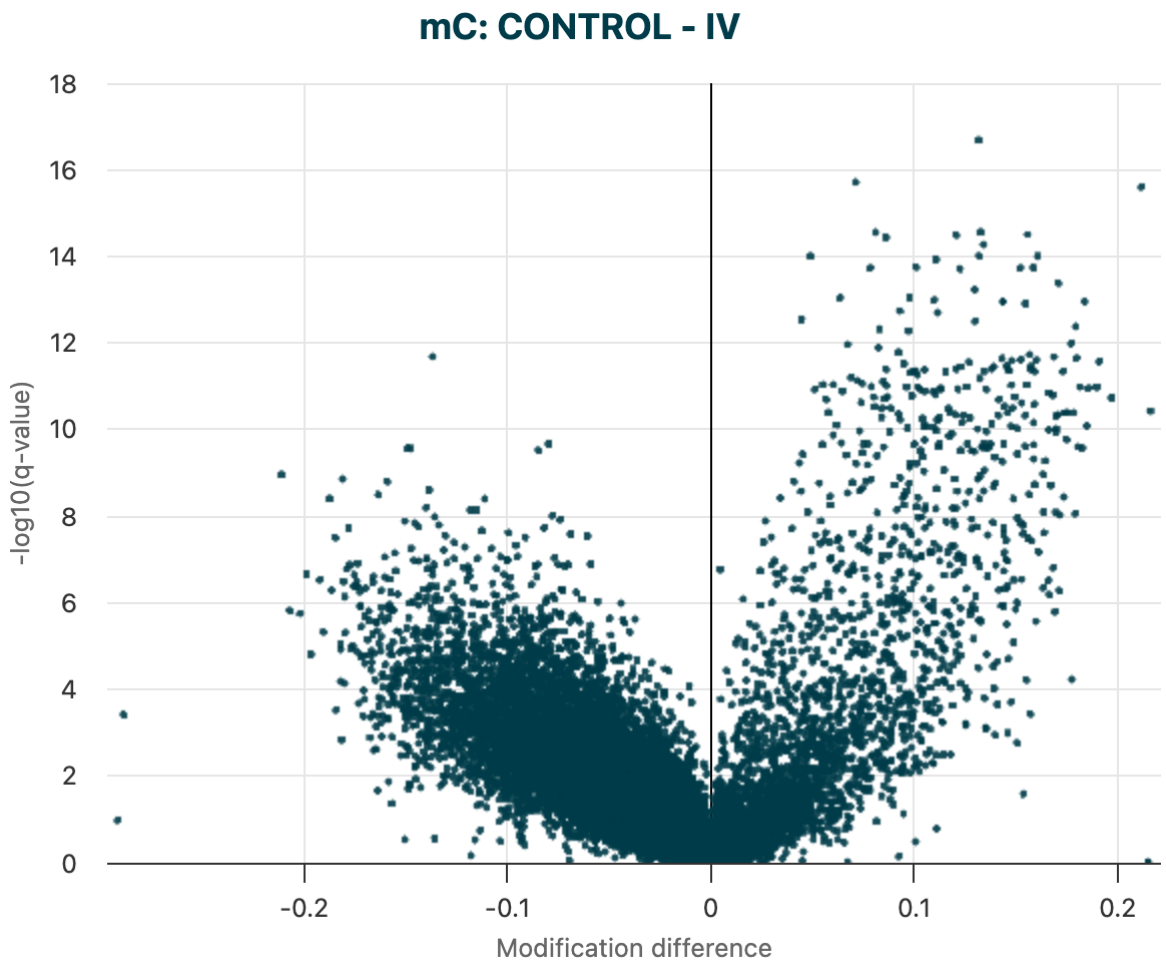

Figure 1.17: The unfiltered volcano plot of DMR results for 5mC DMRs, between Control vs Stage IV cfDNA samples.

1.17.5 4. Filtering DMRs to identify statistically significant changes

Rather than working with all regions, the next step was to identify a subset of regions with statistically significant DMRs meeting a given threshold.

What is being analysed?	Using which visual?
To speed up analysis and to focus on regions with statistically significant differences, filters are applied to create a results file with DMR calls passing a significance threshold and level of change in methylation.	Volcano plot of 5mC DMRs. Data points visible in the plot are determined by the `dmr plot` tool.

Inputs

- **DMR result file(s)**: Paths to the output(s) of the `modality dmr call` tool, in BED3+ format.

Example code

```
modality dmr plot \
        --dmr-results path/to/DMR_mc_CONTROL__IV_20251110_085126.bed \
        --filter-qvalue 0.01
        --filter-mod-difference 0.15
        --output-dir path/to/output/directory
```

Tip: If there is a delay when interacting with the HTML plots, this could be due to having too many data points. This can be addressed by setting more stringent filters to reduce the number of data points to render.

Example outputs including plots

The `dmr plot` command also produces a filtered results `.bed` file, which can be used in the next step of the analysis.

In this example, we are interested in the regions showing significant 5mC hypo-methylation at Stage IV, compared to Healthy Controls. We hypothesise that these regions will give rise to early biomarkers of change when looking at 5hmC levels, because changes in 5hmC are expected to precede the loss of 5mC.

1.17.6 5. Use biological priors to call DMRs in early stage disease

The previous step in the analysis identified DMRs with hypo-methylation in late stage disease. Next, we can ask if these same regions show an increase in 5hmC levels in early stage samples compared to Healthy Controls. This may reveal a biomarker of early change, where 5hmC may signal that a region is primed for later demethylation.

What is being analysed?	Using which visual?
To assess whether hypo-methylation of promoters in Stage IV samples display increase in 5hmC levels at these sites in Stage I vs Healthy Controls.	DMR calling only.

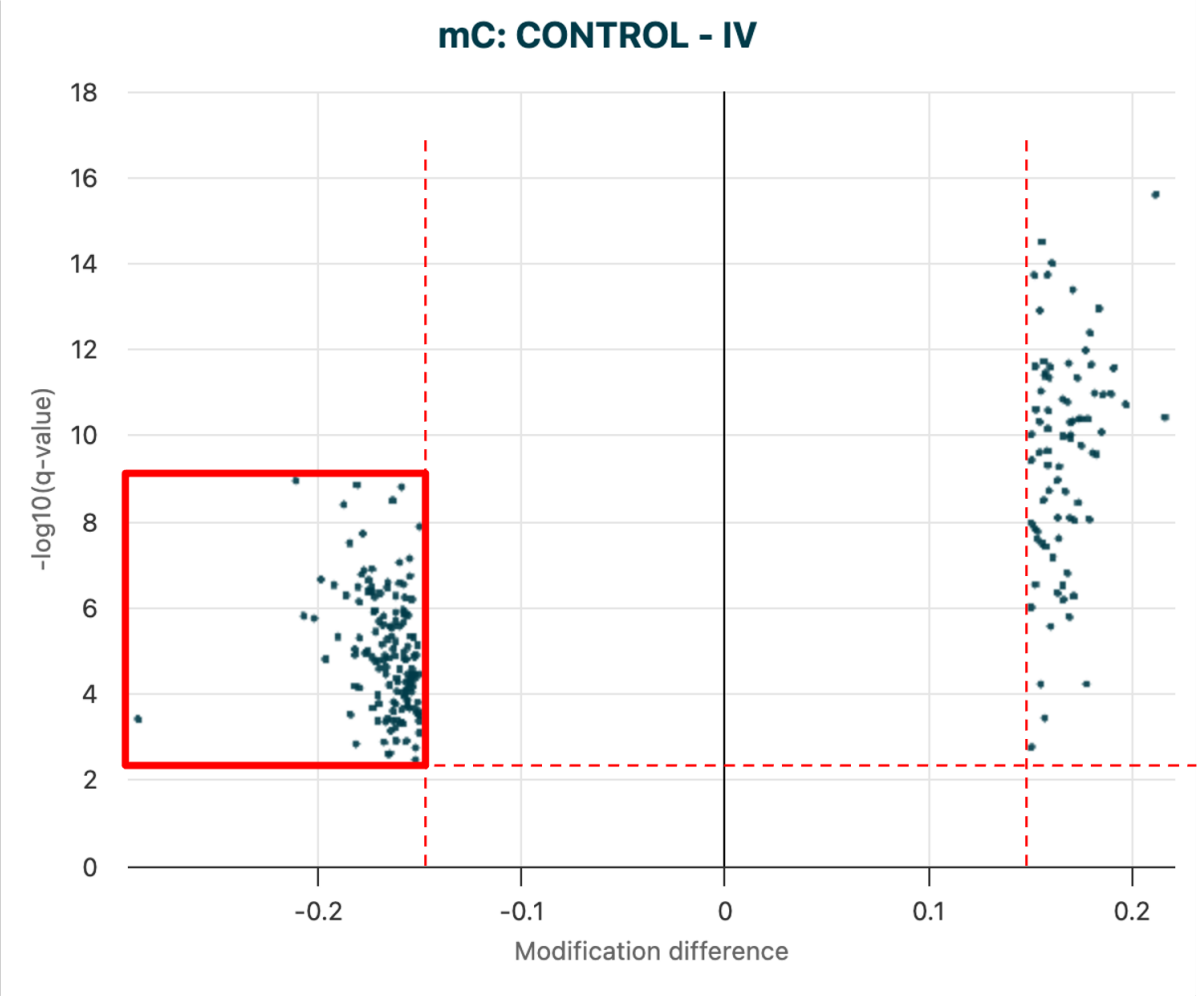

Figure 1.18: The filtered volcano plot of DMR results for 5mC DMRs, between Control vs Stage IV cfDNA samples.

Inputs

- **Filtered 5mC-DMR bedfile**: Path to the output file from above.
- **Sample sheet**: Use the same sample sheet as used in the previous step, which contains metadata about the samples in the Zarr store.
- **Zarr path**: Path to the Zarr file(s) containing methylation data.
- **Output directory (optional)**: Path to the output directory where the results of the analysis will be saved.

Example code

```
modality dmr call \
        --sample-sheet path/to/metadata.csv \
        --condition-array-name label \
        --zarr-path path/to/data.zarrz \
        --methylation-contexts hmc \
        --bedfile path/to/filtered_5mC-DMRs.bed \
        --condition-order "CONTROL" "I" \
        --output-dir path/to/output/directory \
        --min-coverage 10 \
        --overdispersion
```

Outputs

A new timestamped DMR results directory containing the DMR results file with your calls, an `ini` file that can be used to view your DMRs in their genomic context using the `modality tracks` command.

1.17.7 6. Visualisation of key early-stage 5hmC DMRs

Again, this step uses `modality dmr plot` to visualise the DMR results file generated previously (`modality dmr call`).

What is being analysed?	Using which visual?
Visualisation of promoter regions where DMRs show statistically significant increase in 5hmC levels at Stage I versus 5mC hypo-methylation in Stage IV allowing selection of specific regions for further investigation.	Volcano plot of 5hmC DMRs.

Inputs

- **DMR results file**: The DMR results file just created, focussed on the 5hmC levels in the Control and Stage I samples at the hypo-methylated promoter sites from Stage IV samples.
- **Output directory**: Path to the output directory where the results of the analysis will be saved.

Example code

```
modality dmr plot \
        --dmr-results path/to/DMR_hmc_CONTROL__I_20251109_214630.bed \
        --output-dir path/to/output/directory
```

Tip: If there is a delay when interacting with the HTML plots, this could be due to having too many data points. This can be addressed by setting more stringent filters to reduce the number of data points to render.

Example outputs including plots

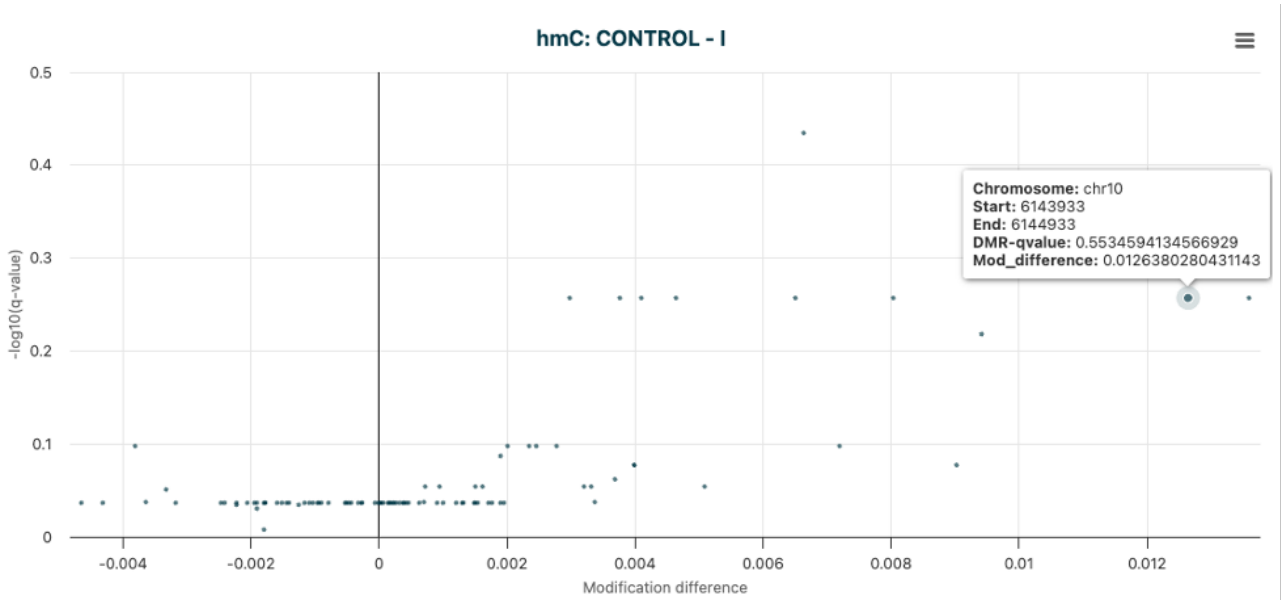

Figure 1.19: The unfiltered volcano plot of DMR results for 5hmC DMRs, between Control vs Stage I cfDNA samples.

From this plot, or directly from the DMR results file, we can identify regions of interest for further investigation. In this example, the promoter region of `PFKFB3` was identified as a region of interest, where 5hmC levels were increased in `Stage I` samples compared to `Controls`, additional analysis showed 5mC levels were significantly decreased in `Stage IV` samples compared to `Controls`. The next step is to bring together the DMR results and genomic annotation to look at biological context of the selected regions.

1.17.8 7. Place methylation changes (DMRs) into genomic context

From the analysis so far, the last DMR plot will identify promoters that warrant further investigation. `modality tracks` allows a combined view of specific 5mC (Stage IV) and 5hmC (Stage I) DMRs by modifying the `.ini` configuration file, including added annotated genomic context.

In this example, we will use the promoter region of `PFKFB3` as a region of interest, which was identified in the previous step. The 1000bp promoter region is defined as `chr10:6143933-6144933`, which we will view in a track plot with an additional 1000bp padding on either side, to allow for visualisation of the surrounding genomic context.

What is being analysed?	Using which visual?
Bringing together the DMR results and genomic annotation to place significant DMRs in biological context.	`modality tracks`

Inputs

- **ini file**: Path to the `.ini` file to be included in the analysis.
- **Output directory**: Path to the output directory where the track plot image will be saved.

Tip: For details on the default `.ini` file, adapting the `.ini` file or creating a specific `.ini` file, refer to the *1. Auto-generated .ini file* and *2. Customising .ini files* sections of the user guide.

Example code

```
modality tracks \
        --region chr10:6143933-6144933 \
        --padding 1000 \
        --tracks-file path/to/config.ini \
        --output path/to/output/directory/track_plot.png
```

Example output

Interpreting the tracks plot:

The output file is named according to the specified path and file extension, and includes:

- **Methylation traces**:
  - In this example the top tracks display the 5mC and 5hmC methylation fraction for each group. The top trace shows the 5mC fraction for CONTROL and Stage IV cfDNA, and the bottom trace shows the 5hmC fraction for CONTROL and Stage I cfDNA.
  - Each line represents the average methylation fraction across the region for a group.
  - Dots indicate per-site values (e.g. CpGs).
  - These traces allow you to see overall methylation patterns and differences between groups. At DMRs, we expect to see divergence between the average methylation trace lines between groups.
  - Trace plots are drawn directly from the Zarr data with the strand merging setting used during DMR calling by default.
- **Modification differences**:
  - Below the methylation traces, the next tracks show the 5mC and 5hmC modification difference between groups. The first shows the 5mC modification difference for CONTROL and Stage IV cfDNA, and the second shows the 5hmC modification difference for CONTROL and Stage I cfDNA.
  - The y-axis represents the difference in methylation fraction between the two groups at each position.
  - Points show raw differences and lines show smoothed differences, highlighting regions of gain or loss.
  - This helps to quantify the relative difference of methylation changes between groups. Note the difference in y-axis scale between the two tracks, as the 5mC hypo-methylation in Stage IV is much greater than the 5hmC hyper-methylation in Stage I.
- **DMR tracks**:
  - The next panels highlight the locations and magnitude of DMRs. The length and position of the bars will match the specified window size or defined regions used as input to DMR calling.
  - Bar colour and height reflect the direction and magnitude of change (e.g., hypo-methylation or hyper-methylation). For example, a blue bar may indicate a region of significant 5mC loss in Stage IV, while a red bar may indicate a region of 5hmC gain in Stage I.
  - The number of asterisks above each bar, describes the level of statistical significance, where:

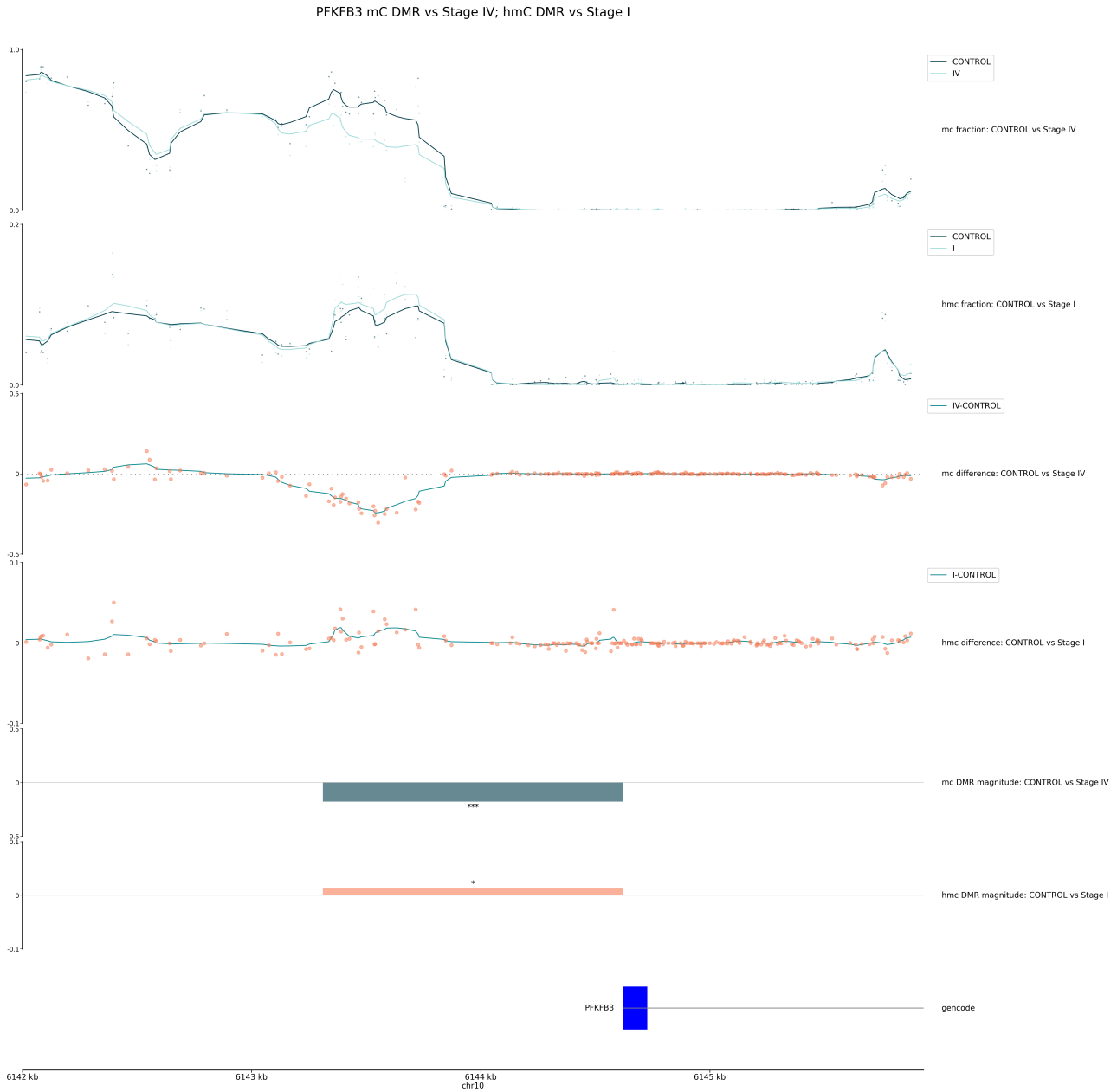

Figure 1.20: A combined track plot showing the methylation levels of 5mC and 5hmC in the promoter region of PFKFB3, with DMRs highlighted.

    - One asterisk (*) indicates a q-value < 0.05
    - Two asterisks (**) indicate a q-value < 0.01
    - Three asterisks (***) indicate a q-value < 0.001
    - **Note:** If a q-value is not listed in the DMR results file, the asterisks will be based on the p-value, with the same significance levels as above.

- **Annotations**:
  - At the bottom, the gene annotation track shows the genomic features in the displayed region.
  - This track is based on the GTF/GFF3 file specified in the `.ini` file.
  - In this example, genes are displayed with exons as blue boxes, and labels (e.g., "PFKFB3") for easy identification.
  - This context allows you to relate methylation changes to known genes or regulatory elements, supporting biological interpretation. We can see that the DMR block matches the 1000bp region upstream of the gene TSS, denoted as a promoter region in the DMR input bedfile.

1.17.9 8. Generate feature sets for building classifiers

Using the methods described above, it is possible to generate feature sets that can be used to build classification models. This can be achieved by extracting different statistical measures for 5mC and 5hmC for your regions of interest. In this case we will be extracting the regional fractions for the regions provided in `--bedfile` with `modality get regional-frac`.

What is being analysed?	Output
The fraction of 5mC in the regions of interest (the promoter regions)	`.tsv` files and HTML report containing violin and correlation plots.
The fraction of 5hmC in the regions of interest (the promoter regions)	`.tsv` files and HTML report containing violin and correlation plots.
[Optional] Co-methylation and clustering analysis of 5mC and 5hmC fractions across a specific subset of regions, e.g informed by gene ontology data.	Additional regional heatmap included in the HTML report, for up to 200 regions.

Inputs

- **Zarr Path**: Path to the Zarr file(s) containing methylation data.
- **fields**: The types of cytosine to extract; in this example both `mc` and `hmc`.
- **Region Definition**: Path to a `.bed` file defining the promoter sites as the regions of interest.
- **Output directory**: Path to the output directory where the result files will be saved.

Example code

To generate regional fraction for both 5mC and 5hmC accross a broad set of regions (e.g. all promoters), the following command can be used:

```
modality get regional-frac \
        --zarr-path path/to/data.zarrz \
        --fields mc hmc \
        --bedfile path/to/gencode.v44.human.promoters.annotation.tsv \
        --min-coverage 10 \
        --disable-strand-merging \
        --output-dir path/to/output/directory
```

To generate regional fraction for a smaller set of regions (e.g. gene pathways), the same command can be used with a different `--bedfile` input and `--heatmap` options. Including a metadata sample sheet allows control for which samples to include.

```
modality get regional-frac \
        --zarr-path path/to/data.zarrz \
        --sample-sheet path/to/metadata.csv \
        --order-by-group label \
        --fields mc hmc \
        --bedfile path/to/gene-region-subset.bed \
        --min-coverage 10 \
        --disable-strand-merging \
        --heatmap \
        --output-dir path/to/output/directory
```

Example data output

track type=bed name=hmc_regional-frac description=hmc_regional-frac visibility=full color=255,0,0 priority=1										
# Chrom	Start	End	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6	Sample7	Sample8
chr1	64418	65418	0.011	0	0.029	0.018	0.016	0.025	0.094	0.03
chr1	451677	452677								
chr1	686653	687653	0.053	0.069	0	0.067	0	0.038	0	0
chr1	922922	923922	0.013	0.007	0.018	0.016	0.009	0.01	0.019	0.013
chr1	959308	960308	0.006	0.007	0.013	0.013	0.017	0.011	0.012	0.008
chr1	959583	960583	0.008	0.007	0.011	0.015	0.016	0.012	0.009	0.009
chr1	965481	966481	0.041	0.013	0.018	0.032	0.025	0.022	0.029	0.034
chr1	982116	983116	0.022	0.02	0.021	0.01	0.023	0.022	0.014	0.016
chr1	1000137	1001137	0.002	0.003	0.003	0.001	0.001	0.004	0.002	0.003
chr1	1000171	1001171	0.002	0.003	0.003	0.001	0.001	0.004	0.003	0.003
chr1	1019119	1020119	0.014	0.006	0.021	0.016	0.027	0.007	0.012	0.029
chr1	1074305	1075305	0.009	0.009	0.027	0.027	0.022	0.016	0.017	0.033
chr1	1116360	1117360	0.004	0.001	0.006	0.009	0.001	0.003	0.003	0.001
chr1	1172879	1173879	0.011	0.017	0.017	0.011	0.007	0.011	0.01	0.014
chr1	1206591	1207591	0.014	0.004	0.017	0.016	0.016	0.015	0.014	0.019
chr1	1214152	1215152	0.044	0.025	0.059	0.064	0.055	0.044	0.065	0.067
chr1	1231236	1232236	0.001	0	0.001	0.001	0	0	0.001	0.002
chr1	1232030	1233030	0.011	0.012	0.012	0.027	0.014	0.014	0.016	0.015

Figure 1.21: Snapshot of an output `tsv.gz` for 8 samples, showing the 5hmC fraction per region, with a depth 10. Rows with no data indicate that the region/sample did not meet the `--min-coverage` requirement. Regions with "0" fraction indicate that CpGs were covered, but no methylation was detected in that region for that methylation context (e.g., 5hmC).

1.17.10 Conclusion

This worked example has shown how to use modality XPLR to run a series of analyses on methylation data, starting from quality control and coverage checks, through to identifying DMRs and visualising them in biological context. The steps outlined above demonstrate how to build a workflow using modality XPLR to address a specific biological question, in this case, the investigation of methylation changes in promoter regions across different stages of disease. The final step of the workflow is to extract features from the DMRs identified, which can be used to build classification models for potential biomarkers. This example highlights the modular nature of modality XPLR, allowing analyses to be tailored to specific needs and biological questions.

Note: This example is intended to provide a general overview of how to use modality XPLR for methylation analysis. The specific commands and parameters may vary depending on the dataset and research question. It is recommended to refer to the modality XPLR user guide for detailed information on each command and its options.

For further information about the utility of 5mC and 5hmC as biomarkers for early disease detection, refer to our recent publication on BioRxiv:

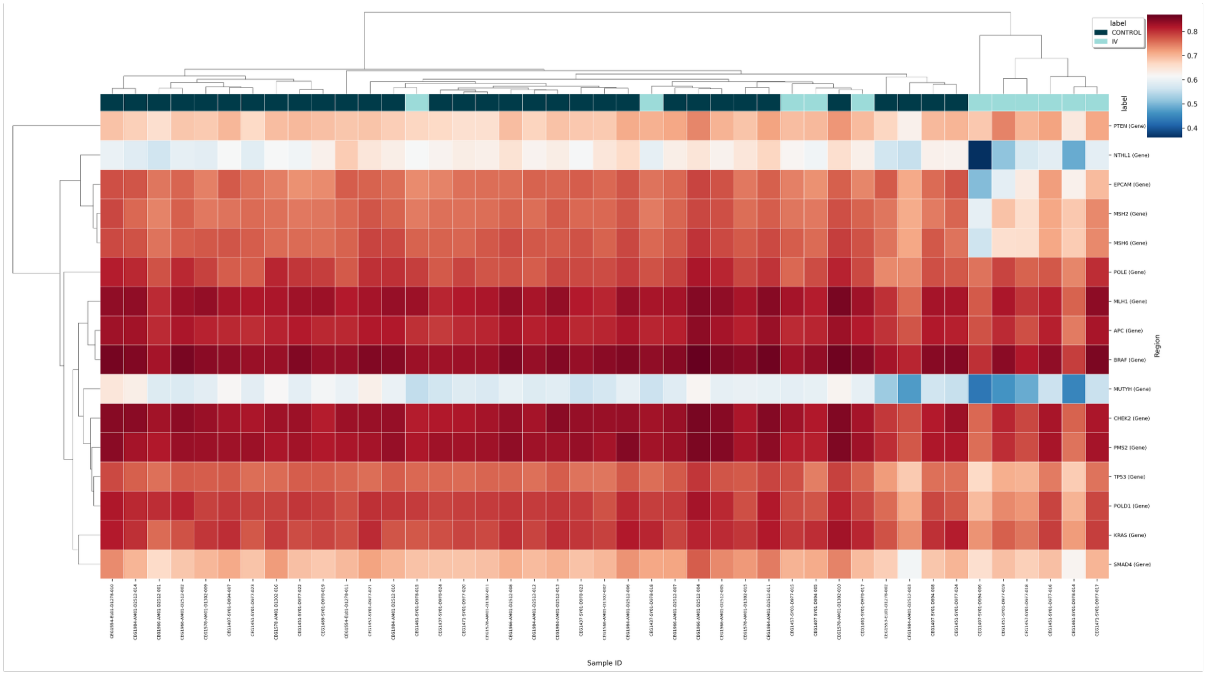

Figure 1.22: Regional co-methylation heatmap for 5mC enrichment in Control and Stage IV samples, for 16 gene regions implicated in CRC.

5-methylcytosine and 5-hydroxymethylcytosine are synergistic biomarkers for early detection of colorectal cancer

1.18 FAQs and troubleshooting

1.18.1 Frequently Asked Questions (FAQs)

This section provides answers to common questions that users may have when using the modality XPLR software. These FAQs are designed to help users navigate the software effectively.

1.18.2 General questions

What is modality XPLR, and what can I use it for?

Modality XPLR is a command-line toolkit for analysing DNA methylation data derived from 5-base (**duet +modC**) and 6-base (**duet evoC**) genomes. It allows you to identify differentially methylated regions (DMRs), extract methylation statistics, and visualise results. It is designed for molecular biologists, bioinformaticians, and researchers who want to analyse methylation data without requiring python experience.

Can I use modality XPLR with other data types?

No, modality XPLR is specifically designed for analysing methylation data from duet Zarr stores. It is not readily compatible with other data types or formats.

Do I need programming experience to use modality XPLR?

No, modality XPLR is designed to be user-friendly for researchers with minimal programming experience. You only need to run commands in the terminal, and the documentation provides step-by-step instructions for each tool.

What is a Zarr store, and why is it important?

A Zarr store is a compressed file format used to store large datasets, such as methylation data. It allows efficient access to subsets of data without loading the entire dataset into memory. modality XPLR uses Zarr stores as the primary input format for analysis.

Suitable Zarr stores to be used with modality XPLR can be created using the duet software and are found in the output of running pipelines with that software.

1.18.3 Installation and setup

What are the system requirements for modality XPLR?

modality XPLR requires macOS or Linux. You need at least 8 GB of RAM, 500 MB of disk space for installation, and Python 3.11.

How do I install modality XPLR?

We provide instructions for installing modality XPLR for macOS and Linux. See the *Installation of modality XPLR* for details.

Do I need to install additional tools like Tabix or Bgzip?

These tools are optional but recommended for performance improvements when working with large genomic files. You can install them using Conda with the following command:

```
conda install -c bioconda htslib
```

See the section *Optional: Install non-python dependencies, tabix and bgzip* for more details.

1.18.4 Input files and formats

What input files do I need to run modality XPLR?

The main input files are:

- Zarr store files (`.zarrz`) containing methylation data.
- Sample sheet (`.csv` or `.tsv`) with metadata about the samples.
- BED files (`.bed`) defining genomic regions of interest (optional).

What formats are supported for the sample sheet?

modality XPLR supports both CSV (comma-separated values) and TSV (tab-separated values) formats for the metadata sample sheet. Examples in this guide are shown for CSV files, but TSV files can be used interchangeably. The file must contain a column named `sample_id` that matches the sample IDs in the Zarr store.

How do I create a sample sheet?

A sample sheet is a CSV file with a column named `sample_id` that matches the sample IDs in the Zarr store. You can include additional columns for metadata, such as `disease_stage` or `smoker_status` which may be used for grouping samples or flagging covariates. modality XPLR is designed so that a single sample sheet for your study can be used for multiple analyses.

Example sample sheet

Table 1.1: Example sample sheet

index	sample_id	Input DNA Quantity (ng/sample)	Protocol Version	sample_type	condition	sex	age	aggregate-sample
0	Sample-01	5	duet evoC	cfDNA	CONTROL	FEMALE	54	Sample-01
1	Sample-02	5	duet evoC	cfDNA	CONTROL	MALE	62	Sample-02
2	Sample-03	5	duet evoC	cfDNA	Stage-II	FEMALE	63	Sample-03
3	Sample-04	5	duet evoC	cfDNA	Stage-II	MALE	71	Sample-04
4	Sample-05_rep1	80	duet evoC	gDNA	CONTROL	FEMALE	68	Sample-05
5	Sample-05_rep2	80	duet evoC	gDNA	CONTROL	FEMALE	59	Sample-05
6	Sample-06_rep1	80	duet evoC	gDNA	TUMOUR	FEMALE	75	Sample-06
7	Sample-06_rep2	80	duet evoC	gDNA	TUMOUR	FEMALE	66	Sample-06

You can also download this example as a CSV file:

`Download sample_sheet.csv`

What is the difference between defining a region or using a BED file?

Both methods define genomic regions of interest:

- A BED3+ file is a standard format with columns for chromosome, start, and end positions. It allows you to specify multiple regions in a single file (e.g. a list of gene promoter regions).
- A region is a single genomic region specified in the format `chr:start-end` (e.g., `chr1:1000-2000`). It is useful for quick analyses of specific regions without creating a separate file.

How many region files (BED3+) can I specify at once?

The `modality dmr call` and `modality get` commands currently accept a single BED3+ file as input per analysis. If you have multiple BED files, you can merge them into a single file before running the analysis, and use the `Annotation` column header to differentiate the region type or origin. However, it is not advisable to combine regions of highly variable lengths (e.g promoters and gene bodies) into a single analysis.

If using the Core Workflow, multiple bedfiles can be specified in the Regions input directory, and will be processed sequentially. For example the `gencode.v44.human.genes.annotation.bed.gz` and `gencode.v44.human.promoters.annotation.bed.gz` are included by default. This allows efficient processing of multiple inputs via bash scripting.

Do annotated BED3+ files need strand information to work correctly?

No, strand information is not required for region-based analyses. However, including this information can be beneficial for downstream utility. Any columns beyond the first three (chromosome, start, end) will be carried through to the output result file.

What is the difference between a sample sheet and a metadata file?

They refer to the same thing! A sample sheet is a CSV file that contains information (metadata) about the samples, such as sample IDs, experimental groups and covariates.

1.18.5 Running Analyses

How do I check the quality of my data before analysis?

Use the `modality biological-qc` command to generate a quality control report. This report includes Pearson correlation heatmaps and PCA plots to assess sample relationships and data structure.

In the Biological QC report, why is the mean CpG coverage value lower than the mean sequencing depth?

The Biological QC report shows the mean CpG coverage per sample, and not the overall sequencing depth. The mean CpG-to-genome-wide coverage ratio is approximately 0.45-0.5.

How do I identify differentially methylated regions (DMRs)?

Use the `modality dmr` command. Provide the Zarr store, sample sheet, and experimental groups as inputs.

What is the difference between `modality get` and `modality dmr call`?

- `modality get` extracts methylation statistics (e.g., count, sum, mean, regional-fraction) for specific regions or windows.
- `modality dmr` identifies regions with statistically significant differences in methylation between groups.

How is the `--min-coverage` filter applied?

The `--min-coverage` filter is an option in `modality dmr call` and `modality get`, to exclude context positions (e.g. CpG sites) from the analysis. The filtering is applied to the mean context coverage value across all samples, prior to any grouping or aggregation steps. Context filtering is applied at the dataset level, not at the sample-level. If the mean coverage is less than the `--min-coverage` value, the context position is excluded from downstream analysis in the current modality XPLR command.

Is DMR calling normalised for coverage?

The method does not explicitly normalise for coverage at the CpG level, but coverage is implicitly accounted for by count aggregation. High coverage sites have more influence, and you can filter out low coverage sites by using the `--filter-context-depth` setting.

How should I interpret DMR results from bulk samples or cfDNA?

When calling DMRs using bulk methylation data, it's important to understand the statistical assumptions underlying the Wald test and how they apply to cell population-level measurements.

Differences between samples reflect genuine biological differences in the proportion of methylated cells, influenced by:

- Cell type composition
- Environmental factors
- Disease states
- Treatment effects
- Individual genetic variation

The logistic regression model assumes that:

- Methylation proportions follow a binomial distribution at the cell population level
- Differences between groups are systematic rather than random
- Sufficient coverage exists to accurately estimate cell population-level methylation

The Wald test in the context of bulk methylation analysis tests whether the population-level methylation proportions differ significantly between experimental groups. Bulk data may exhibit overdispersion (variance greater than expected under a simple binomial model) due to biological variability and technical noise. See *Overdispersion correction* for more details.

What are beta values, and does modality XPLR support them?

"Beta values" is a term that originates from methylation measurements from array-based technologies (e.g., Illumina 450K, EPIC arrays). They represent the ratio of methylated probe intensity to total probe intensity, calculated as M / (M+U), where M and U are the fluorescent signal intensities from methylated and unmethylated cytosines, respectively. Beta values range from 0 (unmethylated) to 1 (fully methylated).

Since modality XPLR analyses whole-genome sequencing data rather than array data, we use **methylation fractions** instead of beta values. Methylation fractions are calculated using the same formula M / (M+U), but here:

- M = count of reads with methylated cytosines at a position or strand-merged context (e.g., CpG).
- M+U = total read coverage at that position (`total_c`)

Methylation fractions provide the same 0-1 scale interpretation as beta values and can be extracted using the `modality get regional-frac` command. See *Extracting methylation statistics* for more details.

While beta values can be used to calculate methylation differences between conditions, modality XPLR goes further with a robust statistical approach to DMR calling:

- Use `modality dmr call` to identify differentially methylated regions (DMRs) with statistical testing

- The output includes the `mod_difference` column, which represents the mean methylation difference between groups (calculated as group2 - group1)
- DMR calling accounts for coverage differences and provides p-values and q-values for statistical significance
- This approach is superior to simple subtraction of beta values/fractions as it incorporates proper statistical modeling

See *Calling differentially methylated regions (DMRs)* for more details.

1.18.6 Outputs and visualisation

What outputs does modality XPLR generate?

modality XPLR generates:

- BED3+ format TSV or BED files with analysis results, depending on the command used.
- HTML reports for quality control and visualisation.
- INI files for configuring track plots.
- Provenance metadata in JSON format.

How do I visualise DMR results?

Once you have the output from the `modality dmr call` command, you can visualise the results using the `modality dmr plot` command. This generates interactive volcano plots to help interpret the DMRs. You can also use the `modality tracks` command to create track plots for specific genomic regions, which can be customised using INI files.

Why do I see a downsample warning in `modality dmr plot`?

The warning `Downsampling to improve html display performance for large datasets` is shown then the number of DMR data points exceed 500,000. In order to draw the volcano plot, downsampling is applied to preserve the overall data distribution while reducing point count. To address this warning, apply more stringent filtering for q-value (`-fq`) or mod-difference (`-fm`) to reduce the number of data points to plot.

Can I view results in IGV?

Yes, you can view the results in the Integrative Genomics Viewer (IGV) by loading the generated BED files. It may be necessary to remove the header lines, or insert a # character at the start of each header line to ensure IGV recognises them correctly.

Can I customise the plots?

You can customise the view of plots using the embedded tools in the HTML report. When you're ready, you can save the plot as a `.png` file.

You can edit the `modality tracks` plots by editing the input `.ini` files to customise figure titles, axis labels, and plot order.

Troubleshooting

What should I do if I encounter an error?

Check the `modality_cli.log` file in your working directory for detailed error messages. Common issues include:

- Missing required inputs: Ensure all required files and arguments are provided.
- Invalid region format: Use the correct format (e.g., `chr1:1000-2000`).
- Sample sheet mismatch: Verify that sample IDs in the sample sheet match those in the Zarr store.

Why is my output directory empty?

Ensure that the `--output-dir` option is specified and that the directory exists. If not, outputs are saved to the current working directory by default.

How do I update modality XPLR to the latest version?

Run the following command to upgrade modality XPLR:

```
pip install --extra-index-url https://europe-python.pkg.dev/prj-biomodal-modality/modality-
↪pypi/simple modality --upgrade
```

Tips and best practices

How can I see what modality XPLR is doing during long analyses?

Use the `-v` or `-vv` flags to increase verbosity. This will provide more detailed output during command execution, helping you understand the progress and any issues that arise.

How can I speed up my analysis?

If your analysis includes consuming a `.bed` file, indexing these files can improve performance. You will need to install Tabix and Bgzip. Then use the `--tabix` option to compress and index output files.

How do I organise my outputs?

Use the `--output-dir` option to specify a dedicated directory for each analysis. Output file directories are timestamped to avoid overwriting previous results.

Can I combine multiple Zarr stores?

Yes, use the `modality join` command to combine multiple Zarr stores into a single file for analysis.

What should I do if I need help?

Refer to the documentation, check the FAQ section, or contact **support@biomodal.com** for assistance.

1.18.7 Error handling

If an error occurs during execution, review the log file (`modality_cli.log`) in the current working directory. The log file contains detailed information about the error, including the command executed, the timestamp, and the error message.

Most errors in modality XPLR are accompanied by descriptive messages that indicate what went wrong and how to resolve it. Common issues include missing or incorrectly formatted input files, mismatched sample IDs, invalid argument values, or problems with output directories.

To resolve errors:

- **Read the error message carefully:** It usually points to the specific problem (e.g., missing input, invalid format).
- **Check your input files:** Ensure all required files exist, are correctly formatted, and match each other (e.g., sample IDs in the sample sheet and Zarr store).
- **Review your command arguments:** Use the `--help` flag to confirm you are using the correct options and formats.
- **Consult the log file:** The `modality_cli.log` file in your working directory contains detailed error information and can help with troubleshooting.
- **Increase verbosity:** Use the `-v` or `-vv` flags for more detailed output during command execution.
- **Validate file paths and permissions:** Make sure output directories exist and are writable.

If you are unable to resolve the issue, refer to the documentation, FAQ, or contact **support@biomodal.com** for further assistance.

1.18.8 Debugging

- Use the `--help` flag with any command to view its required and optional arguments.
- Check the `modality_cli.log` file for detailed error messages and stack traces.
- Enable debug mode by setting the highest verbosity level with `-vv` when running the command.
- Validate input files (e.g., Zarr stores, sample sheets, BED files) before running commands.
- If the issue persists, contact support or refer to the **FAQ section** in the documentation.

1.18.9 biomodal Software License

What license terms govern my use of modality XPLR?

modality XPLR is published under the **biomodal Software License**, accessible via the Customer documentation portal, Software section.

Is modality XPLR "open source"?

No. According to the Open Source Initiative, a licence with non-commercial restrictions cannot be open source. Also the license contains a restriction that you may not remove functionality that displays licensing information through the CLI. That restriction is also not permitted by the Open Source Definition. However, the license permits all non-commercial usage. Some activities which might be considered commercial are also permitted - see the terms of the licence here and the rest of this FAQ.

The license looks similar to the Apache-2.0 license which is open source. Are you sure it isn't open source?

Yes. The biomodal Software License is based on the Apache-2.0 license, but has been significantly modified such that it isn't open source due to the additional restrictions.

Can I build upon modality XPLR?

Yes, the license permits you to make modifications to modality XPLR. However, you may not redistribute modality XPLR (either as-is, or with modifications) unless you comply with the terms of the biomodal Software License. In particular, you must retain the prohibition on commercial use. You may not remove any legal notices that are contained within modality XPLR or packaged alongside it (for example, the NOTICE text file). You may not modify the modality XPLR code so as to remove any CLI functionality which displays the NOTICE text file.

Can I build upon modality XPLR and resell the resulting combined work?

No.

Can I build modality XPLR into a toolkit or other product which I make available commercially to third parties?

No. And this prohibition covers both code you redistribute, or making the functionality of your product available to third parties. That can be over a network, or on a batch basis, for example.

Can I produce research papers based on analyses performed using modality XPLR?

Yes.

Can I use modality XPLR to provide a commercial service to third parties, such as consulting or analytical services, or a service provided on an outsourced basis?

No.

If I modify modality XPLR, do I need to publish my changes under the same license?

No. The license is not a copyleft license, so you do not need to contribute modifications back to the project, or publish them anywhere, and you retain any rights to your modifications.

If you redistribute the code or any modifications to it, see below.

If I redistribute modality XPLR, must I do so under the biomodal Software License?

No, you are permitted to apply your own license terms, provided that these do not conflict with the biomodal Software License. In particular you **must** retain the prohibition on commercial use (as defined in the biomodal Software License), and you must retain the CLI functionality which displays the license and restrictions. You must also ensure any modified files carry prominent notices stating that you modified the files.

In practice, you will almost always find it easier to use the biomodal Software License itself if you redistribute modality XPLR code, rather than trying to redistribute under a different license.

Do I need to do or sign anything in order to use modality XPLR?

Other than to install the software according to the user documentation, there are no additional steps. You do not need to "accept" the license, but your use of the software outside the scope of the license will be a violation of our copyright (this is the way that open source licenses work - the biomodal Software License is not an open source license, but has several characteristics in common with one).

Does biomodal have the right to change the license for modality XPLR?

Yes, future versions of modality XPLR may be released under a different license. We can also issue you an additional permission if the use you envisage falls within the existing non-commercial prohibition. That additional permission may be on paid-for commercial terms, or come with additional conditions, and it's granted entirely at our discretion. Please contact us at support@biomodal.com for more information.

CHAPTER

TWO

HOW TO USE THIS GUIDE

2.1 Navigation + searching

This guide is structured to help you quickly find the information you need. You can navigate through the major sections using the tabs at the top of this page, and then by using the table of contents on the right side of the page.

Additionally, you can use the search function in the top right-hand corner of the page to find specific topics or keywords across all biomodal software pages.

To get back to this page, click the 'home' icon.

2.2 Support links

For any questions or issues, please contact your biomodal support team at **support@biomodal.com**.

CHAPTER

THREE

RELEASE NOTES

A summary of recent changes to the software is provided below. For a full list of changes, please refer to the published release notes PDF by clicking on the version number.

Release version	Date	Description of changes
v1.0.0	2025-Nov-17	• Official stable release following beta testing. • Added authentication. • Added HTML report for `get` commands. • Added optional regional heatmap visualisation for `get regional-frac` commands. • Added `prepare-regions` entrypoint for CpG-informed region segmentation and annotation. • Added `--num-contexts` filter in `dmr call` and `get` commands. • Added controls for merging CpG strands in `biological-qc`, `get` and `export` commands. • Unified behaviour to merge CpG strands by default in all entrypoints, disable with `-dsm`. • Unified behaviour of `--aggregate-by-group` across entrypoints. • Added `--order-by-group` option to improve visualisations by sample metadata. • Improved labelling of annotation types in the DMR Report HTML. • Improved support for handling Zarr store merging. • Updated Core Workflow to v1.3 with improved usability and performance. • Handle DMR calling with insufficient samples for statistical tests. • Bug fixes and improvements.
v1.0.0b4 (beta)	2025-Oct-03	• Resolved issue with v1.0.0b3 installations in new environments due to versioning of CLI dependency. • Bug fixes and improvements.
v1.0.0b3 (beta)	2025-Sep-22	• Ensure consistency of operation names and output file prefixes across entrypoints. • Standardisation of `--min-coverage` method to use mean context coverage across all samples. • Performance improvements to `biological-qc` Pearson correlation and PCA. • `tracks` smoothing method updated to use coverage-weighted Gaussian kernel-based approach. • Core Workflow updated to v1.2 with usability improvements and overdispersion toggle. • Improved documentation, `--help` text and FAQs. • Bug fixes and improvements.
v1.0.0b2 (beta)	2025-Aug-04	• Biological QC `--group-name` option updated to `--aggregate-by-group`. • Include more metadata information in the Biological QC Sample Information table. • Added metadata-based colour formatting for PCA plots on the Biological QC HTML report. • Added DMR statistics table to the HTML report (`dmr plot` function). • Fixed issue with methylation trace and mod-difference smoothing on Tracks plot. • Fixed issue with `--merge-cpg-strands` in `dmr call` and `tracks`. • Updated support for Core Workflow. • Fixed contig name issue with pre-prepared bed files for mouse references and added mm10p6.
v1.0.0b1 (beta)	2025-Jun-30	Initial beta release.