Package ‘anndata’

September 10, 2021

**Type** Package

**Title** 'anndata' for R

**Version** 0.7.5.3

**Description** A 'reticulate' wrapper for the Python package 'anndata'. Provides a scalable way of keeping track of data and learned annotations. Used to read from and write to the h5ad file format.

**License** MIT + file LICENSE

**URL** [https://anndata.dynverse.org](https://anndata.dynverse.org), [https://github.com/dynverse/anndata](https://github.com/dynverse/anndata)

**BugReports** [https://github.com/dynverse/anndata/issues](https://github.com/dynverse/anndata/issues)

**Depends** R (>= 3.5.0)

**Imports** assertthat, Matrix, methods, R6, reticulate (>= 1.17)

**Suggests** stats, testthat, knitr, rmarkdown

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**Config/reticulate** list( packages = list( list(package = ```anndata```))))

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Philipp Angerer [ccp] (<https://orcid.org/0000-0002-0369-2888>, flying-sheep),
Alex Wolf [ccp] (<https://orcid.org/0000-0002-8760-7838>, falexwolf),
Isaac Virshup [ccp] (ivirshup),
Sergei Rybakov [ccp] (Koncopd),
Robrecht Cannoodt [aut, cre, cph]
(<https://orcid.org/0000-0003-3641-729X>, rcannood)

**Maintainer** Robrecht Cannoodt <rcannood@gmail.com>

**Repository** CRAN

**Date/Publication** 2021-09-10 09:30:02 UTC
Description

anndata provides a scalable way of keeping track of data and learned annotations, and can be used to read from and write to the h5ad file format. AnnData() stores a data matrix \( X \) together with annotations of observations obs (obsm, obsp), variables var (varm, varp), and unstructured annotations uns.

Details

This package is, in essence, an R wrapper for the similarly named Python package anndata, with some added functionality to support more R-like syntax. The version number of the anndata R package is synced with the version number of the python version.

Check out ?anndata for a full list of the functions provided by this package.

Creating an AnnData object

- AnnData()
Concatenating two or more AnnData objects

- `concat()`

Reading an AnnData object from a file

- `read_csv()`
- `read_excel()`
- `read_h5ad()`
- `read_hdf()`
- `read_loom()`
- `read_mtx()`
- `read_text()`
- `read_umi_tools()`

Writing an AnnData object to a file

- `write_csvs()`
- `write_h5ad()`
- `write_loom()`

Install the `anndata` Python package

- `install_anndata()`

Examples

```r
## Not run:
ad <- AnnData(
  X = matrix(1:6, nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L, 3L), row.names = c("var1", "var2", "var3")),
  layers = list(
    spliced = matrix(4:9, nrow = 2),
    unspliced = matrix(8:13, nrow = 2)
  ),
  obsm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  varm = list(
    ones = matrix(rep(1L, 12), nrow = 3),
    rand = matrix(rnorm(6), nrow = 3),
    zeros = matrix(rep(0L, 12), nrow = 3)
  ),
  uns = list(
    a = 1,
    b = data.frame(i = 1:3, j = 4:6, value = runif(3)),
  )
)
```
c = list(c.a = 3, c.b = 4)

ad$X

ad$obs

ad$var

ad$obs$ms"ones"

ad$var$nm"rand"

ad$layers$nm"unspliced"

ad$layers$nm"spliced"

ad$uns$nm"b"

ad[,c("var1", "var2")]

ad[-1, , drop = FALSE]

ad[, 2] <- 10

## End(Not run)

---

**all.equal.AnnDataR6**  
*Test if two AnnDataR6 objects are equal*

### Description

Test if two AnnDataR6 objects are equal

### Usage

```r
## S3 method for class 'equal.AnnDataR6'
all(target, current)
```

### Arguments

- **target**: R object.
- **current**: other R object, to be compared with target.
all.equal.LayersR6

Test if two LayersR6 objects are equal

Description

Test if two LayersR6 objects are equal

Usage

## S3 method for class 'equal.LayersR6'
all(target, current)

Arguments

target

R object.

current

other R object, to be compared with target.

AnnData

Create an Annotated Data Matrix

Description

AnnData stores a data matrix X together with annotations of observations obs (obsm, obsp), variables var (varm, varp), and unstructured annotations uns.

An AnnData object adata can be sliced like a data frame, for instance adata_subset <- adata[, list_of_variable_names]

AnnData’s basic structure is similar to R’s ExpressionSet.

If setting an h5ad-formatted HDF5 backing file filename, data remains on the disk but is automatically loaded into memory if needed. See this blog post for more details.

Usage

AnnData(
  X = NULL,
  obs = NULL,
  var = NULL,
  uns = NULL,
  obsm = NULL,
  varm = NULL,
  layers = NULL,
  raw = NULL,
  dtype = "float32",
  shape = NULL,
  filename = NULL,
  filemode = NULL,
  obsp = NULL,


```r
varp = NULL
)

Raw(adata, X = NULL, var = NULL, varm = NULL)
```

## Arguments

- **X**  
  A $\#\text{observations} \times \#\text{variables}$ data matrix.
- **obs**  
  Key-indexed one-dimensional observations annotation of length $\#\text{observations}$.
- **var**  
  Key-indexed one-dimensional variables annotation of length $\#\text{variables}$.
- **uns**  
  Key-indexed unstructured annotation.
- **obsm**  
  Key-indexed multi-dimensional observations annotation of length $\#\text{observations}$. If passing a `~numpy.ndarray`, it needs to have a structured datatype.
- **varm**  
  Key-indexed multi-dimensional variables annotation of length $\#\text{variables}$.
- **layers**  
  Key-indexed multi-dimensional arrays aligned to dimensions of $X$.
- **raw**  
  Store raw version of $X$ and $\text{var}$ as $\text{raw}$X and $\text{raw}$var.
- **dtype**  
  Data type used for storage.
- **shape**  
  Shape list ($\#\text{observations}, \#\text{variables}$). Can only be provided if $X$ is NULL.
- **filename**  
  Name of backing file. See `h5py.File`.
- **filemode**  
  Open mode of backing file. See `h5py.File`.
- **obsp**  
  Pairwise annotation of observations, a mutable mapping with array-like values.
- **varp**  
  Pairwise annotation of observations, a mutable mapping with array-like values.
- **adata**  
  An AnnData object.

## Details

AnnData stores observations (samples) of variables/features in the rows of a matrix. This is the convention of the modern classics of statistic and machine learning, the convention of dataframes both in R and Python and the established statistics and machine learning packages in Python (statsmodels, scikit-learn).

Single dimensional annotations of the observation and variables are stored in the `obs` and `var` attributes as data frames. This is intended for metrics calculated over their axes. Multi-dimensional annotations are stored in `obsm` and `varm`, which are aligned to the objects observation and variable dimensions respectively. Square matrices representing graphs are stored in `obsp` and `varp`, with both of their own dimensions aligned to their associated axis. Additional measurements across both observations and variables are stored in `layers`.

Indexing into an AnnData object can be performed by relative position with numeric indices, or by labels. To avoid ambiguity with numeric indexing into observations or variables, indexes of the AnnData object are converted to strings by the constructor.

Subsetting an AnnData object by indexing into it will also subset its elements according to the dimensions they were aligned to. This means an operation like `adata[\text{list\_of\_obs},]` will also subset `obs`, `obsm`, and `layers`.

Subsetting an AnnData object returns a view into the original object, meaning very little additional memory is used upon subsetting. This is achieved lazily, meaning that the constituent arrays are
AnnData subset on access. Copying a view causes an equivalent “real” AnnData object to be generated. Attempting to modify a view (at any attribute except X) is handled in a copy-on-modify manner, meaning the object is initialized in place. Here’s an example

```r
batch1 <- adata[adata$obs["batch"] == "batch1", ]
batch1$obs["value"] = 0 # This makes batch1 a “real” AnnData object
```

At the end of this snippet: adata was not modified, and batch1 is its own AnnData object with its own data.

Similar to Bioconductor’s ExpressionSet and scipy.sparse matrices, subsetting an AnnData object retains the dimensionality of its constituent arrays. Therefore, unlike with the classes exposed by pandas, numpy, and xarray, there is no concept of a one dimensional AnnData object. AnnDatas always have two inherent dimensions, obs and var. Additionally, maintaining the dimensionality of the AnnData object allows for consistent handling of scipy.sparse matrices and numpy arrays.

Active bindings

- **X** Data matrix of shape n_obs x n_vars.
- **filename** Name of the backing file. Change to backing mode by setting the filename of a .h5ad file.
  - Setting the filename writes the stored data to disk.
  - Setting the filename when the filename was previously another name moves the backing file from the previous file to the new file. If you want to copy the previous file, use `copy(filename='new_filename')`.
- **layers** A list-like object with values of the same dimensions as X. Layers in AnnData are inspired by loompy’s layers.
  - Overwrite the layers:
    ```r
    adata$layers <- list(spliced = spliced, unspliced = unspliced)
    ```
  - Return the layer named “unspliced”:
    ```r
    adata$layers["unspliced"]
    ```
  - Create or replace the "spliced" layer:
    ```r
    adata$layers["spliced"] = example_matrix
    ```
  - Assign the 10th column of layer "spliced" to the variable a:
    ```r
    a <- adata$layers["spliced"][, 10]
    ```
  - Delete the "spliced":
    ```r
    adata$layers["spliced"] <- NULL
    ```
  - Return layers’ names:
    ```r
    names(adata$layers)
    ```
- **T** Transpose whole object.
  - Data matrix is transposed, observations and variables are interchanged. Ignores .raw.
**is_view** TRUE if object is view of another AnnData object, FALSE otherwise.

**isbacked** TRUE if object is backed on disk, FALSE otherwise.

**n_obs** Number of observations.

**obs** One-dimensional annotation of observations (data.frame).

**obs_names** Names of observations.

**obsm** Multi-dimensional annotation of observations (matrix).

Stores for each key a two or higher-dimensional matrix with n_obs rows.

**obsp** Pairwise annotation of observations, a mutable mapping with array-like values.

Stores for each key a two or higher-dimensional matrix whose first two dimensions are of length n_obs.

**n_vars** Number of variables.

**var** One-dimensional annotation of variables (data.frame).

**var_names** Names of variables.

**varm** Multi-dimensional annotation of variables (matrix).

Stores for each key a two or higher-dimensional matrix with n_vars rows.

**varp** Pairwise annotation of variables, a mutable mapping with array-like values.

Stores for each key a two or higher-dimensional matrix whose first two dimensions are of length n_vars.

**shape** Shape of data matrix (n_obs, n_vars).

**uns** Unstructured annotation (ordered dictionary).

**raw** Store raw version of X and var as $raw$X and $raw$var.

The raw attribute is initialized with the current content of an object by setting:

```r
adata$raw = adata
```

Its content can be deleted:

```r
adata$raw <- NULL
```

Upon slicing an AnnData object along the obs (row) axis, raw is also sliced. Slicing an AnnData object along the vars (columns) axis leaves raw unaffected. Note that you can call:

```r
adata$raw[, 'orig_variable_name']$X
```

to retrieve the data associated with a variable that might have been filtered out or "compressed away" in X.

**Methods**

**Public methods:**

- `AnnDataR6$new()`
- `AnnDataR6$obs_keys()`
- `AnnDataR6$obs_names_make_unique()`
- `AnnDataR6$obsm_keys()`
- `AnnDataR6$var_keys()`
- `AnnDataR6$var_names_make_unique()`
Method `new()`: Create a new AnnData object

*Usage:*

```r
AnnDataR6$new(obj)
```

*Arguments:*

- **obj**: A Python anndata object

*Examples:*

```r
\dontrun{
  # use AnnData() instead of AnnDataR6$new()
  ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"))
  )
}
```

Method `obs_keys()`: List keys of observation annotation `obs`.

*Usage:*

```r
AnnDataR6$obs_keys()
```

*Examples:*

```r
\dontrun{
  ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2"))
  )
  ad$obs_keys()
}
```
Method obs_names_make_unique(): Makes the index unique by appending a number string to each duplicate index element: 1, 2, etc.
If a tentative name created by the algorithm already exists in the index, it tries the next integer in the sequence.
The first occurrence of a non-unique value is ignored.

Usage:
AnnDataR6$obs_names_make_unique(join = "-"

Arguments:
join The connecting string between name and integer (default: "-").

Examples:
\dontrun{
ad <- AnnData(
  X = matrix(rep(1, 6), nrow = 3),
  obs = data.frame(field = c(1, 2, 3))
)
ad$obs_names <- c("a", "a", "b")
ad$obs_names_make_unique()
ad$obs_names
}

Method obsm_keys(): List keys of observation annotation obsm.

Usage:
AnnDataR6$obsm_keys()

Examples:
\dontrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  obsm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  )
)
ad$obs_keys()
}

Method var_keys(): List keys of variable annotation var.

Usage:
AnnDataR6$var_keys()

Examples:
\dontrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"))
}
Method `var_names_make_unique()`: Makes the index unique by appending a number string to each duplicate index element: 1, 2, etc.
If a tentative name created by the algorithm already exists in the index, it tries the next integer in the sequence.
The first occurrence of a non-unique value is ignored.

Usage:
AnnDataR6$var_names_make_unique(join = "-")

Arguments:
join The connecting string between name and integer (default: "-").

Examples:
\dontrun{
ad <- AnnData(
  X = matrix(rep(1, 6), nrow = 2),
  var = data.frame(field = c(1, 2, 3))
)
ad$var_names <- c("a", "a", "b")
ad$var_names_make_unique()
ad$var_names
}

Method `varm_keys()`: List keys of variable annotation varm.

Usage:
AnnDataR6$varm_keys()

Examples:
\dontrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  )
)
ad$varm_keys()
}

Method `uns_keys()`: List keys of unstructured annotation uns.

Usage:
AnnDataR6$uns_keys()

Examples:
\dontrun{
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
}

Method \texttt{chunk\_X()}: Return a chunk of the data matrix \(X\) with random or specified indices.

\textit{Usage:}
AnnDataR6$chunk\_X(select = 1000L, replace = TRUE)

\textit{Arguments:}
- \texttt{select} Depending on the values:
  - \(1\) integer: A random chunk with \texttt{select} rows will be returned.
  - \(\text{multiple integers}\): A chunk with these indices will be returned.
- \texttt{replace} if \texttt{select} is an integer then \TRUE means random sampling of indices with replacement, \FALSE without replacement.

\textit{Examples:}
\dontrun{
ad <- AnnData(
    X = matrix(runif(10000), nrow = 50)
)

ad$chunk\_X(select = 10L) # 10 random samples
ad$chunk\_X(select = 1:3) # first 3 samples
}

Method \texttt{chunked\_X()}: Return an iterator over the rows of the data matrix \(X\).

\textit{Usage:}
AnnDataR6$chunked\_X(chunk\_size = NULL)

\textit{Arguments:}
- \texttt{chunk\_size} Row size of a single chunk.

\textit{Examples:}
\dontrun{
ad <- AnnData(
    X = matrix(runif(10000), nrow = 50)
)
ad$chunked\_X(10)
}

Method \texttt{concatenate()}: Concatenate along the observations axis.

\textit{Usage:}
AnnDataR6$concatenate(...)
Method `copy()`: Full copy, optionally on disk.

Usage:
AnnData$copy(filename = NULL)

Arguments:
filename Path to filename (default: NULL).

Examples:
\dontrun{
  ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
  )
  ad$copy()
  ad$copy("file.h5ad")
}

Method `rename_categories()`: Rename categories of annotation key in obs, var, and uns. Only supports passing a list/array-like categories argument. Besides calling self.obs[key].cat.categories = categories – similar for var - this also renames categories in unstructured annotation that uses the categorical annotation key.

Usage:
AnnData$rename_categories(key, categories)

Arguments:
key Key for observations or variables annotation.
categories New categories, the same number as the old categories.

Examples:
\dontrun{
  ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2"))
  )
  ad$rename_categories("group", c(a = "A", b = "B")) # ??
}

Method `strings_to_categoricals()`: Transform string annotations to categoricals. Only affects string annotations that lead to less categories than the total number of observations.

Usage:
AnnData$strings_to_categoricals(df = NULL)

Arguments:

df If df is NULL, modifies both obs and var, otherwise modifies df inplace.

Examples:
```r
\dontrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
)
ad$strings_to_categoricals() # ??  
}

Method to_df(): Generate shallow data frame.
The data matrix $X$ is returned as data frame, where $\text{obs\_names}$ are the rownames, and $\text{var\_names}$ the columns names.
No annotations are maintained in the returned object.
The data matrix is densified in case it is sparse.

Usage:
```
AnnDataR6$to_df(layer = NULL)
```

Arguments:
layer Key for layers

Examples:
```
\dontrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  layers = list(
    spliced = matrix(c(4, 5, 6, 7), nrow = 2),
    unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
  )
)
}
ad$to_df()
ad$to_df("unspliced")
```

Method transpose(): transpose Transpose whole object.
Data matrix is transposed, observations and variables are interchanged.
Ignores .raw.

Usage:
```
AnnDataR6$transpose()
```

Examples:
```
\dontrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"))
)
}```
Method write_csvs(): Write annotation to .csv files.

It is not possible to recover the full AnnData from these files. Use write_h5ad() for this.

Usage:
AnnDataR6$write_csvs(dirname, skip_data = TRUE, sep = ",")

Arguments:
- dirname: Name of the directory to which to export.
- skip_data: Skip the data matrix X.
- sep: Separator for the data.

Examples:
\dontrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)

ad$to_write_csvs("output")

unlink("output", recursive = TRUE)
}

Method write_h5ad(): Write .h5ad-formatted hdf5 file.

Generally, if you have sparse data that are stored as a dense matrix, you can dramatically improve performance and reduce disk space by converting to a csr_matrix.

Usage:
AnnDataR6$write_h5ad(
  filename,
  compression = NULL,
  compression_opts = NULL,
  as_dense = list()
)

Arguments:
compression  See the h5py filter pipeline. Options are "gzip", "lz4" or NULL.
compression_opts  See the h5py filter pipeline.

as_dense  Sparse in AnnData object to write as dense. Currently only supports "X" and "raw/X".

anndata  An AnnData() object

Examples:
\dontrun{
ad <- AnnData(  
  X = matrix(c(0, 1, 2, 3), nrow = 2),  
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),  
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),  
  varm = list(    
    ones = matrix(rep(1L, 10), nrow = 2),  
    rand = matrix(rnorm(6), nrow = 2),  
    zeros = matrix(rep(0L, 10), nrow = 2)  
  ),  
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))  
)

ad$write_h5ad("output.h5ad")

do.remove("output.h5ad")
}

Method write_loom(): Write .loom-formatted hdf5 file.

Usage:
AnnDataR6$write_loom(filename, write_obsm_varm = FALSE)

Arguments:
filename  The filename.
write_obsm_varm  Whether or not to also write the varm and obsm.
anndata  An AnnData() object

Examples:
\dontrun{
ad <- AnnData(  
  X = matrix(c(0, 1, 2, 3), nrow = 2),  
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),  
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),  
  varm = list(    
    ones = matrix(rep(1L, 10), nrow = 2),  
    rand = matrix(rnorm(6), nrow = 2),  
    zeros = matrix(rep(0L, 10), nrow = 2)  
  ),  
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))  
)

ad$write_loom("output.loom")
file.remove("output.loom")
}

**Method print():** Print AnnData object

*Usage:*
AnnDataR6$print(...)

*Arguments:*
... optional arguments to print method.

*Examples:*
\donttrun{
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  layers = list(
    spliced = matrix(c(4, 5, 6, 7), nrow = 2),
    unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
  ),
  obsm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)

dontrun{
ad$print()
print(ad)
}

**Method .set_py_object():** Set internal Python object

*Usage:*
AnnDataR6$.set_py_object(obj)

*Arguments:*
obj A python anndata object

**Method .get_py_object():** Get internal Python object

*Usage:*
AnnDataR6$.get_py_object()
Active bindings

- **X** Data matrix of shape \( n_{\text{obs}} \times n_{\text{vars}} \).
- **n_obs** Number of observations.
- **obs_names** Names of observations.
- **n_vars** Number of variables.
- **var** One-dimensional annotation of variables (data.frame).
- **var_names** Names of variables.
- **varm** Multi-dimensional annotation of variables (matrix).
  - Stores for each key a two or higher-dimensional matrix with \( n_{\text{var}} \) rows.
- **shape** Shape of data matrix \((n_{\text{obs}}, n_{\text{vars}})\).

Methods

**Public methods:**

- `RawR6$new()`
- `RawR6$copy()`
- `RawR6$to_adata()`
- `RawR6$print()`
- `RawR6$.set_py_object()`
- `RawR6$.get_py_object()`

**Method** `new()`: Create a new Raw object

**Usage:**

```
RawR6$new(obj)
```

**Arguments:**

- **obj** A Python Raw object

**Method** `copy()`: Full copy, optionally on disk.

**Usage:**

```
RawR6$copy()
```

**Arguments:**

- **filename** Path to filename (default: NULL).

**Examples:**

```r
\dontrun{
  ad <- AnnData(  
    X = matrix(c(0, 1, 2, 3), nrow = 2)
  )
  ad$copy()
  ad$copy("file.h5ad")
}
```

**Method** `to_adata()`: Create a full AnnData object
Usage:
RawR6$to_adata()

Examples:

```r
\dontrun{
  ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    layers = list(
      spliced = matrix(c(4, 5, 6, 7), nrow = 2),
      unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
    ),
    obsm = list(
      ones = matrix(rep(1L, 10), nrow = 2),
      rand = matrix(rnorm(6), nrow = 2),
      zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    varm = list(
      ones = matrix(rep(1L, 10), nrow = 2),
      rand = matrix(rnorm(6), nrow = 2),
      zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
  )
  ad$raw <- ad

  ad$raw$to_adata()
}
```

Method print(): Print Raw object

Usage:
RawR6$print(…)

Arguments:
… optional arguments to print method.

Examples:

```r
\dontrun{
  ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    layers = list(
      spliced = matrix(c(4, 5, 6, 7), nrow = 2),
      unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
    ),
    obsm = list(
      ones = matrix(rep(1L, 10), nrow = 2),
      rand = matrix(rnorm(6), nrow = 2),
      zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    varm = list(
      ones = matrix(rep(1L, 10), nrow = 2),
      rand = matrix(rnorm(6), nrow = 2),
      zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
  )
  ad$raw <- ad
```
library(reticulate)
scc <- import("scanpy")
sc$pp$normalize_per_cell(ad)

ad[]
ad$raw[]

ad$print()
print(ad)

Method .set_py_object(): Set internal Python object
Usage:
RawR6$.set_py_object(obj)
Arguments:
obj A Python Raw object

Method .get_py_object(): Get internal Python object
Usage:
RawR6$.get_py_object()

See Also
read_h5ad() read_csv() read_excel() read_hdf() read_loom() read_mtx() read_text()
read_umi_tools() write_h5ad() write_csvs() write_loom()

Examples
## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"),
    layers = list(
      spliced = matrix(c(4, 5, 6, 7), nrow = 2),
      unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
    ),
    obsm = list(
      ones = matrix(rep(1L, 10), nrow = 2),
      rand = matrix(rnorm(6), nrow = 2),
      zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    varm = list(
      ones = matrix(rep(1L, 10), nrow = 2),
      rand = matrix(rnorm(6), nrow = 2),
      zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
value <- matrix(c(1, 2, 3, 4), nrow = 2)
ad$X <- value
ad$X

ad$layers
ad$layers["spliced"]
ad$layers["test"] <- value
ad$layers

ad$to.df()
ad$uns

as.matrix(ad)
as.matrix(ad, layer = "unspliced")
dim(ad)
rownames(ad)
colnames(ad)

## End(Not run)

## Method `AnnDataR6$new`

## Not run:
# use AnnData() instead of AnnDataR6$new()
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"))
)

## End(Not run)

## Method `AnnDataR6$obs.keys`

## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2"))
)
ad$obs.keys()

## End(Not run)

## Method `AnnDataR6$obs.names.make.unique`

## Not run:
ad <- AnnData(
    X = matrix(rep(1, 6), nrow = 3),
    obs = data.frame(field = c(1, 2, 3))
)
ad$obs_names <- c("a", "a", "b")
ad$obs_names_make_unique()
ad$obs_names

# End(Not run)

# Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    obsm = list(
        ones = matrix(rep(1L, 10), nrow = 2),
        rand = matrix(rnorm(6), nrow = 2),
        zeros = matrix(rep(0L, 10), nrow = 2)
    )
)
ad$obs_keys()

# End(Not run)

# Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"))
)
ad$var_keys()

# End(Not run)

# Not run:
ad <- AnnData(
    X = matrix(rep(1, 6), nrow = 2),
    var = data.frame(field = c(1, 2, 3))
)
ad$var_names <- c("a", "a", "b")
ad$var_names_make_unique()
ad$var_names
## Method `AnnDataR6$varm_keys`
### Not run:
```r
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  )
)
ad$varm_keys()
```
```r
## Not run:
ad <- AnnData(
  X = matrix(runif(10000), nrow = 50)
)
ad$chunked_X(10)
## End(Not run)

## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2)
  ad$copy()
ad$copy("file.h5ad")
## End(Not run)

## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2"))
)ad$rename_categories("group", c(a = "A", b = "B")) # ??
## End(Not run)

## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"))
)ad$strings_to_categoricals() # ??
## End(Not run)

## Not run:

## Method `AnnDataR6$copy`
## -------------------------------------------

## Method `AnnDataR6$rename_categories`
## -------------------------------------------

## Method `AnnDataR6$strings_to_categoricals`
## -------------------------------------------

## Method `AnnDataR6$to_df`
## -------------------------------------------

## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  layers = list(
    spliced = matrix(c(4, 5, 6, 7), nrow = 2),
    unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
  )
)

ad$to_df()
ad$to_df("unspliced")

# End(Not run)

# ----------------------------------
# Method 'AnnDataR6$transpose'
# ----------------------------------

# Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2"))
)
ad$transpose()

# End(Not run)

# ----------------------------------
# Method 'AnnDataR6$write_csvs'
# ----------------------------------

# Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$to_write_csvs("output")
unlink("output", recursive = TRUE)

# End(Not run)
```r
## Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    varm = list(
        ones = matrix(rep(1L, 10), nrow = 2),
        rand = matrix(rnorm(6), nrow = 2),
        zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$write_h5ad("output.h5ad")
file.remove("output.h5ad")
## End(Not run)

## Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    varm = list(
        ones = matrix(rep(1L, 10), nrow = 2),
        rand = matrix(rnorm(6), nrow = 2),
        zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$write_loom("output.loom")
file.remove("output.loom")
## End(Not run)

## Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    varm = list(
        ones = matrix(rep(1L, 10), nrow = 2),
        rand = matrix(rnorm(6), nrow = 2),
        zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$print()
```

obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
layers = list(
    spliced = matrix(c(4, 5, 6, 7), nrow = 2),
    unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
),
obsm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
),
varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
),
uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$print()
print(ad)

## End(Not run)
## Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
layers = list(
    spliced = matrix(c(4, 5, 6, 7), nrow = 2),
    unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
),
obsm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
),
varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
),
uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$raw <- ad

library(reticulate)
sc <- import("scanpy")
sc$pp$normalize_per_cell(ad)
ad[]
ad$raw[]}
```
## End(Not run)

## Method `RawR6$copy`

## Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2)
)
ad$copy()
ad$copy("file.h5ad")

## End(Not run)

## Method `RawR6$to_adata`

## Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    layers = list(
        spliced = matrix(c(4, 5, 6, 7), nrow = 2),
        unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
    ),
    obsm = list(
        ones = matrix(rep(1L, 10), nrow = 2),
        rand = matrix(rnorm(6), nrow = 2),
        zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    unsupervised = TRUE
)
ad$raw <- ad
ad$raw$to_adata()

## End(Not run)

## Method `RawR6$print`

## Not run:
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    layers = list(
        spliced = matrix(c(4, 5, 6, 7), nrow = 2),
        unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
    ),
    obsm = list(
        ones = matrix(rep(1L, 10), nrow = 2),
        rand = matrix(rnorm(6), nrow = 2),
        zeros = matrix(rep(0L, 10), nrow = 2)
    ),
    unsupervised = TRUE
)
ad$raw <- ad
ad$raw$to_adata()
```

varm = list(  
    ones = matrix(rep(1L, 10), nrow = 2),  
    rand = matrix(rnorm(6), nrow = 2),  
    zeros = matrix(rep(0L, 10), nrow = 2)  
  ),  
uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$raw <- ad

library(reticulate)
s <- import("scanpy")
s$pp$normalize_per_cell(ad)

ad[]
ad$raw[]
ad$print()
print(ad)

## End(Not run)

---

**concat**

Concatenates AnnData objects along an axis.

**Usage**

```r
concat(
    adatas,
    axis = 0L,
    join = "inner",
    merge = NULL,
    uns_merge = NULL,
    label = NULL,
    keys = NULL,
    index_unique = NULL,
    fill_value = NULL,
    pairwise = FALSE
)
```

**Arguments**

- **adatas**: The objects to be concatenated. If a Mapping is passed, keys are used for the keys argument and values are concatenated.
- **axis**: Which axis to concatenate along.
join  How to align values when concatenating. If "outer", the union of the other axis is taken. If "inner", the intersection. See concatenation for more.
merge How elements not aligned to the axis being concatenated along are selected. Currently implemented strategies include: * NULL: No elements are kept. * "same": Elements that are the same in each of the objects. * "unique": Elements for which there is only one possible value. * "first": The first element seen at each position. * "only": Elements that show up in only one of the objects.
uns_merge How the elements of .uns are selected. Uses the same set of strategies as the merge argument, except applied recursively.
label Column in axis annotation (i.e. .obs or .var) to place batch information in. If it’s NULL, no column is added.
keys Names for each object being added. These values are used for column values for label or appended to the index if index_unique is not NULL. Defaults to incrementing integer labels.
index_unique Whether to make the index unique by using the keys. If provided, this is the delimeter between "orig_idxindex_uniquekey". When NULL, the original indices are kept.
fill_value When join="outer", this is the value that will be used to fill the introduced indices. By default, sparse arrays are padded with zeros, while dense arrays and DataFrames are padded with missing values.
pairwise Whether pairwise elements along the concatenated dimension should be included. This is FALSE by default, since the resulting arrays are often not meaningful.

Details
See the concatenation section in the docs for a more in-depth description.

warning: This function is marked as experimental for the 0.7 release series, and will supercede the AnnData$concatenate() method in future releases.

warning: If you use join=’outer’ this fills 0s for sparse data when variables are absent in a batch. Use this with care. Dense data is filled with NaN.

Examples

```r
## Not run:
# Preparing example objects
a <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2, byrow = TRUE),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  uns = list(
    )
```
a = 1,
b = 2,
c = list(
    c.a = 3,
    c.b = 4
)
)b <- AnnData(
    X = matrix(c(4, 5, 6, 7, 8, 9), nrow = 2, byrow = TRUE),
    obs = data.frame(group = c("b", "c"), row.names = c("s3", "s4")),
    var = data.frame(type = c(1L, 2L, 3L), row.names = c("var1", "var2", "var3")),
    varm = list(
        ones = matrix(rep(1L, 15), nrow = 3),
        rand = matrix(rnorm(15), nrow = 3)
    ),
    uns = list(
        a = 1,
        b = 3,
        c = list(
            c.a = 3
        )
    )
)c <- AnnData(
    X = matrix(c(10, 11, 12, 13), nrow = 2, byrow = TRUE),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(3L, 4L), row.names = c("var3", "var4")),
    uns = list(
        a = 1,
        b = 4,
        c = list(
            c.a = 3,
            c.b = 4,
            c.c = 5
        )
    )
)

# Concatenating along different axes
concat(list(a, b))$to_df()
concat(list(a, c), axis = 1L)$to_df()

# Inner and outer joins
inner <- concat(list(a, b))
inner
inner$obs_names
inner$var_names

outer <- concat(list(a, b), join = "outer")
outer
outer$var_names
outer$to_df()

# Keeping track of source objects
concat(list(a = a, b = b), label = "batch")$obs
concat(list(a, b), label = "batch", keys = c("a", "b"))$obs
concat(list(a = a, b = b), index_unique = "-")$obs

# Combining values not aligned to axis of concatenation
concat(list(a, b), merge = "same")
concat(list(a, b), merge = "unique")
concat(list(a, b), merge = "first")
concat(list(a, b), merge = "only")

# The same merge strategies can be used for elements in .uns
concat(list(a, b, c), uns_merge = "same")$uns
concat(list(a, b, c), uns_merge = "unique")$uns
concat(list(a, b, c), uns_merge = "first")$uns
concat(list(a, b, c), uns_merge = "only")$uns

## End(Not run)

---

## dimnames.AnnDataR6  
**AnnData Helpers**

**Description**
AnnData Helpers

**Usage**

```
## S3 method for class 'AnnDataR6'
dimnames(x)

## S3 method for class 'AnnDataR6'
dim(x)

## S3 method for class 'AnnDataR6'
as.data.frame(x, row.names = NULL, optional = FALSE, layer = NULL, ...)

## S3 method for class 'AnnDataR6'
as.matrix(x, layer = NULL, ...)

## S3 method for class 'AnnDataR6'
r_to_py(x, convert = FALSE)

## S3 method for class 'anndata._core.anndata.AnnData'
py_to_r(x)
```
## S3 method for class 'AnnDataR6'
x[oidx, vidx]

## S3 method for class 'AnnDataR6'
t(x)

### Arguments

- **x**: An AnnData object.
- **row.names**: Not used.
- **optional**: Not used.
- **layer**: An AnnData layer. If NULL, will use ad$X, otherwise ad$layers[layer].
- **...**: Parameters passed to the underlying function.
- **convert**: Not used.
- **oidx**: Observation indices
- **vidx**: Variable indices

### Examples

```r
## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3, 4, 5), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L, 3L), row.names = c("var1", "var2", "var3"),
    layers = list(
      spliced = matrix(c(4, 5, 6, 7, 8), nrow = 2),
      unspliced = matrix(c(8, 9, 10, 11, 12), nrow = 2)
    ),
  obsm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  varm = list(
    ones = matrix(rep(1L, 12), nrow = 3),
    rand = matrix(rnorm(6), nrow = 3),
    zeros = matrix(rep(0L, 12), nrow = 3)
  ),
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)

dimnames(ad)
dim(ad)
as.data.frame(ad)
as.data.frame(ad, layer = "unspliced")
as.matrix(ad)
as.matrix(ad, layer = "unspliced")
ad[2,, drop=FALSE]
ad[,-1]
```
ad[,c("var1", "var2")]
## End(Not run)

### Description

Raw Helpers

### Usage

```r
## S3 method for class 'RawR6'
dimnames(x)

## S3 method for class 'RawR6'
dim(x)

## S3 method for class 'RawR6'
as.matrix(x, ...)

## S3 method for class 'RawR6'
r_to_py(x, convert = FALSE)

## S3 method for class 'anndata._core.raw.Raw'
py_to_r(x)

## S3 method for class 'RawR6'
x[...]
```

### Arguments

- `x` An AnnData object.
- `...` Parameters passed to the underlying function.
- `convert` Not used.

### Examples

```r
## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3, 4, 5), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L, 3L), row.names = c("var1", "var2", "var3")),
  layers = list(
    spliced = matrix(c(4, 5, 6, 7, 8, 9), nrow = 2),
    unspliced = matrix(c(8, 9, 10, 11, 12, 13), nrow = 2)
  ),
)
install_anndata

```r
obsm = list(
  ones = matrix(rep(1L, 10), nrow = 2),
  rand = matrix(rnorm(6), nrow = 2),
  zeros = matrix(rep(0L, 10), nrow = 2)
),
varm = list(
  ones = matrix(rep(1L, 12), nrow = 3),
  rand = matrix(rnorm(6), nrow = 3),
  zeros = matrix(rep(0L, 12), nrow = 3)
),
uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
ad$raw <- ad
dimnames(ad$raw)
dim(ad$raw)
as.matrix(ad$raw)
ad$raw[2,,drop=FALSE]
ad$raw[,-1]
ad$raw[,c("var1", "var2")]
## End(Not run)
```

### Description

Needs to be run after installing the anndata R package.

### Usage

```r
install_anndata(method = "auto", conda = "auto")
```

### Arguments

- **method**: Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.

- **conda**: The path to a conda executable. Use "auto" to allow reticulate to automatically find an appropriate conda binary. See Finding Conda for more details.

### Examples

```r
## Not run:
reticulate::conda_install()
install_anndata()
## End(Not run)
```
Layers

Create a Layers object

Description
Create a Layers object

Usage
Layers(parent, vals = NULL)

Arguments
- parent: An AnnData object.
- vals: A named list of matrices with the same dimensions as parent.

Active bindings
- parent: Reference to parent AnnData view

Methods
Public methods:
- LayersR6$new()
- LayersR6$print()
- LayersR6$get()
- LayersR6$set()
- LayersR6$del()
- LayersR6$keys()
- LayersR6$length()
- LayersR6$.set_py_object()
- LayersR6$.get_py_object()

Method new(): Create a new Layers object
Usage:
LayersR6$new(obj)
Arguments:
- obj: A Python Layers object

Method print(): Print Layers object
Usage:
LayersR6$print(...)
Arguments:
- ...: Optional arguments to print method.
Examples:
\dontrun{
ad <- AnnData(
    X = matrix(c(0, 1, 2, 3), nrow = 2),
    obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
    var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
    layers = list(
        spliced = matrix(c(4, 5, 6, 7), nrow = 2),
        unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
    )
)
)

print(ad$layers)
}

Method get(): Get a layer

Usage:
LayersR6$get(name)

Arguments:
name Name of the layer

Method set(): Set a layer

Usage:
LayersR6$set(name, value)

Arguments:
name Name of the layer
value A matrix

Method del(): Delete a layer

Usage:
LayersR6$del(name)

Arguments:
name Name of the layer

Method keys(): Get the names of the layers

Usage:
LayersR6$keys()

Method length(): Get the number of layers

Usage:
LayersR6$length()

Method .set_py_object(): Set internal Python object

Usage:
LayersR6$.set_py_object(obj)
Arguments:
obj A Python layers object

Method .get_py_object(): Get internal Python object

Usage:
LayersR6$.get_py_object()

Examples

## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
layers = list(
  spliced = matrix(c(4, 5, 6, 7), nrow = 2),
  unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
)
)
ad$layers["spliced"]
ad$layers["test"] <- matrix(c(1, 3, 5, 7), nrow = 2)

length(ad$layers)
names(ad$layers)

## End(Not run)

## -------------------------------
## Method 'LayersR6$print'
## -------------------------------

## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2),
obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
layers = list(
  spliced = matrix(c(4, 5, 6, 7), nrow = 2),
  unspliced = matrix(c(8, 9, 10, 11), nrow = 2)
)
)

print(ad$layers)

## End(Not run)
Description
Layers Helpers

Usage
```r
## S3 method for class 'LayersR6'
names(x)

## S3 method for class 'LayersR6'
length(x)

## S3 method for class 'LayersR6'
r_to_py(x, convert = FALSE)

## S3 method for class 'anndata._core.aligned_mapping.Layers'
py_to_r(x)

## S3 method for class 'LayersR6'
x[name]

## S3 replacement method for class 'LayersR6'
x[name] <- value

## S3 method for class 'LayersR6'
x[[name]]

## S3 replacement method for class 'LayersR6'
x[[[name]]] <- value
```

Arguments
- **x**: An AnnData object.
- **convert**: Not used.
- **name**: Name of the layer.
- **value**: Replacement value.

Examples
```r
## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3, 4, 5), nrow = 2),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L, 3L), row.names = c("var1", "var2", "var3")),
  layers = list(
    spliced = matrix(c(4, 5, 6, 7, 8, 9), nrow = 2),
    unspliced = matrix(c(8, 9, 10, 11, 12, 13), nrow = 2)
  )
)
```
r-py-conversion

Convert between Python and R objects

Description

Convert between Python and R objects

Usage

## S3 replacement method for class 'collections.abc.MutableMapping'

x[[name]] <- value

## S3 method for class 'collections.abc.Mapping'

x[name]

## S3 replacement method for class 'collections.abc.MutableMapping'

x[=] <- value

## S3 method for class 'collections.abc.Mapping'

x[name]

## S3 method for class 'collections.abc.Mapping'

names(x)

## S3 method for class 'collections.abc.Set'

py_to_r(x)

## S3 method for class 'pandas.core.indexes.base.Index'

py_to_r(x)

## S3 method for class 'collections.abc.KeysView'

py_to_r(x)

## S3 method for class 'collections.abc.Mapping'

py_to_r(x)

Arguments

x A Python object.
name A name
value A value
Value

An R object, as converted from the Python object.

Description

Read .csv file.

Usage

read_csv(
    filename,
    delimiter = "",
    first_column_names = NULL,
    dtype = "float32"
)

Arguments

filename Data file.

delimiter Delimiter that separates data within text file. If NULL, will split at arbitrary number of white spaces, which is different from enforcing splitting at single white space '. '.

first_column_names Assume the first column stores row names.

dtype Numpy data type.

Details

Same as read_text() but with default delimiter ', '.

Examples

## Not run:
ad <- read_csv("matrix.csv")

## End(Not run)
read_excel

Description
Read .xlsx (Excel) file.

Usage
read_excel(filename, sheet, dtype = "float32")

Arguments
filename  File name to read from.
sheet     Name of sheet in Excel file.
dtype     Numpy data type.

Details
Assumes that the first columns stores the row names and the first row the column names.

Examples
## Not run:
ad <- read_excel("spreadsheet.xls")
## End(Not run)

read_h5ad

Description
Read .h5ad-formatted hdf5 file.

Usage
read_h5ad(filename, backed = NULL)

Arguments
filename  File name of data file.
backed    If 'r', load ~anndata.AnnData in backed mode instead of fully loading it into memory (memory mode). If you want to modify backed attributes of the AnnData object, you need to choose 'r+'.
Examples

```r
## Not run:
ad <- read_h5ad("example_formats/pbmc_1k_protein_v3_processed.h5ad")
## End(Not run)
```

Description

Read .h5 (hdf5) file.

Usage

```r
read_hdf(filename, key)
```

Arguments

- `filename`: Filename of data file.
- `key`: Name of dataset in the file.

Details

Note: Also looks for fields `row_names` and `col_names`.

Examples

```r
## Not run:
ad <- read_hdf("file.h5")
## End(Not run)
```

Description

Read .loom-formatted hdf5 file.
Usage

```r
read_loom(
  filename,
  sparse = TRUE,
  cleanup = FALSE,
  X_name = "spliced",
  obs_names = "CellID",
  obsm_names = NULL,
  var_names = "Gene",
  varm_names = NULL,
  dtype = "float32",
  ...
)
```

Arguments

- **filename**: The filename.
- **sparse**: Whether to read the data matrix as sparse.
- **cleanup**: Whether to collapse all obs/var fields that only store one unique value into `.uns['loom-']`.
- **X_name**: Loompy key with which the data matrix `AnnData.X` is initialized.
- **obs_names**: Loompy key where the observation/cell names are stored.
- **obsm_names**: Loompy keys which will be constructed into observation matrices
- **var_names**: Loompy key where the variable/gene names are stored.
- **varm_names**: Loompy keys which will be constructed into variable matrices
- **dtype**: Numpy data type.
- **...**: Arguments to loompy.connect

Details

This reads the whole file into memory. Beware that you have to explicitly state when you want to read the file as sparse data.

Examples

```r
## Not run:
ad <- read_loom("dataset.loom")
## End(Not run)
```
read_mtx

Description
Read .mtx file.

Usage
read_mtx(filename, dtype = "float32")

Arguments
filename The filename.
dtype Numpy data type.

Examples
## Not run:
ad <- read_mtx("matrix.mtx")
## End(Not run)

read_text

Description
Read .txt, .tab, .data (text) file.

Usage
read_text(
    filename,
    delimiter = NULL,
    first_column_names = NULL,
    dtype = "float32"
)

Arguments
filename Data file, filename or stream.
delimiter Delimiter that separates data within text file. If NULL, will split at arbitrary number of white spaces, which is different from enforcing splitting at single white space "."
first_column_names Assume the first column stores row names.
dtype Numpy data type.
Details

Same as `read_csv()` but with default delimiter `NULL`.

Examples

```r
## Not run:
ad <- read_text("matrix.tab")
## End(Not run)
```

---

**read_umi_tools**

**Description**

Read a gzipped condensed count matrix from umi_tools.

**Usage**

```r
read_umi_tools(filename, dtype = "float32")
```

**Arguments**

- **filename**: File name to read from.
- **dtype**: Numpy data type.

**Examples**

```r
## Not run:
ad <- read_umi_tools("...")
## End(Not run)
```

---

**write_csvs**

**Description**

Write annotation to .csv files.

It is not possible to recover the full AnnData from these files. Use `write_h5ad()` for this.

**Usage**

```r
write_csvs(anndata, dirname, skip_data = TRUE, sep = ",")
```
Arguments

- `anndata`: An `AnnData()` object
- `dirname`: Name of the directory to which to export.
- `skip_data`: Skip the data matrix $X$.
- `sep`: Separator for the data

Examples

```r
## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2, byrow = TRUE),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
write_csvs(ad, "output")
unlink("output", recursive = TRUE)
## End(Not run)
```

Description

Generally, if you have sparse data that are stored as a dense matrix, you can dramatically improve performance and reduce disk space by converting to a csr_matrix:

Usage

```r
write_h5ad(
  anndata, filename,
  compression = NULL,
  compression_opts = NULL,
  as_dense = list()
)
```
Arguments

anndata An `AnnData()` object
filename Filename of data file. Defaults to backing file.
compression See the h5py filter pipeline. Options are "gzip", "lz4" or NULL.
compression_opts See the h5py filter pipeline.
as_dense Sparse in AnnData object to write as dense. Currently only supports "X" and "raw/X".

Examples

```r
## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2, byrow = TRUE),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
write_h5ad(ad, "output.h5ad")
file.remove("output.h5ad")
## End(Not run)
```

write_loom Write .loom-formatted hdf5 file.

Description
Write .loom-formatted hdf5 file.

Usage

```r
write_loom(anndata, filename, write_obsm_varm = FALSE)
```

Arguments

anndata An `AnnData()` object
filename The filename.
write_obsm_varm Whether or not to also write the varm and obsm.
Examples

```r
## Not run:
ad <- AnnData(
  X = matrix(c(0, 1, 2, 3), nrow = 2, byrow = TRUE),
  obs = data.frame(group = c("a", "b"), row.names = c("s1", "s2")),
  var = data.frame(type = c(1L, 2L), row.names = c("var1", "var2")),
  varm = list(
    ones = matrix(rep(1L, 10), nrow = 2),
    rand = matrix(rnorm(6), nrow = 2),
    zeros = matrix(rep(0L, 10), nrow = 2)
  ),
  uns = list(a = 1, b = 2, c = list(c.a = 3, c.b = 4))
)
write_loom(ad, "output.loom")
file.remove("output.loom")
## End(Not run)
```
Index

[.AnnDataR6 (dimnames.AnnDataR6), 32
[.LayersR6 (names.LayersR6), 38
[.RawR6 (dimnames.RawR6), 34
[.collections.abc.Mapping
  (r-py-conversion), 40
[-.LayersR6 (names.LayersR6), 38
[-.collections.abc.MutableMapping
  (r-py-conversion), 40
[[.LayersR6 (names.LayersR6), 38
[[.collections.abc.Mapping
  (r-py-conversion), 40
[[-.LayersR6 (names.LayersR6), 38
[[-.collections.abc.MutableMapping
  (r-py-conversion), 40

all.equal.AnnDataR6, 4
all.equal.LayersR6, 5
AnnData, 5
anndata (anndata-package), 2
AnnData(), 2, 15, 16, 47, 48
anndata-package, 2
AnnDataR6 (AnnData), 5
as.data.frame.AnnDataR6
  (dimnames.AnnDataR6), 32
as.matrix.AnnDataR6
  (dimnames.AnnDataR6), 32
as.matrix.RawR6 (dimnames.RawR6), 34
concat, 29
concat(), 3

dim.AnnDataR6 (dimnames.AnnDataR6), 32
dim.RawR6 (dimnames.RawR6), 34
dimnames.AnnDataR6, 32
dimnames.RawR6, 34

install_anndata, 35
install_anndata(), 3

Layers, 36
LayersR6 (Layers), 36

length.LayersR6 (names.LayersR6), 38
names.collections.abc.Mapping
  (r-py-conversion), 40
names.LayersR6, 38
py_to_r.anndata._core.aligned_mapping.Layers
  (names.LayersR6), 38
py_to_r.anndata._core.anndata.AnnData
  (dimnames.AnnDataR6), 32
py_to_r.anndata._core.raw.Raw
  (dimnames.RawR6), 34
py_to_r.collections.abc.KeysView
  (r-py-conversion), 40
py_to_r.collections.abc.Mapping
  (r-py-conversion), 40
py_to_r.collections.abc.Set
  (r-py-conversion), 40
py_to_r.pandas.core.indexes.base.Index
  (r-py-conversion), 40
r-py-conversion, 40
r_to_py.AnnDataR6 (dimnames.AnnDataR6), 32
r_to_py.LayersR6 (names.LayersR6), 38
r_to_py.RawR6 (dimnames.RawR6), 34
Raw (AnnData), 5
RawR6 (AnnData), 5
read_csv, 41
read_csv(), 3, 20, 46
read_excel, 42
read_excel(), 3, 20
read_h5ad, 42
read_h5ad(), 3, 20
read_hdf, 43
read_hdf(), 3, 20
read_loom, 43
read_loom(), 3, 20
read_mtx, 45
read_mtx(), 3, 20
INDEX

read_text, 45
read_text(), 3, 20, 41
read_umi_tools, 46
read_umi_tools(), 3, 20

t.AnnDataR6(dimnames.AnnDataR6), 32

write_csvs, 46
write_csvs(), 3, 20
write_h5ad, 47
write_h5ad(), 3, 15, 20, 46
write_loom, 48
write_loom(), 3, 20