Package 'scMultiome'

May 2, 2024

Title Collection of Public Single-Cell Multiome (scATAC + scRNAseq) Datasets

Version 1.5.0

Description Single cell multiome data, containing chromatin accessibility (scATAC-seq) and gene expression (scRNA-seq) information analyzed with the ArchR package and presented as MultiAssayExperiment objects.

License CC BY-SA 4.0

Depends AnnotationHub, ExperimentHub (>= 2.8.1), MultiAssayExperiment, SingleCellExperiment, SummarizedExperiment

Imports AzureStor, DelayedArray, GenomicRanges, HDF5Array, S4Vectors, checkmate, methods, rhdf5

Suggests BiocGenerics, IRanges, Matrix, knitr, rmarkdown, rstudioapi, testthat (>= 3.0.0), devtools, BiocStyle, ExperimentHubData

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

biocViews PackageTypeData, ExperimentHub, SingleCellData, ExpressionData, SequencingData, Homo_sapiens_Data, CellCulture, Tissue, GEO

git_url https://git.bioconductor.org/packages/scMultiome

git_branch devel

git_last_commit c30ef4a

git_last_commit_date 2024-04-30

Repository Bioconductor 3.20

Date/Publication 2024-05-02

Author Xiaosai Yao [cre, aut] (<https://orcid.org/0000-0001-9729-0726>),
Aleksander Chlebowski [aut],
Aaron Lun [aut],

Shiqi Xie [ctb],
Tomasz Włodarczyk [aut]

Maintainer Xiaosai Yao <xiaosai.yao@gmail.com>

Contents

`scMultiome-package` *scMultiome: Collection of Public Single-Cell Multiome (scATAC + scRNAseq) Datasets*

Description

Single cell multiome data, containing chromatin accessibility (scATAC-seq) and gene expression (scRNA-seq) information analyzed with the ArchR package and presented as MultiAssayExperiment objects.

Details

Single cell multiome data sets, paired and unpaired, were analyzed with the `ArchR` package in order to obtain epiregulons. `ArchR` projects were converted to `MultiAssayExperiment` objects.

The creation of all datasets is described in detail in respective help files. Run `listDatasets()` to view a list of available data sets or see `Datasets` below. See `?<DATASET_NAME>` for details on particular data sets, e.g. `?prostateENZ`.

Datasets

- **colonHealthy**: Single-cell analysis of samples from healthy human colon
- **hematopoiesis**: scATAC-seq and unpaired scRNA-seq of hematopoetic cells
- **prostateENZ**: LNCaP Cells Treated with Enzalutamide
- **reprogramSeq**: Reprogram-seq of LNCaP cells
- **tfBinding_hg19_atlas**: TF Binding Info hg19 (ChIP-Atlas and ENCODE)
- **tfBinding_hg19_atlas.sample**: TF Binding Info hg19 by sample (ChIP-Atlas)
- **tfBinding_hg19_atlas.tissue**: TF Binding Info hg19 by tissue (ChIP-Atlas)
- **tfBinding_hg19_cistrome**: TF Binding Info hg19 (CistromeDB and ENCODE)
- **tfBinding_hg19_encode.sample**: TF Binding Info hg19 by sample (ENCODE)
- **tfBinding_hg38_atlas**: TF Binding Info hg38 (ChIP-Atlas and ENCODE)
- **tfBinding_hg38_atlas.sample**: TF Binding Info hg38 by sample (ChIP-Atlas)
- **tfBinding_hg38_atlas.tissue**: TF Binding Info hg38 by tissue (ChIP-Atlas)
- **tfBinding_hg38_cistrome**: TF Binding Info hg38 (CistromeDB and ENCODE)
- **tfBinding_hg38_encode.sample**: TF Binding Info hg38 by sample (ENCODE)
- **tfBinding_mm10_atlas**: TF Binding Info mm10 (ChIP-Atlas and ENCODE)
- **tfBinding_mm10_atlas.sample**: TF Binding Info mm10 by sample (ChIP-Atlas)
- **tfBinding_mm10_atlas.tissue**: TF Binding Info mm10 by tissue (ChIP-Atlas)
- **tfBinding_mm10_cistrome**: TF Binding Info mm10 (CistromeDB and ENCODE)
- **tfBinding_mm10_encode.sample**: TF Binding Info mm10 by sample (ENCODE)
- **human_pwms_v2**: TF motifs human
- **mouse_pwms_v2**: TF motifs mouse

Author(s)

Maintainer: Xiaosai Yao <xiaosai.yao@gmail.com> (ORCID)

Authors:

- Aleksander Chlebowski <aleksander.chlebowski@contractors.roche.com>
- Aaron Lun <lun.aaron@gene.com>
- Tomasz Włodarczyk <tomasz.wlodarczyk@contractors.roche.com>

Other contributors:

- Shiqi Xie <xie.shiqi@gene.com> [contributor]

assertHDF5 *check if a file is a valid HDF5 file*

Description

Check if a file path argument points to an non-corrupt HDF5 file.

Usage

```
assertHDF5(path)
```

Arguments

path	path to file to test

Details

Compares the first 8 bytes of a file to those of the standard HDF5 file header.

Value

Returns invisible path if check is successful, otherwise signals an error.

Further development

The HDF5 file header contains 8 bytes, which hold specific meanings. Currently the function only tests that the header of the file specified by path is identical to a healthy HDF5 file and signals a general error if that is not the case. Reporting specific types of corruption can be implemented.

Author(s)

Aleksander Chlebowski

References

http://web.ics.purdue.edu/~aai/HDF5/html/H5.format.html#BootBlock

Examples

```
fileName1 <- tempfile(fileext = ".h5")
rhdf5::h5createFile(fileName1)
rhdf5::h5write(mtcars, fileName1, "mtcars")
rhdf5::h5closeAll()
fileName2 <-  tempfile(fileext = ".csv")
write.csv(mtcars, fileName2)

assertHDF5(fileName1)   # passes
## Not run:
assertHDF5(fileName2)   # fails
```

```
## End(Not run)
```

`colonHealthy` *Single-cell analysis of samples from healthy human colon*

Description

ATACseq and RNAseq data obtained by the colon tissues analysis. Samples were collected from adult human donors.

Usage

```
colonHealthy(
  metadata = FALSE,
  experiments = c("TileMatrix", "GeneIntegrationMatrix", "GeneScoreMatrix",
    "MotifMatrix", "PeakMatrix")
)
```

Arguments

`metadata`	logical flag specifying whether to return data or metadata only
`experiments`	character vector of matrices to return; see `Format`

Format

`MultiAssayExperiment` obtained from an `ArchR` project. Annotated with the Hg38 genome build. Contains the following experiments:

- **TileMatrix**: SingleCellExperiment with 6062095 rows and 59231 columns
- **GeneIntegrationMatrix**: SingleCellExperiment with 19020 rows and 59231 columns
- **GeneScoreMatrix**: SingleCellExperiment with 24919 rows and 59231 columns
- **MotifMatrix**: SingleCellExperiment with 870 rows and 59231 columns
- **PeakMatrix**: SingleCellExperiment with 406946 rows and 59231 columns

Value

`MultiAssayExperiment` made up of `SingleCellExperiments` with assays stored as `DelayedMatrix` objects. If `metadata = TRUE`, an `ExperimentHub` object listing this data set's metadata.

Data preparation

scATAC data was downloaded from Gene Expression Omnibus (acc. no. GSE165659) and analyzed with SingleCell ATAC - 10X pipeline v2.0.0 scRNAseq data in form of Seurat objects was downloaded from https://drive.google.com/drive/folders/12j9ufV1L0uWbUlab-VoXRznDLKDO7PQ. In case of future change in the data storage location, it will be updated in the readme file in project's Github repository (https://github.com/winstonbecker/scCRC_continuum)

Downstream analysis was performed with the `ArchR` package v. 1.0.2:

```
library(ArchR)
library(parallel)

catlas_files <- <FRAGMENT_FILES>
outputDir <- <OUTPUT_DIRECTORY>
arrow_files <- createArrowFiles(inputFiles = catlas_files, sampleNames = <SAMPLE_NAMES>)
doubScores <- addDoubletScores(input = arrow_files)

# cerate ArchR project
project <- ArchRProject(arrow_files, outputDirectory = outputDir)

# filtering out doublet cells
project <- filterDoublets(project)

# add Iterative Latent Semantic Indexing reduced-dimensionality space
project <- addIterativeLSI(ArchRProj = project, useMatrix = "TileMatrix",
                  name = "IterativeLSI", clusterParams = list(resolution = c(0.2),
                                              sampleCells = 10000, n.start = 10))

# batch correction
project  <- addHarmony( ArchRProj = project, reducedDims = "IterativeLSI",
    name = "Harmony", groupBy = "Sample")

project <- addClusters( input = project, reducedDims = "IterativeLSI",
    method = "Seurat", name = "Clusters", resolution = 0.8)

# add clusters after Harmony batch correction
project <- addClusters(input = project, reducedDims = "Harmony",
    method = "Seurat", name = "Clusters_Harmony", resolution = 0.8)

# add UMAP embedding
project <- addUMAP(ArchRProj = project, reducedDims = "IterativeLSI",
    nNeighbors = 30, minDist = 0.5, name ="UMAP_LSI")

project <- addUMAP(ArchRProj = project, reducedDims = "IterativeLSI",
    nNeighbors = 30, minDist = 0.5, name ="UMAP_Harmony")

# add column with log base 10 of the fragment numbers
project <- addCellColData(ArchRProj = project, data = log10(project$nFrags),
                                name = "log10_nFrags", cells = project$cellNames)
```

```
# upload gene expression data
# use files downloaded from
# https://drive.google.com/drive/folders/12j9ufV1L0uWbUlab-VoXRznDLKDO7PQ_?usp=sharing

data_files <- c("Final_scHTAN_colon_normal_epithelial_220213.rds",
                "Final_scHTAN_colon_immune_220213.rds",
                "Final_scHTAN_colon_stromal_220213.rds")

# define object names
RNAseq_se_names <- gsub(".*scHTAN_|_220213.rds", "", data_files)

# create objects as instances of SingleCellExperiment class
for (i in seq_along(RNAseq_se_names)) assign(RNAseq_se_names[i], Seurat::as.SingleCellExperiment(read

# add column with cell types and disease state
for (obj in RNAseq_se_names){
   eval(parse(text = paste0("colData(", obj, ")$CellType <-", obj,"@colData@listData$CellType")))
   eval(parse(text = paste0("colData(", obj, ")$DiseaseState <-", obj,"@colData@listData$DiseaseState'
}

# uniformize colData columns before merging
shared_cols <- purrr::map(list(colon_immune, colon_stromal, colon_normal_epithelial), colData) %>%
    purrr::map(colnames) %>%
    purrr::reduce(intersect)

# remove reducedDims since their column names differ across objects
for (obj in RNAseq_se_names){
   eval(parse(text = paste0(obj, "@colData <- ", obj, "@colData[,colnames(", obj, "@colData) %in% share
   eval(parse(text = paste0("SingleCellExperiment::reducedDim(", obj, ") <- NULL")))
    eval(parse(text = paste0(obj, "@int_colData@listData <- list()")))
}

# merge RNAseq data objects

colon_RNAseq <- cbind(colon_immune, colon_normal_epithelial, colon_stromal)


RNA_se <- SummarizedExperiment(assay = list(counts = as(assay(colon_RNAseq, "counts"), "dgCMatrix")),
                     colData = colData(colon_RNAseq), rowData = rowData(colon_RNAseq))


# select samples from healthy donors (no cancer)
RNA_se <- RNA_se[,colData(RNA_se)$DiseaseState == "Normal"]
```

```
# RNA integration
project <- addGeneIntegrationMatrix(
    ArchRProj = project,
    useMatrix = "GeneScoreMatrix",
    reducedDims = "IterativeLSI",
    seRNA = RNA_se,
    addToArrow = TRUE,
    groupRNA = "CellType",
    nameCell = "predicted_cell_un",
    nameGroup = "predicted_group_un",
    nameScore = "predicted_score_un")


project <- addGroupCoverages(ArchRProj = project, groupBy = "predicted_group_un")

# add pseudo-bulk replicates
## requires MACS2 installation

project <- addReproduciblePeakSet( ArchRProj = project,
    groupBy = "predicted_group_un", pathToMacs2 = <PATH_TO_MACS2>)


# LSI reduced dimensionality based on the GeneIntegrationMatrix

project <- addIterativeLSI(ArchRProj = project, clusterParams = list(resolution = 0.2,
        sampleCells = 1000, n.start = 10), saveIterations = FALSE,
    useMatrix = "GeneIntegrationMatrix", varFeatures = 2500,
    firstSelection = "variable", binarize = FALSE, name = "LSI_RNA")

# add clusters based on the new reduced-dimensionality space
project <- addClusters( input = project, reducedDims = "LSI_RNA",
    method = "Seurat", name = "Clusters_RNA", resolution = 0.8)

# add UMAP embedding
project <- addUMAP(ArchRProj = project, reducedDims = "LSI_RNA",
    nNeighbors = 30, minDist = 0.5, name ="UMAP_LSI_RNA", metric = "cosine",
    force = TRUE)

# batch correction
project  <- addHarmony( ArchRProj = project, reducedDims = "LSI_RNA",
    name = "Harmony_RNA", groupBy = "Sample")

# UMAP embedding after batch correction
project <- addUMAP(ArchRProj = project, reducedDims = "Harmony_RNA",
   nNeighbors = 30, minDist = 0.5, name ="UMAP_LSI_RNA_Harmony", metric = "cosine",
    force = TRUE
)
```

```
# find clutsters after batch correction
project <- addClusters( input = project, reducedDims = "Harmony_RNA",
    method = "Seurat", name = "Clusters_RNA_Harmony", resolution = 0.8)


# combine reduced-dimensionality spaces produced from ATACseq and RNAseq data
project <- addCombinedDims(project, reducedDims = c("IterativeLSI", "LSI_RNA"),
                           name =  "LSI_Combined")

# add UMAP embedding
project <- addUMAP(ArchRProj = project, name = "UMAP_combined", reducedDims = "LSI_Combined",
        nNeighbors = 30, minDist = 0.5, metric = "cosine")


# find clusters in combined reduced space
project <- addClusters(input = project, reducedDims = "LSI_Combined",
                       method = "Seurat", name = "Clusters_combined",
                       resolution = 0.4)

# add information about sequence motifs recognized by known transcriptions factors
project <- addMotifAnnotations(ArchRProj = project,
                                     motifSet = "cisbp", name = "Motif")


# add background peaks to be compared against during peak variation assessement
project <- addBgdPeaks(project)

# calculate per-cell devations of motif annotations
project <- addDeviationsMatrix(project, peakAnnotation = "Motif")

# save project
saveArchRProject(project, outputDir)

# convert project into MultiAssayExperiment object
MAE <- maw.archr::create.mae.with.multiple.sces.from.archr(outputDir, tile.sizes = 500)

saveRDS(MAE, <OUTPUT_PATH>)
```

Data storage and access

The `MultiAssayExperiments` is split into separate `SingleCellExperiment` objects and they in turn are split into components, all of which are stored in a single hdf5 file. Data and can be accessed with a special function that extracts elements of the requested experiment(s), reassembles them, and builds an MAE.

References

1. Zhang K, Hocker JD, Miller M, Hou X, Chiou J, Poirion OB, Qiu Y, Li YE, Gaulton KJ, Wang A, Preissl S, Ren B. A single-cell atlas of chromatin accessibility in the human genome. Cell. 2021 Nov 24;184(24):5985-6001.e19. doi: 10.1016/j.cell.2021.10.024. Epub 2021 Nov 12. PMID: 34774128; PMCID: PMC8664161.
2. Becker, W.R., Nevins, S.A., Chen, D.C. et al. Single-cell analyses define a continuum of cell state and composition changes in the malignant transformation of polyps to colorectal cancer. Nat Genet 54, 985–995 (2022). https://doi.org/10.1038/s41588-022-01088-x

Examples

```
# check metada of dataset
colonHealthy(metadata = TRUE)
# download data
## Not run:
colonHealthy()

## End(Not run)
```

`customClasses` *custom classes*

Description

Additional class definitions.

Classes

- `SingleCellAccessibilityExperiment`, contains `SingleCellExperiment`

`dummies` *create dummy data sets*

Description

Create dummy SCE and MAE objects.

Usage

```
dummySCE(
  features = c("rowData", "rowRanges", "reducedDims", "altExps", "none")
)

dummyMAE(experiments = list(EXP1 = NULL, EXP2 = NULL))
```

Arguments

features	character string specifying which (optional) features to create in the SCE
experiments	named list of character vectors specifying experiments to create and their features

Value

dummySCE returns a SingleCellExperiment. dummyMAE returns a MultiAssayExperiment.

Examples

```
scMultiome:::dummySCE()
scMultiome:::dummySCE("rowData")
scMultiome:::dummySCE("rowRanges")
scMultiome:::dummySCE("reducedDims")
scMultiome:::dummySCE("altExps")

scMultiome:::dummyMAE(list("dummyExperiment" = NULL))
```

---

fileOperations *save and load data sets*

Description

Functions to disassemble and save, and load and reassemble MultiAssayExperiment data sets.

Usage

```
saveMAE(mae, file, experiments = NULL, verbose = TRUE, overwrite = FALSE)

loadMAE(file, experiments, verbose)

saveExp(exp, expName, file, verbose)

## S4 method for signature 'SummarizedExperiment'
saveExp(exp, expName, file, verbose)

## S4 method for signature 'SingleCellExperiment'
saveExp(exp, expName, file, verbose)

loadExp(file, expName, verbose)

testFile(file)

uploadFile(
  file,
```

```
  sasToken,
  endpoint = "https://bioconductorhubs.blob.core.windows.net"
)
```

Arguments

mae	object of class MultiAssayExperiment
file	path to a hdf5 file
experiments	character string specifying which experiments to save/load
verbose	logical flag specifying operation verbosity
overwrite	logical flag specifying whether to allow overwriting the hdf5 file
exp	an experiment object that inherits from class SummarizedExperiment, usually a SingleCellExperiment
expName	name of the experiment, i.e. name of the ArchR Matrix
sasToken	access token to endpoint
endpoint	Bioconductor's data bucket endpoint url

Details

These are utilities for developers to add new data sets to the package. Most will usually be called internally.

saveMAE is used to save a MultiAssayExperiment to a hdf5 file. It creates the file and passes individual experiments to saveExp.

saveExp is called by saveMAE to disassemble experiment exp and save its elements in file. A group hierarchy is created with the top level group called expName, e.g. "GeneScoreMatrix", and lower level groups to store specific elements:

- experiment class is saved in group "class"
- assays are saved as sparse matrices in subgroup "assays", using writeSparseMatrix
- colData and colnames are saved in subgroup "properties"
- if experiment has rownames, the are saved in "properties"
- rowData is saved in "properties", unless it is an empty DataFrame
- if experiment has a rowRanges component, it is converted to a data frame and saved in "properties"
- if experiment has a metadata component, it is deparsed to a string and saved in "properties"
- if experiment has a reducedDims component, they are saved in subgroup "reducedDims"
- if experiment has a altExps component, they are saved in subgroup "altExps" by recursively calling saveExp

DataFrames (e.g. colData, rowData, embeddings) are converted to data.frames and saved as compound type.

loadMAE is called by accessor functions to retrieve data. It locates the hdf5 file in which the data set is stored and uses loadExp to extract the specified experiments.

loadExp first checks which property elements are stored for the experiment in question, loads all elements of the experiment and builds a SummarizedExperiment object. If the experiment was originally a SingleCellExperiment or a subclass thereof, that class as well as possible additional slots are restored.

testFile can be used to test whether a data set loads correctly from a local file. It calls loadMAE and extracts all experiment verbosely.

uploadFile will upload a single file to Bioconductor's staging directory.

Value

saveExp returns TRUE invisibly if the save was successful. saveMAE returns a named list of TRUE values. loadExp returns a SingleCellExperiment or an object of a subclass. loadMAE returns a MultiAssayExperiment. testFile returns the MultiAssayExperiment stored in file in its entirety. uploadFile returns TRUE invisibly.

Author(s)

Aleksander Chlebowski and Xiaosai

See Also

Vignette "rhdf5 - HDF5 interface for R" (vignette or browseVignettes) for details of hdf5 file construction. writeSparseMatrix for details of saving sparse matrices.

Examples

```
# create dummy MultiAssayExperiment
mae <- scMultiome:::dummyMAE()

fileName <- tempfile(fileext = ".h5")
saveMAE(mae, fileName)                                  # save MAE
remae <- loadMAE(fileName, c("EXP1", "EXP2"), TRUE)   # load MAE (internal)
remae_exp1 <- loadMAE(fileName, "EXP1", TRUE)      # load MAE with one experiment (internal)

# create dummy SingleCellExperiment
sce <- scMultiome:::dummySCE()

saveExp(sce, "EXP3", fileName, TRUE)                  # save one experiment (internal)
resce <- loadExp(fileName, "EXP3", TRUE)              # load one experiment (internal)

testFile(fileName)                                    # load whole MAE
```

---

hematopoiesis *scATAC-seq and unpaired scRNA-seq of hematopoetic cells*

Description

Example scATAC-seq data of hematopoietic cells included in ArchR package was integrated with scRNAseq. ScATAC-seq data was obtained from GSE139369 and scRNA-seq obtained from https://jeffgranja.s3.amazonaws.
Hematopoiesis-Granja-2019.rds

Usage

```
hematopoiesis(
  metadata = FALSE,
  experiments = c("TileMatrix500", "GeneScoreMatrix", "GeneIntegrationMatrix",
    "PeakMatrix")
)
```

Arguments

`metadata`	logical flag specifying whether to return data or metadata only
`experiments`	character vector of matrices to return; see `Format`

Format

`MultiAssayExperiment` obtained from an `ArchR` project. Annotated with the hg19 genome build. Contains the following experiments:

- **GeneIntegrationMatrix**: SingleCellExperiment with 17889 rows and 10250 columns
- **GeneScoreMatrix**: SingleCellExperiment with 22217 rows and 10250 columns
- **PeakMatrix**: SingleCellExperiment with 150046 rows and 10250 columns
- **TileMatrix500**: SingleCellExperiment with 5762078 rows and 10250 columns

Value

`MultiAssayExperiment` made up of `SingleCellExperiments` with assays stored as `DelayedMatrix` objects. If `metadata = TRUE`, an `ExperimentHub` object listing this data set's metadata.

Data preparation

Example scATAC-seq data of hematopoietic cells included in ArchR package was integrated with scRNA-seq. ScATAC-seq data was obtained from GSE139369 and scRNA-seq data was obtained from https://jeffgranja.s3.amazonaws.com/ArchR/TestData/scRNA-Hematopoiesis-Granja-2019.rds"

Data storage and access

The `MultiAssayExperiments` is split into separate `SingleCellExperiment` objects and they in turn are split into components, all of which are stored in a single hdf5 file. Data and can be accessed with a special function that extracts elements of the requested experiment(s), reassembles them, and builds an MAE.

References

Single-cell multiomic analysis identifies regulatory programs in mixed-phenotype acute leukemia associated with prostate cancer relapse. Granja *et al.*, *Nature Biotechnology* 2019 Dec;37(12):1458-1465. doi: 10.1038/s41587-019-0332-7

Examples

```
# check metada of dataset
hematopoiesis(metadata = TRUE)
# download data
## Not run:
hematopoiesis()

## End(Not run)
```

`listDatasets` *list all available data sets*

Description

Summary information for all data sets available in the package.

Usage

```
listDatasets()
```

Value

A `DataFrame` listing all available data sets, with one data set per row and the following columns:

- Call: function call used to access the data set directly
- Author: original data set author
- Title: data set name
- Species: species name
- Lineage: sample lineage
- CellNumber: number of cells in the data set
- Multiome: paired or unpaired
- DiskSize: size of the dataset in storage (also size of the download)
- Version: data set version number or upload date

Author(s)

Aleksander Chlebowski

Examples

```
listDatasets()
```

makeDataSetList *create data set list*

Description

Automatically creates the data set list of the package.

Usage

```
makeDataSetList(metadata)
```

Arguments

metadata	a data.frame containing data set metadata

Details

This is an internal helper function for developers and will not be called directly. It creates the file inst/scripts/datasetList.Rmd, which is incorporated into the package help page to automatically list the current data sets.

Value

Invisible TRUE.

prostateENZ *LNCaP Cells Treated with Enzalutamide*

Description

Single-cell ATAC sequencing of parental LNCaP cells (DMSO treated, the control), LNCaP cells treated with 10μM enzalutamide for 48 hours, and LNCaP-derived enzalutamide-resistant RES-A and RES-B cells.

Usage

```
prostateENZ(
  metadata = FALSE,
 experiments = c("TileMatrix", "GeneScoreMatrix", "GeneIntegrationMatrix", "PeakMatrix",
    "MotifMatrix")
)
```

Arguments

metadata	logical flag specifying whether to return data or metadata only
experiments	character vector of matrices to return; see Format

Format

`MultiAssayExperiment` obtained from an `ArchR` project. Annotated with the Hg38 genome build. Contains the following experiments:

- **TileMatrix**: SingleCellExperiment with 6062095 rows and 15522 columns
- **GeneScoreMatrix**: SingleCellExperiment with 24919 rows and 15522 columns
- **GeneIntegrationMatrix**: SingleCellExperiment with 23525 rows and 15522 columns
- **PeakMatrix**: SingleCellExperiment with 80210 rows and 15522 columns
- **MotifMatrix**: SingleCellExperiment with 870 rows and 15522 columns

Value

`MultiAssayExperiment` made up of `SingleCellExperiments` with assays stored as `DelayedMatrix` objects. If `metadata = TRUE`, an `ExperimentHub` object listing this data set's metadata.

Data storage and access

The `MultiAssayExperiments` is split into separate `SingleCellExperiment` objects and they in turn are split into components, all of which are stored in a single hdf5 file. Data and can be accessed with a special function that extracts elements of the requested experiment(s), reassembles them, and builds an MAE.

Data preparation

scATAC data was downloaded from Gene Expression Omnibus (acc. no. GSE168667) and analyzed with SingleCell ATAC - 10X pipeline v2.0.0. scRNAseq data was downloaded from Gene Expression Omnibus (acc.no. GSE168668) and analyzed with SingleCell Gene Expression Analysis - 10X pipeline v6.0.1.

Downstream analysis was performed with the `ArchR` package:

1. Initiate `ArchR` project:

```
# attach ArchR package
library(ArchR)

# configure ArchR
addArchRThreads(16L)
addArchRGenome("hg38")

# create arrow file from fragment files
## list fragment files
fragments <- <FRAGMENT_FILES>
## assign sample names
names(fragments) <- <SAMPLE_IDs>
## create arrows
createArrowFiles(inputFiles = fragments, sampleNames = names(fragments))

# specify output directory
```

```
outDir <- <OUTPUT_DIRECTORY>

# locate arrow files
arrows <- <ARROW_FILES>

# create ArchR project
project <- ArchRProject(arrows, outDir)

# add sample annotation
sampleNames <- c("SRR13927735", "SRR13927736", "SRR13927737", "SRR13927738")
sampleCells <- c("LNCaP", "LNCaP", "LNCaP RES-A", "LNCaP RES-B")
sampleTreatment <- c("0.1% DMSO 48h", "enzalutamide 48h", "enzalutamide", "enzalutamide")
sampleEnzalutamide <- c("sensitive", "sensitive", "resistant", "resistant")
names(sampleCells) <- names(sampleTreatment) <- names(sampleEnzalutamide) <- sampleNames
project$Cells <- sampleCells[project$Sample]
project$Treatment <- sampleTreatment[project$Sample]
project$Enzalutamide <- sampleEnzalutamide[project$Sample]
project$sampleLabels <- sampleLabels[project$Sample]
```

2. Prepare RNA-seq data:

```
# gene expression data is analyzed with `scran.chan` package
# the result is a SingleCellExperiment object
SCE <- <scran.chan ANALYSIS>

# adjust for integration
rownames(SCE) <- rowData(SCE)$Symbol
assay(SCE, "counts") <- as(assay(SCE, "counts"), "dgCMatrix")
# drop duplicates
SCE <- SCE[!duplicated(rowData(SCE)$Symbol), ]
```

3. Commence `ArchR` **analysis:**

```
# reduce dimensionality by iterative LSI
project <- addIterativeLSI(project, useMatrix = "TileMatrix", name = "iLSI_ATAC")

# integrate ATAC and RNAseq
## prepare grouping for constrained integration
groupMapping <- SimpleList(
    sens_NT = SimpleList(
        ATAC = project$cellNames[project$Sample == "SRR13927735"],
        RNA = grep("SRR13927739", colnames(SCE), value = TRUE)
    ),
    sens_Enz = SimpleList(
        ATAC = project$cellNames[project$Sample == "SRR13927736"],
        RNA = grep("SRR13927740", colnames(SCE), value = TRUE)
    ),
    RES_A = SimpleList(
        ATAC = project$cellNames[project$Sample == "SRR13927737"],
        RNA = grep("SRR13927741", colnames(SCE), value = TRUE)
```

```
    ),
    RES_B = SimpleList(
        ATAC = project$cellNames[project$Sample == "SRR13927738"],
        RNA = grep("SRR13927742", colnames(SCE), value = TRUE)
    )
)
## execute
project <- addGeneIntegrationMatrix(project, useMatrix = "GeneScoreMatrix",
                                    matrixName = "GeneIntegrationMatrix",
                                    reducedDims = "iLSI_ATAC", seRNA = SCE,
                          groupATAC = "Sample", groupRNA = "Sample", groupList = groupMapping,
                                    nameCell = "predictedCell",
                                    nameGroup = "predictedGroup",
                                    nameScore = "predictedScore",
                                    addToArrow = TRUE, force = TRUE)

# add LSI for RNAseq
project <- addIterativeLSI(project, useMatrix = "GeneIntegrationMatrix", name = "iLSI_RNAseq")

# combine dim-reduced ATAC and RNAseq
project <- addCombinedDims(project, name = "iLSI_Combined", reducedDims = c("iLSI_ATAC", "iLSI_RNAseq

# add UMAP embedding on combined reduced dimensionality
project <- addUMAP(project, reducedDims = "iLSI_Combined", name = "UMAP_Combined", verbose = FALSE)

# impute weights (for smoother visualizations)
project <- addImputeWeights(project, reducedDims = "iLSI_Combined")

# add group coverages
## inspect available cell numbers
table(project$Sample)
project <- addGroupCoverages(project, groupBy = "Sample", minCells = 30, maxCells = 250)

# add pseudo-bulk replicates
## requires MACS2 installation
project <- addReproduciblePeakSet(project, groupBy = "Sample", pathToMacs2 = "<PATH_TO_MACS2_INSTALLA

# add peak matrix
project <- addPeakMatrix(project)
getAvailableMatrices(project)

# add motif annotation
project <- addMotifAnnotations(project, motifset = "cisbp", name = "Motif")

# add background peaks
project <- addBgdPeaks(project, method = "chromVAR")

# add deviation matrix
```

```
project <- addDeviationsMatrix(project, peakAnnotation = "Motif", force = TRUE)
getAvailableMatrices(project)

# save ArchR project
saveArchRProject(project)
```

4. Save results:

```
# convert project to MultiAssayExperiment object
MAE <- maw.archr::create.mae.with.multiple.sces.from.archr(outDir)

# inspect object
MAE

# remove unpublished class
ind <- which(names(MAE) == "TileMatrix500")
MAElim <- MultiAssayExperiment::MultiAssayExperiment(
    experiments = c(
    TileMatrix500 = as(experiments(MAE)[[ind]], "SingleCellExperiment"),
    as.list(experiments(MAE)[-ind])
))

# save object
saveMAE("inst/extdata/prostateENZ.h5")
```

References

Single-cell ATAC and RNA sequencing reveal pre-existing and persistent cells associated with prostate cancer relapse. Taavitsainen *et al.*, *Nature Communications* 2021 Sep 6;12(1):5307 doi: 10.1038/s41467-021-25624-1

Examples

```
# check metada of dataset
prostateENZ(metadata = TRUE)
# download data
## Not run:
prostateENZ()

## End(Not run)
```

`reprogramSeq` *reprogramSeq*

Description

scMultiome data of LNCaP infected with FOXA1, NKX2-1, GATA6

Usage

```
reprogramSeq(
  metadata = FALSE,
  experiments = c("TileMatrix500", "GeneExpressionMatrix", "GeneScoreMatrix",
    "NEPCMatrix", "PeakMatrix", "TF_bindingMatrix")
)
```

Arguments

metadata	logical flag specifying whether to return data or metadata only
experiments	character vector of matrices to return; see Format

Format

MultiAssayExperiment obtained from an ArchR project. Annotated with the hg38 genome build. Contains the following experiments:

- **TileMatrix500**: SingleCellAccessibilityExperiment with 6062095 rows and 3903 columns
- **GeneExpressionMatrix**: SingleCellExperiment with 36438 rows and 3903 columns
- **GeneScoreMatrix**: SingleCellExperiment with 24919 rows and 3903 columns
- **NEPCMatrix**: SingleCellExperiment with 2 rows and 3903 columns
- **PeakMatrix**: SingleCellExperiment with 126602 rows and 3903 columns
- **TF_bindingMatrix**: SingleCellExperiment with 1274 rows and 3903 columns

Value

MultiAssayExperiment made up of SingleCellExperiments with assays stored as DelayedMatrix objects. If metadata = TRUE, an ExperimentHub object listing this data set's metadata.

Data preparation

scMultiome data was processed by ArchR.

Data storage and access

The MultiAssayExperiments is split into separate SingleCellExperiment objects and they in turn are split into components, all of which are stored in a single hdf5 file. Data and can be accessed with a special function that extracts elements of the requested experiment(s), reassembles them, and builds an MAE.

References

Genentech dataset

Examples

```
# check metada of dataset
reprogramSeq(metadata = TRUE)
# download data
## Not run:
reprogramSeq()

## End(Not run)
```

`retrieve` *retrieve data set or its metadata*

Description

Retrieve and return the data or metadata for the currently queried data set.

Usage

```
retrieve(dataset, metadata, experiments, verbose = FALSE)
```

Arguments

`dataset`	character string specifying the data set name
`metadata`	logical flag specifying whether to return the resource or only its metadata
`experiments`	character string specifying which experiments to extract
`verbose`	logical flag specifying loading verbosity

Details

This is a generic accessor function that is used by user-level accessor functions to access data sets or their metadata.

Value

If `metadata = FALSE`, a `MultiAssayExperiment`, otherwise an `ExperimentHub` object.

RGtools *manipulate GeneRanges*

Description

Prepare and recover GRanges objects for and after storing.

Usage

```
storeGR(x)

restoreGR(df)
```

Arguments

x	object of class GRanges
df	a data.frame

Details

GRanges objects, which can be encountered in rowRanges slots of SingleCellExperiments, are stored as data frames (of type compound).

storeGR converts GRanges to a data frames converts factors to characters. restoreGR resets data types in the basic columns and re-instantiates GRanges.

Value

storeGR returns a data frame, restoreGR returns a GRanges object.

templates *documentation templates*

Description

Create templates for data set documentation.

Usage

```
makeMakeData(dataset)

makeMakeMetadata(dataset)

makeR(dataset)
```

Arguments

dataset	name of data set as character string

Details

Functions to facilitate documenting a new data set. Each function creates a file and attempts to open it in RStudio for editing.

`makeMakeData` creates an Rmarkdown report called `inst/scripts/make-data-<dataset>.Rmd`. `makeMakeMetadata` creates an R script called `inst/scripts/make-metadata-<dataset>.R`. `makeR` creates an R file called `R/<dataset>.Rmd`.

Value

All functions return TRUE invisibly.

Author(s)

Aleksander Chlebowski

---

tfBinding *TF Binding Info*

Description

Combined transcription factor ChIP-seq data from ChIP-Atlas and ENCODE or from CistromeDB and ENCODE.

Usage

```
tfBinding(
  genome = c("hg38", "hg19", "mm10"),
  source = c("atlas", "cistrome", "encode.sample", "atlas.sample", "atlas.tissue"),
  metadata = FALSE
)
```

Arguments

genome	character string specifying the genomic build
source	character string specifying the ChIP-seq data source
metadata	logical flag specifying whether to return data or metadata only

Format

`GRangesList` object containing binding site information of transcription factor ChIP-seq. Contains the following experiments:

- **hg38_atlas**: GRangesList object of length 1558
- **hg19_atlas**: GRangesList object of length 1558
- **mm10_atlas**: GRangesList object of length 768
- **hg38_cistrome**: GRangesList object of length 1269
- **hg19_cistrome**: GRangesList object of length 1271
- **mm10_cistrome**: GRangesList object of length 544
- **hg38_encode.sample**: List object of length 171
- **hg19_encode.sample**: List object of length 171
- **mm10_encode.sample**: List object of length 31
- **hg38_atlas.sample**: List object of length 1112
- **hg19_atlas.sample**: List object of length 1112
- **mm10_atlas.sample**: List object of length 517
- **hg38_atlas.tissue**: List object of length 22
- **hg19_atlas.tissue**: List object of length 22
- **mm10_atlas.tissue**: List object of length 23

Details

This is a special data set that stores transcription factor binding sites for human and mouse genomic builds, which can be used with the package epiregulon to compute regulons.

Value

A list of TF binding sites as a `GrangesList` object.

Data storage and access

Each genomic build is a separate `GRangesList` object, stored in a separate RDS file. All genomic builds can be accessed with the same function `tfBinding`.

Data preparation

1. Data download:

We download public ChIP-seq peak calls from ChIP-Atlas and ENCODE

1.1. ChIP-Atlas:

ChIP-seq binding sites were downloaded from ChIP-Atlas

```
# metatdata
# download fileList.tab from https://dbarchive.biosciencedbc.jp/kyushu-u/metadata/fileList.tab

dir <- "chipAtlas/"
fileLIST <- read.delim(file.path(dir, "metadata/fileList.tab"), header = FALSE)


for (genome in c("hg38", "mm10")){

    TFLIST <- fileLIST[which(fileLIST[,3] == "TFs and others" &
                                 fileLIST[,2] == genome &
                                 fileLIST[,4] != "-" &
                                 fileLIST[,5] == "All cell types" &
                                 fileLIST[,7] == "05"),]

   download.files <- paste0("wget http://dbarchive.biosciencedbc.jp/kyushu-u/",
                            genome,"/assembled/", TFLIST$V1, ".bed")
    write.table(x = download.files,
                file = file.path(dir, genome, ".sh"),
                quote = FALSE,
                col.names = FALSE,
                row.names = FALSE)

    write.table(TFLIST,
                file = file.path(dir, genome,".metadata.txt"),
                quote = FALSE,
                col.names = FALSE,
                row.names = FALSE,
                sep = "\t")

}
```

Download bed files by sample

```
for (genome in c("mm10","hg38")){
   metadata <- read.delim("experimentList.tab", header=FALSE)
   metadata <- metadata[metadata$V2 == genome,]
   metadata <- metadata[metadata$V3 == "TFs and others",]


# write files for download
fileConn <- file(paste0("data/download.tissue.", genome,".txt"))
writeLines(paste0("wget -nc https://chip-atlas.dbcls.jp/data/", genome,"/eachData/bed05/", metadata
close(fileConn)
}
```

1.2. ENCODE:

Transcription factor ChIP-seq peaks were downloaded from ENCODE data portal

2. Merge peaks:

Merge peaks of the same TFs into the same bed files.

2.1. ChIP-Atlas:

```
library(GenomicRanges)
library(rtracklayer)

##################### chipatlas_bedfiles_merge ####################
## Takes in a list of bed files and an accompanying legend that shows
## which BED files correspond to  a specific TF
## Outputs a directory of merged bed files

dir <- 'chipAtlas/'
outdir <- 'chipatlas/data/chipatlas'



genomes <- c("mm10","hg38")
chr_order <- list()
chr_order[["mm10"]] <- c(paste0("chr",1:19),"chrX","chrY","chrM")
chr_order[["hg38"]] <- c(paste0("chr",1:22),"chrX","chrY","chrM")

for (genome in genomes){
  # Get directories of all bed files and make a list of the path of all bed files
  list_beds <- list.files(file.path(dir, paste0(genome, '_1e5')),
                          pattern = "*.bed")

  # Read the metatdata file

  metadata <- read.delim(file.path(dir, genome, ".metadata.txt"), header=FALSE)
  metadata$filename <- file.path(dir, genome, "_1e5", metadata$V1,".bed")
  colnames(metadata)[4] <- "TF"

  ## specify sorting order for chromosomes
  chr_order_genome <- chr_order[[genome]]

  ### get list of TFs represented in BED files
  TF.list <- unique(metadata$TF)

  for (i in seq_along(TF.list)) {

    print(TF.list[i])

    # get bed files associated with TF
    TF.files <- metadata[which(metadata$TF == TF.list[i]), "filename"]

    # merge bed files and sort by chromosome and starting coordinate
```

```
    merged_bed <- as.data.frame(rtracklayer::import.bed(file_name))
   merged_bed$seqnames <- factor(merged_bed$seqnames, levels = chr_order_genome)
    merged_bed <- na.omit(merged_bed)
    merged_bed <- merged_bed[order(merged_bed$seqnames, merged_bed$start),]

    ## convert to granges object and merge overlapping ranges
    gr <- makeGRangesFromDataFrame(merged_bed,
                                   seqnames.field = "seqnames",
                                   start.field = "start",
                                   end.field = "end")
    gr <- reduce(gr, drop.empty.ranges = TRUE)

    # write new bed file to directory
    export.bed(gr, con = file.path(outdir, genome, TF.list[i],".bed"))
  }

}
```

2.2. ENCODE:

Filter and merge ENCODE peaks

```
############### filter peaks ####################
## Takes in bed files and filters the peaks
## in each file based on the enrichment score.
## Low Enrichment peaks are filtered out.

dir = 'encode/'
outdir = 'chipatlas/data/encode/'
for (genome in c("mm10", "hg38")){

    # Make a list of all the bed files
   list_beds <- list.files(file.path(dir, genome, "raw"), pattern = "*.bed.gz")

    #filter each bed file to have p value score > 4
    for (i in seq_along(list_beds)) {

        print(list_beds[i])

        curr_bed <- read.table(file.path(dir, genome, "raw", list_beds[i]))

        post_QC_bed <- curr_bed[curr_bed$V7 >=5,] #Q values

        if (nrow(post_QC_bed) >100){
            write.table(post_QC_bed,
                        file.path('/gstore/scratch/u/yaox19/encode/',
                               genome, "filtered", list_beds[i]),
                        row.names = F, col.names = F, quote = F, sep="\t")
        }
```

```
    }
}



##################### ENCODE_bedfiles_merge ####################
## Takes in a list of bed files and an accompanying legend that shows
## which BED files correspond to  a specific TF
## Outputs a directory of merged bed files



chr_order <- list()
chr_order[["mm10"]] <- c(paste0("chr",1:19),"chrX","chrY","chrM")
chr_order[["hg38"]] <- c(paste0("chr",1:22),"chrX","chrY","chrM")

replacement <- list(mm10 = "-mouse", hg38 = "-human")

for (genome in c("hg38","mm10")){

    # read metadata
   list_beds <- list.files(file.path(dir, genome, "raw"), pattern = "*.bed.gz")
    metadata <- read.delim(file.path(dir, genome, "raw", "metadata.tsv"))
   if(genome == "hg38") metadata <- metadata[metadata$"File.assembly"=="GRCh38",]
    metadata <- metadata[, c("File.accession", "Experiment.target")]
   metadata$"Experiment.target" <-  gsub(replacement[[genome]],"", metadata$"Experiment.target")

    # capitalize the first alphabet for mouse genes
    if (genome == "mm10"){
     metadata$"Experiment.target" <- stringr::str_to_title(metadata$"Experiment.target")
    }

    metadata$"File.accession" <- trimws( metadata$"File.accession")
    metadata$"File.accession" <- file.path(dir, genome, "filtered",
                                           metadata$"File.accession",".bed.gz")
    colnames(metadata) <- c("filename","TF")

    # remove files that do not exist
    metadata <- metadata[which(file.exists(metadata$filename)),]

    # specify sorting order for chromosomes
    chr_order_genome <- chr_order[[genome]]

    # get list of TFs represented in BED files
    TF.list <- unique(metadata$TF)

    for (i in seq_along(TF.list)) {
```

```
        print(TF.list[i])

        # get bed files associated with TF
        TF.files <- metadata[which(metadata$TF == TF.list[i]), "filename"]

        # merge bed files and sort by chromosome and starting coordinate
        extraCols_narrowPeak <- c(singnalValue = "numeric", pValue = "numeric",
                              qValue = "numeric", peak = "integer")
      merged_bed <- lapply(TF.files, function(file) as.data.frame(rtracklayer::import.bed(file, extr
        merged_bed <- do.call('rbind', merged_bed)
      merged_bed$seqnames <- factor(merged_bed$seqnames, levels = chr_order[[genome]])
        merged_bed <- merged_bed[!is.na(merged_bed$seqnames),]
       merged_bed <- merged_bed[order(merged_bed$seqnames, merged_bed$start),]

        # convert to granges object and merge overlapping ranges
        gr <- makeGRangesFromDataFrame(merged_bed,
                                       seqnames.field = "V1",
                                       start.field = "V2",
                                       end.field = "V3")
        gr <- reduce(gr, drop.empty.ranges = TRUE)

        # write new bed file to directory
        export.bed(gr, con=file.path(outdir, genome, TF.list[i],".bed"))
    }




}
```

2.3. Merge both ENCODE and ChIP-Atlas:

```
dir <- "chipatlas/data/"

chr_order <- list()
chr_order[["mm10"]] <- c(paste0("chr",1:19),"chrX","chrY","chrM")
chr_order[["hg38"]] <- c(paste0("chr",1:22),"chrX","chrY","chrM")


for (genome in c("mm10", "hg38")){

    # specify sorting order for chromosomes
    chr_order_genome <- chr_order[[genome]]

    ########### merge with chipatlas bed files
   chipatlas.files <- list.files(file.path(dir, "chipatlas", genome),pattern = "*.bed")
   encode.files <- list.files(file.path(dir, "encode", genome),pattern = "*.bed")
    shared.TFs <- intersect(chipatlas.files, encode.files)
```

```
chipatlas.TFs <- setdiff(chipatlas.files, encode.files)
encode.TFs <- setdiff(encode.files, chipatlas.files)

#### shared.TFs
for (i in seq_along(shared.TFs)) {

    print(shared.TFs[i])

 chipatlas <- as.data.frame(import.bed(file.path(dir, "chipatlas", genome, shared.TFs[i])))
 encode <- as.data.frame(import.bed(file.path(dir, "encode", genome, shared.TFs[i])))

    merged_bed <- rbind(chipatlas[,1:3], encode[,1:3])
 merged_bed$seqnames <- factor(merged_bed$seqnames, levels = chr_order_genome)
    merged_bed <- merged_bed[!is.na(merged_bed$seqnames),]
   merged_bed <- merged_bed[order(merged_bed$seqnames, merged_bed$start),]

    gr <- makeGRangesFromDataFrame(merged_bed,seqnames.field = "seqnames",
                                start.field = "start",end.field = "end")
    gr <- reduce(gr, drop.empty.ranges = TRUE)

 export.bed(gr, con=file.path(dir, "chipatlas_encode_merged", genome, shared.TFs[i]))

}

##### chipatlas
for (i in seq_along(chipatlas.TFs)) {

    print(chipatlas.TFs[i])

 chipatlas <- as.data.frame(import.bed(file.path(dir, "chipatlas", genome, chipatlas.TFs[i])))

    merged_bed <- chipatlas[,1:3]
 merged_bed$seqnames <- factor(merged_bed$seqnames, levels = chr_order_genome)
    merged_bed <- merged_bed[!is.na(merged_bed$seqnames),]
   merged_bed <- merged_bed[order(merged_bed$seqnames, merged_bed$start),]

    gr <- makeGRangesFromDataFrame(merged_bed,seqnames.field = "seqnames",
                                   start.field = "start",end.field = "end")
    gr <- reduce(gr, drop.empty.ranges = TRUE)

 export.bed(gr, con=file.path(dir, "chipatlas_encode_merged", genome, chipatlas.TFs[i]))

}

##### encode
for (i in seq_along(length(encode.TFs))) {

    print(encode.TFs[i])
```

```
      chipatlas <- as.data.frame(import.bed(file.path(dir, "encode", genome, encode.TFs[i])))

         merged_bed <- chipatlas[,1:3]
      merged_bed$seqnames <- factor(merged_bed$seqnames, levels = chr_order_genome)
         merged_bed <- merged_bed[!is.na(merged_bed$seqnames),]
        merged_bed <- merged_bed[order(merged_bed$seqnames, merged_bed$start),]

         gr <- makeGRangesFromDataFrame(merged_bed,seqnames.field = "seqnames",
                                        start.field = "start",end.field = "end")
         gr <- reduce(gr, drop.empty.ranges = TRUE)

      export.bed(gr, con=file.path(dir, "chipatlas_encode_merged", genome, encode.TFs[i]))

    }
}
```

2.4 Merge ChIP-Atlas peaks by sample:

```
library(rtracklayer)
#import metadata

for (genome in c("hg38, mm10")){
   metadata <- read.delim("experimentList.tab", header=FALSE)
   metadata <- metadata[metadata$V2 == genome,]
   metadata <- metadata[metadata$V3 == "TFs and others",]

    # import bed
    extraCols_narrowPeak <- c(singnalValue = "numeric", pValue = "numeric",
                              qValue = "numeric", peak = "integer")

    grl <- list()
    gr <- list()

    celllines <- unique(metadata$V6)
    celllines <- as.vector(na.omit(celllines))
    for (cellline in celllines){
        message("cell line = ", cellline)
        metadata.cellline <- metadata[which(metadata $V6 == cellline),]
        unique.tf <- unique(metadata.cellline$V4)

        for (tf in unique.tf){
            message("tf = ", tf)
            bed <- metadata.cellline$V1[metadata.cellline$V4 ==tf]

            for (i in 1:length(bed)) {
                file <- paste0(bed[i],".05.bed")

                if (file.exists(file)) {
```

```
                gr[[i]] <- try(rtracklayer::import(file, extraCols = extraCols_narrowPeak))

                   }

               }

               if (length(gr) !=0){
                   grl[[cellline]][[tf]] <- reduce(do.call(c, gr))
               }

           }

           grl[[cellline]] <- do.call(GRangesList, grl[[cellline]])

       }

       saveRDS(grl, paste0("grl.chipatlas.sample.", genome,".rds"))

   }
```

2.5 Merge ChIP-Atlas peaks by tissue:

```
library(rtracklayer)
#import metadata

for (genome in c("mm10","hg38")){
   metadata <- read.delim("experimentList.tab", header=FALSE)
   metadata <- metadata[metadata$V2 == genome,]
   metadata <- metadata[metadata$V3 == "TFs and others",]

    # import bed
    extraCols_narrowPeak <- c(singnalValue = "numeric", pValue = "numeric",
                              qValue = "numeric", peak = "integer")

    grl <- list()
    gr <- list()

    tissues <- unique(metadata$V5)
    tissues <- as.vector(na.omit(tissues))
    for (tissue in tissues){
        message("tissue = ", tissue)
        metadata.tissue <- metadata[which(metadata $V5 == tissue),]
        unique.tf <- unique(metadata.tissue$V4)

        for (tf in unique.tf){
            message("tf = ", tf)
            bed <- metadata.tissue$V1[metadata.tissue$V4 ==tf]
```

```
            for (i in 1:length(bed)) {
                file <- paste0(bed[i],".05.bed")

                if (file.exists(file)) {
             gr[[i]] <- try(rtracklayer::import(file, extraCols = extraCols_narrowPeak))
                }

            }

            if (length(gr) !=0){
                grl[[tissue]][[tf]] <- reduce(do.call(c, gr))
            }

        }

        grl[[tissue]] <- do.call(GRangesList, grl[[tissue]])

    }

    saveRDS(grl, paste0("grl.chipatlas.tissue.", genome,".rds"))

}
```

2.6 Merge ENCODE peaks by sample:

```
library(rtracklayer)
#import metadata

metadata.path <- c(hg38="human",
                   mm10="mm10")

for (genome in c("hg38", "mm10")){

   metadata <- read.delim(file.path(metadata.path[genome], "raw","metadata.tsv"), header=TRUE)
    metadata <- metadata[metadata$File.assembly == genome,]
    metadata <- metadata[metadata$Assay == "TF ChIP-seq",]

    # import bed
    extraCols_narrowPeak <- c(singnalValue = "numeric", pValue = "numeric",
                              qValue = "numeric", peak = "integer")

    grl <- list()
    gr <- list()

    celllines <- unique(metadata$Biosample.term.name)
    celllines <- as.vector(na.omit(celllines))
    for (cellline in celllines){
        message("cell line = ", cellline)
```

```
      metadata.cellline <- metadata[which(metadata $Biosample.term.name == cellline),]
         unique.tf <- unique(metadata.cellline$Experiment.target)

         for (tf in unique.tf){
             message("tf = ", tf)
             tf.name <- sapply(strsplit(tf, "-"),"[",1)
         bed <- metadata.cellline$File.accession[metadata.cellline$Experiment.target ==tf]

             for (i in 1:length(bed)) {
            file <- file.path(metadata.path[genome], "filtered", paste0(bed[i],".bed.gz"))

                 if (file.exists(file)) {
              gr[[i]] <- try(rtracklayer::import(file, extraCols = extraCols_narrowPeak))

                 }

             }

             if (length(gr) !=0){
                 grl[[cellline]][[tf.name]] <- reduce(do.call(c, gr))
             }

         }

         grl[[cellline]] <- do.call(GRangesList, grl[[cellline]])

     }

     saveRDS(grl, paste0("grl.encode.sample", genome,".rds"))

 }
```

3. Liftover:

Perform liftover from hg38 to hg19 for the ChIP-seq binding sites

```
# chain downloaded from https://hgdownload.soe.ucsc.edu/goldenPath/hg38/liftOver/hg38ToHg19.over.ch
# need to reformat chain file from space to tab
# sed -r 's/^([0-9]+) ([0-9]+) ([0-9]+)$/\1\t\2\t\3/' hg38ToHg19.over.chain > hg38_to_hg19_tabs.chai

ch <- rtracklayer::import.chain(
    con = "chipatlas/data/hg38_to_hg19_tabs.chain")

# install liftover
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("liftOver")
library(liftOver)
```

```
grl_hg19 <- liftOver(grl, ch)
saveRDS(grl_hg19, file = "hg19_motif_bed_granges.rds")
```

4. Session information:

```
sessionInfo()
```

References

ChIP-Atlas 2021 update: a data-mining suite for exploring epigenomic landscapes by fully integrating ChIP-seq, ATAC-seq and Bisulfite-seq data. Zou Z, Ohta T, Miura F, Oki S. *Nucleic Acids Research. Oxford University Press (OUP);* 2022. doi:10.1093/nar/gkac199

ChIP-Atlas: a data-mining suite powered by full integration of public ChIP-seq data. Oki S, Ohta T, Shioi G, Hatanaka H, Ogasawara O, Okuda Y, Kawaji H, Nakaki R, Sese J, Meno C. *EMBO*; Vol. 19, EMBO reports. 2018. doi:10.15252/embr.201846255

ENCODE: https://www.encodeproject.org/

Cistrome Data Browser: expanded datasets and new tools for gene regulatory analysis. Zheng R, Wan C, Mei S, Qin Q, Wu Q, Sun H, Chen CH, Brown M, Zhang X, Meyer CA, Liu XS *Nucleic Acids Res*, 2018 Nov 20. doi:10.1093/nar/gky1094

Cistrome data browser: a data portal for ChIP-Seq and chromatin accessibility data in human and mouse. Mei S, Qin Q, Wu Q, Sun H, Zheng R, Zang C, Zhu M, Wu J, Shi X, Taing L, Liu T, Brown M, Meyer CA, Liu XS *Nucleic Acids Res*, 2017 Jan 4;45(D1):D658-D662. doi:10.1093/nar/gkw983

Examples

```
# check metada of dataset
tfBinding("mm10", metadata = TRUE)
# download data
## Not run:
tfBinding("mm10", "atlas")
tfBinding("mm10", "cistrome")

## End(Not run)
```

---

tfMotifs *Transcription factor motifs*

Description

Transcription factor motifs sets from the `chromVARmotifs` R package

Usage

```
tfMotifs(species = c("human", "mouse"), metadata = FALSE)
```

Arguments

`species`	character string specifying the species of interest
`metadata`	logical flag specifying whether to return data or metadata only

Format

`PWMatrixList` object containing information on transcription factor motifs. Contains the following experiments:

- **human_pwms_v2**: PWMatrixList object of length 1558
- **mouse_pwms_v2**: PWMatrixList object of length 1558

Details

This data set stores transcription factor motifs for human and mouse genome, which can be used with the package epiregulon to compute scores of transcription factor-regulatory element links.

Value

A list of position weight matrices, one for each transcription factor.

Data storage and access

The transcription factor motifs are stored separately for each species in .rds files encoding `PWMatrixList`. Data for both species are be accessed with the same function `tfMotifs`.

Data preparation

1. Data download:

Data sets were downloaded from https://github.com/GreenleafLab/chromVARmotifs/raw/master/data/ and format was changed to rds.

2. Data preparation:

The motifs were curated from the cisBP database. Position frequency matrices were converted to PWMs by taking the log of the frequencies (after adding a pseudocount of 0.008) divided by 0.25.

3. Session information:

```
sessionInfo()
```

References

Schep, A., Wu, B., Buenrostro, J. & Greenleaf, W. J. (2017) chromVAR: inferring transcription-factor-associated accessibility from single-cell epigenomic data. *Nature Methods* 14, 975–978. doi:0.1038/nmeth.4401

Examples

```
# check metada of dataset
tfMotifs("mouse", metadata = TRUE)
# download data
## Not run:
tfMotifs("human")

## End(Not run)
```

`writeSparseMatrix` *Write a sparse matrix*

Description

Writes a sparse matrix to file in a compressed sparse format.

Usage

```
writeSparseMatrix(
  x,
  file,
  name,
  chunk = 10000,
  column = TRUE,
  tenx = FALSE,
  guess.integer = TRUE
)
```

Arguments

`x`	A sparse matrix of some sort. This includes sparse DelayedMatrix objects.
`file`	String containing a path to the HDF5 file. The file is created if it is not already present.
`name`	String containing the name of the group to store `x`.
`chunk`	Integer scalar specifying the chunk size for the indices and values.
`column`	Logical scalar indicating whether to store as compressed sparse column format.
`tenx`	Logical scalar indicating whether to use the 10X compressed sparse column format.
`guess.integer`	Logical scalar specifying whether to guess an appropriate integer type from `x`.

Details

This writes a sparse matrix to file in various formats:

- `column = TRUE` and `tenx = FALSE` uses H5AD's `csr_matrix` format.
- `column = FALSE` and `tenx = FALSE` uses H5AD's `csc_matrix` format.
- `tenx = TRUE` uses 10X Genomics' HDF5 matrix format.

For the first two formats, the apparent transposition is deliberate, because columns in R are interpreted as rows in H5AD. This allows us to retain consistency the interpretation of samples (columns in R, rows in H5AD) and features (vice versa). Constructors for classes like H5SparseMatrix will automatically transpose so no extra work is required.

If `guess.integer = TRUE`, we attempt to save `x`'s values into the smallest type that will accommodate all of its values. If `x` only contains unsigned integers, we will attempt to save either 8-, 16- or 32-bit unsigned integers. If `x` contains signed integers, we will fall back to 32-bit signed integers. For all other values, we will fall back to double-precision floating point values.

We attempt to save `x`'s indices to unsigned 16-bit integers if the relevant dimension of `x` is small enough. Otherwise we will save it as an unsigned 32-bit integer.

Value

A `NULL` invisibly. The contents of `x` are written to `name` in `file`.

Note

This function has been kindly contributed by Aaron Lun.

Author(s)

Aaron Lun

Examples

```
library(Matrix)
x <- rsparsematrix(100, 20, 0.5)
tmp <- tempfile(fileext = ".h5")
scMultiome:::writeSparseMatrix(x, tmp, "csc_matrix")
scMultiome:::writeSparseMatrix(x, tmp, "csr_matrix", column = FALSE)
scMultiome:::writeSparseMatrix(x, tmp, "tenx_matrix", tenx = TRUE)

rhdf5::h5ls(tmp)
library(HDF5Array)
H5SparseMatrix(tmp, "csc_matrix")
H5SparseMatrix(tmp, "csr_matrix")
H5SparseMatrix(tmp, "tenx_matrix")
```

Index