Package ‘CellBench’

May 29, 2024

Type Package
Title Construct Benchmarks for Single Cell Analysis Methods
Version 1.20.0
Description This package contains infrastructure for benchmarking analysis methods and access to single cell mixture benchmarking data. It provides a framework for organising analysis methods and testing combinations of methods in a pipeline without explicitly laying out each combination. It also provides utilities for sampling and filtering SingleCellExperiment objects, constructing lists of functions with varying parameters, and multithreaded evaluation of analysis methods.

biocViews Software, Infrastructure, SingleCell

URL https://github.com/shians/cellbench
BugReports https://github.com/Shians/CellBench/issues
License GPL-3
Encoding UTF-8
Depends R (>= 3.6), SingleCellExperiment, magrittr, methods, stats, tibble, utils
Imports assertthat, BiocGenerics, BiocFileCache, BiocParallel, dplyr, rlang, glue, memoise, purrr (>= 0.3.0), rappdirs, tidyr, tidyselect, lubridate
Suggests BiocStyle, covr, knitr, rmarkdown, testthat, limma, ggplot2

VignetteBuilder knitr
RoxygenNote 7.2.3
git_url https://git.bioconductor.org/packages/CellBench
git_branch RELEASE_3_19
git_last_commit 4713b56
git_last_commit_date 2024-04-30
Repository Bioconductor 3.19
Date/Publication 2024-05-29
**Author**  Shian Su [cre, aut],  
Saskia Freytag [aut],  
Luyi Tian [aut],  
Xueyi Dong [aut],  
Matthew Ritchie [aut],  
Peter Hickey [ctb],  
Stuart Lee [ctb]  

**Maintainer**  Shian Su <su.s@wehi.edu.au>  

**Contents**

- CellBench-package ................................................................. 3  
- all_unique ................................................................. 3  
- any_task_errors ......................................................... 4  
- apply_methods ............................................................... 4  
- arrow_sep ...................................................... 6  
- as_pipeline_list ............................................................... 6  
- cache_method ............................................................... 7  
- cellbench_case_study .......................................................... 8  
- cellbench_file ............................................................... 8  
- check_class ............................................................... 9  
- clear_cached_datasets ...................................................... 10  
- clear_cellbench_cache ...................................................... 10  
- collapse_pipeline ........................................................... 11  
- data_list ................................................................. 12  
- filter_zero_genes ............................................................. 13  
- fn_arg_seq ................................................................. 13  
- fn_list ................................................................. 14  
- is.task_error ............................................................... 15  
- keep_high_count_cells ...................................................... 15  
- keep_high_count_genes ...................................................... 16  
- keep_high_var_genes ...................................................... 16  
- load_sc_data ................................................................. 17  
- mhead ................................................................. 18  
- print.fn_arg_seq .............................................................. 18  
- print.task_error ............................................................. 19  
- sample_cells ................................................................. 19  
- sample_genes ................................................................. 20  
- sample_sce_data .............................................................. 21  
- set_cellbench_bpparam ...................................................... 21  
- set_cellbench_cache_path .................................................. 22  
- set_cellbench_threads ..................................................... 23  
- split_step ................................................................. 23  
- strip_timing ................................................................. 24  
- summary.benchmark_tbl ..................................................... 25  
- time_methods ................................................................. 26  
- unpack_timing ................................................................. 27
CellBench-package

A framework for benchmarking combinations of methods in multi-stage pipelines

Description

This package contains a framework for benchmarking combinations of methods in a multi-stage pipeline. It is mainly based around the apply_methods function, which takes lists of functions to be applied in stages of a pipeline.

Author(s)

Shian Su <https://www.github.com/shians>

See Also

The core function in this package is apply_methods, see vignette("Introduction", package = "CellBench") for basic usage. Run cellbench_case_study() to see a case study using CellBench. The data loading functions from load_all_data may also be of interest.

all_unique

Check if all values in a vector are unique

Description

Check if all values in a vector are unique

Usage

all_unique(x)

Arguments

x the vector to check

Value

TRUE if all values in the vector are unique

Examples

all_unique(c(1, 2, 3)) # TRUE
all_unique(c(1, 2, 2)) # FALSE
any_task_errors  

Check if any tasks produced errors

Description
Check the results column of a benchmark tibble for any task_error objects.

Usage
any_task_errors(x, verbose)

## S3 method for class 'benchmark_tbl'
any_task_errors(x, verbose = FALSE)

Arguments

x  
the tibble to check

verbose  
TRUE if the rows with errors should be reported

Value
TRUE if any entry in the result column is a task_error object

Methods (by class)

- any_task_errors(benchmark_tbl):

apply_methods  

Apply methods

Description
apply_methods() and its aliases apply_metrics and begin_benchmark take either lists of datasets or benchmark_tbl objects and applies a list of functions. The output is a benchmark_tbl where each method has been applied to each dataset or preceding result.

Usage
apply_methods(x, fn_list, name = NULL, suppress.messages = TRUE)

## S3 method for class 'list'
apply_methods(x, fn_list, name = NULL, suppress.messages = TRUE)

## S3 method for class 'benchmark_tbl'
apply_methods(x, fn_list, name = NULL, suppress.messages = TRUE)
apply_methods

```r
## S3 method for class 'tbl_df'
apply_methods(x, fn_list, name = NULL, suppress.messages = TRUE)

apply_metrics(x, fn_list, name = NULL, suppress.messages = TRUE)

begin_benchmark(x, fn_list, name = NULL, suppress.messages = TRUE)
```

### Arguments

- `x`: the list of data or benchmark tibble to apply methods to
- `fn_list`: the list of methods to be applied
- `name`: (optional) the name of the column for methods applied
- `suppress.messages`: TRUE if messages from running methods should be suppressed

### Value

Benchmark tbl object containing results from methods applied, the first column is the name of the dataset as factors, middle columns contain method names as factors and the final column is a list of results of applying the methods.

### See Also

- `time_methods`

### Examples

```r
# list of data
datasets <- list(
  set1 = rnorm(500, mean = 2, sd = 1),
  set2 = rnorm(500, mean = 1, sd = 2)
)

# list of functions
add_noise <- list(
  none = identity,
  add_bias = function(x) { x + 1 }
)

res <- apply_methods(datasets, add_noise)
```
Description
Utility function for generating unicode arrow separators.

Usage
arrow_sep(towards = c("right", "left"), unicode = FALSE)

Arguments
towards the direction the unicode arrow points towards
unicode whether unicode arrows should be used. Does not work inside plots within knitted PDF documents.

Value
a string containing an unicode arrow surrounded by two spaces

Examples
arrow_sep("left") # left arrow
arrow_sep("right") # right arrow

Description
convert a benchmark_tbl to a list where the name of the elements represent the pipeline steps separated by ".". This can be useful for using the apply family of functions.

Usage
as_pipeline_list(x)

Arguments
x the benchmark_tbl object to convert

Value
list containing the results with names set to data and pipeline steps separated by ..
**cache_method**

Create a cached function for CellBench

### Description

Take a function and return a cached version. The arguments and results of a cached method is saved to disk and if the cached function is called again with the same arguments then the results will be retrieved from the cache rather than be recomputed.

### Usage

```r
cache_method(f, cache = getOption("CellBench.cache"))
```

### Arguments

- **f**: the function to be cached
- **cache**: the cache information (from memoise package)

### Details

(CAUTION) Because cached functions called with the same argument will always return the same output, pseudo-random methods will not return varying results over repeated runs as one might expect.

This function is a thin wrapper around `memoise`

### Value

function whose results are cached and is called identically to f

---

**See Also**

- `collapse_pipeline`

**Examples**

```r
# list of data
datasets <- list(
  set1 = rnorm(500, mean = 2, sd = 1),
  set2 = rnorm(500, mean = 1, sd = 2)
)

# list of functions
add_noise <- list(
  none = identity,
  add_bias = function(x) { x + 1 } 
)

res <- apply_methods(datasets, add_noise)
as_pipeline_list(res)
```
See Also

set_cellbench_cache_path

Examples

# sets cache path to a temporary directory
set_cellbench_cache_path(file.path(tempdir(), ".CellBenchCache"))
f <- function(x) { x + 1 }
cached_f <- cache_method(f)

---

cellbench_case_study  Open vignette containing a case study using CellBench

description

Open vignette containing a case study using CellBench

Usage

cellbench_case_study()

value

opens a vignette containing a case study

Examples

## Not run:
cellbench_case_study()
## End(Not run)

---

cellbench_file  Get path to CellBench packaged data

description

Search CellBench package for packaged data, leaving argument empty will list the available data.

Usage

cellbench_file(filename = NULL)
**check_class**

**Arguments**

- **filename**
  - the name of the file to look for

**Value**

- string containing the path to the packaged data

**Examples**

cellbench_file() # shows available files
cellbench_file("10x_sce_sample.rds") # returns path to 10x sample data

---

**check_class**  
*Check class of object*

**Description**

Check an object against a vector of class names. Testing if they match any or all of the classes. For is_all_of, the object needs to be at least every class specified, but it can have addition classes and still pass the check.

**Usage**

- is_one_of(x, classes)
- is_any_of(x, classes)
- is_all_of(x, classes)

**Arguments**

- **x**
  - the object to check
- **classes**
  - the vector of strings of class names

**Value**

- boolean value for the result of the check

**Examples**

- is_one_of(1, c("numeric", "logical")) # TRUE
- is_one_of(1, c("character", "logical")) # FALSE

- is_all_of(1, c("numeric", "logical")) # FALSE
- is_all_of(tibble::tibble(), c("tbl", "data.frame")) # TRUE
clear_cached_datasets  Clear cached datasets

Description
Delete the datasets cached by the load_*_data set of functions

Usage
clear_cached_datasets()

Value
None

Examples
## Not run:
clear_cached_datasets()
## End(Not run)

clear_cellbench_cache  Clear CellBench Cache

Description
Clears the method cache for CellBench

Usage
clear_cellbench_cache()

Value
None

Examples
## Not run:
clear_cellbench_cache()
## End(Not run)
collapse_pipeline

Collapse benchmark_tbl into a two column summary

Description

Collapse benchmark_tbl into two columns: "pipeline" and "result". The "pipeline" column will be the concatenated values from the data and methods columns while the "result" column remains unchanged from the benchmark_tbl. This is useful for having a string summary of the pipeline for annotating.

Usage

collapse_pipeline(
  x,
  sep = arrow_sep("right"),
  drop.steps = TRUE,
  data.name = TRUE
)

pipeline_collapse(
  x,
  sep = arrow_sep("right"),
  drop.steps = TRUE,
  data.name = TRUE
)

Arguments

x  the benchmark_tbl to collapse
sep the separator to use for concatenating the pipeline steps
drop.steps if the data name and methods steps should be dropped from the output. TRUE by default.
data.name if the dataset name should be included in the pipeline string. Useful if only a single dataset is used.

Value

benchmark_tbl with pipeline and result columns (and all other columns if drop.steps is FALSE)

See Also

as_pipeline_list
data_list

Constructor for a data list

Description

Constructor for a list of data, a thin wrapper around list() which checks that all the inputs are of the same type and have names.

Usage

data_list(...)

Arguments

... objects, must all be named

Value

a list of named data

Examples

data(iris)
flist <- data_list(
    data1 = iris[1:20, ],
    data2 = iris[21:40, ]
)
filter_zero_genes  
*Filter out zero count genes*

**Description**
Remove all genes (rows) where the total count is 0

**Usage**
`filter_zero_genes(x)`

**Arguments**
- `x`  
  the SingleCellExperiment or matrix to filter

**Value**
object of same type as input with all zero count genes removed

**Examples**
```r
x <- matrix(rep(0:5, times = 5), nrow = 6, ncol = 5)
filter_zero_genes(x)
```

---

fn_arg_seq  
*Create a list of functions with arguments varying over a sequence*

**Description**
Generate a list of functions where specific arguments have been pre-applied from a sequences of arguments, i.e. a function `f(x, n)` may have the `n` argument pre-applied with specific values to obtain functions `f1(x, n = 1)` and `f2(x, n = 2)` stored in a list.

**Usage**
`fn_arg_seq(func, ..., .strict = FALSE)`

**Arguments**
- `func`  
  function to generate list from
- `...`  
  vectors of values to use as arguments
- `.strict`  
  TRUE if argument names are checked, giving an error if specified argument does not appear in function signature. Note that functions with multiple methods generally have only `f(x, ...)` as their signature, so the check would fail even if the arguments are passed on.
Details

If multiple argument vectors are provided then the combinations of arguments in the sequences will be generated.

Value

list of functions with the specified arguments pre-applied. Names of the list indicate the values that have been pre-applied.

Examples

```r
f <- function(x) {
  cat("x:", x)
}

f_list <- fn_arg_seq(f, x = c(1, 2))
f_list
f_list[[1]]() # x: 1
f_list[[2]]() # x: 2

g <- function(x, y) {
  cat("x:", x, "y:", y)
}

g_list <- fn_arg_seq(g, x = c(1, 2), y = c(3, 4))
g_list
g_list[[1]]() # x: 1 y: 3
g_list[[2]]() # x: 1 y: 4
g_list[[3]]() # x: 2 y: 3
g_list[[4]]() # x: 2 y: 4
```

---

**fn_list**  
Constructor for a function list

Description

Constructor for a list of functions, a thin wrapper around list() which checks that all the inputs are functions and have names

Usage

```r
fn_list(...)  
```

Arguments

...  
objects, must all be named

Value

a list of named functions
is.task_error

Examples

```r
flist <- fn_list(
    mean = mean,
    median = median
)
```

---

is.task_error  Check for task errors

Description

This is a helper function for checking the result column of a benchmark_tbl for task_error objects. This is useful for filtering out rows where the result is a task error.

Usage

```r
is.task_error(x)
```

Arguments

- `x` the object to be tested

Value

vector of logicals denoting if elements of the list are task_error objects

---

keep_high_count_cells  Filter down to the highest count cells

Description

Filter a SingleCellExperiment or matrix down to the cells (columns) with the highest counts

Usage

```r
keep_high_count_cells(x, n)
```

Arguments

- `x` the SingleCellExperiment or matrix
- `n` the number of highest count cells to keep

Value

object of same type as input containing the highest count cells
keep_high_var_genes

Description
Filter a SingleCellExperiment or matrix down to the most variable genes (rows), variability is determined by var() scaled by the total counts for the gene.

Usage
keep_high_var_genes(x, n)

Arguments
x the SingleCellExperiment or matrix
n the number of most variable genes to keep

Value
object of same type as input containing the highest count genes

Examples
data(sample_sce_data)
keep_high_var_genes(sample_sce_data, 300)

keep_high_count_genes  Filter down to the highest count genes

Description
Filter a SingleCellExperiment or matrix down to the genes (rows) with the highest counts

Usage
keep_high_count_genes(x, n)

Arguments
x the SingleCellExperiment or matrix
n the number of highest count genes to keep

Value
object of same type as input containing the highest count genes

Examples
data(sample_sce_data)
keep_high_count_genes(sample_sce_data, 10)
load_sc_data

Value

object of same type as input containing the most variable genes

Examples

data(sample_sce_data)
keep_high_var_genes(sample_sce_data, 50)

load_sc_data  Load CellBench Data

Description

Load in all CellBench data described at <https://github.com/LuyiTian/CellBench_data/blob/master/README.md>.

Usage

load_sc_data()
load_cell_mix_data()
load_mrna_mix_data()
load_all_data()

Value

list of SingleCellExperiment

Functions

• load_sc_data(): Load single cell data
• load_cell_mix_data(): Load cell mixture data
• load_mrna_mix_data(): Load mrna mixture data

Examples

## Not run:
cellbench_file <- load_all_data()

## End(Not run)
mhead

Get head of 2 dimensional object as a square block

Description

head prints all columns which may flood the console, mhead takes a square block which can look nicer and still provide a good inspection of the contents

Usage

mhead(x, n = 6)

Arguments

x the object with 2 dimensions
n the size of the n-by-n block to extract

Value

an n-by-n sized subset of x

Examples

x <- matrix(runif(100), nrow = 10, ncol = 10)
mhead(x)
mhead(x, n = 3)

print.fn_arg_seq

Print method for fn_arg_seq output

Description

Print method for fn_arg_seq output

Usage

## S3 method for class 'fn_arg_seq'
print(x, ...)

Arguments

x fn_arg_seq object
... addition arguments for print
**print.task_error**

**Value**

None

**Examples**

```r
fn_seq <- fn_arg_seq(kmeans, centers = 1:3)
fn_seq
```

---

**print.task_error**  
*Print method for task_error object*

**Description**

Task error objects that result from failed methods

**Usage**

```r
## S3 method for class 'task_error'
print(x, ...)
```

**Arguments**

- `x`  
  a task_error object
- `...`  
  not used

**Value**

None

---

**sample_cells**  
*Sample cells from a SingleCellExperiment*

**Description**

Sample n cells from a SingleCellExperiment object with no replacement.

**Usage**

```r
sample_cells(x, n)
```

**Arguments**

- `x`  
  the SingleCellExperiment object
- `n`  
  the number of cells to sample
sample_genes

Description
Sample n genes from a SingleCellExperiment object with no replacement

Usage
sample_genes(x, n)

Arguments
x the SingleCellExperiment object
n the number of genes to sample

Value
SingleCellExperiment object

Examples
```r
sample_sce_data <- readRDS(cellbench_file("celseq_sce_sample.rds"))
dim(sample_sce_data)
x <- sample_cells(sample_sce_data, 10)
dim(x)
```
sample_sce_data

This is data for testing functions in CellBench.

Description

A dataset containing 200 genes and 50 cells randomly sampled from the CelSeq mRNA mixture dataset, each sample is a mixture of mRNA material from 3 different human adenocarcinoma cell lines. Useful for quick prototyping of method wrappers.

Usage

data(sample_sce_data)

Format

A SingleCellExperiment object with sample annotations in colData(sample_sce_data). The annotation contains various QC metrics as well as the cell line mixture proportions

- **H2228_prop** proportion of mRNA from H2228 cell line
- **H1975_prop** proportion of mRNA from H1975 cell line
- **HCC827_prop** proportion of mRNA from HCC827 cell line

See Also

- `load_mrna_mix_data`

set_cellbench_bpparam

Set BiocParallel parameter used CellBench

Description

This is a more advanced interface for changing CellBench’s parallelism settings. Internally CellBench uses BiocParallel for parallelism, consult the documentation of BiocParallel to see what settings are available.

Usage

set_cellbench_bpparam(param)

Arguments

- **param** the BiocParallel parameter object

Value

None
See Also

`set_cellbench_threads` for more basic interface

Examples

```r
set_cellbench_threads(1) # CellBench runs on a single thread
```

---

### set_cellbench_cache_path

Set CellBench cache path

**Description**

Set CellBench cache path

**Usage**

```r
set_cellbench_cache_path(path = "./.CellBenchCache")
```

**Arguments**

- `path` the path to where method caches should be stored

**Value**

None

**See Also**

`cache_method` for constructing cached methods.

**Examples**

```r
## Not run:
# hidden folder in local path
set_cellbench_cache_path("./.CellBenchCache")

## End(Not run)
# store in temp directory valid for this session
set_cellbench_cache_path(file.path(tempdir(), ".CellBenchCache"))
```
set_cellbench_threads  
Set number of threads used by CellBench

Description
Sets global parameter for CellBench to use multiple threads for applying methods. If any methods applied are multi-threaded then it's recommended to set CellBench threads to 1. It only recommended to use CellBench with multiple threads if methods applied can be set to run on single threads.

Usage
set_cellbench_threads(nthreads = 1)

Arguments
nthreads the number of threads used by CellBench

Value
None

See Also
set_cellbench_bpparam for more advanced interface

Examples
set_cellbench_threads(1) # CellBench runs on a single thread

split_step  
Split combined pipeline step

Description
Some methods perform multiple steps of a pipeline. This function assists with splitting the combined pipeline step into multiple steps with duplicated method names.

Usage
split_step(x, step, into)
Arguments

x  a results data.frame from `apply_methods()`.
step  the name of the column to split.
into  the name of the columns to split into.

Value

a results data.frame where the ‘step’ column has been split into the ‘into’ columns with duplicated values.

Examples

datasets <- list(
  set1 = rnorm(500, mean = 2, sd = 1),
  set2 = rnorm(500, mean = 1, sd = 2)
)
# list of functions
add_noise <- list(
  none = identity,
  add_bias = function(x) { x + 1 }
)
res <- apply_methods(datasets, add_noise)
res %>%
  split_step("add_noise", c("split1", "split2"))

strip_timing
Strip timing information

Description

Takes the result of a time_methods() call and remove timing information from the ‘timed_result’ column, replacing it with a ‘result’ column and converting it to a benchmark_tbl.

Usage

stripTiming(x)

## S3 method for class 'benchmark_timing_tbl'
strip_timing(x)

Arguments

x  the benchmark_timing_tbl object
Value

benchmark_tbl

See Also

unpack_timing

Examples

```r
## Not run:
datasets <- list(
  data1 = 1:1e8,
)
transforms <- list(
  log = log,
  sqrt = sqrt
)
datasets %>%
  time_methods(transforms) %>%
  strip_timing()

## End(Not run)
```

### S3 method for class 'benchmark_tbl'

```r
summary(object, ...)
```

Arguments

- `object` the benchmark_tbl to be summarised
- `...` additional arguments affecting the summary produced.

Value

None
Examples

```r
# list of data
datasets <- list(
    set1 = rnorm(500, mean = 2, sd = 1),
    set2 = rnorm(500, mean = 1, sd = 2)
)

# list of functions
add_noise <- list(
    none = identity,
    add_bias = function(x) { x + 1 }
)

res <- apply_methods(datasets, add_noise)
summary(res)
```

Description

time_methods() take either lists of datasets or benchmark_timing_tbl objects and applies a list of functions. The output is a benchmark_timing_tbl where each method has been applied to each dataset or preceding result. Unlike apply_methods(), time_methods() is always single threaded as to produce fair and more consistent timings.

Usage

```r
time_methods(x, fn_list, name = NULL, suppress.messages = TRUE)
```

## S3 method for class 'list'
```r
time_methods(x, fn_list, name = NULL, suppress.messages = TRUE)
```

## S3 method for class 'benchmark_timing_tbl'
```r
time_methods(x, fn_list, name = NULL, suppress.messages = TRUE)
```

Arguments

- `x` the list of data or benchmark timing tblle to apply methods to
- `fn_list` the list of methods to be applied
- `name` (optional) the name of the column for methods applied
- `suppress.messages` TRUE if messages from running methods should be suppressed

Value

benchmark_timing_tbl object containing results from methods applied, the first column is the name of the dataset as factors, middle columns contain method names as factors and the final column is a list of lists containing the results of applying the methods and timings from calling system.time().
**unpack_timing**

See Also

- apply_methods

Examples

```r
datasets <- list(
    set1 = 1:1e7
)

transform <- list(
    sqrt = sqrt,
    log = log
)

time_methods(datasets, transform) %>%
  unpack_timing() # extract timings out of list
```

**unpack_timing**

Unpack timing information

**Description**

Takes the result of a time_methods() call and remove the 'timed_result' column, replacing it with three columns of durations representing the 'system', 'user' and 'elapsed' times from a system.time() call.

**Usage**

```r
unpack_timing(x)
```

```r
## S3 method for class 'benchmark_timing_tbl'
unpack_timing(x)
```

**Arguments**

- `x` : the benchmark_timing_tbl object

**Value**

a tibble containing pipeline steps and timing information

**See Also**

- strip_timing
Examples

```r
## Not run:

datasets <- list(
  data1 = c(1, 2, 3)
)

transforms <- list(
  log = function(x) { Sys.sleep(0.1); log(x) },
  sqrt = function(x) { Sys.sleep(0.1); sqrt(x) }
)

datasets %>%
  time_methods(transforms) %>%
  unpack_timing()

## End(Not run)
```
Index

* datasets
  sample_sce_data, 21
* internal
  all_unique, 3
  check_class, 9
  print.fn_arg_seq, 18
  print.task_error, 19
  strip_timing, 24
  summary.benchmark_tbl, 25
  unpack_timing, 27
  all_unique, 3
  any_task_errors, 4
  apply_methods, 3, 4, 27
  apply_metrics (apply_methods), 4
  arrow_sep, 6
  as_pipeline_list, 6, 11
  begin_benchmark (apply_methods), 4
  cache_method, 7, 22
  CellBench (CellBench-package), 3
  CellBench-package, 3
  cellbench_case_study, 8
  cellbench_file, 8
  check_class, 9
  clear_cached_datasets, 10
  clear_cellbench_cache, 10
  collapse_pipeline, 7, 11
  data_list, 12
  filter_zero_genes, 13
  fn_arg_seq, 13
  fn_list, 14
  is.task_error, 15
  is_all_of (check_class), 9
  is_any_of (check_class), 9
  is_one_of (check_class), 9
  keep_high_count_cells, 15
  keep_high_count_genes, 16
  keep_high_var_genes, 16
  load_all_data, 3
  load_all_data (load_sc_data), 17
  load_cell_mix_data (load_sc_data), 17
  load_mrna_mix_data (load_sc_data), 21
  load_mrna_mix_data (load_sc_data), 17
  load_sc_data, 17
  memoise, 7
  mhead, 18
  pipeline_collapse (collapse_pipeline), 11
  print.fn_arg_seq, 18
  print.task_error, 19
  sample_cells, 19
  sample_genes, 20
  sample_sce_data, 21
  set_cellbench bpparam, 21, 23
  set_cellbench_cache_path, 8, 22
  set_cellbench_threads, 22, 23
  split_step, 23
  strip_timing, 24, 27
  summary.benchmark_tbl, 25
  time_methods, 5, 26
  unpack_timing, 25, 27