# Package 'scMultiome'

November 10, 2026

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

**Version** 1.12.0

**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, GenomicRanges, HDF5Array, S4Vectors, checkmate, methods, rhdf5, alabaster.matrix

**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** 7.3.2

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

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

**git_branch** RELEASE_3_23

**git_last_commit** 92c60a9

**git_last_commit_date** 2026-04-28

**Repository** Bioconductor 3.23

**Date/Publication** 2026-11-10

**Author** Xiaosai Yao [cre, aut] (ORCID: <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 . . . . . 2
AR_drug . . . . . 4
assertHDF5 . . . . . 13
colonHealthy . . . . . 14
customClasses . . . . . 19
dummies . . . . . 20
fileOperations . . . . . 21
hematopoiesis . . . . . 23
listDatasets . . . . . 25
makeDataSetList . . . . . 26
PBMC_10x . . . . . 26
prostateENZ . . . . . 31
reprogramSeq . . . . . 36
retrieve . . . . . 37
RGtools . . . . . 38
TEADi_resistance . . . . . 38
templates . . . . . 48
tfBinding . . . . . 49
tfMotifs . . . . . 62

**Index** **64**

---

`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

- **AR_drug**: Response of prostate cancer cells to drug treatment
- **colonHealthy**: Single-cell analysis of samples from healthy human colon
- **hematopoiesis**: scATAC-seq and unpaired scRNA-seq of hematopoetic cells
- **PBMC_10x**: PBMC Data Set
- **prostateENZ**: LNCaP Cells Treated with Enzalutamide
- **reprogramSeq**: Reprogram-seq of LNCaP cells
- **TEADi_resistance**: Resistance of TEAD inhibitor to drug
- **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.sample_v2**: TF Binding Info hg19 by sample (ChIP-Atlas)
- **tfBinding_hg19_atlas.tissue**: TF Binding Info hg19 by tissue (ChIP-Atlas)
- **tfBinding_hg19_atlas.tissue_v2**: TF Binding Info hg19 by tissue (ChIP-Atlas)
- **tfBinding_hg19_encode.sample**: TF Binding Info hg19 by sample (ENCODE)
- **tfBinding_hg19_encode.sample_v2**: 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.sample_v2**: TF Binding Info hg38 by sample (ChIP-Atlas)
- **tfBinding_hg38_atlas.tissue**: TF Binding Info hg38 by tissue (ChIP-Atlas)
- **tfBinding_hg38_atlas.tissue_v2**: TF Binding Info hg38 by tissue (ChIP-Atlas)
- **tfBinding_hg38_encode.sample**: TF Binding Info hg38 by sample (ENCODE)
- **tfBinding_hg38_encode.sample_v2**: 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.sample_v2**: TF Binding Info mm10 by sample (ChIP-Atlas)
- **tfBinding_mm10_atlas.tissue**: TF Binding Info mm10 by tissue (ChIP-Atlas)
- **tfBinding_mm10_atlas.tissue_v2**: TF Binding Info mm10 by tissue (ChIP-Atlas)
- **tfBinding_mm10_encode.sample**: TF Binding Info mm10 by sample (ENCODE)
- **tfBinding_mm10_encode.sample_v2**: 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]

AR_drug *AR-dependent gene expression in prostate cancer cells*

## Description

Single cell gene expression and ATACseq from prostate cancer cell lines (LNCaP, VCaP, DU145, MDA-PCA-2B, 22Rv1 and NCI-H660). after 24h of drug treatment (enzalutamide, ARV110 (AR degrader) or SMARCA2_4.1 (SMARCA2/4 degrader) or DMSO control.

## Usage

```
AR_drug(
  metadata = FALSE,
 experiments = c("TileMatrix", "GeneScoreMatrix", "GeneExpressionMatrix", "PeakMatrix",
    "MotifMatrix", "TFPeaksDeviationsMatrix")
)
```

## 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 6068436 rows and 23118 columns
- **GeneScoreMatrix**: SingleCellExperiment with 57765 rows and 23118 columns
- **GeneExpressionMatrix**: SingleCellExperiment with 36451 rows and 23118 columns
- **PeakMatrix**: SingleCellExperiment with 237856 rows and 23118 columns
- **MotifMatrix**: SingleCellExperiment with 870 rows and 23118 columns
- **TFPeaksDeviationsMatrix**: SingleCellExperiment with 1533 rows and 23118 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

Data can be downloaded Gene Expression Omnibus (acc. no. GSE251977)

Downstream analysis was performed with the `ArchR` package:

1. **Initiate `ArchR` project:**

```
# attach ArchR package
library(ArchR)

# configure ArchR
addArchRGenome("hg38")

# create arrow file from fragment files
## list fragment files
fragments <- <FRAGMENT_FILES> # available in Gene Expression Omnibus (GSE251977)
## assign sample names
names(fragments) <- <SAMPLE_IDs>
## create arrows
createArrowFiles(inputFiles = fragments, sampleNames = names(fragments),
                     minTSS = 4, minFrags = 1000)

# specify output directory
outDir <- <OUTPUT_DIRECTORY>

# locate arrow files
arrows <- <ARROW_FILES>

doublet.score <- addDoubletScores(
    input = arrows,
    k = 10, #Refers to how many cells near a 'pseudo-doublet' to count.
   knnMethod = 'UMAP', #Refers to the embedding to use for nearest neighbor search.
    LSIMethod = 1
)

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

**2. HTO demultiplexing:**

```
library(ggplot2)
library(DropletUtils)
library(SingleCellExperiment)
library(Matrix)
library(ArchR)
library(zellkonverter)
library(scater)
library(gridExtra)
library(BiocParallel)


temp_dir <- tempdir()

destfiles_HTO <- c(file.path(temp_dir, "LIB5458339_SAM24418230.hashing.csv"),
                   file.path(temp_dir, "LIB5458340_SAM24418231.hashing.csv"),
```

```
                    file.path(temp_dir, "LIB5463784_SAM24425416.hashing.csv"),
                    file.path(temp_dir, "LIB5467656_SAM24427130.hashing.csv")
                    )

download.file(c("https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE251977&format=file&file=GSE2519
          "https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE251977&format=file&file=GSE251977%5F
          "https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE251977&format=file&file=GSE251977%5F
          "https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE251977&format=file&file=GSE251977%5F
             "",
             method = "libcurl"
             )

# download GEX data

destfiles_GEX <- c(file.path(temp_dir, "LIB5458339_SAM24418230_raw_feature_bc_matrix.h5"),
            file.path(temp_dir, "LIB5458340_SAM24418231_raw_feature_bc_matrix.h5"),
            file.path(temp_dir, "LIB5463784_SAM24425416_raw_feature_bc_matrix.h5"),
            file.path(temp_dir, "LIB5467656_SAM24427130_raw_feature_bc_matrix.h5")
                  )

download.file(c(),destfiles_GEX,method = "libcurl")

HTO_uri <- data.frame(SAMID = c("SAM24418230", "SAM24418231", "SAM24425416", "SAM24427130"), HTO_uri

arcseq_info <-data.frame(SAMID = c("SAM24418230", "SAM24418231", "SAM24425416", "SAM24427130"), uri =

# merge arcseq and HTO paths
merge_file_info <- merge(HTO_uri, arcseq_info, by.x = "SAMID" , by.y = "sampleName")
merge_file_info <- merge_file_info[order(merge_file_info$SAMID), ]



######## load matrices
# load RNA matrix

hashing_qc <- list()
umapplot <- list()
seRNA_final <- list()
for (i in seq_len(nrow(merge_file_info))) {
    message(merge_file_info$SAMID[i])
    # import gex
    seRNA <- ArchR::import10xFeatureMatrix(
       input = file.path( merge_file_info$uri[i], "raw_feature_bc_matrix.h5"),
        names = merge_file_info$SAMID[i])
    names(assays(seRNA)) <- "counts"

    # import HTO and convert to sce
    HTO <- data.table::fread(merge_file_info$HTO_uri[i])
```

```
  rownames_HTO <- HTO$Antibody
  HTO <- HTO[,-1]
  colnames(HTO) <- paste0(merge_file_info$SAMID[i], "#", colnames(HTO),"-1")
  HTO <- as(as.matrix(HTO), "dgCMatrix")
  rownames(HTO) <- rownames_HTO
  HTO <- SingleCellExperiment(assays = list(counts=HTO))

  # merge GEX and HTO into a SCE
  common_cells <- intersect(colnames(HTO), colnames(seRNA))
  HTO <- HTO[, common_cells]
  seRNA <- seRNA[, common_cells]
  seRNA <- as(seRNA, "SingleCellExperiment")
  altExp(seRNA, "HTO") <- HTO

  # call empty droplets to define ambient droplets
  set.seed(10010)
  e.out.gene <- emptyDrops(counts(seRNA), by.rank = 30000 )
  is.cell <- e.out.gene$FDR <= 0.001
  summary(is.cell)

  # plot empty droplet assignments
  par(mfrow=c(1,2))
  r <- rank(-e.out.gene$Total)
plot(r, e.out.gene$Total, log="xy", xlab="Rank", ylab="Total gene count", main="")
  abline(h=metadata(e.out.gene)$retain, col="darkgrey", lty=2, lwd=2)
  hist(log10(e.out.gene$Total[is.cell]), xlab="Log[10] gene count", main="")

  # Estimate HTO ambient proportions using empty droplets
  hto.mat <- assay(altExp(seRNA),"counts")[,which(is.cell)]
ambient <- proportions(rowSums(assay(altExp(seRNA), "counts")[,is.na(e.out.gene$FDR)]))
  # plot ambient proportions
  barplot(ambient,las=2, main="ambient proportion")
  hash.stats <- hashedDrops(hto.mat, ambient=ambient)
  table(hash.stats$Best[hash.stats$Confident])

  # examine hashing
  colors <- rep("grey", nrow(hash.stats))
  colors[hash.stats$Doublet] <- "red"
  colors[hash.stats$Confident] <- "black"

hashing_qc[[merge_file_info$SAMID[i]]] <- plot(hash.stats$LogFC, hash.stats$LogFC2,
                                xlab="Log fold-change from best to second HTO",
                                ylab="Log fold-change of second HTO over ambient",
                                               col=colors,
                                            main=merge_file_info$SAMID[i])

  # keep only non-empty cells
  seRNA <- seRNA[, which(is.cell)]
```

```
    colData(seRNA) <- cbind(colData(seRNA), hash.stats)
   colData(seRNA)$library <- sapply(strsplit(colnames(seRNA), split = "#"), "[",1)

   assay(altExp(seRNA), "logcounts") <- log10(assay(altExp(seRNA), "counts")+1)
    assay(altExp(seRNA), "clr") <- sweep(assay(altExp(seRNA), "logcounts"), 2,
                              colMeans(assay(altExp(seRNA), "logcounts")), "-")
   seRNA <- runUMAP(seRNA, altexp = "HTO", name="UMAP_HTO", assay.type = "clr", exprs_values = "clr")
    seRNA$hash_assignment <- rownames_HTO[seRNA$Best]
   umapplot[[merge_file_info$SAMID[i]]] <- plotReducedDim(seRNA[, which(seRNA$Doublet == FALSE & seRN
                                                          dimred = "UMAP_HTO",
                                                  color_by = "hash_assignment",
                                              point_size=0.5, rasterise=TRUE) +
        ggtitle(merge_file_info$SAMID[i])

    # save seRNA
    seRNA_final[[merge_file_info$SAMID[i]]] <- seRNA
}

seRNA_final <- do.call(cbind, seRNA_final)


library(ArchR)
seRNA_final$hash_assignment2 <- paste0(seRNA_final$library, seRNA_final$hash_assignment)


# copy ArchR project from /gstore/data/genomics/congee_rest_runs/6679b364f79a145e53521a61/ArchR_outp
proj <- ArchR::loadArchRProject("OUTPUT/ArchRProject/")
common <- intersect(proj$cellNames, colnames(seRNA_final))
proj <- proj[common,]

# add HTO information

for (row_data in colnames(colData(seRNA_final))){
    proj <- addCellColData(
        ArchRProj = proj,
        data = colData(seRNA_final)[common, row_data],
        cells = common,
        name = row_data,
        force = TRUE
    )
}

#filter out doublets and non-confident calls
proj <- proj[which(proj$Confident == TRUE & proj$Doublet == FALSE), ]
```

**3. Clustering:**

```
# filter doublets
archr.proj <- filterDoublets(
```

```
    ArchRProj = archr.proj,
    cutEnrich = 1,
    cutScore = -Inf,
    filterRatio = 1
)

# filter cells that do not contain rna
archr.proj <- archr.proj[!is.na(archr.proj$Gex_nUMI)]

# add reduced dims
archr.proj <- addIterativeLSI(
    ArchRProj = archr.proj,
    useMatrix = 'TileMatrix',
    name = 'IterativeLSI_TileMatrix',
    threads = 4,
    seed = 2,
    force = TRUE
)

archr.proj <- addIterativeLSI(
    ArchRProj = archr.proj,
    useMatrix = 'GeneExpressionMatrix',
    name = 'IterativeLSI_GeneExpressionMatrix',
    firstSelection = "variable",
    depthCol = "Gex_nUMI",
    varFeatures = 2500,
    binarize = FALSE,
    threads = 4,
    seed = 2,
    force = TRUE
)

archr.proj <- addCombinedDims(
    archr.proj,
   reducedDims = c('IterativeLSI_TileMatrix', 'IterativeLSI_GeneExpressionMatrix'),
    name =  'IterativeLSI_Combined'
)


# add clusters
archr.proj <- addClusters(
input = archr.proj,
reducedDims = 'IterativeLSI_Combined',
name = 'Clusters_Combined',
seed = 2,
prefix = 'Combined_C',
force = TRUE,
method = "scran"
```

```
)
cM <- confusionMatrix(getCellColData(archr.proj)[,'hash_assignment2'], archr.proj$Sample)
cM <- cM / Matrix::rowSums(cM)
pheatmap::pheatmap(
    mat = as.matrix(cM),
    color = paletteContinuous('whiteBlue'),
    border_color = 'black',
   filename = paste0(getOutputDirectory(archr.proj),'/Plots/GPSA-Sample-hash_assignment2-pheatmap.p
)


# add embeddings

archr.proj <- addUMAP(
    ArchRProj = archr.proj,
    reducedDims = 'IterativeLSI_Combined',
    name = 'UMAP_Combined',
    seed = 2,
    threads = 1,
    force = TRUE
)

archr.proj <- addUMAP(
    ArchRProj = archr.proj,
    reducedDims = 'IterativeLSI_TileMatrix',
    name = 'UMAP_ATAC',
    seed = 2,
    threads = 1,
    force = TRUE
)


archr.proj <- addUMAP(
    ArchRProj = archr.proj,
    reducedDims = 'IterativeLSI_GeneExpressionMatrix',
    name = 'UMAP_RNA',
    seed = 2,
    threads = 1
)



# tSNE
archr.proj <- addTSNE(
    ArchRProj = archr.proj,
    reducedDims = 'IterativeLSI_Combined',
    name = 'TSNE_Combined',
    perplexity = 30,
```

```
    seed = 2,
    force = TRUE,
    threads = max(floor(4/2), 1)
)
```

**4. Clustering and motif annotation:**

```
# Peak calling

library(BSgenome.Hsapiens.Genentech.GRCh38)

archr.proj <- addGroupCoverages(
    ArchRProj = archr.proj,
    groupBy = 'hash_assignment2',
    threads = 4
)


archr.proj <- ArchR.helper::addReproduciblePeakSet(
    ArchRProj = archr.proj,
    groupBy = 'hash_assignment2',
    peakMethod = "MACSr",
    excludeChr = c('chrMT','chrY'),
    genomeSize = 2.7e9,
    threads = 4,
    force = TRUE
)

archr.proj <- addPeakMatrix(
    ArchRProj = archr.proj,
    binarize = FALSE,
    threads = 1,
    force = TRUE
)

# TF annotation
peaks.anno <- scMultiome::tfBinding()
archr.proj <- addPeakAnnotations(
    ArchRProj = archr.proj,
    regions = peaks.anno,
    name = 'TF_peaks',
    force = TRUE
)

archr.proj <- addDeviationsMatrix(
    ArchRProj = archr.proj,
    peakAnnotation = 'TF_peaks',
    matrixName = 'TFPeaksDeviationsMatrix',
```

```
    threads = 1,
    force = TRUE
)


# motif annotation
archr.proj <- addMotifAnnotations(ArchRProj = archr.proj, motifSet = 'cisbp', name = 'Motif', species

archr.proj <- addDeviationsMatrix(
    ArchRProj = archr.proj,
    peakAnnotation = 'Motif',
    threads = 1,
    force = TRUE
)



#add BigWigs

getGroupBW(
    ArchRProj = archr.proj,
    groupBy = "hash_assignment2",
    normMethod = "ReadsInTSS",
    threads = 1,
)


getGroupBW(
    ArchRProj = archr.proj,
    groupBy = 'hash_assignment2',
    normMethod = 'ReadsInTSS',
    threads = 1
)

#add extra  cell information
sample_info <- read.csv("HTO_SAMID.csv") # to be dowloaded from Gene Expression Omnibus, GSE251977
sample_info$SAMID_HTO <- paste0(sample_info$SAMID, sample_info$HTO)

archr.proj$TREATMENT <- sample_info$TREATMENT[match(archr.proj$hash_assignment2, sample_info$SAMID_
archr.proj$Cell <- unlist(lapply(strsplit(archr.proj$TREATMENT, split = "-"),"[",1))
archr.proj$TEST_ARTICLE <- unlist(lapply(strsplit(archr.proj$TREATMENT, split = "-"),"[",2))

archr.proj$Cell[archr.proj$Cell == "22RV1"] <- "22Rv1"
```

**5. Save results:**

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

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

### 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

Tomasz Włodarczyk, Aaron Lun, Diana Wu, Shreya Menon, Shushan Toneyan, Kerstin Seidel, Liang Wang, Jenille Tan, Shang-Yang Chen, Timothy Keyes, Aleksander Chlebowski, Yu Guo, Ciara Metcalfe, Marc Hafner, Christian W. Siebel, M. Ryan Corces, Robert Yauch, Shiqi Xie, Xiaosai Yao. 2023. "Inference of single-cell transcription factor activity to dissect mechanisms of lineage plasticity and drug response" bioRxiv 2023.11.27.568955; doi: https://doi.org/10.1101/2023.11.27.568955

### Examples

```
# check metada of dataset
AR_drug(metadata = TRUE)

# download data
 ## Not run:
AR_drug()

## End(Not run)
```

---

## `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.

## PBMC_10x *10k PBMC data*

### Description

PBMC from a Healthy Donor downloaded from 10x Genomics. Granulocytes were removed by cell sorting. Paired ATAC and Gene Expression libraries were generated from the isolated nuclei. Targeted nuclei recovery was 10,000. Data source: https://www.10xgenomics.com/datasets/pbmc-from-a-healthy-donor-granulocytes-removed-through-cell-sorting-10-k-1-standard-2-0-0

### Usage

```
PBMC_10x(
  metadata = FALSE,
 experiments = c("TileMatrix", "GeneScoreMatrix", "GeneExpressionMatrix", "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 9702 columns
- **GeneScoreMatrix**: SingleCellExperiment with 24919 rows and 9702 columns
- **GeneExpressionMatrix**: SingleCellExperiment with 36438 rows and 9702 columns
- **PeakMatrix**: SingleCellExperiment with 159290 rows and 9702 columns
- **MotifMatrix**: SingleCellExperiment with 870 rows and 9702 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

**1. Download data:**

```
# specify output directory
outDir <- <OUTPUT_DIRECTORY>

# dowbnload data from 10XGenomics server
download.file(c("https://cf.10xgenomics.com/samples/cell-arc/1.0.0/pbmc_granulocyte_sorted_10k/pbm
           "https://cf.10xgenomics.com/samples/cell-arc/1.0.0/pbmc_granulocyte_sorted_10k/pbmc_gra
              method="libcurl")
```

**2. Initiate ArchR project:**

```
library(ArchR)

# configure ArchR
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))

# locate arrow files
arrows <- <ARROW_FILES>

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

**3. Add gene expression data:**

```
seRNA <- import10xFeatureMatrix(
 input = c(file.path(outDir, "pbmc_granulocyte_sorted_10k_filtered_feature_bc_matrix.h5")),
  names = c("PBMC_10k")
)

# filter out genes which are expressed in less than 3 cells
seRNA <- seRNA[colSums(assay(seRNA))>2,]

proj <- addGeneExpressionMatrix(input = proj, seRNA = seRNA, force = TRUE)
```

**4. Quality control:**

```
#Filter Cells
proj <- proj[proj$TSSEnrichment > 6 & proj$nFrags > 2500 & !is.na(proj$Gex_nUMI)]

#Doublet Filtration
proj <- addDoubletScores(proj)
proj <- filterDoublets(proj)
```

**5. Clustering and dimensionality reduction:**

```
#LSI-ATAC
proj <- addIterativeLSI(
  ArchRProj = proj,
  clusterParams = list(
    resolution = 0.2,
    sampleCells = 10000,
    n.start = 10
  ),
  saveIterations = FALSE,
  useMatrix = "TileMatrix",
  depthCol = "nFrags",
  name = "LSI_ATAC"
)

#LSI-RNA
proj <- addIterativeLSI(
  ArchRProj = proj,
  clusterParams = list(
    resolution = 0.2,
    sampleCells = 10000,
    n.start = 10
  ),
  saveIterations = FALSE,
  useMatrix = "GeneExpressionMatrix",
  depthCol = "Gex_nUMI",
  varFeatures = 2500,
  firstSelection = "variable",
```

```
  binarize = FALSE,
  name = "LSI_RNA"
)

#Combined Dims
proj <- addCombinedDims(proj, reducedDims = c("LSI_ATAC", "LSI_RNA"), name =  "LSI_Combined")

#UMAPs
proj <- addUMAP(proj, reducedDims = "LSI_ATAC", name = "UMAP_ATAC", minDist = 0.8, force = TRUE)

proj <- addUMAP(proj, reducedDims = "LSI_RNA", name = "UMAP_RNA", minDist = 0.8, force = TRUE)

proj <- addUMAP(proj, reducedDims = "LSI_Combined", name = "UMAP_Combined", minDist = 0.8, force = TRUE

#Add Clusters
proj <- addClusters(proj, reducedDims = "LSI_Combined", name = "Clusters", resolution = 0.4, force = TR
#proj <- addClusters(proj, reducedDims = "LSI_RNA", name = "Clusters_genes", resolution = 0.4, force =
```

**6. Peak calling:**

```
library(BSgenome.Hsapiens.UCSC.hg38)

proj<- addGroupCoverages(ArchRProj = proj, groupBy = "Clusters")

proj <- addReproduciblePeakSet(
  ArchRProj = proj,
  groupBy = "Clusters",
  pathToMacs2 = pathToMacs2,
  force = TRUE,
  threads = 10
)

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

proj <- addPeakMatrix(proj)

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

# calculate per-cell devations of motif annotations
proj <- addDeviationsMatrix(proj, peakAnnotation = "Motif", force = TRUE)
```

**7. Cell type annotation:**

```
GeneExpressionMatrix <- getMatrixFromProject(proj, useMatrix = "GeneExpressionMatrix")

# Loading reference data with Ensembl annotations.
library(celldex)
```

```
#ref.data <- HumanPrimaryCellAtlasData(ensembl=FALSE)
bpe_data <- BlueprintEncodeData()


expr_assay <- assays(GeneExpressionMatrix)[[1]]
rownames(expr_assay) <- rowData(GeneExpressionMatrix)$name

# Performing predictions.
library(SingleR)
predictions <- SingleR(test=expr_assay, assay.type.test=1,
               ref=bpe_data, labels=bpe_data$label.fine) # could be switched to label.main

proj$cell_type_SingleR <- predictions$labels



library(ArchR)
library(scater)
library(SingleCellExperiment)
library(epiregulon.archr)

GeneExpressionMatrix <- getMatrixFromProject(proj, "GeneExpressionMatrix")
GeneExpressionMatrix <- ArchRMatrix2SCE(GeneExpressionMatrix, rename="normalizedCounts")
reducedDim(GeneExpressionMatrix, "UMAP_Combined") <- getEmbedding(proj, embedding = "UMAP_Combined")
GeneExpressionMatrix$cell_type[is.na(GeneExpressionMatrix$cell_type)] <- "unknown"
plotReducedDim(GeneExpressionMatrix, dimred="UMAP_Combined", colour_by = "Clusters", text_by = "Clus
plotReducedDim(GeneExpressionMatrix, dimred="UMAP_Combined", colour_by = "cell_type_SingleR", text_


# identify cell types in cluster 10
GeneExpressionMatrix$C10 <- GeneExpressionMatrix$cell_type_SingleR
GeneExpressionMatrix$C10[GeneExpressionMatrix$Clusters!="C10"] <- NA
plotReducedDim(GeneExpressionMatrix, dimred="UMAP_Combined", colour_by = "C10", text_by = "C10")

# identify cell types in cluster 11
GeneExpressionMatrix$C11 <- GeneExpressionMatrix$cell_type_SingleR
GeneExpressionMatrix$C11[GeneExpressionMatrix$Clusters!="C11"] <- NA
plotReducedDim(GeneExpressionMatrix, dimred="UMAP_Combined", colour_by = "C11", text_by = "C11")
table(GeneExpressionMatrix$C11)

# identify cell types in cluster 1
GeneExpressionMatrix$C1 <- GeneExpressionMatrix$cell_type_SingleR
GeneExpressionMatrix$C1[GeneExpressionMatrix$Clusters!="C1"] <- NA
table(GeneExpressionMatrix$C1)

clusters <- proj$Clusters
manual_annotation <- rep(NA, length(clusters))
manual_annotation[clusters %in% "C6"] <- "Naive CD4+ T"
```

```
manual_annotation[clusters %in% c("C13")] <- "CD14+ Mono"
manual_annotation[clusters %in% "C12"] <- "Monocytes"
manual_annotation[clusters %in% c("C2", "C3")] <- "B"
manual_annotation[clusters %in% "C4"] <- "Memory CD8+ T"
manual_annotation[clusters %in% "C14"] <- "FCGR3A+ Mono"
manual_annotation[clusters %in% "C5"] <- "NK"
manual_annotation[clusters %in% c("C9", "C10")] <- "Memory CD4+ T"
manual_annotation[clusters %in% c("C7", "C8")] <- "Naive CD8+ T"
manual_annotation[clusters %in% c("C1")] <- "DC"
proj$cell_type <- manual_annotation
proj <- proj[!is.na(proj$cell_type)]
```

### 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.

### Examples

```
# check metada of dataset
PBMC_10x(metadata = TRUE)
# download data
 ## Not run:
PBMC_10x()

## End(Not run)
```

---

## 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.

## TEADi_resistance *TEAD inhibitor resistance*

### Description

Single cell multiomics on mesothelioma cell line H226 sensitive and resistant to the TEAD inhibitor GNE-7883

### Usage

```
TEADi_resistance(
  metadata = FALSE,
 experiments = c("TileMatrix", "GeneScoreMatrix", "GeneExpressionMatrix", "PeakMatrix",
    "MotifMatrix", "TFPeaksDeviationsMatrix", "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:

- **TileMatrix**: SingleCellExperiment with 6068436 rows and 4952 columns
- **GeneScoreMatrix**: SingleCellExperiment with 57765 rows and 4952 columns
- **GeneExpressionMatrix**: SingleCellExperiment with 36451 rows and 4952 columns
- **PeakMatrix**: SingleCellExperiment with 103723 rows and 4952 column
- **MotifMatrix**: SingleCellExperiment with 870 rows and 4952 columns
- **TFPeaksDeviationsMatrix**: SingleCellExperiment with 1269 rows and 4952 columns
- **TF_bindingMatrix**: SingleCellExperiment with 1504 rows and 4952 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

The following workflow was created based on the original code written by Julien Tremblay (julien.tremblay@contractors.roch

**1. Initiate ArchR project:**

```
# attach ArchR package
library(ArchR)

# configure ArchR
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),
                     minTSS = 4, minFrags = 1000)

# specify output directory
outDir <- <OUTPUT_DIRECTORY>

# locate arrow files
```

```
arrows <- <ARROW_FILES>

doublet.score <- addDoubletScores(
    input = arrows,
    k = 10, #Refers to how many cells near a 'pseudo-doublet' to count.
   knnMethod = 'UMAP', #Refers to the embedding to use for nearest neighbor search.
    LSIMethod = 1
)

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

**2. Add further standard analysis to the ArchR Project:**

This part of code is wrapped by one of the internal libraries.

```
# .h5 files to be downloaded from GEO (GSE247442)
se.rna <- import.archr.10x.se(h5.files = rna.files, sample.names = names(rna.files))

# filter RNA so that only cells which also have ATAC-seq are included
se.rna <- se.rna[,which(colnames(se.rna) %in% rownames(ArchR::getCellColData(archr.proj)))]

# Filter out scaffolds that don't have at least 3 genes
filtered.chr.names <- names(which(table(seqnames(se.rna)) >= 3));
se.rna <- se.rna[as.character(seqnames(se.rna)) %in% filtered.chr.names,];
rowRanges(se.rna) <- keepSeqlevels(rowRanges(se.rna), filtered.chr.names, pruning.mode = 'coarse');

archr.proj <- addGeneExpressionMatrix(
    input = project,
    seRNA = se.rna,
    threads = num.threads
);

# LSI dimensionality reduction
archr.proj <- addIterativeLSI(
    ArchRProj = project,
    useMatrix = 'TileMatrix',
    name = paste0('IterativeLSI_ATAC'),
    seed = 2,
    threads = num.threads
)

# Add clusters to colData
archr.proj <- addClusters(
    input = project,
    reducedDims = paste0('IterativeLSI_ATAC'),
    name = paste0('Clusters',atac.name.suffix),
    seed = 2
)
```

**3. Demultiplex HTO counts:**

```
# HTO counts were generated using internal cumulus cellranger workflow and the results are
# available on Gene Expression Omnibus, GSE247442
# This was equivalent to run in the command line:
#generate_count_matrix_ADTs \
#   /gstore/apps/CellRanger/7.1.0/lib/python/cellranger/barcodes/737K-arc-v1.txt.gz \
#    ../data/hashing_index.csv \
#    ../data/hto/raw_reads_LIB5457060_SAM24417357 \
#    LIB5457060_SAM24417357 \
#   -p 4 --max-mismatch-feature 2 --feature antibody --max-mismatch-cell 1 --umi-length 12
#
#generate_count_matrix_ADTs \
#   /gstore/apps/CellRanger/7.1.0/lib/python/cellranger/barcodes/737K-arc-v1.txt.gz \
#    ../data/hashing_index.csv \
#    ../data/hto/raw_reads_LIB5457058_SAM24417355 \
#    LIB5457058_SAM24417355 \
#   -p 4 --max-mismatch-feature 2 --feature antibody --max-mismatch-cell 1 --umi-length 12

library(zellkonverter)
library(ArchR)
library(DropletUtils)
library(SingleCellExperiment)
library(Matrix)
library(BiocParallel)
library(data.table)
library(dplyr)

hto_files = list.files(hto_path, pattern="*.hashing.csv", recursive=TRUE, full.names=TRUE)
hto_info2 = data.frame(hto_files)
hto_info2$sample = gsub(".*_(SAM\\d+).*", "\\1", hto_info2$hto_files)


arcseq_files = list.files(paste0(arcseq_info$uri, "/"), pattern="*raw_feature_bc_matrix", recursive=
if(!is.null(skip_sample)){
    arcseq_files = arcseq_files[!grepl("SAM24417356", arcseq_files)]
}
arcseq_info2 = data.frame(arcseq_files)
arcseq_info2$sample = gsub(".*(SAM\\d+).*", "\\1", arcseq_info2$arcseq_files)

samples = gsub(".*_(SAM\\d+).*", "\\1", hto_files)
summary_df_final = NULL
seRNA_objects = list()
#figures = list()
tables = list()
empty_drops = list()
hashed_drops = list()
```

```
for(i in 1:length(samples)){
        summary_df = NULL

        curr_sample = samples[i]
        message("Processing ", curr_sample)
        dir.create(paste0(outdir, "/", curr_sample, "/rds"), recursive=TRUE)

     curr_arcseq_file = arcseq_info2[arcseq_info2$sample == curr_sample,]$arcseq_files
      curr_hto_file   = hto_info2[hto_info2$sample == curr_sample,]$hto_files
        ############################
        # import gex              #
        ############################
        message("...Importing arcseq matrix")
        seRNA <- ArchR::import10xFeatureMatrix(
            input = file.path(curr_arcseq_file),
            names = curr_sample
        )
        names(assays(seRNA)) = "counts"
      # coerce seRNA obs to a SingleCellExperiment because it is a RangedSingleCellExperiment at this p
        seRNA = as(seRNA, "SingleCellExperiment")
      # Here if you look at the seRNA obj, there are approx 725k single-cells: at this point, they are NO
        # vast majority of them are empty (processed below)

        ##########################################
        # import HTO and convert to sce object.  #
        ##########################################
        hto = data.frame(fread(curr_hto_file, header=T), check.names=FALSE)
        rownames_hto = hto$Antibody
        hto = hto[,-1]
      # Here remember that each hto also contains a cell barcode sequence. in the next line,
      # we are formatting each cell barcode the way as it is formatted in the seRNA object
      # will be used later to only keep cell barcodes for which we have an assigned hto.
        colnames(hto) = paste0(curr_sample, "#", colnames(hto), "-1")
        hto = as(as.matrix(hto), "dgCMatrix")
        rownames(hto) = rownames_hto
        hto = SingleCellExperiment(assays = list(counts=hto))

        # merge GEX and HTO into a SCE
        # Only keep seRNA cells that are found in hto data.
        common_cells = intersect(colnames(hto), colnames(seRNA))

        # Then, narrow selection to common cells.
        hto = hto[, common_cells] # row=features, col=cells
      # Same selection with seRNA obj (remember both SingleCellExperiment and SummarizedExperiment cla
        seRNA = seRNA[, common_cells]
      # coerce seRNA obs to a SingleCellExperiment because it is a RangedSingleCellExperiment at this p
        seRNA = as(seRNA, "SingleCellExperiment")
        # Add HTO data as altExp attribute, see doc for more details.
```

```
   altExp(seRNA, "HTO") = hto

   ##################################################
   # Distinguish single cells from empty droplets.   #
   ##################################################
   # Remove all zeros
# We can't remove rows at this point, only col (i.e. single cells) at this point. because downstre
   seRNA = seRNA[, Matrix::colSums(assays(seRNA)$counts) > 0]
   # Have a look at barcode ranks:
   empty_thresh = 100 # remove more invalid cells.
   bc_ranks = barcodeRanks(counts(seRNA), lower=empty_thresh)

# Add colData to seRNA obj (i.e. bc_ranks hold barcodes in the same order as it was in the seRNA ob
   colData(seRNA)$BarcodeRank   = bc_ranks$rank
   colData(seRNA)$BarcodeTotal  = bc_ranks$total
   colData(seRNA)$BarcodeFitted = bc_ranks$fitted


# Barcodes that contain more than retain total counts are always retained. This ensures that large
# are very similar to the ambient pool are not inadvertently discarded. If retain is not specified
# count at the knee point detected by barcodeRanks. Manual specification of retain may be useful i
# identified in complex log-rank curves. Users can also set retain=Inf to disable automatic reten

   curr_rank = 30000
   emp_drops = emptyDrops(counts(seRNA), lower=NULL, niters=100000,
               test.ambient = TRUE, BPPARAM=SerialParam(), by.rank=curr_rank)
   emp_fdr = 0.01
   is_cell = emp_drops$FDR <= emp_fdr
is_cell[is.na(is_cell)] = FALSE # NA means its not a cell, so force NAs to FALSE
   empty_drops[[curr_sample]] = emp_drops

seRNA_filt = runUMAP(seRNA_filt, altexp = "HTO", name="UMAP_HTO", exprs_values = "clr")

   seRNA_filt = seRNA[, which(is_cell)]

   # Estimate HTO ambient proportions using empty droplets
   hto_mat = assay(altExp(seRNA),"counts")[, which(is_cell)]
# confirm that by ambient, we mean non-cells... or just really low abundant stuff.
   # Ok, from the documentation, I believe ambient = non-cells HTOs.
   # Get % of each HTO
   ambient_hto_mat = assay(altExp(seRNA), "counts")[,!is_cell]
ambient_hto_prop = proportions(rowSums(assay(altExp(seRNA), "counts")[,!is_cell]))
   hash_stats = hashedDrops(hto_mat, ambient=ambient_hto_prop,
                            doublet.nmads=3,
                            doublet.min=2,
                            confident.nmads=1,
                            confident.min=1
   )
```

```
    colData(seRNA_filt) = cbind(colData(seRNA_filt), hash_stats)
   colData(seRNA_filt)$library = sapply(strsplit(colnames(seRNA_filt), split = "#"), "[",1)

   assay(altExp(seRNA_filt), "logcounts") = log10(assay(altExp(seRNA_filt), "counts")+1)
   # Here see the doc on sweep. But basically for each row the logcount is subtracted from mean log count
    assay(altExp(seRNA_filt), "clr") = sweep(
                                        assay(altExp(seRNA_filt), "logcounts"),
                                         2,
                                colMeans(assay(altExp(seRNA_filt), "logcounts")),
                                         "-"
                                       )
   seRNA_filt = runUMAP(seRNA_filt, altexp = "HTO", name="UMAP_HTO", exprs_values = "clr")
    seRNA_filt$hash_assignment = rownames_hto[seRNA_filt$Best]
    # save output
   saveRDS(seRNA_filt, paste0(outdir, "/", curr_sample, "/rds/", curr_sample, "_demultiplex_hto_and_
}
```

**4. Merge HTO data into the ArchR project:**

```
library(ArchR)
library(SingleCellExperiment)
library(data.table)

seRNA_objs = list()
for(sample in samples){
   curr_seRNA_obj = readRDS(paste0("./output/", sample, "/rds/", sample, "_demultiplex_hto_and_scRNA
    seRNA_objs[[sample]] = curr_seRNA_obj
}

# Here load the multiple seRNA objects from last step and combine them.
seRNA_final = do.call(cbind, seRNA_objs)

seRNA_final$hash_assignment2 = paste0(seRNA_final$library, "_", seRNA_final$hash_assignment)

common = intersect(project$cellNames, colnames(seRNA_final))
project = project[common,]

# Here at this point, we have the archr object and seRNA obj with the same cells.
# add HTO information
for(curr_cell_attribute in colnames(colData(seRNA_final))){
    proj = addCellColData(
        ArchRProj = project,
        data = colData(seRNA_final)[common, curr_cell_attribute],
        cells = common,
        name = curr_cell_attribute,
        force = TRUE
```

```
    )
}

#filter out doublets and non-confident calls
project = project[which(project$Confident == TRUE & proj$Doublet == FALSE), ]

# mapping_file.tsv can be dowloaded from GEO (GSE247442)

mapping = data.frame(fread("mapping_file.tsv", header=T, sep="\t"), check.names=F)
row.names(mapping) = mapping$SampleID
mapping$hash_assignment2 = mapping$SampleID

tmp_df = data.frame(getCellColData(project, select=c("hash_assignment2")))
tmp_df2 = join(tmp_df, mapping, by="hash_assignment2")
project$CNAME = tmp_df2$CNAME
project$TEST_ARTICLE = tmp_df2$TEST_ARTICLE
project$HTO = tmp_df2$HTO
project$Treatment = paste0(project$CNAME, "_", project$TEST_ARTICLE)
project$Treatment_HTO = paste0(project$CNAME, "_", project$TEST_ARTICLE, " (", project$HTO, ")")
project$Treatment_library = paste0(project$Treatment, "_", project$library)
```

**5. Cluster annotation:**

```
# mapping_file.tsv can be dowloaded from GEO (GSE247442)
mapping = data.frame(fread("mapping_file.tsv", header=T, sep="\t"), check.names=F)
samples_vColors = vColors[1:length(unique(mapping[["Treatment_HTO"]]))]
names(samples_vColors) = unique(mapping[["Treatment_HTO"]])

embed = getEmbedding(project, embedding=paste0("UMAP_TileMatrix"))
embed = cbind(embed, getCellColData(project, select=c("Treatment", "HTO", "Treatment_HTO", "Treatmen
colnames(embed)[1] = "UMAP_dim_1"
colnames(embed)[2] = "UMAP_dim_2"

my_labels_df = data.frame(Treatment_HTO=unique(embed$Treatment_HTO),label=paste0("", unique(embed$T
my_labels = embed %>%
    dplyr::group_by(Treatment_HTO) %>%
   dplyr::summarize(UMAP_dim_1 = median(UMAP_dim_1), UMAP_dim_2 = median(UMAP_dim_2)) %>%
    dplyr::left_join(my_labels_df) %>% as.data.frame()

p_init_samples = NULL; p_init_samples = ggplot(embed, aes(x=UMAP_dim_1, y=UMAP_dim_2, color=Treatmen
    geom_point(size=0.3) +
    scale_color_manual(values=samples_vColors) +
    xlab(paste0("UMAP 1")) +
    ylab(paste0("UMAP 2")) +
    #ggtitle(paste0("UMAP; ", curr_signature)) +
    guides(color=guide_legend(ncol=2, override.aes=list(size=1.5))) +
   ggrepel::geom_label_repel(data=my_labels, box.padding=0.0, alpha=0.8, label.size=0.05, fill="gray
    theme_minimal() +
```

```
   theme(panel.border=element_rect(fill=NA, linetype="solid", colour = "black", linewidth=0.5), axis
       axis.text=element_text(size=8, angle=0), strip.text.y=element_text(size=8, angle=0), legend.p
       legend.text=element_text(size=8, angle=0), plot.title=element_text(size=10), legend.key.size

my_labels_df2 = data.frame(Clusters_TileMatrix=unique(embed$Clusters_TileMatrix),label=paste0("", u
my_labels2 = embed %>%
    dplyr::group_by(Clusters_TileMatrix) %>%
   dplyr::summarize(UMAP_dim_1 = median(UMAP_dim_1), UMAP_dim_2 = median(UMAP_dim_2)) %>%
    dplyr::left_join(my_labels_df2) %>% as.data.frame()

clusters_vColors = vColors20[1:length(unique(mapping[["Clusters_TileMatrix"]]))]
names(clusters_vColors) = unique(mapping[["Clusters_TileMatrix"]])

p_init_clusters = NULL; p_init_clusters = ggplot(embed, aes(x=UMAP_dim_1, y=UMAP_dim_2, color=.data[[
    geom_point(size=0.3) +
    scale_color_manual(values=clusters_vColors) +
    xlab(paste0("UMAP 1")) +
    ylab(paste0("UMAP 2")) +
    #ggtitle(paste0("UMAP; ", curr_signature)) +
   guides(color = guide_legend(override.aes=list(size=1.5))) + guides(fill=guide_legend(ncol=3)) + #
   ggrepel::geom_label_repel(data=my_labels2, fill="gray70", box.padding=0.0, alpha=0.8, label.size=
    theme_minimal() +
   theme(panel.border=element_rect(fill=NA, linetype="solid", colour = "black", linewidth=0.5), axis
       axis.text=element_text(size=8, angle=0), strip.text.y=element_text(size=8, angle=0), legend.p
       legend.text=element_text(size=8, angle=0), plot.title=element_text(size=10), legend.key.size
#p_init_clusters
p1 = ggarrange(p_init_samples, p_init_clusters)
#ggsave(paste0(curr_outdir, "/marker_UMAP_TilleMatrix_sample_vs_cluster_ATAC", ".pdf"), plot=p1, de
#ggsave(paste0(curr_outdir, "/marker_UMAP_TileMatrix_sample_vs_cluster_ATAC", ".png"), plot=p1, dev

# C1 = Sensitive_common_small;   Sensitive_side
# C2 = Sensitive_no_agent_big    Sensitive_DMSO
# C3 = Sensitive_agent;          Sensitive_GNE7883
# C4 = Sensitive_agent;          Sensitive_GNE7883
# C5 = Resistant_small;          Resistant_side
# C6 = Resistant_big;            Resistant_main
# C7 = Resistant_big;            Resistant_main
#From XY: resistant_main (big), resistant_side (small), sensitive_DMSO, sensitive_GNE7883, sensitive_
project$Clusters_TileMatrix_named = plyr::mapvalues(project$Clusters_TileMatrix,
                                from = paste0("C_TileMatrix",1:7),
                      to = c("Sensitive_side", "Sensitive_DMSO", "Sensitive_GNE7883",
                           "Sensitive_GNE7883",  "Resistant_side", "Resistant_main",
                                        "Resistant_main")
)
```

**6. Generate ordinations:**

```
# add reduced dims TileMatrix
project = addIterativeLSI(project, useMatrix='TileMatrix', name='IterativeLSI_TileMatrix', seed=2, f
```

```
project = addClusters(project, reducedDims='IterativeLSI_TileMatrix', name='Clusters_TileMatrix', se
project = addUMAP(project, reducedDims='IterativeLSI_TileMatrix', name='UMAP_TileMatrix', seed=2, fo

# add reduced dims GeneExpressionMatrix
project = addIterativeLSI(project, useMatrix='GeneExpressionMatrix', name='IterativeLSI_GeneExpress
project = addClusters(project, reducedDims='IterativeLSI_GeneExpressionMatrix', name='Clusters_Gene
project = addUMAP(project, reducedDims='IterativeLSI_GeneExpressionMatrix', name='UMAP_GeneExpressi

# Combine both dimensionalities, add cluster to resulting combined dim and add umap
project = addCombinedDims(project, reducedDims=c('IterativeLSI_TileMatrix', 'IterativeLSI_GeneExpre
project = addClusters(project, reducedDims='IterativeLSI_Combined', name='Clusters_Combined', seed=2
project = addUMAP(project, reducedDims='IterativeLSI_Combined', name='UMAP_Combined', seed=2, force=

# Add tSNE embeddings
project = addTSNE(project, reducedDims='IterativeLSI_TileMatrix', name='TSNE_TileMatrix', perplexity
project = addTSNE(project, reducedDims='IterativeLSI_GeneExpressionMatrix', name='TSNE_GeneExpressi
project = addTSNE(project, reducedDims='IterativeLSI_Combined', name='TSNE_Combined', perplexity=30
project = addTSNE(project, reducedDims='IterativeLSI_ATAC', name='TSNE_ATAC', perplexity=30, seed=2,
```

**7. Peak calling:**

```
library(MASS)
library(chromVARmotifs)

project = addGroupCoverages(project, groupBy='Clusters_TileMatrix', threads=num_threads)

project = addReproduciblePeakSet(project, groupBy="Clusters_TileMatrix", method="q", cutOff=0.05,
                        pathToMacs2="/gstore/home/tremblj2/software/macs2/macs2_venv/bin/macs2",
                        excludeChr=c('chrMT','chrY'), genomeSize=2.7e9, threads=num_threads)

project = addPeakMatrix(project, binarize=FALSE, threads=num_threads, force=TRUE)

# TF annotation
peaks_anno = genomitory::getFeatures('GMTY162:hg38_motif_bed_granges@REVISION-3')
project = addPeakAnnotations(project, regions=peaks_anno, name='ENCODE_and_cistromeDB_TF_peaks', for
project = addDeviationsMatrix(project, peakAnnotation='ENCODE_and_cistromeDB_TF_peaks',
                      matrixName='TFPeaksDeviationsMatrix', threads=num_threads, force=TRUE)

# motif annotation
project = addMotifAnnotations(project, motifSet='cisbp', annoName='Motif', species='Homo sapiens', f
project = addDeviationsMatrix(project, peakAnnotation='Motif', threads=num_threads, force=TRUE)

# TF binding matrix
project = addPeakAnnotations(project, regions=epiregulon::getTFMotifInfo(genome="hg38"), name="TF_b
project = addDeviationsMatrix(project, peakAnnotation="TF_binding", threads=num_threads)
```

**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

Manuscript under review...

## Examples

```
# check metada of dataset
TEADi_resistance(metadata = TRUE)

# download data
 ## Not run:
TEADi_resistance()

## End(Not run)
```

---

`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

### Usage

```
tfBinding(
  genome = c("hg38", "hg19", "mm10"),
  source = c("atlas", "encode.sample", "atlas.sample", "atlas.tissue"),
  metadata = FALSE,
  version = 1,
  peak_number = 1000
)
```

### 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 |
| `version` | numeric indicating data version number (see Details) |
| `peak_number` | numeric indicating threshold to be applied number of peaks per transcription factor in the combined version of GenomicRanges (from all samples and tissues). |

### 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_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.

In version 2 chipatlas sample and tissue GenomicRanges were build using experiments with at least 2E7 unique reads and 1E3 peaks. In sample specific encode chip-seq data a quality metrics was number of peaks only.

## 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)
    combined_grl <- GRangesList()
    #### 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_merged, drop.empty.ranges = TRUE)
    combined_grl <- c(combined_grl, GRangesList(gr))
  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)
    combined_grl <- c(combined_grl, GRangesList(gr))
  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)
    combined_grl <- c(combined_grl, GRangesList(gr))
  export.bed(gr, con=file.path(dir, "chipatlas_encode_merged", genome, encode.TFs[i]))
    saveRDS(grl, paste0(output_dir, "tfBinding_", genome,"_atlas.rds"))
  }
}
```

**2.4 Merge ChIP-Atlas peaks by sample:**

```
library(rtracklayer)
#import metadata

for (genome in c("hg38","mm10")){
   metadata <- read.delim(paste0(data_dir, "chipAtlas/experimentList.tab"), header=FALSE)
    metadata <- metadata[metadata$V2 == genome,]
    metadata <- metadata[metadata$V3 == "TFs and others",]

    # add QC metrics - new in version 2
    metadata_qc <- do.call(rbind, strsplit(metadata$V8, split = ","))
    metadata_qc <- data.frame(apply(metadata_qc, 2, as.numeric))
   colnames(metadata_qc) <- c("Number.of.reads", "Percentage.mapped", "Percentage.duplicates", "Numb

    metadata <- cbind(metadata, metadata_qc)
    metadata$unique.reads <- round(metadata[,"Number.of.reads"]*
                                       metadata[,"Percentage.mapped"]*
                                (100-metadata[,"Percentage.duplicates"])/10000)

    metadata <- metadata[which(metadata$unique.reads >= 20000000 &
                                    #metadata_qc$Percentage.mapped >= 70 &
                                    metadata$Number.of.peaks >= 1000), ]

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

    grl <- list()

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

        for (tf in unique.tf){

            message("cell line = ", cellline, " ", "tf = ", tf)
            gr <- list()
            beds <- metadata.cellline$V1[metadata.cellline$V4 ==tf]

            if (length(beds) >0) {
                counter <- 1
                for (bed in beds) {
             file <- paste0(data_dir, "chipAtlas/", genome, "_1e5_individual/", bed,".05.bed")

                    if (file.exists(file)) {
                        # each gr list is per TF per tissue
                    gr[[counter]] <- try(rtracklayer::import(file, extraCols = extraCols_narrowPeak))
```

```
                        counter <- counter + 1
                    }
                }

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

        if (length(grl[[cellline]])>0){
            grl[[cellline]] <- GRangesList(grl[[cellline]])

        }
    }
   saveRDS(grl, paste0(output_dir, "tfBinding_", genome,"_atlas.sample_v2.rds"))
}
```

### 2.5 Merge ChIP-Atlas peaks by tissue:

```
library(rtracklayer)
#import metadata

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

    # add QC metrics - new in version 2
    metadata_qc <- do.call(rbind, strsplit(metadata$V8, split = ","))
    metadata_qc <- data.frame(apply(metadata_qc, 2, as.numeric))
   colnames(metadata_qc) <- c("Number.of.reads", "Percentage.mapped", "Percentage.duplicates", "Numb

    metadata <- cbind(metadata, metadata_qc)
    metadata$unique.reads <- round(metadata[,"Number.of.reads"]*
                                       metadata[,"Percentage.mapped"]*
                                (100-metadata[,"Percentage.duplicates"])/10000)

    metadata <- metadata[which(metadata$unique.reads >= 20000000 &
                                    metadata$Number.of.peaks >= 1000), ]
    # import bed
    extraCols_narrowPeak <- c(singnalValue = "numeric", pValue = "numeric",
                              qValue = "numeric", peak = "integer")

    grl <- list()
=
    tissues <- unique(metadata$V5)
    tissues <- as.vector(na.omit(tissues))
```

```
    for (tissue in tissues){
        metadata.tissue <- metadata[which(metadata $V5 == tissue),]
        unique.tf <- unique(metadata.tissue$V4)
        #
        for (tf in unique.tf){

            message("tissue = ", tissue," ", "tf = ", tf)
            gr <- list()
            beds <- metadata.tissue$V1[metadata.tissue$V4 ==tf]

            if (length(beds) >0) {
                counter <- 1
                for (bed in beds) {
             file <- paste0(data_dir, "chip_peaks/", genome, "_1e5_individual/", bed,".05.bed")

                    if (file.exists(file)) {
                    # each gr list is per TF per tissue
                gr[[counter]] <- try(rtracklayer::import(file, extraCols = extraCols_narrowPeak))
                        counter <- counter + 1
                    }

                }

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

            }
        }

        if (length(grl[[tissue]]) > 0) {
            grl[[tissue]] <- GRangesList(grl[[tissue]])
        }
    }

   saveRDS(grl, paste0(output_dir, "tfBinding_", genome,"_atlas.tissue_v2.rds"))
}
```

### 2.6 Merge ENCODE peaks by sample:

```
library(rtracklayer)

metadata.path <- c(GRCh38=paste0(data_dir,"encode/human"),
                   mm10=paste0(data_dir, "encode/mm10/encode"))

for (genome in c("GRCh38", "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",]
 metadata <- metadata[metadata$Audit.NOT_COMPLIANT == "",]
 metadata <- metadata[metadata$Audit.ERROR == "",]
 metadata$Peaks <- NA
 rownames(metadata) <- metadata$File.accession

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

 grl <- list()

 celllines <- unique(metadata$Biosample.term.name)
 celllines <- as.vector(na.omit(celllines))
 for (cellline in celllines){
  metadata.cellline <- metadata[which(metadata $Biosample.term.name == cellline),]
     unique.tf <- unique(metadata.cellline$Experiment.target)

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

         gr <- list()

         if (length(beds) >0) {

             for (bed in beds) {
          file <- file.path(metadata.path[genome], "raw", paste0(bed,".bed.gz"))

                 if (file.exists(file)) {
             temp_bed <- try(rtracklayer::import(file, extraCols = extraCols_narrowPeak))
                     peak_no <- length(temp_bed)
                     metadata[bed, "Peaks"] <- peak_no
                     if (peak_no > 1000) {              # new in version 2
                         gr[[bed]] <- temp_bed
                     }
                     rm(temp_bed)
                 }
             }
         }

         if (length(gr) > 0){
        grl[[cellline]][[tf.name]] <- GenomicRanges::reduce(do.call(c, unname(gr)))
         }
     }
```

```
        if (length(grl[[cellline]])>0){
            grl[[cellline]] <- GRangesList(grl[[cellline]])

        }
    }
   saveRDS(grl, paste0(output_dir,"tfBinding_", genome,"_encode.sample_v2.rds"))
   write.table(metadata, paste0(output_dir, "encode.metadata.", genome, ".txt"), quote = FALSE, row.r
}
```

**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")

library(liftOver)

# convert binding sites from ENCODE
grl_hg38 <- readRDS(paste0(output_dir,"tfBinding_", genome,"_encode.sample_v2.rds"))
grl_hg19 <- lapply(grl_hg38, liftOver, ch)
saveRDS(grl_hg19, file = paste0(output_dir,"tfBinding_hg19_encode.sample_v2.rds"))

# convert binding sites from ChIP Atlas

# convert grl by cell line
grl_hg38 <- readRDS(paste0(output_dir, "tfBinding_hg38_atlas.sample_v2.rds"))
grl_hg19 <- lapply(grl_hg38, liftOver, ch)
saveRDS(grl_hg19, file = paste0(output_dir, "tfBinding_hg19_atlas.sample_v2.rds")))

#convert grl by tissue
grl_hg38 <- readRDS(paste0(output_dir, "tfBinding_hg38_atlas.tissue_v2.rds"))
grl_hg19 <- lapply(grl_hg38, liftOver, ch)
saveRDS(grl_hg19, file = paste0(output_dir, "tfBinding_hg19_atlas.tissue_v2.rds"))

#convert combined grl
grl_hg38 <- readRDS(paste0(output_dir, "tfBinding_hg38_atlas.rds")
grl_hg19 <- lapply(grl_hg38, liftOver, ch)
saveRDS(grl_hg19, file = paste0(output_dir, "tfBinding_hg19_atlas.rds"))
```

## 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/

## Examples

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

## 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)
```

# Index


∗ **internal**
    customClasses, 19
    dummies, 20
    makeDataSetList, 26
    retrieve, 37
    RGtools, 38
    scMultiome-package, 2

AR_drug, 4
assertHDF5, 13

colonHealthy, 14
customClasses, 19

dummies, 20
dummyMAE *(*dummies*)*, 20
dummySCE *(*dummies*)*, 20

fileOperations, 21

hematopoiesis, 23

listDatasets, 25
loadExp *(*fileOperations*)*, 21
loadMAE *(*fileOperations*)*, 21

makeDataSetList, 26
makeMakeData *(*templates*)*, 48
makeMakeMetadata *(*templates*)*, 48
makeR *(*templates*)*, 48

PBMC_10x, 26
prostateENZ, 31

reprogramSeq, 36
restoreGR *(*RGtools*)*, 38
retrieve, 37
RGtools, 38

saveExp *(*fileOperations*)*, 21
saveExp,SingleCellExperiment-method
        *(*fileOperations*)*, 21
saveExp,SummarizedExperiment-method
        *(*fileOperations*)*, 21
saveMAE *(*fileOperations*)*, 21
scMultiome *(*scMultiome-package*)*, 2
scMultiome-package, 2
storeGR *(*RGtools*)*, 38

TEADi_resistance, 38
templates, 48
testFile *(*fileOperations*)*, 21
tfBinding, 49
tfMotifs, 62

uploadFile *(*fileOperations*)*, 21