methylprep Documentation

Release 1.6

Life Epigenetics

May 13, 2022

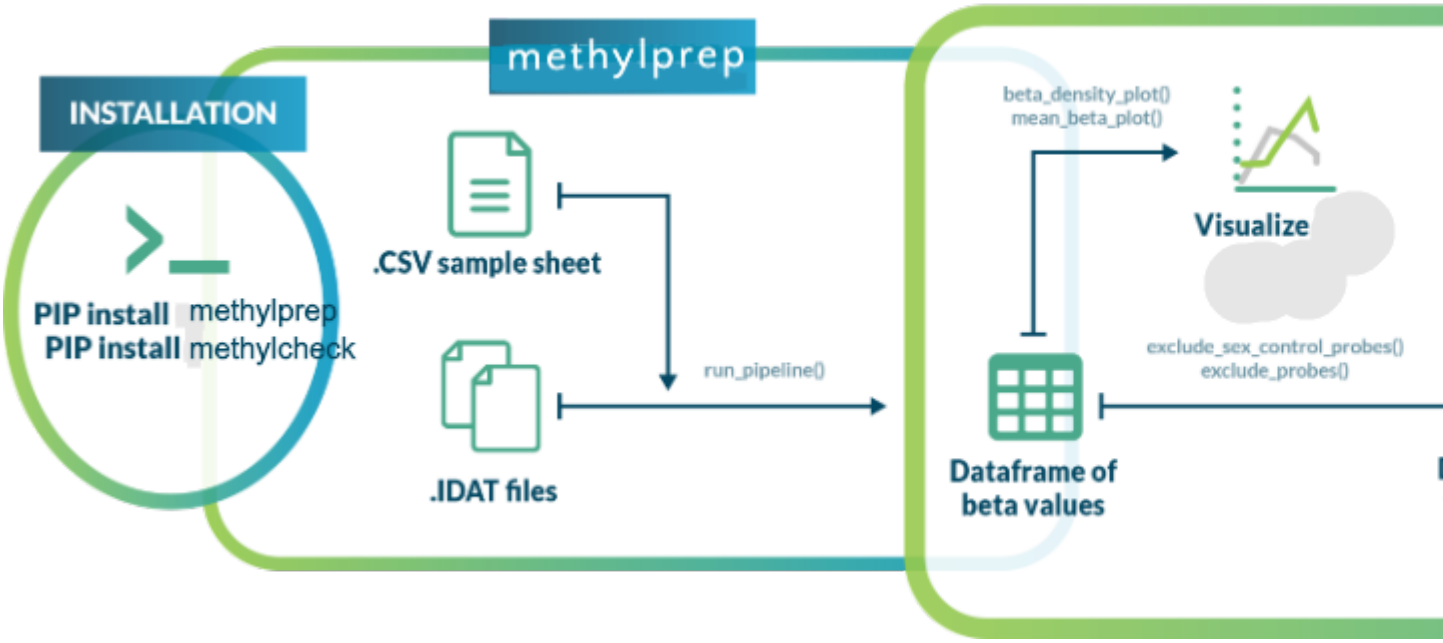
Contents:

CHAPTER 1

Getting Started

`methylprep` is a python package for processing Illumina methylation array data. View on ReadTheDocs.

1.1 Methylprep is part of the methylsuite



`methylprep` is part of the methylsuite of python packages that provide functions to process and analyze DNA methylation data from Illumina's Infinium arrays (27k, 450k, and EPIC, as well as mouse arrays). The `methylprep` package contains functions for processing raw data files from arrays and downloading/processing public data sets from GEO (the NIH Gene Expression Omnibus database repository), or from ArrayExpress. It contains both a command line interface (CLI) for processing data from local files, and a set of functions for building a custom pipeline in a jupyter notebook or python scripting environment. The aim is to offer a standard process, with flexibility for those who want it.

`methylprep` data processing has also been tested and benchmarked to match the outputs of two popular R packages: sesame (v1.10.4) and minfi (v1.38).

1.2 Methylsuite package components

You should install all three components, as they work together. The parts include:

- `methylprep`: (this package) for processing `idat` files or downloading GEO datasets from NIH. Processing steps include
  - infer type-I channel switch
  - NOOB (normal-exponential convolution on out-of-band probe data)
  - poobah (p-value with out-of-band array hybridization, for filtering lose signal-to-noise probes)
  - qualityMask (to exclude historically less reliable probes)
  - nonlinear dye bias correction (AKA signal quantile normalization between red/green channels across a sample)
  - calculate beta-value, m-value, or copy-number matrix
  - large batch memory management, by splitting it up into smaller batches during processing

- `methylcheck`: for quality control (QC) and analysis, including
  - functions for filtering out unreliable probes, based on the published literature
    - Note that `methylprep process` will exclude a set of unreliable probes by default. You can disable that using the –no_quality_mask option from CLI.
  - sample outlier detection
  - array level QC plots, based on Genome Studio functions
  - a python clone of Illumina's Bead Array Controls Reporter software (QC)
  - data visualization functions based on `seaborn` and `matplotlib` graphic libraries.
  - predict sex of human samples from probes
  - interactive method for assigning samples to groups, based on array data, in a Jupyter notebook
- `methylize` provides more analysis and interpretation functions
  - differentially methylated probe statistics (between treatment and control samples)
  - volcano plots (which probes are the most different?)
  - manhattan plots (where in genome are the differences?)

1.3 Installation

`methylprep` maintains configuration files for your Python package manager of choice: pipenv or pip. Conda install is coming soon.

```
>>> pip install methylprep
```

or if you want to install all three packages at once:

```
>>> pip install methylsuite
```

1.4 Tutorials and Guides

If you're new to DNA methylation analysis, we recommend reading through *this introduction* in order get the background knowledge needed to best utilize `methylprep` effectively. Otherwise, you're ready to use `methylprep` for:

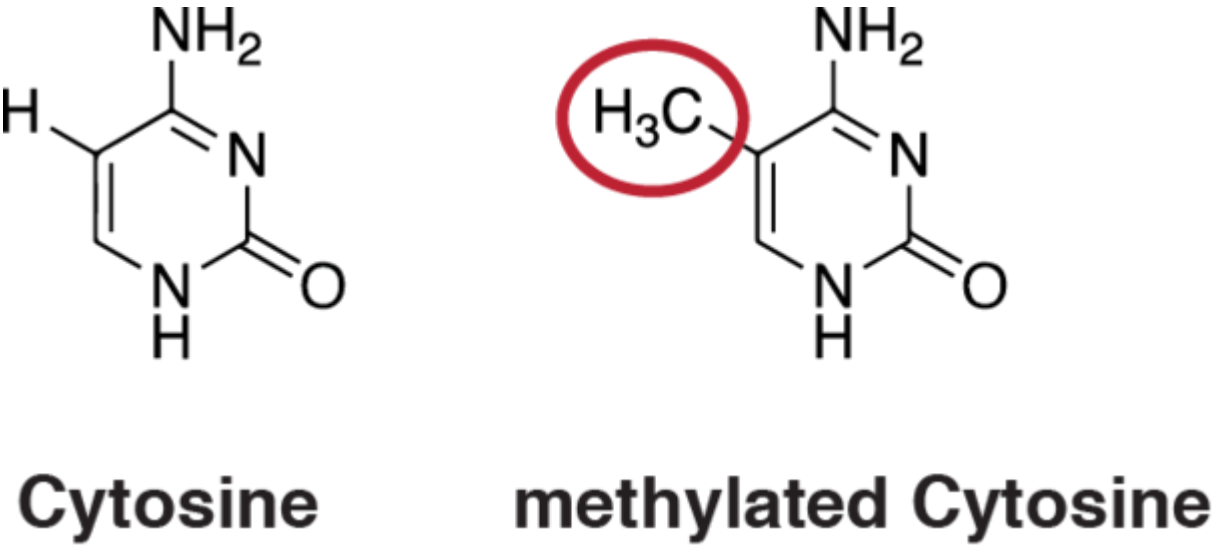
- processing *your own methylation data*
- downloading *unprocessed data* (like IDAT files) from GEO.
- downloading *preprocessed data* (like beta values) from GEO.
- building a composite dataset *using control samples* from GEO.
- building a composite dataset from GEO data *with any keyword you choose* (e.g. combining all GEO datasets that have methylation data from patients with brain cancer).

1.4.1 Introduction to DNA Methylation Analysis

In this introduction, we'll cover what DNA methylation is, where it occurs, how we measure it, and common methods for cleaning/pre-processing data before analysis. At the end of this introduction, we also provide a list of papers, videos, and documentation pages that provide more detail on these topics than we can go into in this quick primer.

Introduction

DNA methylation occurs when a methyl group (CH3) is transferred to the C5 position of a cytosine base. This is a mechanism for gene regulation. Methylation can be a signal for the cell to recruit inhibiting proteins. The methyl group can also prevent transcription factors from binding to DNA, thus preventing transcription factors from upregulating the affected gene.



So where does methylation commonly occur?

The human genome has areas which have a high ratio of CG basepairs. These GC-rich regions are known as CpG islands (**C**ytosine-**p**hosphate-**G**uanine), or CGIs. These regions are generally 500-1500bp with >60% GC-content. CpGs are not to be confused with CG basepair bonds. A CpG island references the nucleotides in sequence and on one strand of DNA (linked by the phosphodiester bond, hence the p in CpG), *not* a C linked to a G in a basepair bond. See the example below, which shows a CpG site on the left and a CG basepair bond on the right.

CGIs tend to be in the promoters of genes and usually contain the 5' end of the transcript. In mammals, it is estimated that 70-80% of cytosines in CGIs are methylated. Unmethylated CpG islands are typically associated with active promoter regions *[1]*.

However, CpG islands are not the only places where methylation occurs. Differential methylation has also been observed in the "shores," "shelves," and the "open sea" (these terms are the names of regions that are varying distances

from the CpG islands) *[2]*. Shores are up to 2kb from the CpG island and shelves are from 2kb to 4kb from the CpG island. The open sea refers to isolated regions that do not have a specific designation. See figure below.

Methylation also plays an important role in cellular development by silencing some genes and shaping the pathway the cell uses to differentiate itself. The unique and stable methylation patterns of various types of tissue have been documented, but differential methylation has also increasingly been studied in several diseases in recent years *[[3]*, *[4]*, *[5]*]. DNA methylation occurs over time in normal, healthy cells as a response to environmental stimuli. Another important note is that methylation is reversible, and there is ongoing research into how lifestyle changes can affect methylation patterns in the genome *[6]*.

Measuring Methylation

One of the common methods of measuring DNA methylation is a methylation array, and the most commonly used arrays are manufactured by Illumina. Illumina has released several array types for humans–27k, 450k, EPIC, and EPIC+–as well as a mouse array. (`methylsuite` supports all human and mouse arrays, with options for custom arrays as well). 450k arrays have been discontinued; 90% of the probes that were on 450k arrays are covered by the new EPIC array. The EPIC+ array covers all of the EPIC and 450k probes, in addition to double coverage on some probes of interest for quality control purposes.

These arrays make use of bisulfite sequencing to detect whether specific CpG loci are methylated. With bisulfite sequencing, methylated cytosines are unaffected. Unmethylated cytosines are converted to uracil and then a thymine. So in the new DNA sequence, instead of an unmethylated cytosine, there is a thymine. See below for an example.

```
Original DNA sequence:          A C G A T C G C G A
Bisulfite converted sequence:   A C G A T T G T G A
```

We see only the first cytosine remains the same after bisulfite conversion, so we can assume that that particular C is methylated. We can also assume the other two cytosines at the end of the sequence are unmethylated because they were converted to thymines. However, it has been observed that most CpG loci are "in phase" with each other: if one locus is methylated, there is a high probability that the nearby CpG sites are also methylated.

Array Design

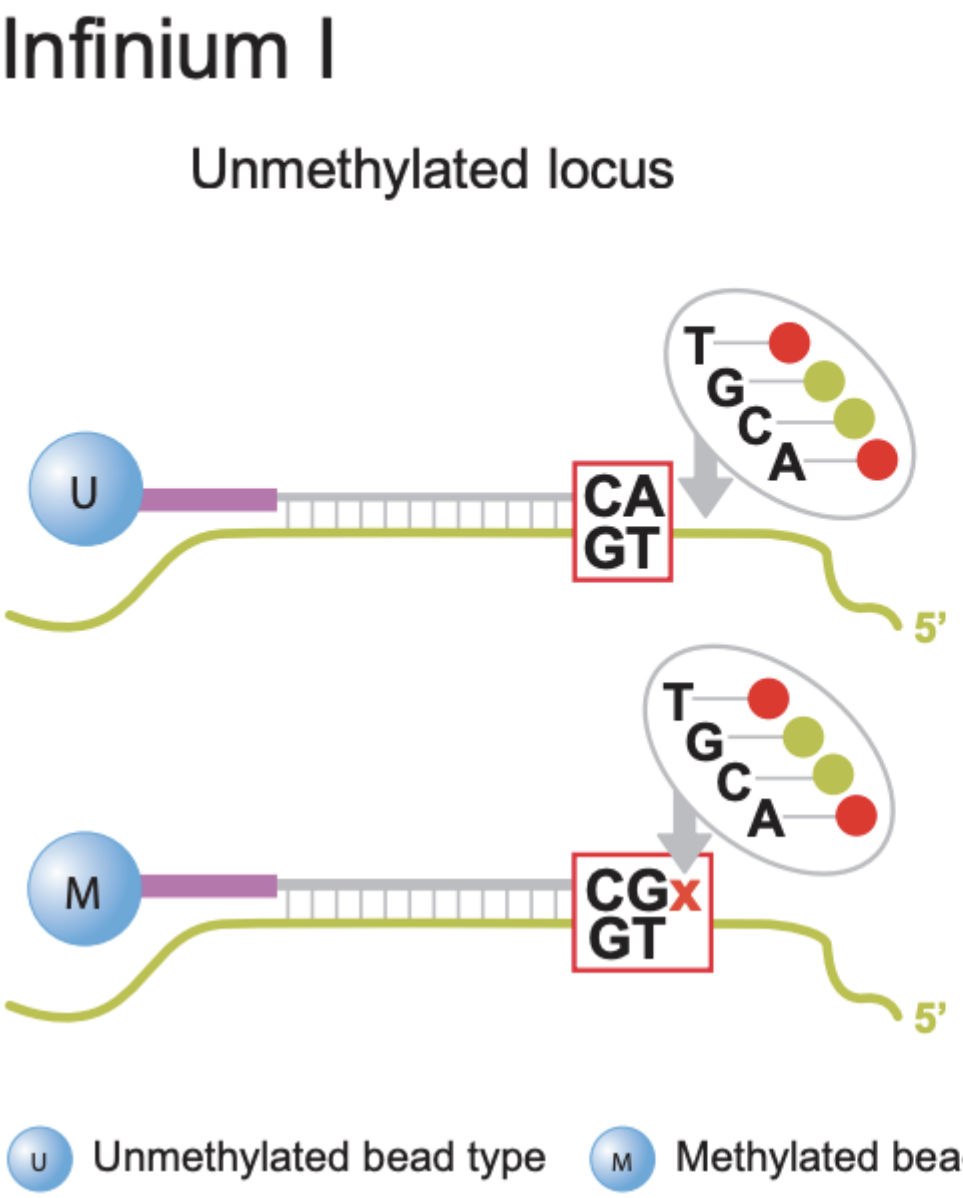
Each methylation array is covered in thousands of microwells. Each well houses a silica bead with an oligonucleotide probe targeting a specific DNA sequence. The probe sequence will stop one nucleotide short of the target locus. The target DNA binds to the probe and a fluorescently labelled ddNTP will bind to the end of the sequence, complementing

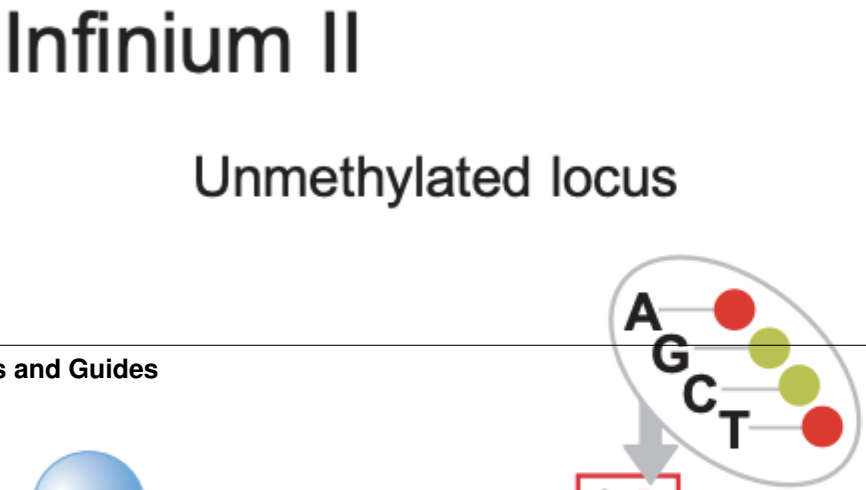
whatever base was present at the target locus (cytosine or thymine). The probes are then excited by a laser and the output frequency is measured.

To confuse things further, there are two types of probes. Both probes are approximately 50bp of target DNA sequence and both types of probe are specific to ONE nucleotide pair's methylation state. For example, if the probe's sequence is as follows:

```
CTACAAATACGACACCCGCAACCCATATTTCATATATTATCTCATTTAAC
```

We would be evaluating the methylation state of the cytosine in a CpG site immediately following this target sequence.


Infinium I
Unmethylated locus
T
G
C
A
U
CA
GT
5'
T
G
C
A
M
CG
x
GT
5'
U
Unmethylated bead type
M
Methylated bead type


Infinium II
Unmethylated locus
A
G
C
T

Infinium I: Type I probes require two beads for each locus. One bead type (M) is for the methylated state of that locus. If the cytosine is methylated and does *not* convert after bisulfite conversion, the M bead will bind at that locus (see top right panel of figure above). However, if the cytosine is not methylated and *does* convert to a thymine, the second bead type (U) will bind to the thymine (see top left panel of figure above). Type I probes are measured by a single channel. These probes are based on the assumption that CpG methylation is correlated in a 50bp span. Underlying CpG sites (other CpGs in the target DNA sequence that are not the final target CpG) are treated as "in phase" with the target locus. In other words, if we revisit our example probe sequence above, that means we would assume that ALL the CpGs in that sequence are in the same methylation state as the one we targeted.

Infinium II: Type II probes will bind to methylated OR unmethylated bases at the target locus (see bottom panel of figure above). The ddNTP that is incorprated at the end (either an A to match the T for an unmethylated state, or a G to match the C that remains unchanged for a methylated state) will be excited by a laser. Thus, this probe type needs to be measured by two channels (green for methylated, red for unmethylated). Type II probes are more sensitive to underlying CpG sites because the same probe is used to evaluate methylated sequence (where the cytosines are preserved) and unmethylated sequence (where the cytosines are converted to thymines). Thus, type II probes tolerate a maximum of 3 other CpGs in a probe sequence and they will be assigned degenerate "R" bases so that the complimentary target DNA sequence is still able to bind to the probe whether those bases are converted to thymines or not (i.e. regardless of their methylation state). This means that, unlike type I probes, the target locus can be evaluated independently of nearby CpG sites and the "all or nothing" methylation assumption of Type I probes is not applicable.

There is a distinct advantage to the use of type II probes in that they only need one probe per locus, so building an array of type II probes could cover twice as many loci as an array of only type I probes. However, the drawback of type II probes, as covered above, is that they do not function well in regions of high CpG density (such as CpG islands). Also, due to the difference in chemistry, type I probes are considered advantageous for the extremes of methylation states (either completely methylated or completely unmethylated) because the peaks of the beta distribution are spread further apart.

Betas and M-Values

Beta values and M-Values are two ways to measure methylation. Betas are typically easier to interpet, as they range from 0 to 1 and represent the proportion of how many cells had a methylated base for that probe site. We would expect 0's and 1's in a perfect experiment (0 for an unmethylated locus and 1 for a methylated locus), but the reality is that technical noise and other types of variation make it very uncommon to encounter either of those scenarios. More often, beta values lie somewhere between 0.1-0.9, and we see a bimodal distribution with peaks at either end of that range when beta values are plotted on one line.

The beta value is calculated with the following formula:

Where and represent the methylated and unmethylated signals respectively. And is a constant, typically set to 100, that stabilizes the beta value when both the signals are low. Note that is sometimes left out of common preprocessing piplines, which results in being equal to the simple proportion of M/(M+U). `minfi` uses Illumina's recommended constant of 100, while `SeSAMe` and `methylprep` both set to 0 by default (although, in the case of `methylprep`, users may set `minfi=True` when processing raw data, which will mimic `minfi`'s output).

M-values take the ratio of methylated and unmethylated intensities and apply a log transformation. M-values are unbounded, so they can be more difficult to interpret and compare. A value around 0 means that the methylated and unmethylated signals are close to equal. A positive M-value means that the methylated signal is higher, while a negative M-value means that the unmethylated signal is higher.

Where and represent the methylated and unmethylated signals respectively. And is a constant, typically set to 1, that stabilizes the M-value when both the signals are low. Again, `methylprep`'s is set to 0 by default, to match `SeSAMe`'s output.

Beta values have been shown to have heteroscedasticity outside of the middle methylation range (0.2-0.8). Additionally, they are bounded, which means they violate the assumptions of the Gaussian distribution that many statistical models apply. The M-value does not have these challenges, but has a much less intuitive biological meaning. Depend-

ing on what analyses are being run, either beta or M-values (or both) may be used. In practice, either value works well. `methylsuite` focuses on using beta values because they are more intuitive.

Other Important Considerations

background correction, normalization methods, filtering, etc

One of the first processing steps of `methylprep` is to **infer the color channel** for type I probes. Recall that type I probes operate on a single color channel. Each probe's color channel should be provided as part of the manifest file. Occasionally, these manifest files are inaccurate. `methylprep` will infer and reset the probe's channel in the manifest based on the actual signals from each channel.

Another issue that arises with multi-channel usage is **dye-bias**. This is a phenomenon where the red and green channels have different signal intensities. They have to be corrected in order for the rest of the processing pipeline to be accurate. `methylprep` uses quantile normalization between red and green channels within a sample to correct this. This is also known as a nonlinear method of dye bias correction (comparable to the one `SeSAMe` uses).

Out-of-band (OOB) probe data is commonly used as part of the preprocessing pipleline. Recall that type I probes utilize only one color channel but have two probes for one locus. The probe that does not match the actual methylation state of the sample still captures a measure of fluorescence that can be used to help estimate background noise. In other words, if a type I probe is operating on the green channel to capture the methylation state of the locus, we will still have flourescence measures from the red channel (and that fluorescence from the red channel is "OOB" data).

There are many ways to normalize methylation data, but the most widely used method is normal-exponential convolution on out of band probes (**NOOB**). The normal-exponential (normexp) convolution model was developed as part of the RMA algorithm for Affymetrix microarray data (see recommended reading section below for more details on this). The model assumes that the observed intensities are the sum of background noise and true signal components. The background is *normally* distributed and the signal is *exponentially* distributed. NOOB is simply using the normexp model on out of band probe data.

Detection P-values are a measure of how likely it is that a given signal is background fluorescence. There are a few methods of calculating these detection p-values. Illumina's GenomeStudio uses negative control probes in the array to parametrize a Guassian distribution and recommends p-values>0.05 to be excluded. `minfi`, another popular package for methylation array analysis in R, uses a similar method but with the M and U probes combined into one signal and a background distribution to calculate the Z-score. The background distribution is calculated by combining the color channel(s) of the corresponding probe (type II probes will have both red and green). And they recommend the more stringent exclusion criteria of p-values>0.01 being excluded. `SeSAMe` and `methylprep` implement a different method, known as **pOOBAH** (**P** value **O**ut **O**f **B**and probes for **A**rray **H**ybridization) where they use the OOB signal of all type I probes to calculate an empirical cumulative distribution function. `methylprep` users have the option to save pOOBAH values as an output file after sample processing.

Recommended Reading

While we have done our best to give a concise yet detailed introduction that should cover all you need to know to use `methylsuite`, there is a chance that you are confused (or intrigued) and wish to cover some of these concepts in more detail. Here we provide a few suggested resources:

- For more about CpG Islands see this article
- For more about bisulfite sequencing, see this short video
- For more about methylation arrays, see Illumina's product sheet or Illumina's webpages on microarrays
- For more about type I vs. type II probes, see this piece from Illumina
- For more information on beta values and M-values, see this article

- For more information about background correction, here is a helpful video.
- This paper is a good resource to learn more about normalization.
- The paper on the RMA algorithm that was used to develop NOOB is located here here.
- `methylsuite` is partially modeled after R's minfi and SeSAMe packages (with expansions and improvements), so their documentation may prove helpful. Also recommended are these two papers that describe SeSAMe and minfi in greater detail.
- Dr. Wanding Zhou (SeSAMe's creator) has published a few other helpful papers, as well. This one includes more information on pOOBAH normalization and this one is a comprehensive characterization of the EPIC and 450k microarrays.

References

1. Jung, M., and G. P. Pfeifer. "CpG Islands." In Brenner's Encyclopedia of Genetics (Second Edition), edited by Stanley Maloy and Kelly Hughes, 205–7. San Diego: Academic Press, 2013. https://doi.org/10.1016/B978-0-12-374984-0.00349-1.
2. Cross, S. H., and A. P. Bird. "CpG Islands and Genes." Current Opinion in Genetics & Development 5, no. 3 (June 1995): 309–14. https://doi.org/10.1016/0959-437x(95)80044-1.
3. Fan, Jian, Jun Li, Shicheng Guo, Chengcheng Tao, Haikun Zhang, Wenjing Wang, Ying Zhang, Dake Zhang, Shigang Ding, and Changqing Zeng. "Genome-Wide DNA Methylation Profiles of Low- and High-Grade Adenoma Reveals Potential Biomarkers for Early Detection of Colorectal Carcinoma." Clinical Epigenetics 12 (April 21, 2020): 56. https://doi.org/10.1186/s13148-020-00851-3.
4. Reinius, Lovisa E., Nathalie Acevedo, Maaike Joerink, Göran Pershagen, Sven-Erik Dahlén, Dario Greco, Cilla Söderhäll, Annika Scheynius, and Juha Kere. "Differential DNA Methylation in Purified Human Blood Cells: Implications for Cell Lineage and Studies on Disease Susceptibility." PLOS ONE 7, no. 7 (July 25, 2012): e41361. https://doi.org/10.1371/journal.pone.0041361.
5. Moss, Joshua, Judith Magenheim, Daniel Neiman, Hai Zemmour, Netanel Loyfer, Amit Korach, Yaacov Samet, et al. "Comprehensive Human Cell-Type Methylation Atlas Reveals Origins of Circulating Cell-Free DNA in Health and Disease." Nature Communications 9, no. 1 (November 29, 2018): 5068. https://doi.org/10.1038/s41467-018-07466-6.
6. Hibler, Elizabeth, Lei Huang, Jorge Andrade, and Bonnie Spring. "Impact of a Diet and Activity Health Promotion Intervention on Regional Patterns of DNA Methylation." Clinical Epigenetics 11, no. 1 (September 11, 2019): 133. https://doi.org/10.1186/s13148-019-0707-0.

1.4.2 General Walkthrough

Here we will provide a few examples of how to use various `methylprep` functions. We'll focus on examples using the CLI, as that is the recommended interface for `methylprep`, but there's also a section near the end of the tutorial that demonstrates how to run `methlyprep` from an IDE.

Note: sample sheets are recommended but not necessary to run `methylprep`. We will cover a few ways to work with a data set without a sample sheet.

Set Up

If you haven't done so already, run this command from your terminal to install methylprep:

```
>>> pip install methylprep
```

Downloading from GEO

The first step in this tutorial will be using `methylprep` to acquire a dataset from GEO. `methylcheck` and `methylize` work best when users include a sample sheet along with the IDAT files. The meta data contained in sample sheets is useful when running QC or analyses.

When downloading from a GEO dataset, `download` will attempt to find and download the associated sample sheet. If there is none, `methylprep` will automatically generate one. Users may make one of their own as well. The Illumina sample sheet is the standard format.

Note: if you already know you'd like to work with pre-processed GEO data (like beta values), check the beta_bake section of the special cases tutorial.

Our GEO dataset

For our tutorial, we will download GEO data from this experiment where researchers examined differential methylation among high and low grade gliomas (an aggressive form of brain tumors). This is a relatively small dataset (16 samples) of surgical resections processed using the Infinium MethylationEPIC assay.

The authors analyzed methylation data from 6 II-grade, 2 III-grade and 8 IV-grade gliomas from individual surgical resections. Stage I and II gliomas are defined as low grade, while stage III and IV are defined as high grade.

We will run the `download` command with the `-v` option to get more information about what processes are running. This should be run from the command line.

```
  >>> python -m methylprep -v download -i GSE147391 -d <filepath>

Where ``<filepath>`` is the directory you would like the processed data and output␣
↪files to be stored.
```

`methylprep` will search the GEO database and find the dataset we indicated. It will also find the file with associated metadata and parse through it, indicating any extra fields in the sample sheet. In our case, there are 5 additional fields, which will come in handy in later analysis:

```
INFO:methylprep.processing.pipeline:Found 5 additional fields in sample_sheet: source␣
↪| histological diagnosis --> histological_diagnosis | gender | description | Sample_
↪ID
```

`methylprep` will begin downloading, unpacking, and processing the IDAT files after downloading the file manifest and the parsing sample sheet. The automatic processing can be turned off with the `-o` option.

Processing time depends on the type of array (450k arrays process faster than EPIC arrays) and on the size of the dataset. Datasets with more than 100 samples will, by default, be chunked into batches of 100 for more efficient processing. Batch size is adjustable with the `--batch_size` argument.

After the files are processed, we're ready to load the files into `methylcheck` for QC. See `methylcheck` documentation for instructions and more details.

Processing Your Own Data

It is often the case that users have their own idat files that they would like to process. Instead of using the `download` command, we will use the `process` command. This is the main workhorse of the `methylprep` package, and includes extra options for customization that the `download` command lacks.

Users should note that the sample sheet is optional, but the ***manifest file*** is not. Make sure there is a manifest file included with the IDAT data, *especially* if you are analyzing data from a custom array, as `methlyprep` will use the manifest file to determine what probes are included in the custom array, which control probes are present, etc.

Note: If users are interested in processing their own files, the folder containing the IDATs, manifest, and sample sheet needs to be unzipped before being processed.

Once again, we'll run this from the command line:

```
>>> python -m methylprep process -d <filepath> --all
```

`<filepath>` specifies where the manifest and IDAT files (and sample sheet, if any) are stored.

The `--all` option at the end tells `methylprep` to save output for ALL of the associated processing steps.

- beta_values.pkl
- poobah_values.pkl
- control_probes.pkl
- m_values.pkl
- noob_meth_values.pkl
- noob_unmeth_values.pkl
- meth_values.pkl
- unmeth_values.pkl
- sample_sheet_meta_data.pkl

By default, the output is usually:

- beta_values.pkl
- noob_meth_values.pkl
- noob_unmeth_values.pkl
- control_probes.pkl

These files will be saved in the same directory as the data. `methylprep` will also create two folders to store processed .csv files. If users are interested in the probes that failed poobah, they are included in the .csv file outputs.

The default output is all that is needed to run qc reports from `methylcheck`. However, other useful outputs like the poobal_values.pkl can optionally be included in the qc functions for greater details on sample quality.

Using methylprep from a Jupyter Notebook

`methylprep` also offers a scikit-learn style interface for users to run within jupyter notebooks or a similar IDE. We recommend using the CLI for `methylprep`–IDEs tend to process more slowly, especially for large batches–but users are able to get most of the package's functionality from within an IDE.

`methylprep.run_pipeline`

This command is the IDE equivalent of `methylprep process`. The output, if left to default, is a set of data containers. Alternatively, users may specify `betas=True` or `m_values=True` to get a dataframe of their chosen values. We also recommend setting `export=True` to get output files saved as well. That way you can easily load the data in the future instead of running this command every time you open the notebook. The only required argument for this function is the directory where the raw data is stored.

`run_pipeline` will take almost all of the same arguments that `process` will take from the CLI.

```
from methylprep import run_pipeline
from pathlib import Path
filepath = Path('~/tutorial/GPL21145')

data_containers = run_pipeline(filepath, export=True)
```

For more information on optional arguments, exports, etc, run `help(methylprep.run_pipeline)`

methylprep.make_pipeline

Users may specify ['all'] for both the 'steps' and 'exports' arguments to get the same functionality as the CLI code `methylprep process -d <filepath> --all`. Alternatively, if users are interested in customizing their pipeline, they may list the steps they'd like to include and leave out any they would like to skip.

```
import methylprep
from pathlib import Path
filepath = Path('~/tutorial')

# choose any combination of:
# ['infer_channel_switch', 'poobah', 'quality_mask', 'noob', 'dye_bias']
mysteps = ['all']

# choose any combination of:
# ['csv', 'poobah', 'meth', 'unmeth', 'noob_meth', 'noob_unmeth', 'sample_sheet_meta_
↪data', 'mouse', 'control']
myexports = ['all']

# make_pipeline will auto-run the pipeline
# make_pipeline will also take the same kwargs as run_pipeline
methylprep.make_pipeline(data_dir=filepath, steps=mysteps, exports=myexports,␣
↪estimator=None)
```

For more information on this function, run `help(methylprep.make_pipeline)`

1.4.3 Specialized Functions Walkthrough

We cover the most high level use cases in our general walkthrough. However, there are some functions available in `methylprep` for less common (more specialized) use cases that we'll cover here.

Building a composite dataset with `alert` and `composite`

`methylprep` includes a few functions that can be used in a pipeline to merge datasets from GEO into one larger dataset. Say, for example, that you are particularly interested in investigating data from glioblastoma (an aggressive form of cancer that can occur in the brain or spine).

You may use the `alert` command to compile a csv of GEO datasets that mention the keyword glioblastoma. The csv will contain, at most, 500 rows (a query limit set by GEO).

```
>>> python -m methylprep alert -k "glioblastoma"
```

After looking through the .csv file and eliminating any datasets that aren't useful, you can pull the column of GSE id's and run `methylprep composite` from the CLI. `composite` will run the `download` command in a loop given a text file with a list of GSE id's.

From our geo alert glioblastoma .csv file, here are 4 datasets that we might pull into one larger composite dataset. One thing to be aware of is to make sure that the platform types are all the same. Otherwise you may be combining 450k array data with EPIC array data.

Here is a peek at the txt file with the list of GSEs. Make sure that the format is such that there is just one GSE id per line.

```
>>> cat glioblastoma.txt
GSE161175
GSE122808
GSE143842
GSE143843
```

Now we can run the `composite` command and it'll download/process data from each GSE in the list. You need to use `-l` to specify the .txt file that contains the GSEs and `-d` to specify the directory where to save the data.

```
>>> python -m methylprep composite -l ~/tutorial/glioblastoma.txt -d ~/tutorial
```

The `meta_data` function has a similar application for building a composite dataset when using the `--control` option. See below for the walkthrough of how to use that command to build a dataset of healthy control samples instead of experimental/disease case samples.

using the `meta_data` function

Creating a meta_data.pkl file

GEO data will always be accompanied by a MINiML file (an .xml file that contains information about the platform, series, and samples). `methylprep` will handle downloading and parsing this as part of the `download` or `process` functions, with the output stored in the meta_data.pkl file. So if you're going to run either of those, you don't need to worry about running `meta_data` separately.

However, if you are interested in creating a meta_data.pkl file without downloading/processing data, the `meta_data` function will come in handy.

```
>>> python -m methylprep meta_data -i GSE147391 -d ~/tutorial
INFO:methylprep.download.miniml:found 16 idat files; updated meta_data.
INFO:methylprep.download.miniml:Final samplesheet contains 16 rows and 9 columns
```

Filtering samples with `meta_data`

`meta_data` also has a few experimental commands that may be useful in filtering unwanted samples and reducing the time needed to process datasets. For example, consider the glioma dataset from GSE147391 (the dataset discussed in the general walkthrough). This dataset contains methylation profiles of high and low grade gliomas from 16 patients (8 high grade gliomas and 8 low grade gliomas). If we wanted to ignore the high grade gliomas and focus on the low grade gliomas, we could use `meta_data`'s `--keyword` (`-k` for short) command to search for samples that are indicated as "grade II" in the sample sheet and only retain those values.

```
>>> python -m methylprep meta_data -i GSE147391 -d ~/tutorial -k "grade II"
INFO:methylprep.download.miniml:found 16 idat files; updated meta_data.
INFO:methylprep.download.miniml:Filtering keyword: Retained 8/16 samples matching␣
↪`grade II``.
INFO:methylprep.download.miniml:Final samplesheet contains 8 rows and 9 columns
```

Now our meta_data.pkl file only contains information on the low grade gliomas, and we can exclude the other samples from our future analyses. If we haven't already used `methylprep process` on our data, we can also include the `--sample_name` or `-n` flag with the list of the samples we want to run (the low grade gliomas) to save time processing the data.

building a composite dataset using `meta_data`

To better demonstrate the use case for `meta_data's --control` command, we will work with a new dataset. GSE163970 is a dataset that examines differential methylation in 253 patients with Paget's disease of bone (PDB) as compared to 280 controls.

We might be able to cut down processing time for this large dataset if we are just interested in the control data and want to exclude the PDB samples. One example of a scenario in which this would be helpful is building a large composite dataset that includes only healthy control samples from multiple GEO datasets.

Before downloading any data from GEO, we'll run this:

```
>>> python -m methylprep meta_data --control -i GSE163970
```

`methylprep` will look through our computer and realize that we don't have any data or metadata files downloaded. It will then pull the MINiML file from the GEO database and search it for samples with the word "control" or "ctrl" in the metadata.

```
INFO:methylprep.download.miniml:Downloading GSE163970_family.xml.tgz
INFO:methylprep.download.miniml:Filtering controls: Retained 260/492 samples matching␣
↪['control', 'ctrl'].
INFO:methylprep.download.miniml:Final samplesheet contains 260 rows and 13 columns
```

As we noted above, these functions may not work on every dataset. `methylprep` does its best to identify controls based on keywords like 'ctrl' or 'control', but controls may be labelled differently in various datasets.

using `beta_bake` for preprocessed data

`beta_bake` is a function intended for users who may want to build a composite dataset with mixed data formats. Say we ran the `alert` command and found 2 candidate datasets that we are interested in combining into a larger dataset: GSE164149 and GSE158089. The problem? GSE158089 doesn't have IDAT files to download!

Using `beta_bake`, we can download the available data from GEO (whether it is raw IDAT files or has already been processed into beta values).

GSE164149 is a dataset of 16 samples run on EPIC arrays. This data shows the methylome of CRISPR-mediated gene knock-ins on Tregs with high expansion (the researchers chose FOXP3 as the target benchmark because it is a master transcription factor). This dataset includes pre-processed beta values in the GEO data, but `beta_bake` will preferentially pull the raw IDATs (also included) so that we are able to process them with `methylprep`.

The command will look like this:

```
>>> python -m methylprep beta_bake -i GSE164149 -d ~/tutorial/GSE164149
```

The `-d` option specifies which directory to store the downloaded files in.

The output files in this case include:

- `geo_alert GSE164149.csv`: a csv file of the results of running the `methylprep alert` command with the keyword "GSE164149"
- `GSE164149_family.xml`: the MINiML file that contains sample metadata.

- `GSE164149_GPL21145_meta_data.pkl`: a python-optimized .pkl file that holds the same metadata found in the MINiML file.
- `GSE164149_GPL21145_samplesheet.csv`: the sample sheet for this dataset.
- 32 `.idat` files: the raw IDAT data for our 16 samples.
- `GSE164149.zip`: a compressed folder that holds of all of the files above.

After downloading these files, we run `methylprep process` in this folder, and we will have a `beta_values.pkl` file to use for analysis:

```
>>> python -m methylprep process --all -d ~/tutorial/GSE164149
```

The second dataset, GSE158089, is a set of 14 samples that investigates DNA methylomic trajectories of neuronal aging. They sampled induced pluripotent stem cell lines at 4 different points during development and ran those samples on the EPIC array. This dataset only includes the processed beta values, so `beta_bake` will just pull those into a .pkl format for us.

```
>>> python -m methylprep beta_bake -i GSE158089 -d ~/tutorial/GSE158089
```

The output files we have for this are:

- `GSE158089_beta_values.pkl.gz`: beta values stored in a compressed .pkl file (run `gunzip` from the command line to unzip this, or just double click on a Mac).
- `GSE158089_GPL21145_meta_data.pkl.gz`: a similarly formatted file that holds the meta_data found for these samples.

Now that we have two `beta_values.pkl` files for two datasets from the same array types, we can load these into an IDE and easily combine them, the same way you would combine any two pandas dataframes.

```
import pandas as pd
import methylcheck

GSE158089_betas = pd.read_pickle('~/tutorial/GSE158089/GSE158089_beta_values.pkl')

GSE164149_betas = pd.read_pickle('~/tutorial/GSE164149/beta_values.pkl')

betas = pd.concat([GSE158089_betas, GSE164149_betas], axis=1)

betas.head()
```

1.4.4 Using methylprep from the Command Line

The most common use case is processing `.idat` files on a computer within a command line interface. This can also be done in a Jupyter notebook, but large data sets take hours to run and Jupyter will take longer to run these than command line.

Getting Help from Command Line

`methylprep` provides a command line interface (CLI) so the package can be used directly in bash/batchfile or windows/cmd scripts as part of building your custom processing pipeline.

All invocations of the `methylprep` CLI will provide contextual help, supplying the possible arguments and/or options available based on the invoked command. If you specify verbose logging the package will emit log output of DEBUG levels and above.

For more information about `methylprep` as a package:

```
>>> python -m methylprep

usage: methylprep [-h] [-v] {process,sample_sheet} ...

Utility to process methylation data from Illumina IDAT files

positional arguments:
  {process,sample_sheet}
    process             process help
    sample_sheet        sample sheet help

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Enable verbose logging
```

For more information about any of `methylprep`'s functions, simply type the name of the function:

```
>>> python -m methylprep process
[Error]:
the following arguments are required: -d/--data_dir

usage: methylprep process [-h] -d DATA_DIR
                          [--array_type {custom,27k,450k,epic,epic+,mouse}]
                          [-m MANIFEST] [-s SAMPLE_SHEET] [--no_sample_sheet]
                          [-n [SAMPLE_NAME [SAMPLE_NAME ...]]] [-b] [-v]
                          [--batch_size BATCH_SIZE] [-u] [-e] [-x]
                          [-i {float64,float32,float16}] [-c] [--poobah]
                          [--export_poobah] [--minfi] [--no_quality_mask] [-a]

Process Illumina IDAT files, producing NOOB, beta-value, or m_value corrected
scores per probe per sample

optional arguments:
  -h, --help            show this help message and exit
  -d DATA_DIR, --data_dir DATA_DIR
                        Base directory of the sample sheet and associated IDAT
                        files. If IDAT files are in nested directories, this
                        will discover them.
  --array_type {custom,27k,450k,epic,epic+,mouse}
                        Type of array being processed. If omitted, this will
                        autodetect it.
  -m MANIFEST, --manifest MANIFEST
                        File path of the array manifest file. If omitted, this
                        will download the appropriate file from `s3`.
  -s SAMPLE_SHEET, --sample_sheet SAMPLE_SHEET
                        File path of the sample sheet. If omitted, this will
                        discover it. There must be only one CSV file in the
                        data_dir for discovery to work.
  --no_sample_sheet     If your dataset lacks a sample sheet csv file, specify
                        --no_sample_sheet to have it create one on the fly.
                        This will read .idat file names and ensure processing
                        works. If there is a matrix file, it will add in
```

(continues on next page)

(continued from previous page)

```
                      sample names too. If you need to add more meta data
                      into the sample_sheet, look at the create sample_sheet
                      CLI option.
  -n [SAMPLE_NAME [SAMPLE_NAME ...]], --sample_name [SAMPLE_NAME [SAMPLE_NAME ...]]
                      Sample(s) to process. You can pass multiple sample
                      names like this: `python -m methylprep process -d .
                      --all --no_sample_sheet -n Sample_1 Sample_2 Sample_3`
  -b, --betas         If passed, output returns a dataframe of beta values
                      for samples x probes. Local file beta_values.npy is
                      also created.
  -v, --m_value       If passed, output returns a dataframe of M-values for
                      samples x probes. Local file m_values.npy is also
                      created.
  --batch_size BATCH_SIZE
                      If specified, samples will be processed and saved in
                      batches no greater than the specified batch size
  -u, --uncorrected   If specified, processed csv will contain two
                      additional columns (meth and unmeth) that have not
                      been NOOB corrected.
  -e, --no_export     Default is to export data to csv in same folder where
                      IDAT file resides. Pass in --no_export to suppress
                      this.
  -x, --no_meta_export  Default is to convert the sample sheet into a pickled
                      DataFrame, recognized in methylcheck and methylize.
                      Pass in --no_meta_export to suppress this.
  -i {float64,float32,float16}, --bit {float64,float32,float16}
                      Change the processed beta or m_value data_type output
                      from float64 to float16 or float32, to save disk
                      space.
  -c, --save_control  If specified, saves an additional "control_probes.pkl"
                      file that contains Control and SNP-I probe data in the
                      data_dir.
  --poobah            By default, any beta-values or m-values output will
                      contain all probes. If True, those probes that fail
                      the p-value signal:noise detection are replaced with
                      NaNs in dataframes in beta_values and m_value output.
  --export_poobah     If specified, exports a pickled dataframe of the
                      poobah p-values per sample.
  --minfi             If specified, processing uses legacy parameters based
                      on minfi. By default, v1.4.0 and higher mimics sesame
                      output.
  --no_quality_mask   If specified, processing to RETAIN all probes that
                      would otherwise be excluded using the quality_mask
                      sketchy-probe list from sesame. --minfi processing
                      does not use a quality_mask.
  -a, --all           If specified, saves everything: (beta_values.pkl,
                      m_value.pkl, control_probes.pkl, CSVs for each sample,
                      uncluding uncorrected raw values, and meta data, and
                      poobah_values.pkl). And removes failed probes using
                      sesame pOOBah method from these files. This overrides
                      individual CLI settings.
```

process

```
python -m methylprep -v process -d <filepath> --all
```

`-d` (data file path) is the only required option. `-v` (short for `--verbose`) specifies verbose logging. And the `--all` option tells `methylprep process` to save output for ALL of the associated processing steps:

- beta_values.pkl
- poobah_values.pkl
- control_probes.pkl
- m_values.pkl
- noob_meth_values.pkl
- noob_unmeth_values.pkl
- meth_values.pkl
- unmeth_values.pkl
- sample_sheet_meta_data.pkl

By default, the output is usually:

- beta_values.pkl
- noob_meth_values.pkl
- noob_unmeth_values.pkl
- control_probes.pkl

The default settings are designed to match the output of `R`'s `sesame` processing. Prior to `methylprep v1.4.0`, the defaults matched `minfi`'s output.

Here are some high level options:

Argument	Type	Default	Description
`data_dir`	`str`, `Path`	*Required*	Base directory of the sample sheet and associated IDAT files
`minfi`	`bool`	`False`	Changes many settings to match `minfi` output. Default is `sesame`.

Use these options to specify file locations and array type:

Argument	Type	Default	Description
array_type	str	None	Code of the array type being processed. Possible values are custom, 27k, 450k, epic, and epic+. If not provided, the package will attempt to determine the array type based on the number of probes in the raw data. If the batch contains samples from different array types, this may not work. Our data download function attempts to split different arrays into separate batches for processing to accommodate this.
sample_names	str, list	None	List of sample names to process, in the CLI format of -n sample1 sample2 sample3 etc. If provided, only those samples specified will be processed. Otherwise all samples found in the sample sheet will be processed.
manifest_filepath	str, Path	None	File path for the array's manifest file. If not provided, this file will be downloaded from a Life Epigenetics archive.
no_sample_sheet	bool	None	pass in "–no_sample_sheet" from command line to trigger sample sheet auto-generation. Sample names will be based on idat filenames. Useful for public GEO data sets that lack sample sheets.
sample_sheet_filepath	str, Path	None	File path of the project's sample sheet. If not provided, the package will try to find one based on the supplied data directory path.

Use these options to specify what gets saved from processing, and how it gets saved:

Argument	Type	Default	Description
no_export	bool	False	Add to prevent saving the processed samples to CSV files.
no_metadata_export	bool	False	Add to prevent saving the meta data to pickle files.
betas	bool	False	Add flag to output a pickled dataframe of beta values of sample probe values.
m_value	bool	False	Add flag to output a pickled dataframe of m_values of samples probe values.
uncorrected	bool	False	Saves raw florescence intensities in CSV and pickle output.
save_control	bool	False	Add to save control probe data. Required for some methylcheck QC functions.
export_poobah	bool	False	Include probe p-values in output files.
bit	str	float32	Specify data precision, and file size of output files (float16, float32, or float64)
batch_size	int	None	Optional: splits the batch into smaller sized sets for processing. Useful when processing hundreds of samples that can't fit into memory. This approach is also used by the package to process batches that come from different array types.
poobah	bool	True	calculates probe detection p-values and filters failed probes from pickled output files, and includes this data in a column in CSV files.

data_dir is the one required parameter. If you do not provide the file path for the project's sample_sheet CSV, it will find one based on the supplied data directory path. It will also auto detect the array type and download the corresponding manifest file for you.

Other Commands

The methylprep CLI provides these top-level commands, which make it easier to use GEO datasets:

- sample_sheet will find/read/validate/create a sample sheet for a data set, or display its contents (given the directory of the data). This is also part of process and can be applied using the --no_sample_sheet flag.
- download download and process public data sets in NIH GEO or ArrayExpress collections. Provide the public Accession ID and methylprep will handle the rest.

- `beta_bake` combines `download`, `meta_data`, and file format conversion functions to produce a package that can be processed (with `process`) or loaded with `methylcheck.load` for analysis.
- `alert` scan GEO database and construct a CSV / dataframe of sample meta data and phenotypes for all studies matching a keyword
- `composite` download a bunch of datasets from a list of GEO ids, process them all, and combine into a large dataset
- `meta_data` will download just the meta data for a GEO dataset (using the MINiML file from the GEO database) and convert it to a samplesheet CSV

sample_sheet

Find and parse the sample sheet in a given directory and emit the details of each sample. This is not required for actually processing data. `methylprep` will automatically create a sample sheet as part of the `process` or `download` pipelines.

optional arguments:

Argument	Type	Description
-h, –help		show this help message and exit
-d, –data_dir	`str`	Base directory of the sample sheet and associated IDAT files
-c, –create	`bool`	If specified, this creates a sample sheet from idats instead of parsing an existing sample sheet. The output file will be called “samplesheet.csv”.
-o OUTPUT_FILE, –output_file OUTPUT_FILE	`str`	If creating a sample sheet, you can provide an optional output filename (CSV).

download

There are thousands of publically accessible DNA methylation data sets available via the GEO (US NCBI NIH) https://www.ncbi.nlm.nih.gov/geo/ and ArrayExpress (UK) https://www.ebi.ac.uk/arrayexpress/ websites. This function makes it easy to import them and build a reference library of methylation data.

The CLI includes a `download` option. Supply the GEO ID or ArrayExpress ID and it will locate the files, download the idats, process them, and build a dataframe of the associated meta data. This dataframe format is compatible with `methylcheck` and `methylize`.

Argument	Type	Default	Description
-h, –help	•	•	show this help message and exit
-d , –data_dir	`str`	[required path]	path to where the data series will be saved. Folder must exist already.
-i ID, –id ID	`str`	[required ID]	The dataset's reference ID (Starts with `GSE` for GEO or `E-MTAB-` for ArrayExpress)
-l LIST, –list LIST	`multiple strings`	optional	List of series IDs (can be either GEO or ArrayExpress), for partial downloading
-o, –dict_only	`True`	pass flag only	If passed, will only create dictionaries and not process any samples
-b BATCH_SIZE, –batch_size BATCH_SIZE	`int`	optional	Number of samples to process at a time, 100 by default.

When processing large batches of raw `.idat` files, specify `--batch_size` to break the processing up into smaller batches so the computer's memory won't overload. This is off by default when using `process` but is ON when using `download` and set to batch_size of 100. Set to 0 to force processing everything as one batch. The output files will be split into multiple files afterwards, and you can recomine them using `methylcheck.load`.

beta_bake

`beta_bake` is a function intended for combining data that are not processed in exactly the same way. If IDAT files are present, it will download them for you to run `process` on. If there are no idats, but there is uncorrected methylated/unmethylated data, it will download that instead. If there is no unprocessed data, it will parse processed beta values for you.

This is intended for creating datasets that sacrifice some data quality in exchange for size. For example, using a machine learning algorithm on 10,000 noisy samples can yield better results than using that algorithm on a more curated set of 1,000 samples. ML algorithms can be trained to read through the noise and benefit from more data to train on.

Note: less than half of the GEO datasets include raw idat files! Most of the data on GEO has already been processed into beta values. This is a part of why `beta_bake` is so useful.

Argument	Type	Default	Description
-h, –help			show this help message and exit
-i ID, –id ID	str		GEO_ID of the dataset to download
-d DATA_DIR, –data_dir DATA_DIR	str		Folder where series data will appear.
-v, –verbose			if specified, this will turn on more verbose processing messages.
-s, –save_source			if specified, this will retain .idat and/or -tbl-1.txt files used to generate beta_values dataframe pkl files.
-b BUCKET, –bucket BUCKET	str		AWS S3 bucket where downloaded files are stored
-e EFS, –efs EFS	str		AWS elastic file system name, for lambda or AWS batch processing
-p PROCESSED_BUCKET, –processed_bucket PROCESSED_BUCKET	str		AWS S3 bucket where final files are saved
-n, –no_clean			If specified, this LEAVES processing and raw data files in temporary folders. By default, these files are removed during processing, and useful files moved to data_dir.

```
>>> python -m methylprep beta_bake -i GSE74013 -d GSE74013
INFO:methylprep.download.miniml:Downloading GSE74013_family.xml.tgz
INFO:methylprep.download.miniml:MINiML file does not provide (Sentrix_id_R00C00) for␣
↪24/24 samples.
INFO:methylprep.download.miniml:Final samplesheet contains 24 rows and 9 columns
```

Output file containing a beta_values pickled dataframe: GSE74013_beta_values.pkl.gz

Output file containing meta data: GSE74013_GPL13534_meta_data.pkl.gz

composite

A tool to build a data set from a list of public datasets. This function basically just loops `download` through the provided list of datasets.

optional arguments:

Argument	Type	Description
-h, –help		show this help message and exit
-l LIST, –list LIST	`str`, filepath	A text file containg several GEO/ArrayExpress series ids. One ID per line in file. Note: The GEO Accession Viewer lets you export search results in this format.
-d DATA_DIR, –data_dir DATA_DIR	`str`, filepath	Folder where to save data (and read the ID list file).
-c, –control	`bool`	If flagged, this will only save samples that have the word "control" in their meta data.
-k KEYWORD –keyword KEYWORD	`str`	Only retain samples that include this keyword (e.g. blood) somewhere in their meta data.
-e, –export	`bool`	If passed, saves raw processing file data for each sample. (unlike meth-process, this is off by default)
-b, –betas	`bool`	If passed, output returns a dataframe of beta values for samples x probes. Local file beta_values.npy is also created.
-m, –m_value	`bool`	If passed, output returns a dataframe of M-values for samples x probes. Local file m_values.npy is also created.

alert

Function to check for new datasets on GEO and update a csv each time it is run. Usable as a weekly cron command line function. Saves data to a local csv to compare with old datasets in _meta.csv. Saves the dates of each dataset from GEO; calculates any new ones as new rows. updates csv.

optional arguments:

Argument	Type	Description
keyword	`str`	Specify a word or phrase to narrow the search, such as "spleen blood".

meta_data

Provides a more feature-rich meta data parser for public MINiML (formatted) GEO datasets. You can use `meta_data` to identify 'controls' or samples containing a specific keyword (e.g. blood, tumor, etc) and remove any samples from sheet that lack these criteria, and delete the associated idats that don't have these keywords. After, run `process` on the rest, saving time. You can effectively ignore the parts of datasets that you don't need based on the associated meta data.

optional arguments:

Argument	Type	Description
-h, –help		show this help message and exit
-i ID, –id ID	str	Unique ID of the series (the GEO GSExxxx ID)
-d DATA_DIR, –data_dir DATA_DIR	str or path	Directory to search for MINiML file.
-c, –control	str	[experimental]: If flagged, this will look at the sample sheet and only save samples that appear to be"controls".
-k KEYWORD, –keyword KEYWORD	str	[experimental]: Retain samples that include this keyword (e.g. blood, case insensitive) somewhere in samplesheet values.
-s, –sync_idats	bool	[experimental]: If flagged, this will scan the data_dir and remove all idat files that are not in the filtered samplesheet, so they won't be processed.
-o, –dont_download	bool	By default, this will first look at the local filepath (–data-dir) for GSE..._family.xml files. IF this is specified, it wont later look online to download the file. Sometimes a series has multiple files and it is easier to download, extract, and point this parser to each file instead.

1.4.5 Using methylprep from within an IDE

The primary methylprep API provides methods for the most common data processing and file retrieval functionality.

run_pipeline

Run the complete methylation processing pipeline for the given project directory, optionally exporting the results to file.

Returns: A collection of DataContainer objects for each processed sample

```
from methylprep import run_pipeline

data_containers = run_pipeline(data_dir, array_type=None, export=False, manifest_
↪filepath=None, sample_sheet_filepath=None, sample_names=None)
```

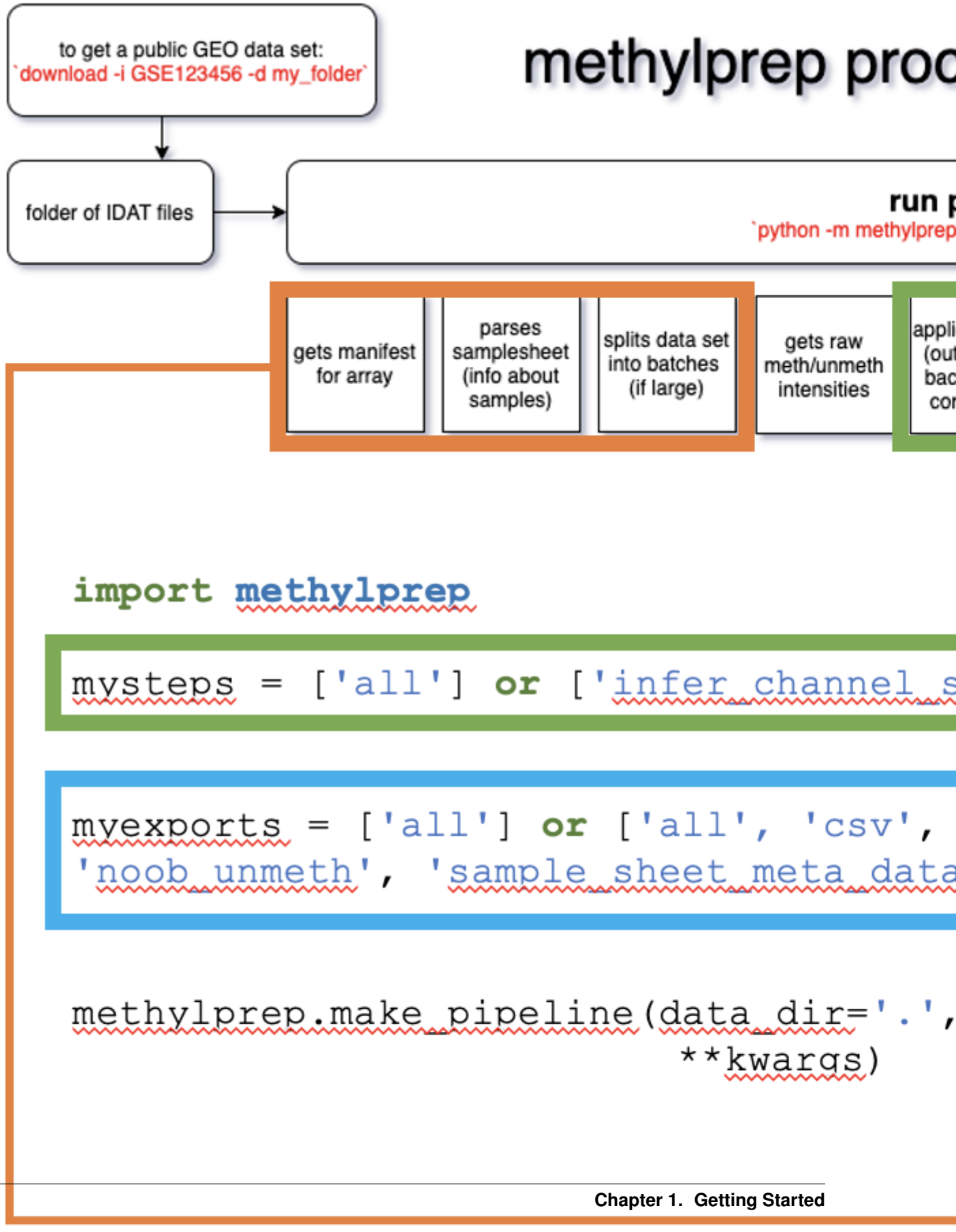
Argument	Type	Default	Description
data_dir	str, Path	•	Base directory of the sample sheet and associated IDAT files
array_type	str	None	Code of the array type being processed. Possible values are custom, 450k, epic, and epic+. If not provided, the pacakage will attempt to determine the array type based on the number of probes in the raw data.
export	bool	False	Whether to export the processed data to CSV
manifest_filepath	str, Path	None	File path for the array's manifest file. If not provided, this file will be downloaded from a Life Epigenetics archive.
sample_sheet_filepath	str, Path	None	File path of the project's sample sheet. If not provided, the package will try to find one based on the supplied data directory path.
sample_names	str collection	None	List of sample names to process. If provided, only those samples specified will be processed. Otherwise all samples found in the sample sheet will be processed.

make_pipeline

This function works much like run_pipeline, but users are offered the flexibility to customize the pipeline to their liking. Any steps that they would like to skip may be excluded from the steps argument. Similarly, users may pick and choose what exports (if any) they would like to save.

Argument	Type	Default	Description
`data_dir`	`str`, `Path`	•	Base directory of the sample sheet and associated IDAT files
`steps`	`str`, `list`	`None`	List of processing steps: [‘all’, ‘infer_channel_switch’, ‘poobah’, ‘quality_mask’, ‘noob’, ‘dye_bias’]
`exports`	`str`, `list`	`None`	file exports to be saved. Anything not specified is not saved. [‘all’, ‘csv’, ‘poobah’, ‘meth’, ‘unmeth’, ‘noob_meth’, ‘noob_unmeth’, ‘sample_sheet_meta_data’, ‘mouse’, ‘control’]
`estimator`	`str`	`beta`	what the pipeline should return [beta, m_value, copy_number, None] (returns data containers instead)

If customizing the data processing steps interests you, you may also want to look at using the SampleDataContainer object, which is the output of processing when run in notebooks and `beta_value` or `m_value` is False. Each SampleDataContainer class object includes all of the `sesame` SigSet data sets and additional information about how the sample was processed.


to get a public GEO data set:
`download -i GSE123456 -d my_folder`
methylprep proc
folder of IDAT files
run
`python -m methylprep
gets manifest for array
parses samplesheet (info about samples)
splits data set into batches (if large)
gets raw meth/unmeth intensities
import methylprep
mysteps = ['all'] or ['infer_channel_s
myexports = ['all'] or ['all', 'csv',
'noob_unmeth', 'sample_sheet_meta_data
methylprep.make_pipeline(data_dir='.',
**kwargs)

get_sample_sheet

Find and parse the sample sheet for the provided project directory path.

Returns: A SampleSheet object containing the parsed sample information from the project's sample sheet file

```
from methylprep import get_sample_sheet

sample_sheet = get_sample_sheet(dir_path, filepath=None)
```

Argument	Type	Default	Description
data_dir	str, Path	•	Base directory of the sample sheet and associated IDAT files
sample_sheet_filepath	str, Path	None	File path of the project's sample sheet. If not provided, the package will try to find one based on the supplied data directory path.

1.4.6 Troubleshooting, Tips, and FAQs

Loading Processed Data

If you have a small data set (under 200 samples), and did not use the `batch_size` option:

```
import pandas as pd
data = pd.read_pickle('{path_to_file}/beta_values.pkl')
```

Otherwise, loading hundreds of samples can be slow. But there's a helper function that loads a bunch of smaller batches into one dataframe, if they were processed with methylprep's `batch_size` option:

```
import methylize
data = methylize.load(path_to_file)
```

That will load 1600 samples in 8 seconds, compared to taking minutes if you `read_pickle` them and `concat` them using pandas.

Processing Notes

1. DataFrames are the format that methylcheck/methylize functions expect. For saving and loading, the probe names belong in columns and the sample names belong in the index. But when processing data, some functions auto-transpose this to decrease processing time.
2. By default, `methylprep process` also creates a file called `sample_sheet_meta_data.pkl` from various data sources
   - if there is a GEO series MINiML xml file, it reads this data preferentially
   - if there is a samplesheet, it will convert this to a meta DataFrame
   - if these sources are missing required data, such as Sentrix_Position and Sentrix_ID, it will look for idat files and read these from the filenames.

idat filenames

Some GEO datasets have unconventionally named IDATs, and cannot be processed without renaming them first.

- There are two acceptable formats:
  - `<GSM_ID>_<Sentrix_ID>_<Sentrix_Position>_<Red|Grn>.idat<.gz>`
  - `<Sentrix_ID>_<Sentrix_Position>_<Red|Grn>.idat<.gz>`
  - Methylprep will convert `.gz` files to `.idat` uncompressed files when processing.
- methylprep does not recognize the older 27k array filename format: `<GSM_ID>_<Sentrix_ID>_<Sentrix_Position>_<SOME_LETTER>.idat`

Why didn't the `methylprep download` function work for GEO dataset GSEnnn?

A significant number of GEO datasets do not store their data in a consistent format. Here are some reasons a GEO dataset fails to download:

1. `idat` filename format is off (missing R00C00 position) The `download` function ONLY supports `.idat` files.
2. no raw idats in zip, only processed data
3. The meta data in MINiML format xml file is incomplete. (There are ways to make it work without meta data)
4. the NIH GEO FTP server is down (yes, we've experienced this whilst testing too)
5. `idat` files in dataset have varying number of probes. If a dataset combines results from two array types (EPIC and 450k), it can sometimes split the data into two sample sheets and you can process each folder separately. `methylprep` usually resolves this for you by moving each batch of files into separate folders by array, so you can process them separately.

In most cases where `download` fails, there will be processed data available in some other format. Use `methylprep beta_bake` to download and convert this data instead.

In other cases where `beta_bake` also fails, we've made an effort to giving you clear error messages, detailing why. IF you detect a pattern that could be parsed with our code, we're happy to support additional file formats for methylation data found on GEO.

What if `methylprep download` and `methylprep beta_bake` fails. How do I use the data anyway?

1. Download the raw idat zipfile manually to a folder.
2. Uncompress it.
3. Confirm that there are `.idat` files present.
4. If the files end in `.idat.gz`, gunzip them first. (In a mac/linux bash window you can navigate to the folder and type `gunzip *` to uncompress all those files. On a PC, use some software like `7zip` or `winzip`.)
5. IF the filenames don't include Sentrix IDs and Sentrix array positions, like `<GSM ID>_<some long number>_<R01C01>_<Red or Grn>.idat`, you'll need to manually edit the samplesheet to match the files.
6. Run `methylprep process` on the folder, possibly with the `--no_sample_sheet` option. It should work, but you won't have any of the sample meta data bundled for you for analysis with datasets in nonstandard formats.

How to process only part of a GEO dataset

The `methylprep meta_data` command line interface (CLI) option allows you to specify which samples to process using keyword matching against the GEO meta data. You can use this before `methylprep download` and `methylprep process` to create smaller data sets, faster.

You can also specify sample names individually from a large set like this: `methylprep process --sample_name Bob -n Suzy -n Doug`. This will reduce processing time.

Examples

(1) Grab a samplesheet for a GEO data set

```
python -m methylprep meta_data -i GSE125105
```

Yields three files on local disk:

- GSE125105_family.xml (the original GEO meta data file in MINiML format)
- GSE125105_GPL13534_samplesheet.csv (used for processing)
- GSE125105_GPL13534_meta_data.pkl (used in analysis to describe samples)

GSM_ID	Sample_Name	Sentrix_ID	Sentrix_Position	source	diagnosis	age	Sex	tissue	cellcount-cd8t	cellcount-cd4t	cellcount-nk	cellcount-bcell	cellcount-mono	cellcount-gran	cellcount-description
GSM3562834	genomic DNA from sample291	3999840035	R01C01	control_whole blood	control	73	F	whole blood	0.07679	0.09099	0.06041	0.08542	0.09072	0.62266	whole blood control sample
GSM3562835	genomic DNA from sample612	3999840036	R02C02	case_whole blood	case	32	M	whole blood	0.05546	0.07946	0.01596	0.09557	0.05519	0.72663	whole blood case sample
GSM3562836	genomic DNA from sample611	3999840026	R01C01	case_whole blood	case	51	F	whole blood	0.08279	0.22216	0.03107	0.07699	0.07919	0.54165	whole blood case sample
GSM3562837	genomic DNA from sample375	3999840026	R02C02	case_whole blood	case	30	M	whole blood	0.03779	0.07368	0.00385	0.07548	0.08910	0.74809	whole blood case sample

(2) Grab just the samplesheet and create a csv and dataframe pkl of it.

```
python -m methylprep -v meta_data -i GSE84727 -d GSE84727
```

Verbose CLI output

```
INFO:methylprep.download.miniml:Downloading GSE84727_family.xml
INFO:methylprep.download.miniml:Downloaded GSE84727_family.xml
INFO:methylprep.download.miniml:Unpacking GSE84727_family.xml
INFO:methylprep.download.miniml:Downloaded and unpacked GSE84727
INFO:methylprep.download.miniml:MINiML file does not provide `methylprep_name`␣
↪(sentrix_id_R00C00) for 847/847 samples.
INFO:methylprep.download.miniml:dropped Sentrix_ID (empty)
INFO:methylprep.download.miniml:dropped Sentrix_Position (empty)
INFO:methylprep.download.miniml:source == description; dropping source
INFO:methylprep.download.miniml:dropped `platform` ({'GPL13534'})
INFO:methylprep.download.miniml:title == Sample_Name; dropping title
INFO:methylprep.download.miniml:Final samplesheet contains 847 rows and 7 columns
INFO:methylprep.download.miniml:['GSM_ID', 'Sample_Name', 'sentrixids', 'Sex', 'age',
↪'disease_status', 'description']
```

Resulting samplesheet

GSM_ID	Sample_Name	sentrixids	Sex	age	disease_status	description
GSM2250273	33262604	3998567027_R01C01	M	47.3	1	control blood
GSM2250274	33261623	3998567027_R02C01	M	60.4	1	control blood
GSM2250275	33262614	3998567027_R04C01	M	30.1	1	control blood

You'll notice that one column `sentrixids` is misnamed. It should be split into `Sentrix_Position` and `Sentrix_ID` columns for processing to work on this GEO series. You can edit the csv and fix that prior to running the pipeline with `methylprep process`. If you don't, you'll get an error saying, "methylprep could not find the samplesheet." This error is caused by the researcher putting an arbitrary name into the GSE84727_family.xml `MINiML` meta data file:

```
      <Characteristics tag="sentrixids">
3998567027_R02C02
      </Characteristics>
```

If you use the `methylprep download` option by itself, it can generally avoid this type of XML parsing error, but it will download everything. Doing analysis on just part of a dataset requires some debugging like this.

(3) Samplesheet with only "normal" samples

```
python -m methylprep -v meta_data -i GSE52270 -d GSE52270 -k normal
```

`-k` is shorthand for `--keyword`. The resulting sample sheet only includes samples that include the keyword `normal`

GSM_ID	Sample_Name	source	disease state	description
GSM1278809	Colon 61	Large Intestine	normal	1011N
GSM1278812	Colon 64	Large Intestine	normal	1082N
GSM1278823	Colon 75	Large Intestine	normal	1184N
GSM1278825	White matter 77	Central Nervous System	normal	12_03

(4) Generate filtered samplesheet with only control samples from blood

```
python -m methylprep  meta_data -i GSE125105 -d GSE125105 --control -k blood
```

GSM_ID	Sample_Name	Sentrix_ID	Sentrix_Position	source	diagnosis	age	Sex	tissue	cellcount-cd8t	cellcount-cd4t	cellcount-nk	cellcount-bcell	cellcount-mono	cellcount-gran	description
GSM3562834	genomic DNA from sample291	3999840036	R01C01	control_whole blood	control	73	F	whole blood	0.07679	0.09099	0.06041	0.08542	0.09072	0.62266	whole blood control sample
GSM3562839	genomic DNA from sample176	3999840036	R02C02	control_whole blood	control	43	M	whole blood	0.06946	0.12989	0.04703	0.09808	0.14105	0.54663	whole blood control sample
GSM3562842	genomic DNA from sample161	3999840036	R03C01	control_whole blood	control	44	F	whole blood	0.10986	0.13566	0.07657	0.09126	0.12520	0.49317	whole blood control sample
GSM3562846	genomic DNA from sample270	3999840037	R01C01	control_whole blood	control	64	M	whole blood	0.11508	0.14116	0.06790	0.09416	0.15310	0.47829	whole blood control sample
GSM3562855	genomic DNA from sample162	3999840037	R06C01	control_whole blood	control	65	M	whole blood	0.01668	0.14318	0.10960	0.05545	0.09695	0.60283	whole blood control sample

This only retains 211 of the 699 samples in a samplesheet. Next, you download the `.idat` files with with `methylprep download` and then remove the `idat` files you won't need like this:

```
python -m methylprep -v download -i GSE125105 -d GSE125105
python -m methylprep -v meta_data -i GSE125105 -d GSE125105 --sync_idats --control -k␣
↪blood
```

And then process the 6.1GB file using this samplesheet, like this:

```
python -m methylprep -v process -d GSE125105 --betas --m_value -e
```

Those options will create two big files. One is a dataframe of beta_values for each sample. The other, m_values for each sample (kind of the same thing, but sometimes you want m_values). the `-e` or `--no_export` option will suppress the function creating files of probe values for each sample, as these are not needed by most `methylize`

and `methylcheck` functions. There is also a `--save_uncorrected` option that prevents any sort of background and NOOB signal enhancement during processing. Uncorrected files are needed for a few analysis functions, namely `p-value probe detection`.

In general, partial-dataset processing fails because the meta data for a GEO dataset is incomplete. Either the array positions are missing, or misnamed. Careful checking can allow one to fix this and build a large data set from multiple GEO datasets.

(4b) Another condensed example of downloading GEO data and only processing control samples

```
python -m methylprep -v download -i GSE130030 -d GSE130030
# next, remove the treatment samples using `-c` and remove extra idats with `-s`
python -m methylprep -v meta_data -i GSE130030 -d GSE130030 --control -s
# finally, process it
python -m methylprep -v process -d GSE130030 --betas --m_value --no_export
```

This creates two files, `beta_values.pkl` and `GSE130030_GPL13534_meta_data.pkl`, that you can work with in `methylize` like this:

Navigate to the `GSE130030` folder created by `methylrep`, and start a python interpreter:

```
import methylize
data,meta = methylize.load_both()
INFO:methylize.helpers:Found several meta_data files; using: GSE130030_GPL13534_meta_
↪data.pkl)
INFO:methylize.helpers:loaded data (485512, 14) from 1 pickled files (0.159s)
INFO:methylize.helpers:meta.Sample_IDs match data.index (OK)
```

Or if you are running in a notebook, specify the full path:

```
import methylize
data,meta = methylize.load_both('<path_to...>/GSE105018')
```

Why won't `methylprep composite` parse some GEO data sets' meta data?

Here are some examples of logical, but unexpected ways data can be stored in the MINiML file format:

The composite expects "control" to be in one of the rows in a spreadsheet. Instead, the authors have recoded "control" as a number, "1" in the column header name. Our parser just isn't smart enough to read that.

```
['GSM_ID', 'Sample_Name', 'diseasestatus (1=control, 2=scz patient)', 'source',
↪'gender', 'sample type', 'plate', 'sentrix barcode', 'sentrix position', 'well id',
↪'age',  'used_in_analysis', 'description']
```

Here, instead of having the same names for data for each sample, they have split the smoking status into a bunch of columns, and not provided values for every sample. (smoking_evernever and smoke_free_years don't add up to 95.) Fixing this requires putting in null values for each incomplete column in a sample sheet.

```
ValueError - array lengths vary in sample meta data: [('GSM_ID', 95), ('Sample_Name',␣
↪95),
('smoking_evernever', 52), ('smoke_free_years', 30), ('Sentrix_ID', 95), ('Sentrix_
↪Position', 95), ('source', 95), ('gender', 95), ('slide', 95), ('array', 95), (
↪'array_name', 95), ('sample_group', 52),  ('smoking_5years', 52), ('ms_case_control
↪', 52), ('sample_year', 52), ('age_sampling', 52), ('py', 52),  ('description', 95),
↪ ('Sample_ID', 95)]
```

1.4.7 API Reference

methylprep.processing	
methylprep.run_pipeline(data_dir[, ...])	The main CLI processing pipeline.
methylprep.files.create_sample_sheet(dir_path)	Creates a samplesheet.csv file from the .IDAT files of a GEO series directory
methylprep.download	
methylprep.run_series(id, path[, dict_only, ...])	Downloads the IDATs and metadata for a series then generates one metadata dictionary and one beta value matrix for each platform in the series
methylprep.read_geo	
methylprep.build_composite_dataset(...[, ...])	A wrapper function for convert_miniml() to download a list of GEO datasets and process only those samples that meet criteria.
methylprep.models	
methylprep.files	

class methylprep.**ArrayType**

Bases: enum.Enum

This class stores meta data about array types, such as numbers of probes of each type, and how to guess the array from probes in idat files.

CUSTOM = 'custom'

ILLUMINA_27K = '27k'

ILLUMINA_450K = '450k'

ILLUMINA_EPIC = 'epic'

ILLUMINA_EPIC_PLUS = 'epic+'

ILLUMINA_MOUSE = 'mouse'

from_probe_count = <bound method ArrayType.from_probe_count of <enum 'ArrayType'>>

num_controls

num_probes

Used to load normal cg+ch probes from start of manifest until this point. Then start control df.

num_snps

class methylprep.**Manifest**(*array_type*, *filepath_or_buffer=None*, *on_lambda=False*, *verbose=True*)

Bases: object

Provides an object interface to an Illumina array manifest file.

Arguments: array_type {ArrayType} – The type of array to process. values are styled like ArrayType.ILLUMINA_27K, ArrayType.ILLUMINA_EPIC or ArrayType('epic'), ArrayType('mouse')

Keyword Arguments: filepath_or_buffer {file-like} – a pre-existing manifest filepath (default: {None})

Raises: ValueError: The sample sheet is not formatted properly or a sample cannot be found.

columns

control_data_frame

data_frame

static download_default(*array_type*, *on_lambda=False*)
Downloads the appropriate manifest file if one does not already exist.

Arguments: array_type {ArrayType} – The type of array to process.

Returns: [PurePath] – Path to the manifest file.

get_data_types()

get_probe_details(*probe_type*, *channel=None*)
used by infer_channel_switch. Given a probe type (I, II, SnpI, SnpII, Control) and a channel (Channel.RED | Channel.GREEN), this will return info needed to map probes to their names (e.g. cg0031313 or rs00542420), which are NOT in the idat files.

mouse_data_frame

read_control_probes(*manifest_file*)
Unlike other probes, control probes have no IlmnID because they're not locus-specific. they also use arbitrary columns, ignoring the header at start of manifest file.

read_mouse_probes(*manifest_file*)
ILLUMINA_MOUSE contains unique probes whose names begin with 'mu' and 'rp' for 'murine' and 'repeat', respectively. This creates a dataframe of these probes, which are not processed like normal cg/ch probes.

read_probes(*manifest_file*)

read_snp_probes(*manifest_file*)
Unlike cpg and control probes, these rs probes are NOT sequential in all arrays.

static seek_to_start(*manifest_file*)
find the start of the data part of the manifest. first left-most column must be "IlmnID" to be found.

snp_data_frame

methylprep.**get_sample_sheet**(*dir_path*, *filepath=None*)
Generates a SampleSheet instance for a given directory of processed data.

Arguments: dir_path {string or path-like} – Base directory of the sample sheet and associated IDAT files.

Keyword Arguments:

filepath {string or path-like} – path of the sample sheet file if provided, otherwise one will try to be found. (default: {None})

Returns: [SampleSheet] – A SampleSheet instance.

methylprep.**parse_sample_sheet_into_idat_datasets**(*sample_sheet*, *sample_name=None*, *from_s3=None*, *meta_only=False*)
Generates a collection of IdatDatasets from samples in a sample sheet.

Arguments: sample_sheet {SampleSheet} – The SampleSheet from which the data originates.

Keyword Arguments: sample_name {string} – Optional: one sample to process from the sample_sheet. (default: {None}) from_s3 {zip_reader} – pass in a S3ZipReader object to extract idat files from a zipfile hosted on s3. meta_only {True/False} – doesn't read idat files, only parses the meta data about them. (RawMetaDataset is same as RawDataset but has no idat probe values stored in object, because not needed in pipeline)

Raises: ValueError: If the number of probes between raw datasets differ.

Returns: [RawDatasets] – A list of idat data pairs, each a dict like {'green_idat': green_idat, 'red_idat': red_idat}

methylprep.**consolidate_values_for_sheet**(*data_containers*, *postprocess_func_colname='beta_value'*, *bit='float32'*, *poobah=True*, *poobah_sig=0.05*, *exclude_rs=True*)

Transforms results into a single dataframe with all of the function values, with probe names in rows, and sample beta values for probes in columns.

Input: data_containers – the output of run_pipeline() is this, a list of data_containers. (a list of processed SampleDataContainer objects)

Arguments for postprocess_func_colname: calculate_beta_value –> 'beta_value' calculate_m_value –> 'm_value' calculate_copy_number –> 'cm_value'

note: these functions are hard-coded in pipeline.py as part of process_all() step. note: if run_pipeline included 'sesame' option, then quality mask is automatically applied to all pickle outputs, and saved as column in processed CSV.

Options:

bit (float16, float32, float64) – change the default data type from float32 to another type to save disk space. float16 works fine, but might not be compatible with all numnpy/pandas functions, or with outside packages, so float32 is default. This is specified from methylprep process command line.

poobah If true, filters by the poobah_pval column. (beta m_val pass True in for this.)

data_container.quality_mask (True/False) If 'quality_mask' is present in df, True filters these probes from pickle output.

exclude_rs as of v1.5.0 SigSet keeps snp ('rs') probes with other probe types (if qualityMask is false); need to separate them here before exporting to file.

methylprep.**run_series**(*id*, *path*, *dict_only=False*, *batch_size=100*, *clean=True*, *abort_if_no_idats=True*, *decompress=True*)

Downloads the IDATs and metadata for a series then generates one metadata dictionary and one beta value matrix for each platform in the series

Arguments:

id [required] the series ID (can be a GEO or ArrayExpress ID)

path [required] the path to the directory to download the data to. It is assumed a dictionaries and beta values directory has been created for each platform (and will create one for each if not)

dict_only if True, downloads idat files and meta data and creates data dictionaries for each platform, but does not process them further.

batch_size the batch_size to use when processing samples (number of samples run at a time). By default is set to the constant 100.

clean if True, removes intermediate processing files

methylprep.**run_series_list**(*list_file*, *path*, *dict_only=False*, *batch_size=100*, ***kwargs*)

Downloads the IDATs and metadata for a list of series, creating metadata dictionaries and dataframes of sample beta_values

Arguments:

list_file [required] the name of the file containing a list of GEO_IDS and/or Array Express IDs to download and process. This file must be located in the directory data is downloaded to. Each line of the file should contain the name of one data series ID.

path [required] the path to the directory to download the data to. It is assumed a dictionaries and beta values directory has been created for each platform (and will create one for each if not)

dict_only if True, only downloads data and creates dictionaries for each platform

batch_size the batch_size to use when processing samples (number of samples run at a time). By default is set to the constant 100.

methylprep.**convert_miniml**(*geo_id*, *data_dir='.'*, *merge=True*, *download_it=True*, *extract_controls=False*, *require_keyword=None*, *sync_idats=False*, *remove_tgz=False*, *verbose=False*)

This scans the datadir for an xml file with the geo_id in it. Then it parses it and saves the useful stuff to a dataframe called "sample_sheet_meta_data.pkl". DOES NOT REQUIRE idats.

CLI version: python -m meta_data -i GSExxxxx -d <my_folder>

Arguments:

merge: If merge==True and there is a file with 'samplesheet' in the folder, and that sheet has GSM_IDs, merge that data into this samplesheet. Useful for when you have idats and want one combined samplesheet for the dataset.

download_it: if miniml file not in data_dir path, it will download it from web.

extract_controls [experimental]: if you only want to retain samples from the whole set that have certain keywords, such as "control" or "blood", this experimental flag will rewrite the samplesheet with only the parts you want, then feed that into run_pipeline with named samples.

require_keyword [experimental]: another way to eliminate samples from samplesheets, before passing into the processor. if specified, this keyword needs to appear somewhere in the values of a samplesheet.

sync_idats: If flagged, this will search *data_dir* for idats and remove any of those that are not found in the filtered samplesheet. Requires you to run the download function first to get all idats, before you run this *meta_data* function.

Attempts to also read idat filenames, if they exist, but won't fail if they don't.

methylprep.**build_composite_dataset**(*geo_id_list*, *data_dir*, *merge=True*, *download_it=True*, *extract_controls=False*, *require_keyword=None*, *sync_idats=True*, *betas=False*, *m_value=False*, *export=False*)

A wrapper function for convert_miniml() to download a list of GEO datasets and process only those samples that meet criteria. Specifically - grab the "control" or "normal" samples from a bunch of experiments for one tissue type (e.g. "blood"), process them, and put all the resulting beta_values and/or m_values pkl files in one place, so that you can run *methylize.load_both()* to create a combined reference dataset for QC, analysis, or meta-analysis.

Arguments:

geo_id_list (required): A list of GEO "GSEnnn" ids. From command line, pass these in as separate values

data_dir: folder to save data

merge (True): If merge==True and there is a file with 'samplesheet' in the folder, and that sheet has GSM_IDs, merge that data into this samplesheet. Useful for when you have idats and want one combined samplesheet for the dataset.

download_it (True): if miniml file not in data_dir path, it will download it from web.

extract_controls (True)): if you only want to retain samples from the whole set that have certain keywords, such as "control" or "blood", this experimental flag will rewrite the samplesheet with only the parts you want, then feed that into run_pipeline with named samples.

require_keyword (None): another way to eliminate samples from samplesheets, before passing into the processor. if specified, the "keyword" string passed in must appear somewhere in the values of a samplesheet for sample to be downloaded, processed, retained.

sync_idats: If flagged, this will search *data_dir* for idats and remove any of those that are not found in the filtered samplesheet. Requires you to run the download function first to get all idats, before you run this *meta_data* function.

betas: process beta_values

m_value: process m_values

- Attempts to also read idat filenames, if they exist, but won't fail if they don't.
- removes unneeded files as it goes, but leaves the xml MINiML file and folder there as a marker if a geo dataset fails to download. So it won't try again on resume.

methylprep.**run_pipeline**(*data_dir*, *array_type=None*, *export=False*, *manifest_filepath=None*, *sample_sheet_filepath=None*, *sample_name=None*, *betas=False*, *m_value=False*, *make_sample_sheet=False*, *batch_size=None*, *save_uncorrected=False*, *save_control=True*, *meta_data_frame=True*, *bit='float32'*, *poobah=False*, *export_poobah=False*, *poobah_decimals=3*, *poobah_sig=0.05*, *low_memory=True*, *sesame=True*, *quality_mask=None*, ***kwargs*)

The main CLI processing pipeline. This does every processing step and returns a data set.

Required Arguments:

data_dir [required] path where idat files can be found, and a samplesheet csv.

Optional file and sub-sampling inputs:

manifest_filepath [optional] if you want to provide a custom manifest, provide the path. Otherwise, it will download the appropriate one for you.

sample_sheet_filepath [optional] it will autodetect if ommitted.

make_sample_sheet [optional] if True, generates a sample sheet from idat files called 'samplesheet.csv', so that processing will work. From CLI pass in "–no_sample_sheet" to trigger sample sheet auto-generation.

sample_name [optional, list] if you don't want to process all samples, you can specify individual samples as a list. if sample_names are specified, this will not also do batch sizes (large batches must process all samples)

Optional processing arguments:

sesame [default: True] If True, applies offsets, poobah, noob, infer_channel_switch, nonlinear-dye-bias-correction, and qualityMask to imitate the output of openSesame function. If False, outputs will closely match minfi's processing output. Prior to version 1.4.0, file processing matched minfi.

array_type [default: autodetect] 27k, 450k, EPIC, EPIC+ If omitted, this will autodetect it.

batch_size [optional] if set to any integer, samples will be processed and saved in batches no greater than the specified batch size. This will yield multiple output files in the format of "beta_values_1.pkl ... beta_values_N.pkl".

bit [default: float32] You can change the processed output files to one of: {float16, float32, float64}. This will make files & memory usage smaller, often with no loss in precision. However, using float16 masy cause an overflow error, resulting in "inf" appearing instead of numbers, and numpy/pandas functions do not universally support float16.

low_memory [default: True] If False, pipeline will not remove intermediate objects and data sets during processing. This provides access to probe subsets, foreground, and background probe sets in the SampleDataContainer object returned when this is run in a notebook (not CLI).

quality_mask [default: None] If False, process will NOT remove sesame's list of unreliable probes. If True, removes probes. The default None will defer to sesamee, which defaults to true. But if explicitly set, it will override sesame setting.

Optional export files:

meta_data_frame [default: True] if True, saves a file, "sample_sheet_meta_data.pkl" with samplesheet info.

export [default: False] if True, exports a CSV of the processed data for each idat file in sample.

save_uncorrected [default: False] if True, adds two additional columns to the processed.csv per sample (meth and unmeth), representing the raw fluorescence intensities for all probes. It does not apply NOOB correction to values in these columns.

save_control [default: False] if True, adds all Control and SnpI type probe values to a separate pickled dataframe, with probes in rows and sample_name in the first column. These non-CpG probe names are excluded from processed data and must be stored separately.

poobah [default: False] If specified as True, the pipeline will run Sesame's p-value probe detection method (poobah) on samples to remove probes that fail the signal/noise ratio on their fluorescence channels. These will appear as NaNs in the resulting dataframes (beta_values.pkl or m_values.pkl). All probes, regardless of p-value cutoff, will be retained in CSVs, but there will be a 'poobah_pval' column in CSV files that methylcheck.load uses to exclude failed probes upon import at a later step.

poobah_sig [default: 0.05] the p-value level of significance, above which, will exclude probes from output (typical range of 0.001 to 0.1)

poobah_decimals [default: 3] The number of decimal places to round p-value column in the processed CSV output files.

mouse probes Mouse-specific will be saved if processing a mouse array.

Optional final estimators:

betas if True, saves a pickle (beta_values.pkl) of beta values for all samples

m_value if True, saves a pickle (m_values.pkl) of beta values for all samples

Note on meth/unmeth: if either betas or m_value is True, this will also save two additional files: 'meth_values.pkl' and 'unmeth_values.pkl' with the same dataframe structure, representing raw, uncorrected meth probe intensities for all samples. These are useful in some methylcheck functions and load/produce results 100X faster than loading from processed CSV output.

Returns: By default, if called as a function, a list of SampleDataContainer objects is returned, with the following execptions:

betas if True, will return a single data frame of betavalues instead of a list of SampleDataContainer objects. Format is a "wide matrix": columns contain probes and rows contain samples.

m_value if True, will return a single data frame of m_factor values instead of a list of SampleDataContainer objects. Format is a "wide matrix": columns contain probes and rows contain samples.

if batch_size is set to more than ~600 samples, nothing is returned but all the files are saved. You can recreate/merge output files by loading the files using methylcheck.load().

Processing notes: The sample_sheet parser will ensure every sample has a unique name and assign one (e.g. Sample1) if missing, or append a number (e.g. _1) if not unique. This may cause sample_sheets and processed data in dataframes to not match up. Will fix in future version.

pipeline steps: 1 make sample sheet or read sample sheet into a list of samples' data 2 split large projects into batches, if necessary, and ensure unique sample names 3 read idats 4 select and read manifest 5 put everything into SampleDataContainer class objects 6 process everything, using the pipeline steps specified

idats -> channel_swaps -> poobah -> quality_mask -> noob -> dye_bias

7 apply the final estimator function (beta, m_value, or copy number) to all data 8 export all the data into multiple files, as defined by pipeline

methylprep.**make_pipeline**(*data_dir='.'*, *steps=None*, *exports=None*, *estimator='beta'*, ***kwargs*)

Specify a list of processing steps for run_pipeline, then instantiate and run that pipeline.

steps: list of processing steps ['all', 'infer_channel_switch', 'poobah', 'quality_mask', 'noob', 'dye_bias']

exports: list of files to be saved; anything not specified is not saved; ['all'] saves everything. ['all', 'csv', 'poobah', 'meth', 'unmeth', 'noob_meth', 'noob_unmeth', 'sample_sheet_meta_data', 'mouse', 'control']

estimator: which final format? [beta | m_value | copy_number | None (returns containers instead)]

This feeds a Class that runs the run_pipeline function of transforms with a final estimator. It replaces all of the kwargs that are in run_pipeline() and adds a few more options:

[steps] – you can set all of these with ['all'] or any combination of these in a list of steps: Also note that adding "sesame=True" to kwargs will enable: infer_channel_switch, poobah, quality_mask, noob, dye_bias 'infer_channel_switch' 'poobah' 'quality_mask' 'noob' 'dye_bias' – specifying this select's sesame's nonlinear-dye-bias correction. Omitting causes NOOB to use minfi's linear-dye-correction, unless NOOB is missing.

[exports] export=False, make_sample_sheet=False, export_poobah=False, save_uncorrected=False, save_control=False, meta_data_frame=True,

[final estimator] – default: return list of sample data containers. betas=False, m_value=False, -copy_number- You may override that by specifying 'estimator'= ('betas' or 'm_value').

[how it works] make_pipeline calls run_pipeline(), which has a **kwargs final keyword that maps many additional esoteric settings that you can define here.

These are used for more granular unit testing on methylsuite, but could allow you to change how data is processed in very fine-tuned ways.

The rest of these are additional optional kwargs you can include:

[inputs] – omitting these kwargs will assume the defaults, as shown below data_dir, array_type=None, manifest_filepath=None, sample_sheet_filepath=None, sample_name=None,

[processing] – omitting these kwargs will assume the defaults, as shown below batch_size=None, — if you have low RAM memory or >500 samples, you might need to process the batch in chunks. bit='float32', — float16 or float64 also supported for higher/lower memory/disk usage low_memory=True, — If True, processing deletes intermediate objects. But you can save them in the SampleDataContainer by setting this to False. poobah_decimals=3 — in csv file output poobah_sig=0.05

[logging] – how much information do you want on the screen? Default is minimal information. verbose=False (True for more) debug=False (True for a LOT more info)

processing

class methylprep.processing.**SampleDataContainer**(*idat_dataset_pair*, *manifest=None*, *retain_uncorrected_probe_intensities=False*, *bit='float32'*, *pval=False*, *poobah_decimals=3*, *poobah_sig=0.05*, *do_noob=True*, *quality_mask=True*, *switch_probes=True*, *do_nonlinear_dye_bias=True*, *debug=False*, *sesame=True*)

Wrapper that provides easy access to red+green idat datasets, the sample, manifest, and processing params.

Arguments: raw_dataset {RawDataset} – A sample's RawDataset for a single well on the processed array. manifest {Manifest} – The Manifest for the correlated RawDataset's array type. bit (default: float32) – option to store data as float16 or float32 to save space. pval (default: False) – whether to apply p-value-detection algorithm to remove

unreliable probes (based on signal/noise ratio of fluoresence) uses the sesame method (pOOBah) based on out of band background levels

Jan 2020: added .snp_(un)methylated property. used in postprocess.consolidate_crontrol_snp() Mar 2020: added p-value detection option Mar 2020: added mouse probe post-processing separation June 2020: major refactor to use SigSet, like sesame. Removed raw_dataset and methylationDataset. - SigSet is now a Superclass of SampleDataContainer.

export(*output_path*)

Saves a CSV for each sample with all processing intermediate data

process_all()

Runs all pre and post-processing calculations for the dataset. Combines the SigSet methylated and unmethylated parts of SampleDataContainer, and modifies them, whilst creating self.__data_frame with noob/dye processed data.

Order:

- poobah
- quality_mask
- noob (background correction)
- build data_frame
- nonlinear dye-bias correction
- reduce memory/bit-depth of data
- copy over uncorrected values
- split out mouse probes

process_beta_value(*input_dataframe*, *quality_mask_probes=None*)

Calculate Beta value from methylation data

process_copy_number(*input_dataframe*)

Calculate copy number value from methylation data

process_m_value(*input_dataframe*)

Calculate M value from methylation data

methylprep.processing.**preprocess_noob**(*container*, *offset=15*, *pval_probes_df=None*, *quality_mask_df=None*, *nonlinear_dye_correction=True*, *debug=False*, *unit_test_oob=False*)

NOOB pythonized copy of https://github.com/zwdzwd/sesame/blob/master/R/background_correction.R - The function takes a SigSet and returns a modified SigSet with the background subtracted. - Background is modelled in a normal distribution and true signal in an exponential distribution. - The Norm-Exp deconvolution is parameterized using Out-Of-Band (oob) probes. - includes snps, but not control probes yet - output should replace the container instead of returning debug dataframes - II RED and II GREEN both have data, but manifest doesn't have a way to track this, so function tracks it. - keep IlmnID as index for meth/unmeth snps, and convert fg_green

if nonlinear_dye_correction=True, this uses a sesame method in place of minfi method, in a later step. if unit_test_oob==True, returns the intermediate data instead of updating the SigSet/SampleDataContainer.

methylprep.processing.**run_pipeline**(*data_dir*, *array_type=None*, *export=False*, *manifest_filepath=None*, *sample_sheet_filepath=None*, *sample_name=None*, *betas=False*, *m_value=False*, *make_sample_sheet=False*, *batch_size=None*, *save_uncorrected=False*, *save_control=True*, *meta_data_frame=True*, *bit='float32'*, *poobah=False*, *export_poobah=False*, *poobah_decimals=3*, *poobah_sig=0.05*, *low_memory=True*, *sesame=True*, *quality_mask=None*, ***kwargs*)

The main CLI processing pipeline. This does every processing step and returns a data set.

Required Arguments:

data_dir [required] path where idat files can be found, and a samplesheet csv.

Optional file and sub-sampling inputs:

manifest_filepath [optional] if you want to provide a custom manifest, provide the path. Otherwise, it will download the appropriate one for you.

sample_sheet_filepath [optional] it will autodetect if ommitted.

make_sample_sheet [optional] if True, generates a sample sheet from idat files called 'samplesheet.csv', so that processing will work. From CLI pass in "–no_sample_sheet" to trigger sample sheet autogeneration.

sample_name [optional, list] if you don't want to process all samples, you can specify individual samples as a list. if sample_names are specified, this will not also do batch sizes (large batches must process all samples)

Optional processing arguments:

sesame [default: True] If True, applies offsets, poobah, noob, infer_channel_switch, nonlinear-dye-bias-correction, and qualityMask to imitate the output of openSesame function. If False, outputs will closely match minfi's processing output. Prior to version 1.4.0, file processing matched minfi.

array_type [default: autodetect] 27k, 450k, EPIC, EPIC+ If omitted, this will autodetect it.

batch_size [optional] if set to any integer, samples will be processed and saved in batches no greater than the specified batch size. This will yield multiple output files in the format of "beta_values_1.pkl ... beta_values_N.pkl".

bit [default: float32] You can change the processed output files to one of: {float16, float32, float64}. This will make files & memory usage smaller, often with no loss in precision. However, using float16 masy cause an overflow error, resulting in "inf" appearing instead of numbers, and numpy/pandas functions do not universally support float16.

low_memory [default: True] If False, pipeline will not remove intermediate objects and data sets during processing. This provides access to probe subsets, foreground, and background probe sets in the SampleDataContainer object returned when this is run in a notebook (not CLI).

quality_mask [default: None] If False, process will NOT remove sesame's list of unreliable probes. If True, removes probes. The default None will defer to sesamee, which defaults to true. But if explicitly set, it will override sesame setting.

Optional export files:

meta_data_frame [default: True] if True, saves a file, "sample_sheet_meta_data.pkl" with samplesheet info.

export [default: False] if True, exports a CSV of the processed data for each idat file in sample.

save_uncorrected [default: False] if True, adds two additional columns to the processed.csv per sample (meth and unmeth), representing the raw fluorescence intensities for all probes. It does not apply NOOB correction to values in these columns.

save_control [default: False] if True, adds all Control and SnpI type probe values to a separate pickled dataframe, with probes in rows and sample_name in the first column. These non-CpG probe names are excluded from processed data and must be stored separately.

poobah [default: False] If specified as True, the pipeline will run Sesame's p-value probe detection method (poobah) on samples to remove probes that fail the signal/noise ratio on their fluorescence channels. These will appear as NaNs in the resulting dataframes (beta_values.pkl or m_values.pkl). All probes, regardless of p-value cutoff, will be retained in CSVs, but there will be a 'poobah_pval' column in CSV files that methylcheck.load uses to exclude failed probes upon import at a later step.

poobah_sig [default: 0.05] the p-value level of significance, above which, will exclude probes from output (typical range of 0.001 to 0.1)

poobah_decimals [default: 3] The number of decimal places to round p-value column in the processed CSV output files.

mouse probes Mouse-specific will be saved if processing a mouse array.

Optional final estimators:

betas if True, saves a pickle (beta_values.pkl) of beta values for all samples

m_value if True, saves a pickle (m_values.pkl) of beta values for all samples

Note on meth/unmeth: if either betas or m_value is True, this will also save two additional files: 'meth_values.pkl' and 'unmeth_values.pkl' with the same dataframe structure, representing raw, uncorrected meth probe intensities for all samples. These are useful in some methylcheck functions and load/produce results 100X faster than loading from processed CSV output.

Returns: By default, if called as a function, a list of SampleDataContainer objects is returned, with the following execptions:

betas if True, will return a single data frame of betavalues instead of a list of SampleDataContainer objects. Format is a "wide matrix": columns contain probes and rows contain samples.

m_value if True, will return a single data frame of m_factor values instead of a list of SampleDataContainer objects. Format is a "wide matrix": columns contain probes and rows contain samples.

if batch_size is set to more than ~600 samples, nothing is returned but all the files are saved. You can recreate/merge output files by loading the files using methylcheck.load().

Processing notes: The sample_sheet parser will ensure every sample has a unique name and assign one (e.g. Sample1) if missing, or append a number (e.g. _1) if not unique. This may cause sample_sheets and processed data in dataframes to not match up. Will fix in future version.

pipeline steps: 1 make sample sheet or read sample sheet into a list of samples' data 2 split large projects into batches, if necessary, and ensure unique sample names 3 read idats 4 select and read manifest 5 put everything into SampleDataContainer class objects 6 process everything, using the pipeline steps specified

idats -> channel_swaps -> poobah -> quality_mask -> noob -> dye_bias

7 apply the final estimator function (beta, m_value, or copy number) to all data 8 export all the data into multiple files, as defined by pipeline

methylprep.processing.**consolidate_values_for_sheet**(*data_containers*, *postprocess_func_colname='beta_value'*, *bit='float32'*, *poobah=True*, *poobah_sig=0.05*, *exclude_rs=True*)

Transforms results into a single dataframe with all of the function values, with probe names in rows, and sample beta values for probes in columns.

Input: data_containers – the output of run_pipeline() is this, a list of data_containers. (a list of processed SampleDataContainer objects)

Arguments for postprocess_func_colname: calculate_beta_value –> 'beta_value' calculate_m_value –> 'm_value' calculate_copy_number –> 'cm_value'

note: these functions are hard-coded in pipeline.py as part of process_all() step. note: if run_pipeline included 'sesame' option, then quality mask is automatically applied to all pickle outputs, and saved as column in processed CSV.

Options:

bit (float16, float32, float64) – change the default data type from float32 to another type to save disk space. float16 works fine, but might not be compatible with all numnpy/pandas functions, or with outside packages, so float32 is default. This is specified from methylprep process command line.

poobah If true, filters by the poobah_pval column. (beta m_val pass True in for this.)

data_container.quality_mask (True/False) If 'quality_mask' is present in df, True filters these probes from pickle output.

exclude_rs as of v1.5.0 SigSet keeps snp ('rs') probes with other probe types (if qualityMask is false); need to separate them here before exporting to file.

models

class methylprep.models.**ArrayType**

This class stores meta data about array types, such as numbers of probes of each type, and how to guess the array from probes in idat files.

num_probes

Used to load normal cg+ch probes from start of manifest until this point. Then start control df.

class methylprep.models.**Channel**

idat probes measure either a red or green fluorescence. This specifies which to return within idat.py: red_idat or green_idat.

class methylprep.models.**ControlProbe**(*address*, *control_type*, *color*, *extended_type*)

NOT USED ANYWHERE

class methylprep.models.**ControlType**

An enumeration.

methylprep.models.**parse_sample_sheet_into_idat_datasets**(*sample_sheet*, *sample_name=None*, *from_s3=None*, *meta_only=False*)

Generates a collection of IdatDatasets from samples in a sample sheet.

Arguments: sample_sheet {SampleSheet} – The SampleSheet from which the data originates.

Keyword Arguments: sample_name {string} – Optional: one sample to process from the sample_sheet. (default: {None}) from_s3 {zip_reader} – pass in a S3ZipReader object to extract idat files from a zipfile hosted on s3. meta_only {True/False} – doesn't read idat files, only parses the meta data about them. (RawMetaDataset is same as RawDataset but has no idat probe values stored in object, because not needed in pipeline)

Raises: ValueError: If the number of probes between raw datasets differ.

Returns: [RawDatasets] – A list of idat data pairs, each a dict like {'green_idat': green_idat, 'red_idat': red_idat}

class methylprep.models.**ProbeType**

probes can either be type I or type II for CpG or Snp sequences. Control probes are used for background correction in different fluorescence ranges and staining efficiency. Type I probes record EITHER a red or a green value. Type II probes record both values together. NOOB uses the red fluorescence on a green probe and vice versa to calculate background fluorescence.

class methylprep.models.**Sample**(*data_dir*, *sentrix_id*, *sentrix_position*, ***addl_fields*)

Object representing a row in a SampleSheet file

Arguments: data_dir {string or path-like} – Base directory of the sample sheet and associated IDAT files. sentrix_id {string} – The slide number of the processed array. sentrix_position {string} – The position on the processed slide.

Keyword Arguments: addl_fields {} – Additional metadata describing the sample. including experiment subject meta data:

name (sample name, unique id) Sample_Type Control GSM_ID (same as sample name if using GEO public data)

array meta data:

group plate pool well

alternate_base_filename

GEO data sets using this file name convention.

get_export_filepath()

Called by run_pipeline to find the folder/filename to export data as CSV, but CSV file doesn't exist yet.

get_file_s3(*zip_reader*, *extension*, *suffix=None*)

replaces get_filepath, but for *s3* context. Since these files are compressed within a single zipfile in the bucket, they don't resolve to PurePaths.

get_filepath(*extension*, *suffix=None*, *verify=True*)

builds the filepath based on custom file extensions and suffixes during processing.

Params (verify): tests whether file exists, either in data_dir or somewhere in recursive search path of data_dir.

Export: uses this later to fetch the place where a file ought to be created – but doesn't exist yet, so use verify=False.

Notes: _suffix – used to create the <*file*>*_processed* files.

class methylprep.models.**SigSet**(*sample*, *green_idat*, *red_idat*, *manifest*, *debug=False*)

I'm gonna try to create a fresh methylprep "SigSet" to replace our methylationDataset and RawDataset objects, which are redundant, and even have redundant functions within them. Part of why I have been frustrated/confused by our code. Central to the SeSAMe platform is the SigSet data structure, an S4 class with slots containing signals for six different classes of probes: [x] II - Type-II probes; [x] IR - Type-I Red channel probes; [x] IG - Type-I Grn channel probes; [x] oobG - Out-of-band Grn channel probes (matching Type-I Red channel probes in number); [x] oobR - Out-of-band Red channel probes (matching Type-I Grn channel probes in number); [x] ctrl_green, ctrl_red - control probes. [x] methylated, unmethylated, snp_methylated, snp_unmethylated [x] fg_green, fg_red (opposite of oobG and oobR) AKA ibG, ibR for in-band probes.

- just tidying up how we access this stuff, and trying to stick to IlmnID everywhere because the illumina_id within IDAT files is no longer unique as a ref.
- I checked again, and no other array breaks these rules. But sounds like Bret won't stick to this pattern going forward with designs. So I suspect other software will break with new arrays, unless they rewrite for this too.
- this combines every layer of objects between IdatDatasets and SampleDataContainers.
- this avoids looping through probe subsets, instead referring to a lookup-dataframe of how these relate.
- **avoids probes.py** probe_type is a derived label, not in manifest (I, II, SnpI, SnpII, control)

address_code = None

SigSet EPIC ## - @IG probes: 49989 - 332 4145 70 7094 599 2958 ... ## - @IR probes: 92294 - 183 8040 1949 6152 833 89 ... ## - @II probes: 724612 - 6543 1596 3133 1011 3035 2837 ... ## - @oobG probes: 92294 - 138 277 107 218 232 80 ... ## - @oobR probes: 49989 - 1013 150 81 910 448 183 ... ## - @ctl probes: 635 ... ## - @pval: 866895 - 0.005141179 0.04914081 0.002757492 ...

SigSet 450k @II 350076 ..... methylated 485512 @IG 46298 ... oobR 46298 ..... unmethylated 485512 @IR 89203 ... oobG 89203 ..... snp_methylated 65 ..... snp_unmethylated 65 fg_green 396325 |vs| ibG 396374 (incl 40 + 9 SNPs) –(flattened)–> 442672 fg_red 439223 |vs| ibR 439279 (incl 40 + 16 SNPs) –(flattened)–> 528482

check_for_probe_loss(*stage=''*)

Debugger runs this during processing to see where mouse probes go missing or get duplicated.

detect_and_drop_duplicates()

as of v1.5.0, mouse manifest includes a few probes that cause duplicate values, and breaks processing. So this removes them. About 5 probes in all.

Note: This runs during SigSet__init__, and might fail if any of these probes are affected by inter_type_I_probe_switch(), which theoretically should never happen in mouse. But infer-probes affects the idat probe_means directly, and runs before SigSet is created in SampleDataContainer, to avoid double-reading confusion.

set_noob(*red_factor*)

same method as update_probe_means, but simply applies a linear correction to all RED channel values

update_probe_means(*noob_green*, *noob_red*, *red_factor=None*)

pass in two dataframes (green and red) with IlmnIDs in index and a 'bg_corrected' column in each.

because __init__ has created each subset as a dataframe with IlmnID in index, this matches to index. and uses decoder to parse whether 'Meth' or 'Unmeth' values get updated.

upstream: container.sigset.update_probe_means(noob_green, noob_red)

replaces 'bg_corrected' column with 'noob_Meth' or 'noob_Unmeth' column.

does NOT update ctrl_red or ctrl_green; these are updated within the NOOB function because structually different.

class methylprep.models.**RawMetaDataset**(*sample*)
Wrapper for a sample and meta data, without its pair of raw IdatDataset values.

methylprep.models.**get_array_type**(*idat_dataset_pairs*)
provide a list of idat_dataset_pairs and it will return the array type, confirming probe counts match in batch.

files

class methylprep.files.**IdatDataset**(*filepath_or_buffer*, *channel*, *idat_id='IDAT'*, *idat_version=3*, *verbose=False*, *std_dev=False*, *nbeads=False*, *bit='float32'*)
Validates and parses an Illumina IDAT file.

Arguments: filepath_or_buffer {file-like} – the IDAT file to parse. channel {Channel} – the fluorescent channel (Channel.RED or Channel.GREEN) that produced the IDAT dataset.

Keyword Arguments: idat_id {string} – expected IDAT file identifier (default: {DEFAULT_IDAT_FILE_ID}) idat_version {integer} – expected IDAT version (default: {DEFAULT_IDAT_VERSION}) bit {string, default 'float32'} – 'float16' will pre-normalize intensities,

capping max intensity at 32127. This cuts data size in half, but will reduce precision on ~0.01% of probes. [effectively downscaling fluorescence]

Raises: ValueError: The IDAT file has an incorrect identifier or version specifier.

meta(*idat_file*)
To enable this, initialize idatDataset with verbose=True

read(*idat_file*)
Reads the IDAT file and parses the appropriate sections. Joins the mean probe intensity values with their Illumina probe ID.

Arguments: idat_file {file-like} – the IDAT file to process.

Returns: DataFrame – mean probe intensity values indexed by Illumina ID.

class methylprep.files.**Manifest**(*array_type*, *filepath_or_buffer=None*, *on_lambda=False*, *verbose=True*)
Provides an object interface to an Illumina array manifest file.

Arguments: array_type {ArrayType} – The type of array to process. values are styled like ArrayType.ILLUMINA_27K, ArrayType.ILLUMINA_EPIC or ArrayType('epic'), ArrayType('mouse')

Keyword Arguments: filepath_or_buffer {file-like} – a pre-existing manifest filepath (default: {None})

Raises: ValueError: The sample sheet is not formatted properly or a sample cannot be found.

static download_default(*array_type*, *on_lambda=False*)
Downloads the appropriate manifest file if one does not already exist.

Arguments: array_type {ArrayType} – The type of array to process.

Returns: [PurePath] – Path to the manifest file.

get_probe_details(*probe_type*, *channel=None*)
used by infer_channel_switch. Given a probe type (I, II, SnpI, SnpII, Control) and a channel (Channel.RED | Channel.GREEN), this will return info needed to map probes to their names (e.g. cg0031313 or rs00542420), which are NOT in the idat files.

read_control_probes(*manifest_file*)
Unlike other probes, control probes have no IlmnID because they're not locus-specific. they also use arbitrary columns, ignoring the header at start of manifest file.

read_mouse_probes(*manifest_file*)
ILLUMINA_MOUSE contains unique probes whose names begin with 'mu' and 'rp' for 'murine' and 'repeat', respectively. This creates a dataframe of these probes, which are not processed like normal cg/ch probes.

read_snp_probes(*manifest_file*)
Unlike cpg and control probes, these rs probes are NOT sequential in all arrays.

static seek_to_start(*manifest_file*)
find the start of the data part of the manifest. first left-most column must be "IlmnID" to be found.

class methylprep.files.**SampleSheet**(*filepath_or_buffer*, *data_dir*)
Validates and parses an Illumina sample sheet file.

Arguments: filepath_or_buffer {file-like} – the sample sheet file to parse. dir_path {string or path-like} – Base directory of the sample sheet and associated IDAT files.

Raises: ValueError: The sample sheet is not formatted properly or a sample cannot be found.

build_meta_data(*samples=None*)
Takes a list of samples and returns a data_frame that can be saved as a pickle.

build_samples()
Builds Sample objects from the processed sample sheet rows.

Added to Sample as class_method: if the idat file is not in the same folder, (check if exists!) looks recursively for that filename and updates the data_dir for that Sample.

contains_column(*column_name*)
helper function to determine if sample_sheet contains a specific column, such as GSM_ID. SampleSheet must already have __data_frame in it.

get_sample(*sample_name*)
scans all samples for one matching sample_name, if provided. If no sample_name, then it returns all samples.

get_samples()
Retrieves Sample objects from the processed sample sheet rows, building them if necessary.

methylprep.files.**get_sample_sheet**(*dir_path*, *filepath=None*)
Generates a SampleSheet instance for a given directory of processed data.

Arguments: dir_path {string or path-like} – Base directory of the sample sheet and associated IDAT files.

Keyword Arguments:

filepath {string or path-like} – path of the sample sheet file if provided, otherwise one will try to be found. (default: {None})

Returns: [SampleSheet] – A SampleSheet instance.

methylprep.files.**get_sample_sheet_s3**(*zip_reader*)
reads a zipfile and considers all filenames with 'sample_sheet' but will test all csv. the zip_reader is an amazon S3ZipReader object capable of reading the zipfile header.

methylprep.files.**create_sample_sheet**(*dir_path*, *matrix_file=False*, *output_file='samplesheet.csv'*, *sample_type=''*, *sample_sub_type=''*)
Creates a samplesheet.csv file from the .IDAT files of a GEO series directory

Arguments: dir_path {string or path-like} – Base directory of the sample sheet and associated IDAT files. matrix_file {boolean} – Whether or not a Series Matrix File should be searched for names. (default: {False})

========== | ========= | ==== | ======= parameter | required | type | effect ========== | ========= ==== | ======= sample_type | optional | string | label all samples in created sheet as this type (i.e. blood, saliva, tumor cells) sample_sub_type | optional | string | further detail sample type for batch controls | optional | list of sample_names | assign all samples in controls list to be "control samples", not treatment samples. ========== | ========= | ==== | =======

Note: Because sample_names are only generated from Matrix files, this method won't let you assign controls to samples from CLI. Would require all sample names be passed in from CLI as well, a pretty messy endeavor.

Raises: FileNotFoundError: The directory could not be found.

methylprep.files.**find_sample_sheet**(*dir_path*, *return_all=False*)

Find sample sheet file for Illumina methylation array.

Notes: looks for csv files in {dir_path}. If more than one csv file found, returns the one that has "sample_sheet" or 'samplesheet' in its name. Otherwise, raises error.

Arguments: dir_path {string or path-like} – Base directory of the sample sheet and associated IDAT files. return_all – if True,

returns a list of paths to samplesheets, if multiple present, instead of raising an error.

Raises: FileNotFoundError: [description] Exception: [description]

Returns: [string] – Path to sample sheet in base directory

geo download

methylprep.download.**run_series**(*id*, *path*, *dict_only=False*, *batch_size=100*, *clean=True*, *abort_if_no_idats=True*, *decompress=True*)

Downloads the IDATs and metadata for a series then generates one metadata dictionary and one beta value matrix for each platform in the series

Arguments:

id [required] the series ID (can be a GEO or ArrayExpress ID)

path [required] the path to the directory to download the data to. It is assumed a dictionaries and beta values directory has been created for each platform (and will create one for each if not)

dict_only if True, downloads idat files and meta data and creates data dictionaries for each platform, but does not process them further.

batch_size the batch_size to use when processing samples (number of samples run at a time). By default is set to the constant 100.

clean if True, removes intermediate processing files

methylprep.download.**run_series_list**(*list_file*, *path*, *dict_only=False*, *batch_size=100*, ***kwargs*)

Downloads the IDATs and metadata for a list of series, creating metadata dictionaries and dataframes of sample beta_values

Arguments:

list_file [required] the name of the file containing a list of GEO_IDS and/or Array Express IDs to download and process. This file must be located in the directory data is downloaded to. Each line of the file should contain the name of one data series ID.

path [required] the path to the directory to download the data to. It is assumed a dictionaries and beta values directory has been created for each platform (and will create one for each if not)

dict_only if True, only downloads data and creates dictionaries for each platform

batch_size the batch_size to use when processing samples (number of samples run at a time). By default is set to the constant 100.

methylprep.download.**convert_miniml**(*geo_id*, *data_dir='.'*, *merge=True*, *download_it=True*, *extract_controls=False*, *require_keyword=None*, *sync_idats=False*, *remove_tgz=False*, *verbose=False*)

This scans the datadir for an xml file with the geo_id in it. Then it parses it and saves the useful stuff to a dataframe called "sample_sheet_meta_data.pkl". DOES NOT REQUIRE idats.

CLI version: python -m meta_data -i GSExxxxx -d <my_folder>

Arguments:

merge: If merge==True and there is a file with 'samplesheet' in the folder, and that sheet has GSM_IDs, merge that data into this samplesheet. Useful for when you have idats and want one combined samplesheet for the dataset.

download_it: if miniml file not in data_dir path, it will download it from web.

extract_controls [experimental]: if you only want to retain samples from the whole set that have certain keywords, such as "control" or "blood", this experimental flag will rewrite the samplesheet with only the parts you want, then feed that into run_pipeline with named samples.

require_keyword [experimental]: another way to eliminate samples from samplesheets, before passing into the processor. if specified, this keyword needs to appear somewhere in the values of a samplesheet.

sync_idats: If flagged, this will search *data_dir* for idats and remove any of those that are not found in the filtered samplesheet. Requires you to run the download function first to get all idats, before you run this *meta_data* function.

Attempts to also read idat filenames, if they exist, but won't fail if they don't.

methylprep.download.**build_composite_dataset**(*geo_id_list*, *data_dir*, *merge=True*, *download_it=True*, *extract_controls=False*, *require_keyword=None*, *sync_idats=True*, *betas=False*, *m_value=False*, *export=False*)

A wrapper function for convert_miniml() to download a list of GEO datasets and process only those samples that meet criteria. Specifically - grab the "control" or "normal" samples from a bunch of experiments for one tissue type (e.g. "blood"), process them, and put all the resulting beta_values and/or m_values pkl files in one place, so that you can run *methylize.load_both()* to create a combined reference dataset for QC, analysis, or meta-analysis.

Arguments:

geo_id_list (required): A list of GEO "GSEnnn" ids. From command line, pass these in as separate values

data_dir: folder to save data

merge (True): If merge==True and there is a file with 'samplesheet' in the folder, and that sheet has GSM_IDs, merge that data into this samplesheet. Useful for when you have idats and want one combined samplesheet for the dataset.

download_it (True): if miniml file not in data_dir path, it will download it from web.

extract_controls (True)): if you only want to retain samples from the whole set that have certain keywords, such as "control" or "blood", this experimental flag will rewrite the samplesheet with only the parts you want, then feed that into run_pipeline with named samples.

require_keyword (None): another way to eliminate samples from samplesheets, before passing into the processor. if specified, the "keyword" string passed in must appear somewhere in the values of a samplesheet for sample to be downloaded, processed, retained.

sync_idats: If flagged, this will search *data_dir* for idats and remove any of those that are not found in the filtered samplesheet. Requires you to run the download function first to get all idats, before you run this *meta_data* function.

betas: process beta_values

m_value: process m_values

- Attempts to also read idat filenames, if they exist, but won't fail if they don't.
- removes unneeded files as it goes, but leaves the xml MINiML file and folder there as a marker if a geo dataset fails to download. So it won't try again on resume.

methylprep.download.**search**(*keyword*, *filepath='.'*, *verbose=True*)

CLI/cron function to check for new datasets. set up as a weekly cron. uses a local storage file to compare with old datasets in <pattern>_meta.csv. saves the dates of each dataset from GEO; calculates any new ones as new rows. updates csv.

options: pass in -k keyword verbose (TruelFalse) — reports to page; saves csv too

returns: saves a CSV to disk and returns a dataframe of results

methylprep.download.**pipeline_find_betas_any_source**(***kwargs*)

beta_bake: Sets up a script to run methylprep that saves directly to path or S3. The slowest part of processing GEO datasets is downloading, so this handles that.

STEPS

- **uses *methylprep alert -k <keywords>* to curate a list of GEO IDs worth grabbing.** note that version 1 will only process idats. also runs methylcheck.load on processed files, if installed.
- downloads a zipfile, uncompresses it,
- creates a samplesheet,
- moves it into foxo-test-pipeline-raw for processing.
- You get back a zipfile with all the output data.

required kwargs:

- **project_name: string, like GSE123456, to specify one GEO data set to download.** to initialize, specify one GEO id as an input when starting the function. - beforehand, you can use *methylprep alert* to verify the data exists. - OR you can pass in a string of GEO_ID separated by commas without any spaces and it will split them.

optional kwargs:

- function: 'geo' (optional, ignored; used to specify this pipeline to run from command line)
- **data_dir:**
  - default is current working directory ('.') if omitted
  - use to specify where all files will be downloaded, processed, and finally stored, unless *–cleanup=False*.
  - if using AWS S3 settings below, this will be ignored.
- verbose: False, default is minimal logging messages.

- save_source: if True, it will retain .idat and/or -tbl-1.txt files used to generate beta_values dataframe pkl files.
- **compress: if True, it will package everything together in a {geo_id}.zip file, or use gzip if files are too big for zip.**
  - default is False
- clean: If True, removes files from folder, except the compressed output zip file. (Requires compress to the True too)

It will use local disk by default, but if you want it to run in AWS batch + efs provide these:

- efs (AWS elastic file system name, for lambda or AWS batch processing)
- **clean: default True. If False, does not explicitly remove the tempfolder files at end, or move files into data_dir ou**
  - if you need to keep folders in working/efs folder instead of moving them to the data_dir.
  - use cleanup=False when embedding this in an AWS/batch/S3 context,

  then use the *working tempfolder* path and filenames returned to copy these files into S3.

returns:

- if a single GEO_ID, returns a dict with "error", "filenames", and "tempdir" keys.
- if mulitple GEO_IDs, returns a dict with "error", "geo_ids" (nested dict), and "tempdir" keys. Uses same tempdir for everything, so clean should be set to True.
- "error" will be None if it worked okay.
- "filenames" will be a list of filenames that were created as outputs (type=string)
- **"tempdir" will be the python tempfile tempory-directory object. Passing this out prevents** garbage collector from removing it when the function ends, so you can retrive these files and run tempdir.cleanup() manually. Otherwise, python will remove the tempdir for you when python closes, so copy whatever you want out of it first. This makes it possible to use this function with AWS EFS (elastic file systems) as part of a lambda or aws-batch function where disk space is more limited.

NOTE: v1.3.0 does NOT support multiple GEO IDs yet.

1.4.8 Release History

v1.6.2

- Minor bug fixes

v1.6.1

- samplesheet: ensures csv and meta_data pickle match
- better error message when multiple samplesheet csvs are detected, and more stringent detection parameters
- updated CI/CD with github actions, faster testing, dropped flaky unit tests
- updated documentation

v1.6.0

- qualityMask: All versions of 1.5.x used a different convention for probes to drop or keep. In version 1.6.0 and above, methylprep reverts to the previous convention used in v1.4.6 and below: For the `quality_mask` column in the processed output CSV and the in-memory SampleDataContainer data_frame, 1.0 means keep the probe (good) and 0.0 means drop. (This reverses a previous convention and fixes bugs where the 0.0 were sometimes NaN.)

v1.5.9

- Fixed bug: the SampleDataContainer returned by run_pipeline did not have 1.0 and 0 for quality_mask. Matches the CSV export now.

v1.5.8

- Fixed bug in CSV export; quality_mask (probes to be excluded from pickles) were NaN instead of 1.0

v1.5.7

- **Merely a maintenance release to deal with some dependency conflicts in the python science stack.** Pandas version 1.3+ now supported.

v1.5.6

- completely rewritten and updated documentation, including a more extensive tutorial
- updated all manifests to include probe-to-locus mapping for two genome builds
  - Uses **OLD_** in front of 4 genomic columns to denote the older genome build in each respective array/organism
  - mouse manifest has mm39 (newer) and mm10 (older) genome assemblies.
  - human hg38 is a corrected and improved version of (OLD) hg19.
- **27k array is no longer supported. (It turns out it was never supported, but we just never unit-tested it. And** because there are no control probes and no type-II probes on the 27k first generation array, it would be a lot of work to support it, and nobody has ever asked about it.)
- **removed read_geo and detect_header_pattern from methylprep, because it is maintained in methylcheck and imported** into methylprep now.
- **new beta_bake CLI option `-z / --compress` will put all downloaded files into a zipfile. This used to be the default** behavior, and now it is optional.
- fixed minor bug where malformed custom user-supplied manifest files will return a useful error message.
- better processing error detection with a `check_for_probe_loss` functions that warns if probes are dropped

v1.5.5

- Fixed Reading IDATs progress bar in 'process'
- `download` now uses HTTPS requests to fetch GSExxxxxx_RAW.TAR data files instead of FTP, because this is way more reliable and avoids errors.

- Better error messages when trying to download and process GEO datasets containing multiple array types. The process splits these samples into separate folders based on the meta data, and tries to run each array type separately.
- Improved documentation everywhere
- Improved support for GEO series_matrix.txt.gz files and _family.xml -tbl-1.txt files
- Fixed bug where quality_mask was removing SNP (rs) probes from CSV or SampleDataContainer output.
- Removed detect_header_pattern and read_geo from methylprep. These are maintained in methylcheck and not called anywhere. Instead, methylprep tries to import methylcheck in the Download functions that need to read GEO data files. But if a user has not installed methylcheck, they will only see an error if they run specific commands in Download that require it.

v1.5.2

- Bug fix: added 'Extended_Type' into control_probes.pkl output. Required by methylcheck.plot_controls().
- Minor bug fixes and improved unit test coverage.
- Fixed bug where `process --minfi` was not working with `--all`. Added more test coverage for CLI.
- updated read_geo to handle more edge cases
- deprecated some never-used functions.
  - instead of methylprep.files.idat.RunInfo use IdatDataset(verbose=True)

v1.5.0, v1.5.1

- MAJOR refactor/overhaul of all the internal classes. This was necessary to fully support the mouse array.
- new SigSet class object that mirror's sesame's SigSet and SigDF object.
- Combines idats, manifest, and sample sheet into one object that is inherited by SampleDataContainer
- RawDataset, MethylationDataset, ProbeSubtype all deprecated and replaced by SigSet
- SampleDataContainer class is now basically the SigSet plus all pipeline processing settings
- new mouse manifest covers all probes and matches sesame's output
- Processing will work even if a batch of IDATs have differing probe counts for same array_type, though those differing probes in question may not be saved.
- unit tests confirm that methylprep, sesame, and minfi beta values output match to within 1% of each other now. Note that the intermediate stages of processing (after NOOB and after DYE) do not match with sesame in this version. Can be +/- 100 intensity units, likely due to differences in order of steps and/or oob/mask probes used.

v1.4.7

- mouse manifest updated to conform with illumina Genome Studio / sesame probe naming convention.
- mouse_probes.pkl now includes different probe types. Previously, if a probe type was 'mu' (multi) or 'rp' (repeat) or IlmnID started with 'uk' (unknown?), it was moved to experimental mouse_probes.pkl. This was about 6300 probes. Now, all 'Multi' and 'Random' probes are moved and stored in mouse_probes.pkl, about 30,000.
- mouse manifest has a 'design' column with tons of human-readable notes on different probe origins, including analogous EPIC human-mapped probes.

v1.4.6

- **pipeline CSV output will now include meth, unmeth, beta, and m-values for all probes, including failed probes.** version 1.4.0 to 1.4.5 was replacing these values with NaN if a probe was filtered by the quality_mask. Pickled beta, M-value, noob_meth, noob_unmeth output files will continue to exclude (e.g. show NaN) probes that failed poobah_pval or quality_mask.

v1.4.5

- fixed qualityMask for epic+

v1.4.4

- faster circleci testing
- mouse probes have duplicate names, breaking dye-bias step, so it will fallback to linear-dye when duplicates are present
- added more mouse array test coverage

v1.4.0

- now uses sesame's infer_type_I_channel function to detect and correct probe switching, if sesame=True
- **uses sesame's nonlinear dye bias correction function, if sesame=True** instead of the previous linear-dye-correction in the NOOB function.
- **as part of the run_pipeline(sesame=True) default ON settings, it will apply sesame's "quality_mask"** that automatically removes probes that are unreliable from all data.
- reads more IDAT raw data (run_info, probe nbeads, probe standard deviation)
  - idat.py IdatDataset has new kwargs, including bit='float16' option to cut file/memory usage in half by clipping max intensity at 32127 (which cuts off ~0.01% of probes)
- processing will mirror sesame more closely now, instead of minfi (to revert, use sesame=False in run_pipeline)
- adds sesame quality_mask, which auto-hides known set of sketchy probes.
- **internal objects updated so that values align in every stage of processing** (i.e. if you apply the sesame quality mask, the output files and the SampleDataContainer will exclude those probes)
- make_pipeline provides a scikit-learn style interface, as alternative to run_pipeline

v1.3.3

- ensures methylprep output matches sesame output
- order of probes in CSVs, pickles, and SampleDataContainer doesn't match
- **fixes bug where are few probes had negative meth/unmeth values because of int16 limits.** Now it uses unsigned int16 data type and unit tests confirm no negative values appear.

v1.3.2

- updated support for Illumina mouse array
- summarized processing warnings at end, to make tqdm progress bar cleaner

v1.3.1

- run_pipeline() has 50% shorter processing time due to user-submitted changes
- idats can be processed while gzipped (.idat.gz) and saved this way using –no_uncompress flag
- 'download' function manages FTP connection better
- improved unit testing: download and process_series
- run_pipeline() function has two new optional parameters
  - poobah_decimals: if you want more than the default 3 decimals saved in poobah_values.pkl and _processed.csv files-, then specify a higher limit.
  - poobah_sig: default significance level for excluding probes is 0.05. You can set it to something else in the 0.1 to 0.000001 range, if you want.

v1.3.0

- improved methylprep's run_pipeline() process to use less memory and avoid memory usage spikes
- files created are also smaller too (because they use float32 or int16 instead of 64 bit data)
- **output files like beta_values.pkl are automatically consolidated at end of pipeline.** batch_size will split large batches into multiple files during processing to save memory, but the output will be merged at the end.
- improved support for Illumina Mouse Arrays.
- **methylprep.download has a new all-encompassing pipeline that will read GEO data sets and convert** any data file type into a pickle of beta_values, whether from idats or processed matrix files.

Older versions exist on pypi, but no changelog

CHAPTER 2

Indices and tables

- genindex

Python Module Index

m

Index

I

M

N

P

R

S

U