Package ‘httrlib’

September 19, 2023

Title pipeline for automated in vitro screening platform

Version 0.7.3

Description US EPA has designed a rapid and automated in vitro screening platform using the TempO-seq targeted RNA-seq assay to profile chemical bioactivity across a range of concentrations and cell types. The scale and complexity of these data (measurements of ~20,000 genes across thousands of samples) has necessitated the development of a bioinformatics pipeline to ensure standardized and reproducible results across these screening efforts. This package includes R and Python code for rapid and robust processing of TempO-seq data, leveraging existing open-source tools wherever possible (e.g. HISAT2, samtools, DESeq2). This pipeline also includes a battery of QC metrics developed specifically for TempO-seq data generated in large-scale automated experiments, and additional functions to store and access multiple levels of analysis results in an intuitive NoSQL database (MongoDB). The package is intended for both internal EPA/ORD use and to allow external researchers to reproduce our results, process their own data, and ultimately facilitate the development of community standards for high-throughput transcriptomic analysis in a regulatory context.

License `use_mit_license()`, `use_gpl3_license()` or friends to pick a license

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

exports DESeq2_cut
addQCplotCols
as.mongo.date
countGini
countQC
countQCflag
countStats
degFrame
deleteByID
filterProbesByMean
filterSamplesByTotal
findByID
findIDs
flattenDEG
getAnlName
getChemTrts
getCredentials
getDB

getDEGs
getDocIDs
getFCmatrix
getProbeManifest
getQCdefault
getTrts
getWellCounts
getWellInfo
insertByID
insertManyDEG
insertOneDEG
iterate
mongoQuery
mongoURL
openMongo
outerFences
plotQCjitter
plotQCviolin
qcBatch
runDESeq2
runDESeq2ForChemCond
runDESeq2Single
sameCtrlWells
sampleID_wrapper
splitIDs
trtGrpWells
tss_deg_count
tss_l2_norm
update_attenuation_factors
updateqcFlags
validateProbeManifest
validate_httr_well_trt_schema

Suggests BiocParallel,
DESeq2,
ggplot2,
reldist,
rlist,
stats,
testthat

Imports data.table,
jsonlite,
mongolite,
foreach

R topics documented:

`.onLoad` *options set upon package loading Options used for the QC flag calls - to skip any of these remove them from httrQCflags httrMinMapdFrac = Flag wells with mapd_frac < this value, default 0.5 httrMinMapdN = Flag wells with n_reads_mapd < this value, default 300k httrMinNsig80 = Flag wells with n_sig80 < this value, default 1000 httrMinNcov5 = Flag wells with n_cov5 < this value, default 5000 httrMaxTop10Prop = Flag wells with top10_prop > this value, default 0.1 (but NOT used) httrMaxGini = Flag wells with gini_coef > this value, default 0.95 httrQCflags = Vector of QC flags, order defines flag priority (first flag that hits will be used) - default is to skip HIGH_TOP10 flag min_colsum = floor value under which sample_id are removed mean_cnt (*integer*) = floor value to exclude samples with colsum below this value*

Description

options set upon package loading Options used for the QC flag calls - to skip any of these remove them from httrQCflags httrMinMapdFrac = Flag wells with mapd_frac < this value, default 0.5 httrMinMapdN = Flag wells with n_reads_mapd < this value, default 300k httrMinNsig80 = Flag wells with n_sig80 < this value, default 1000 httrMinNcov5 = Flag wells with n_cov5 < this value, default 5000 httrMaxTop10Prop = Flag wells with top10_prop > this value, default 0.1 (but NOT used) httrMaxGini = Flag wells with gini_coef > this value, default 0.95 httrQCflags = Vector of QC flags, order defines flag priority (first flag that hits will be used) - default is to skip HIGH_TOP10 flag min_colsum = floor value under which sample_id are removed mean_cnt (*integer*) = floor value to exclude samples with colsum below this value

Usage

```
.onLoad(lib, pkg)
```

---

addQCplotCols — *addQCplotCols Function to adjust well treatment data in a way that is better for QC plotting Reclassifies stype/rna_src column to "scat" column, and converts to a factor and creates log10 scale columns from n_reads and n_reads_mapd, and bad_probe_prop if necessary cols present NOTE: Requires data.table package*

Description

addQCplotCols Function to adjust well treatment data in a way that is better for QC plotting Reclassifies stype/rna_src column to "scat" column, and converts to a factor and creates log10 scale columns from n_reads and n_reads_mapd, and bad_probe_prop if necessary cols present NOTE: Requires data.table package

Usage

```
addQCplotCols(data)
```

Arguments

`data`	(*data.frame* or *data.table*) = Table containing all well treatment and QC info, e.g. as returned by getWellInfo()

Value

(*data.table*) = data with additional columns `allBspLysateGroups` *allBspLysateGroups Construct documents for httr_trt_grp_cmp collection corresponding to each BioSpyder Reference QC Lysate sample comparison (single conc treatment). Loop over all plate groups and create a httr_trt_grp_cmp document for each pair of reference RNAs.*

Description

allBspLysateGroups Construct documents for httr_trt_grp_cmp collection corresponding to each BioSpyder Reference QC Lysate sample comparison (single conc treatment). Loop over all plate groups and create a httr_trt_grp_cmp document for each pair of reference RNAs.

Usage

```
allBspLysateGroups(
  DB = getOption("DB_NAME"),
  well = getOption("HTTR_WELL_NAME"),
  ...
)
```

Arguments

`DB`	(string) = database name with httr_well and httr_trt_grp_cmp collections
`well`	(string) = Name of collection with individual well treatment data, default is "httr_well"
`...`	= All additional args passed to chemTreatGroup

Value

(list) = List of documents that were inserted into trt_grp collection

`allBspRNAGroups` *allBspRNAGroups Construct documents for httr_trt_grp_cmp collection corresponding to each BioSpyder Reference RNA QC sample comparison (single conc treatment). Loop over all plate groups and create a httr_trt_grp_cmp document for each pair of reference RNAs.*

Description

allBspRNAGroups Construct documents for httr_trt_grp_cmp collection corresponding to each BioSpyder Reference RNA QC sample comparison (single conc treatment). Loop over all plate groups and create a httr_trt_grp_cmp document for each pair of reference RNAs.

Usage

```
allBspRNAGroups(
  DB = getOption("DB_NAME"),
  well = getOption("HTTR_WELL_NAME"),
  ...
)
```

Arguments

`DB`	(string) = database name with httr_well and httr_trt_grp_cmp collections
`well`	(string) = Name of collection with individual well treatment data, default is "httr_well"
`...`	= All additional args passed to chemTreatGroup

Value

(list) = List of documents that were inserted into trt_grp collection

---

`allBulkLysateGroups` *allBulkLysateGroups Construct documents for httr_trt_grp_cmp collection corresponding to each bulk lysate QC sample comparison (single conc treatment).*

Description

Loop over all plate groups, chem_id (bulk lysate chemicals only) and create a httr_trt_grp_cmp document for each one.

Usage

```
allBulkLysateGroups(
  DB,
  trt_type = "QC sample",
  rna_src = "Bulk Lysate",
  ctrl_chem = "DMSO",
  well = "httr_well",
  ...
)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections collections
`trt_type`	(str) = Filter stype to get list of plate groups, chem_id
`rna_src`	(str) = Filter rna_src to get list of plate groups, chem_id
`ctrl_chem`	(str) = Skip this chemical when looping over chem_id
`well`	(str) = Name of collection with individual well treatment data, default is "httr_well ... = All additional args passed to chemTreatGroup

Details

Parameters:

Value

(list) = List of documents that were inserted into trt_grp collection Store all documents in a list, also track the number that were skipped due to insufficient replicates with passing QC flags:

allRefDoseGroups

allRefDoseGroups Construct documents for httr_trt_grp_cmp collection corresponding to each reference chemical (single or multi conc).

Description

Loop over all plate groups, media, timeh, chem_id, dose_level (reference chemicals only) and create a httr_trt_grp_cmp document for each one. Alternate version of allRefGroups that also loops over and handles reference chemicals with multiple dose levels Note that to use this version, all reference chemical wells in httr_well_trt MUST have dose_level defined (even single conc reference chemicals) For studies with only single conc reference chemicals that do NOT have a dose_level field, use allRefGroups instead (e.g. for the original MCF-7 screens) Also, unlike allRefGroups, this will automatically append plate group ID as part of the trt_name when creating docs

Usage

```
allRefDoseGroups(
  DB,
  trt_type = "reference chemical",
  trt_prop_fields = c("chem_id", "conc", "conc_unit", "dose_level", "stype"),
  grp_id_opts = c("pg"),
  well,
  exp_doses = NULL,
  ...
)
```

Arguments

`DB`	(pymongo.MongoClient) = Open connection to database with httr_well and httr_trt_grp_cmp collections
`trt_type`	(str) = Filter stype to get list of plate groups, media, timeh, and chem_id
`trt_prop_fields`	(list of str) = Fields in httr_well collection that should match up for all trt_wells and the singular value should be propagated to field of same name in httr_trt_grp_cmp collection. Default is to propagate chem_id, conc, conc_unit, dose_level, and stype fields
`grp_id_opts`	(list) = Options to modify trt_grp_id generation, "pg" is usually sufficient for reference chemicals to create distinct group IDs for each plate group
`well`	(str) = Name of collection with individual well treatment data, default is "httr_well"
`exp_doses(list`	of str) = Expected number of doses for each reference chemical. This will suppress some debug messages and only warn if the number of doses does not match. For example, c("GEN" = 8, "SIRO" = 8, "TSA" = 1)
`log`	(httrplcore.PipelineLogger) = Log handler
`**kwargs`	= All additional args passed to chemTreatGroup

Details

Parameters:

Value

(list of dict) = List of documents that were inserted into trt_grp collection

---

`allRefGroups` *allRefGroups Function to loop over all plate groups, media, and times and then call refChemGroup Construct documents for httr_trt_grp_cmp collection corresponding to each reference chemical (single conc).*

Description

Loop over all plate groups, media, timeh, chem_id (reference chemicals only) and create a httr_trt_grp_cmp document for each one.

Usage

```
allRefGroups(DB, trt_type = "reference chemical", well = "httr_well", ...)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections collections
`trt_type`	(str) = Filter stype to get list of plate groups, media, timeh, and chem_id
`well`	(str) = Name of collection with individual well treatment data, default is "httr_well" ... = All additional args passed to chemTreatGroup

Details

Parameters:

Value

(list) = List of documents that were inserted into trt_grp collection Store all documents in a list, also track the number that were skipped due to insufficient replicates with passing QC flags:

`allRefRNAGroups` *allRefRNAGroups Construct documents for httr_trt_grp_cmp collection corresponding to each reference RNA QC sample comparison (single conc treatment).*

Description

Loop over all plate groups and create a httr_trt_grp_cmp document for each pair of reference RNAs.

Usage

```
allRefRNAGroups(DB, well = "httr_well", ...)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections collections
`well`	(str) = Name of collection with individual well treatment data, default is "httr_well" ... = All additional args passed to chemTreatGroup

Details

Parameters:

Value

(list) = List of documents that were inserted into trt_grp collection Store all documents in a list, also track the number that were skipped due to insufficient replicates with passing QC flags:

`allTestGroups` *allTestGroups*

Description

Construct documents for httr_trt_grp_cmp collection corresponding to each test chemical and dose level treatment.

Usage

```
allTestGroups(DB, well = "httr_well", exp_doses, ...)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections collections
`well`	(str) = Name of collection with individual well treatment data, default is "httr_well"
`exp_doses`	(int) = Expected number of doses for each test chemical - this will suppress some debug messages and only warn if the number of doses does not match ... = All additional args passed to chemTreatGroup

Details

Loop over all plate groups, media, timeh, chem_id, and dose_level and create a httr_trt_grp_cmp document for each one.

Parameters:

Value

(list) = List of documents that were inserted into trt_grp collection Store all documents in a list, also track the number that were skipped due to insufficient replicates with passing QC flags:

as.mongo.date — *as.mongo.date Convert "POSIXct" to a list object with one member: '$date'=(ISODate str)*

Description

Given an R datetime object, e.g. as returned by Sys.time(), convert to a structure appropriate for conversion to JSON and mongo insert

Usage

```
as.mongo.date(x)
```

Arguments

x	(POSIXct) = R datetime object, e.g. as returned by Sys.time()

Value

(list) = Has one member: '$date'=(ISODate str), converting this to JSON with auto_unbox=T will insert this as a proper date

bspLysateGroup — *bspLysateGroup Construct a document for httr_trt_grp_cmp collection corresponding to the BioSpyder reference QC lysate sample comparison (A vsB). Given a specific plate group, plus any optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This function is just a convenenience wrapper to chemTreatGroup with modified defaults. Note that unlike the original MCF7 bulk lysates, these samples are annotated more like the reference RNA samples.*

Description

bspLysateGroup Construct a document for httr_trt_grp_cmp collection corresponding to the BioSpyder reference QC lysate sample comparison (A vsB). Given a specific plate group, plus any optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This function is just a convenenience wrapper to chemTreatGroup with modified defaults. Note that unlike the original MCF7 bulk lysates, these samples are annotated more like the reference RNA samples.

Usage

```
bspLysateGroup(
  DB = getOption("DB_NAME"),
  pg_id,
  trt_name = "BSP_LYSATE_B",
  ctrl_name = "BSP_LYSATE_A",
  trt_type = "QC sample",
  ctrl_type = "QC sample",
  trt_src = "Bulk Lysate",
    ctrl_src = "Bulk Lysate",
    trt_prop_fields = c(),
    both_prop_fields = c("stype", "rna_src", "pg_id", "block_id"),
    ctrl_desc_field = "trt_name",
    grp_id_opts = c("vs", "pg"),
    ...
  )
```

Arguments

`DB`	(string) database name with httr_well and httr_trt_grp_cmp collections
`pg_id`	(string) = Filter both trt and ctrl wells, match to pg_id in httr_well, required for ref RNA groups
`trt_type`	(string) = Match to stype in httr_well, defaults to "QC sample"
`ctrl_type`	(string) = Match to stype in httr_well, defaults to "QC sample"
`trt_src`	(string) = Match to rna_src in httr_well, defaults to "BSP_RNA_B"
`ctrl_src`	(string) = Match to rna_src in httr_well, defaults to "BSP_RNA_A"
`trt_prop_fields`	(list) = Fields in httr_well collection that should match up for all trt_wells and the singular value should be propagated to field of same name in httr_trt_grp_cmp collection. Default is to propagate just the stype field (exclude all chemical and dose information).
`both_prop_fields`	(list) = Fields in httr_well collection that should match up for all trt_wells AND ctrl_wells, and the singular value should be propagated to to field of same name in httr_trt_grp_cmp collection. Default is to propagate pg_id and block_id only for ref RNA comparisons
`ctrl_desc_field`	(string) = Field to propagate from ctrl_wells to ctrl field in httr_trt_grp_cmp document, defaults to "trt_name" for ref RNA comparisons
`grp_id_opts`	(list) = Options to modify trt_grp_id generation, "vs" will capture the pairwise comparison type, and "pg" is usually sufficient for ref RNA to create distinct group IDs for each plate group
`...`	All additional args passed to treatGroupFromSamples

Value

(list) = Document that was inserted into trt_grp collection

`bspRNAGroup`	*bspRNAGroup Construct a document for httr_trt_grp_cmp collection corresponding to the BioSpyder purified reference RNA QC sample comparison (A vsB). Given a specific plate group, plus any optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This function is just a convenenience wrapper to chemTreatGroup with modified defaults.*

Description

bspRNAGroup Construct a document for httr_trt_grp_cmp collection corresponding to the BioSpyder purified reference RNA QC sample comparison (A vsB).

Given a specific plate group, plus any optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This function is just a convenenience wrapper to chemTreatGroup with modified defaults.

Usage

```
bspRNAGroup(
  DB = getOption("DB_NAME"),
  pg_id,
  trt_name = "BSP_RNA_B",
  ctrl_name = "BSP_RNA_A",
  trt_type = "QC sample",
  ctrl_type = "QC sample",
  trt_src = "Reference RNA",
  ctrl_src = "Reference RNA",
  trt_prop_fields = c(),
  both_prop_fields = c("stype", "rna_src", "pg_id", "block_id"),
  ctrl_desc_field = "trt_name",
  grp_id_opts = c("vs", "pg"),
  ...
)
```

Arguments

`DB`	(db name) = Open connection to database with httr_well and httr_trt_grp_cmp collections
`pg_id`	(string) = Filter both trt and ctrl wells, match to pg_id in httr_well, required for ref RNA groups
`trt_type`	(string) = Match to stype in httr_well, defaults to "QC sample"
`ctrl_type`	(string) = Match to stype in httr_well, defaults to "QC sample"
`trt_src`	(string) = Match to rna_src in httr_well, defaults to "BSP_RNA_B"
`ctrl_src`	(string) = Match to rna_src in httr_well, defaults to "BSP_RNA_A"
`trt_prop_fields`	(vector) = Fields in httr_well collection that should match up for all trt_wells and the singular value should be propagated to field of same name in httr_trt_grp_cmp collection. Default is to propagate just the stype field (exclude all chemical and dose information).
`both_prop_fields`	(vector) = Fields in httr_well collection that should match up for all trt_wells AND ctrl_wells, and the singular value should be propagated to to field of same name in#' @param httr_trt_grp_cmp collection. Default is to propagate pg_id and block_id only for ref RNA comparisons
`ctrl_desc_field`	(string) = Field to propagate from ctrl_wells to ctrl field in httr_trt_grp_cmp document, defaults to "trt_name" for ref RNA comparisons
`grp_id_opts`	(vector) = Options to modify trt_grp_id generation, "vs" will capture the pairwise comparison type, and "pg" is usually sufficient for ref RNA to create distinct group IDs for each plate group

Value

(list) = Document that was inserted into trt_grp collection

`bulkLysateGroup` *bulkLysateGroup Construct a document for httr_trt_grp_cmp collection corresponding to a specific bulk lysate chemical treatment.*

Description

Given a specific plate group, plus any optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This function is just a convenenience wrapper to chemTreatGroup with modified defaults.

Usage

```
bulkLysateGroup(
  DB,
  pg_id,
  trt_chem = "TSA",
  trt_dose = NULL,
  trt_type = "QC sample",
  ctrl_type = "QC sample",
  ctrl_chem = "DMSO",
  rna_src = "Bulk Lysate",
  trt_prop_fields = c("chem_id", "conc", "conc_unit", "stype"),
  both_prop_fields = c("pg_id", "block_id"),
  ctrl_desc_field = "chem_id",
  grp_id_opts = c("pg"),
  ...
)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections collections
`pg_id`	(str) = Filter both trt and ctrl wells, match to pg_id in httr_well, required for bulk lysate groups
`trt_chem`	(str) = Match to chem_id in httr_well, defaults to "TSA"
`trt_dose`	(int) = Match to dose_level in httr_well - Excluded by default
`trt_type`	(str) = Match to stype in httr_well, defaults to "QC sample"
`ctrl_type`	(str) = Match to stype in httr_well, defaults to "QC sample"
`ctrl_chem`	(str) = Match to chem_id in httr_well, defaults to "DMSO"
`rna_src`	(str) = Match to rna_src in httr_well for both trt and ctrl wells, defaults to "Bulk Lysate"
`trt_prop_fields`	(vector) = Fields in httr_well collection that should match up for all trt_wells and the singular value should be propagated to field of same name in httr_trt_grp_cmp collection. Default is to propagate chem_id, conc, conc_unit, and stype fields (exclude dose_level here).

`both_prop_fields`	(vector) = Fields in httr_well collection that should match up for all trt_wells AND ctrl_wells, and the singular value should be propagated to to field of same name in httr_trt_grp_cmp collection. Default is to propagate pg_id and block_id only for bulk lysate comparisons
`ctrl_desc_field`	(str) = Field to propagate from ctrl_wells to ctrl field in httr_trt_grp_cmp document, defaults to "chem_id" for bulk lysate comparisons
`grp_id_opts`	(vector) = Options to modify trt_grp_id generation, "pg" is usually sufficient for bulk lysate to create distinct group IDs for each plate group ... = All additional args passed to treatGroupFromSamples

Details

Parameters:

Value

(list) = Document that was inserted into trt_grp collection Pass to chemTreatGroup:

`check_collection_remapped` *check_collection_remapped Check if any collections are re-mapped, fill in defaults for everything else*

Description

check_collection_remapped Check if any collections are re-mapped, fill in defaults for everything else

Usage

```
check_collection_remapped(collections)
```

Arguments

`collections`	(*named character vector*) = vector to re-map collection names used, where name=default collection; value=new collection name

Value

collections (*named character vector*) = vector to re-map collection names used, where name=default collection; value=new collection name

`chemTreatGroup`	*chemTreatGroup Function that generates the trt_grp_cmp for a specific test chemical treatment vs DMSO Construct a document for httr_trt_grp_cmp collection corresponding to a specific chemical and dose level treatment.*

Description

Given a specific chemical and dose, plus optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. Note, if filtering reduces either trt or ctrl group to < min_reps samples this treatment group will be skipped.

Usage

```
chemTreatGroup(
  DB,
  trt_chem,
  trt_dose,
  trt_type = "test sample",
  ctrl_type = "vehicle control",
  ctrl_chem = "DMSO",
  cell_type = NULL,
  pg_id = NULL,
  media = NULL,
  timeh = NULL,
  trt_src = NULL,
  ctrl_src = NULL,
  qc_flags = c("OK"),
  min_reps = 2,
  well = "httr_well",
  ...
)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections
`trt_chem`	(str) = Match to chem_id in httr_well
`trt_dose`	(int) = Match to dose_level in httr_well (set to None to exclude)
`trt_type`	(str) = Match to stype in httr_well, defaults to "test sample"
`ctrl_type`	(str) = Match to stype in httr_well
`ctrl_chem`	(str) = Match to chem_id in httr_well
`pg_id`	(str) = Filter both trt and ctrl wells, match to pg_id in httr_well, defaults to None
`media`	(str) = Filter both trt and ctrl wells, match to media in httr_well, defaults to None
`timeh`	(str) = Filter both trt and ctrl wells, match to timeh in httr_well, defaults to None
`trt_src`	(str) = Filter trt wells, match to rna_src field
`ctrl_src`	(str) = Filter ctrl wells, match to rna_src field

`qc_flags`	(vector) = Filter both trt and ctrl wells, match to qc_flag field, defaults to OK only
`min_reps`	(int) = Minimum number of replicates in trt and ctrl groups, respectively - if either is less, will generate an output message and return empty dict
`well`	(str) = Name of collection with individual well treatment data, default is "httr_well"
`...`	= All additional args passed to treatGroupFromSamples

Details

Parameters:

Value

(list) = Document that was inserted into trt_grp collection Build the query for treatment and ctrl wells

---

`countGini` *countGini Computing Gini coefficient Use reldist::gini(counts) where counts = probe counts for a single sample This metric DOES depend on whether 0 counts are present or missing from the vector If zero counts are missing, should specify proben to fill in missing 0s*

Description

countGini Computing Gini coefficient Use reldist::gini(counts) where counts = probe counts for a single sample This metric DOES depend on whether 0 counts are present or missing from the vector If zero counts are missing, should specify proben to fill in missing 0s

Usage

```
countGini(
  counts,
  proben = getOption("httrProbeN"),
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`counts`	(*integer*) = a vector of counts for a single sample, w/ or w/out 0 counts
`proben`	(*integer*) = Total number of probes used for Gini coefficients across samples, ignored if NULL
`debug`	(*logical*) = Whether or not to report extra debug messages

Value

(*numeric*) = Gini coefficient returned from reldist::gini `countQC` *countQC Given a document from httr_counts collection, compute all QC stats and flags and format according to httr_counts_qc schema option() function should be used to change defaults for underlying QC functions*

Description

countQC Given a document from httr_counts collection, compute all QC stats and flags and format according to httr_counts_qc schema option() function should be used to change defaults for underlying QC functions

Usage

```
countQC(
  counts_doc,
  bad_probes = c(),
  proben = getOption("httrProbeN"),
  calc_gini = getOption("httrStatGini", default = TRUE),
  calc_flag = TRUE,
  min_mapd_frac = get_global_option("min_mapd_frac"),
  min_n_reads_mapd = get_global_option("min_n_reads_mapd"),
  min_n_sig80 = get_global_option("min_n_sig80"),
  min_n_cov5 = get_global_option("min_n_cov5"),
  max_top10_prop = get_global_option("max_top10_prop"),
  max_gini_coef = get_global_option("max_gini_coef"),
  qc_flags = get_global_option("qc_flags"),
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`counts_doc`	(*list*) = Follows DB schema for httr_counts, msut include the _id field
`bad_probes`	(*character vector*) = List of probe IDs that should be masked
`proben`	(*integer*) = Total number of probes on the platform (length bad_probes will be subtracted before passing to gini function)
`calc_gini`	(*logical*) = Whether or not to calculate gini, default True but can override if no reldist package installed
`calc_flag`	(*logical*) = Whether or not to calculate qc_flag, default True but when set to False all flags will be set to "OK"
`debug`	(*logical*) = Whether to report additional debug info

`countQCflag`	*countQCflag Given a list structure with all the quantitative fields in the httr_counts_qc schema, determine the qc_flag field Flags are checked in priority order according to qc_flags parameter NOTE: If any required fields are missing from qc_stats, it will lead to an error*

Description

countQCflag Given a list structure with all the quantitative fields in the httr_counts_qc schema, determine the qc_flag field Flags are checked in priority order according to qc_flags parameter NOTE: If any required fields are missing from qc_stats, it will lead to an error

Usage

```
countQCflag(
  qc_stats,
  min_mapd_frac = get_global_option("min_mapd_frac"),
  min_n_reads_mapd = get_global_option("min_n_reads_mapd"),
  min_n_sig80 = get_global_option("min_n_sig80"),
  min_n_cov5 = get_global_option("min_n_cov5"),
  max_top10_prop = get_global_option("max_top10_prop"),
  max_gini_coef = get_global_option("max_gini_coef"),
  qc_flags = get_global_option("qc_flags"),
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`qc_stats`	(*list*) = list of all QC stats computed by countStats and countGini above
`min_mapd_frac`	(*numeric*) = minimum mapd_frac, flag with LOW_MAPD_FRAC below this cutoff
`min_n_reads_mapd`	(*integer*) = minimum n_reads_mapd, flag with LOW_READS below this cutoff
`min_n_sig80`	(*integer*) = minimum n_sig80, flag with LOW_NSIG80 below this cutoff
`min_n_cov5`	(*integer*) = minimum n_cov5, flag with LOW_NCOV5 below this cutoff
`max_top10_prop`	(*numeric*) = maximum top10_prop, flag with HIGH_TOP10 above this cutoff
`max_gini_coef`	(*numeric*) = maximum gini_coef, flag with HIGH_GINI above this cutoff
`qc_flags`	(*character vector*) = list of flags to apply in priority order
`debug`	(*logical*) = Whether to report some debug info

Value

(*character*) = a single QC flag for the well, "OK" if not thresholds were hit

`countStats` *countStats Function to compute count-based QC statistics for a single sample*

Description

countStats Function to compute count-based QC statistics for a single sample

Usage

```
countStats(
  counts,
  ncov = getOption("httrStatNCov", default = 5),
  nsig = getOption("httrStatNsig", default = 0.8),
  topn = getOption("httrStatTopN", default = 10)
)
```

Arguments

`counts`	(*integer*) = a vector of counts for a single sample, w/ or w/out 0 counts All of these statistics should be invariant to the probe labels, ordering of the counts, or whether 0 counts are missing
`ncov`	(*integer*) = Minimum read count(s) for Ncov statistic (number of probes covered at a minimum read count level)
`nsig`	(*numeric*) = Proportion of signal for Nsig statistic (minimum number of probes capturing top X% of signal)
`topn`	(*integer*) = Number of probes to use for TopN statistic (proportion of signal captured by top probe)

Value

(*named list*) = each of the requested statistics, member names now match DB schema

`degFrame` *degFrame - Take just the degs member of a httr_deg doc and reformat as a data.frame with appropriate classes for each column This a low-level function, is automatically applied by getDEGs when pulling from database.*

Description

degFrame - Take just the degs member of a httr_deg doc and reformat as a data.frame with appropriate classes for each column This a low-level function, is automatically applied by getDEGs when pulling from database.

Usage

```
degFrame(degs)
```

Arguments

`degs`	(*list*) - The raw degs member of a doc in httr_deg collection, as returned directly from mongo

Value

(*data.frame*) - Table with 7 columns matching the underlying deg fields

---

`deleteByID` *deleteByID*

- *Delete documents from a collection. Given a list of IDs, delete all matching documents from a collection, or delete ALL documents. The primary purpose is to remove documents from a collection before replacing them when rerun=True in a particular pipeline step.*

Description

deleteByID

- Delete documents from a collection. Given a list of IDs, delete all matching documents from a collection, or delete ALL documents. The primary purpose is to remove documents from a collection before replacing them when rerun=True in a particular pipeline step.

Usage

```
deleteByID(
  DB,
  ids = c(),
  id_field = "sample_id",
  delete_all = FALSE,
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`DB`	(mongo object) = Open connection to database and collection
`ids`	(*vector*) = Vector (usually character) of IDs for documents to delete, default=c(), which defers to delete_all
`id_field`	(*character*) = Alternate ID field to use, default: "sample_id"
`delete_all`	(*logical*) = Setting to True will delete the whole collection, default: False
`debug`	(*logical*) = Whether to post debug messages

Value

(*numeric*) = Number of documents that were deleted, will also warn if != length(ids) when delete_all=False

DESeq2_cut *DESeq2_cut- (compartmentalized unit test) - test whether a localled generated DESeq2 result matches the the saved result*

Description

DESeq2_cut- (compartmentalized unit test) - test whether a localled generated DESeq2 result matches the the saved result

Usage

```
DESeq2_cut(saved_results, treatments)
```

Arguments

`saved_results`	(*list*) = object with archived DESeq2 results and all necessary objects to recreate that object with local instalation
`treatments`	(d*ata.frame*) = dataframe containing information about the analysis parameters that went into calculating DESeq2 results (shrinkage, normalization)

Value

(*boolean*) = if all tests pass, returns true, if *any* fail, returns false

extract_and_run *extract_and_run - Run DESeq2 on a single chemical dose-response series Uses DB access functions to extract all httr_trt_grp_id entries corresponding to a dose-response series, then extracts counts for all wells, filters by min read depth and mean floor, and passes through runDESeq2.*

Description

extract_and_run - Run DESeq2 on a single chemical dose-response series Uses DB access functions to extract all httr_trt_grp_id entries corresponding to a dose-response series, then extracts counts for all wells, filters by min read depth and mean floor, and passes through runDESeq2.

Usage

```
extract_and_run(
  chem_id,
  db_host,
  db_name,
  pg_id = NULL,
  media = NULL,
  timeh = NULL,
  max_dose_level = 8,
  min_colsum = getOption("min_colsum"),
    mean_cnt = getOption("mean_cnt"),
    plate_effect = F,
    shrinkage = "normal",
    threads = 1,
    collections = character(0),
    debug = getOption("debug", default = FALSE)
  )
```

Arguments

`chem_id`	(*character*) = ID for the chemical to run, matched against chem_id field in httr_trt_grp_cmp
`db_host`	(*character*) = Host for DB connection
`db_name`	(*character*) = Name of DB
`pg_id`	(*character*) = Limit httr_trt_grp_cmp to this plate group, useful if multiple dose response series for same chem_id
`media`	(*character*) = Limit httr_trt_grp_cmp to this media type, useful if multiple dose response series for same chem_id
`timeh`	(*integer*) = Limit httr_trt_grp_cmp to this timeh, useful if multiple dose response series for same chem_id
`max_dose_level`	(*integer*) = Max number of dose levels expected
`min_colsum`	(*integer*) = Filter samples by min read total, set to NULL to skip, defaults to 100k
`mean_cnt`	(*integer*) = Filter probes by mean floor, set to NULL to skip, defaults to 5
`plate_effect`	(*logical*) = Whether or not to include plate effect in model, defaults to FALSE
`shrinkage`	(*character*) = Type of shrinkage to use, defaults to new default, a.k.a. "normal"
`threads`	(*integer*) = Number of threads to use for core DESeq2 functions, defaults to 1 (no multi-threading)
`collections`	(*named character vector*) = Optional vector to re-map collection names used, where name=default collection; value=new collection name
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

vector of my_results, chemTrts, trt_grp_name

`fill_dose_for_well_trt` *fill_dose_for_well_trt This function takes a data.table and fills the dose column with calculated values*

Description

fill_dose_for_well_trt This function takes a data.table and fills the dose column with calculated values

Usage

```
fill_dose_for_well_trt(sampleKey)
```

Arguments

`sampleKey:`	(*data.table*) the data.table conforming with the normal welltrt columns process: for each conc found ordered by pg_id/chem_id assign a corresponding dose_level

Value

the modified data.table

`fill_dose_for_well_trt_wrapper` *fill_dose_for_well_trt_wrapper This function takes a sample_id file and converts it to a data.table and does some checks then calls fill_dose_for_well_trt*

Description

fill_dose_for_well_trt_wrapper This function takes a sample_id file and converts it to a data.table and does some checks then calls fill_dose_for_well_trt

Usage

```
fill_dose_for_well_trt_wrapper(sampleID_file)
```

Arguments

`sampleKey:`	(*file*) the file conforming with the normal welltrt columns

Value

the dat.frame resulted from fill_dose_for_well_trt

`filterProbesByMean` *filterProbesByMean - Function to prefilter DESeq2 input data by removing probes with low mean count Removes any probes with mean count < mean_cnt*

Description

filterProbesByMean - Function to prefilter DESeq2 input data by removing probes with low mean count Removes any probes with mean count < mean_cnt

Usage

```
filterProbesByMean(
  COUNTS,
  mean_cnt = getOption("mean_cnt"),
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`COUNTS`	(*data.frame*) = Data frame containing all counts to be analyzed, row.names = probes, colnames = samples
`mean_cnt`	(*integer*) = Remove any samples with colsum below this value
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*data.frame*) = COUNTS after removing the probes with low mean counts

---

`filterSamplesByTotal` *filterSamplesByTotal - Function to prefilter DESeq2 input data by removing low-depth samples Removes any sample with colsum < min_colsum*

Description

filterSamplesByTotal - Function to prefilter DESeq2 input data by removing low-depth samples Removes any sample with colsum < min_colsum

Usage

```
filterSamplesByTotal(
  COUNTS,
  min_colsum = getOption("min_colsum"),
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`COUNTS`	(*data.frame*) = Data frame containing all counts to be analyzed, row.names = probes, colnames = samples
`min_colsum`	(*integer*) = Remove any samples with colsum below this value
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*data.frame*) = COUNTS after removing the samples with low colsum

`Find` *title Find : with DB object as well as fields and potentially query named list, inquire db for matching documents*

Description

title Find : with DB object as well as fields and potentially query named list, inquire db for matching documents

Usage

```
Find(DB = NULL, ...)
```

Arguments

`DB`	(*mongo object*) = Opened connection to specific Mongo Database and Collection
`optional`	fields: json list of fields to be included in our search other parameter (optional): query JSON query to use when querying the db

Value

list of documents matching search specified by query/fields

`findByID` *findByID Find all documents in a collection with optional ID filtering.*

Description

Given a vector of IDs (typically either _id or sample_id), return an iterable object (mongo iterator or list) corresponding to all matching documents in a collection. If ids is an empty list, return all documents in the collection. Also has options to limit the result to just the id_field or any other subset of fields, and to dump out the data as a list (stored in local memory) instead of returning the pymongo Cursor object.

Usage

```
findByID(
  DB,
  ids = c(),
  id_field = "sample_id",
  id_only = FALSE,
  fields = "{}",
  dump = FALSE,
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`DB`	(emphmongo object) = Open connection to specific Mongo Database and Collection
`ids`	(character vector) = Vector of IDs (usually character) to query on, gets all values if empty, default: empty vector.
`id_field`	(character) = Which ID field to match against ids, default: sample_id.
`id_only`	(logical) = If True, return only the values in the ID field being matched against, otherwise return the whole document
`fields`	(Mongo query, as character) = If id_only = False, this will be used to filter the return fields instead.
`dump`	(logical) = If True, loop over the initial return object (pymongo.cursor.Cursor) and extract all results to a new list object.
`debug`	(logical) = If True, dump out some extra debugging messages, can also set using options(debug=True)

Value

mongo iterator, list, or vector) = Either: an iterator object to return results from Mongo (dump=False), a list of all results (dump=True, id_only=False) a vector of IDs (dump=True, id_only=True)

`findIDs` *findIDs Generic function to get IDs matching any query from any collection Given an open DB connection or connection parameters, run a query and get all distinct values of id_field*

Description

findIDs Generic function to get IDs matching any query from any collection Given an open DB connection or connection parameters, run a query and get all distinct values of id_field

Usage

```
findIDs(
  DB = NULL,
  db_host = NULL,
  db_name = NULL,
  collection = NULL,
  id_field = "_id",
  debug = getOption("debug", default = FALSE),
  ...
)
```

Arguments

`DB`	(*mongo object*) = If specified, use this open connection to a collection, ignore db_host, db_name, collection
`db_host`	(*character*) = If DB is NULL, this specifies the host to open a connection to
`db_name`	(*character*) = If DB is NULL, this specifies the DB name to connect to

`collection`	(*character*) = If DB is NULL, this specifies the collection to connect to
`id_field`	(*character*) = What field in the DB to return IDs from, defaults to the generic mongo _id field
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...) ... = All additional params passed to mongoQuery to form the query string

Value

(character vector) = The ID values returned by DB$distinct call

---

`flattenDEG` *flattenDEG Convert a single httr_deg schema doc to a data.frame This is a low-level function that is automatically called by getDEGs whenever flatten=T This takes all other fields besides degs and makes them columns in the data.frame anl_opt and update_notes are flattened using jsonlite::toJSON*

Description

flattenDEG Convert a single httr_deg schema doc to a data.frame This is a low-level function that is automatically called by getDEGs whenever flatten=T This takes all other fields besides degs and makes them columns in the data.frame anl_opt and update_notes are flattened using jsonlite::toJSON

Usage

```
flattenDEG(deg_doc)
```

Arguments

`deg_doc`	(*list*) = Single mongo doc conforming to httr_deg schema

Value

(data.frame) = Flatted version of deg_doc

`format_for_db_insert` *format_for_db_insert description: optionally saves the results provided by prior call to extract_and_run function back to the DB in addition to returning a single data.frame with all results that can be written to a TSV file or saved in Rdata format.*

Description

format_for_db_insert description: optionally saves the results provided by prior call to extract_and_run function back to the DB in addition to returning a single data.frame with all results that can be written to a TSV file or saved in Rdata format.

Usage

```
format_for_db_insert(
  extract_and_run_return,
  mean_cnt,
  plate_effect,
  shrinkage,
  collections,
  db_host,
  db_name,
  note,
  rerun,
  debug,
  db_insert
)
```

Arguments

`extract_and_run_return`	vector containing the results computed by prior call to runDESeq2 function and to be inserted into db
`mean_cnt`	(*integer*) = Filter probes by mean floor, set to NULL to skip, defaults to 5
`plate_effect`	(*logical*) = Whether or not to include plate effect in model, defaults to FALSE
`shrinkage`	(*character*) = Type of shrinkage to use, defaults to new default, a.k.a. "normal"
`collections`	(*named character vector*) = Optional vector to re-map collection names used, where name=default collection; value=new collection name
`db_host`	(*character*) = Host for DB connection
`db_name`	(*character*) = Name of DB
`note`	(*character*) = Overwrite default note fields for insert if this is specified.
`rerun`	(*logical*) = Whether to re-run the DB insertion if results are in there already (ideally this should check the DB before running...)
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)
`db_insert`	(*logical*) = Whether to insert the results back into the DB, defaults to TRUE

Value

(*data.frame*) = Contains DESeq2 results for all treatments in the dose-response series

---

`getAnlName` *getAnlName Convert analysis options to standardized anl_name for httr_deg docs*

Description

getAnlName Convert analysis options to standardized anl_name for httr_deg docs

Usage

```
getAnlName(
  mean_cnt = getOption("mean_cnt"),
  plate_effect = F,
  shrinkage = "normal",
  ...
)
```

Arguments

`mean_cnt`	(*numeric*)
`plate_effect`	(*character*)
`shrinkage`	(*character*)

`getChemTrts` *getChemTrts*

- *Pull out documents from httr_trt_grp_cmp collection corresponding to a single chemical dose-response series Returns the relevant documents from httr_trt_grp_cmp as a named list, where names come from trt_grp_id field, and are sorted by dose_level*

Description

getChemTrts

- Pull out documents from httr_trt_grp_cmp collection corresponding to a single chemical dose-response series Returns the relevant documents from httr_trt_grp_cmp as a named list, where names come from trt_grp_id field, and are sorted by dose_level

Usage

```
getChemTrts(
  DB = NULL,
  chem_id,
  db_host = NULL,
  db_name = NULL,
  collection = "httr_trt_grp_cmp",
  pg_id = NULL,
  media = NULL,
  timeh = NULL,
  max_dose_level = 8,
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`DB`	(*mongo object*) = If specified, use this open connection to httr_trt_grp_cmp collection, ignore db_host, db_name, collection
`chem_id`	(*character*) = Match against chem_id field, this is required
`db_host`	(*character*) = If DB is NULL, this specifies the host to open a connection to
`db_name`	(*character*) = If DB is NULL, this specifies the DB name to connect to
`collection`	(*character*) = If DB is NULL, this specifies the collection to connect to, default: httr_trt_grp_cmp
`pg_id`	(*character*) = Optional addl filter on pg_id field, useful if there is more than one series for same chem_id
`media`	(*character*) = Optional addl filter on media field, useful if there is more than one series for same chem_id
`timeh`	(*numeric*) = Optional addl filter on timeh field, useful if there is more than one series for same chem_id
`max_dose_level`	(*numeric*) = Expected max number of doses, will give a warning if more than this number of treatment groups returned, default is 8
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*named list*) = Names come from trt_grp_id field, each member is a complete httr_trt_grp_cmp doc (list type), members are sorted by dose_level field

---

`getCredentials` *getCredentials This function replaces the Keychain class in lib/db/passwords.py Rather than load entire keychains into memory, it just quickly scans ~/.mongopw and ~/.mngdb/passwd for relevant user,passwd combo*

Description

getCredentials This function replaces the Keychain class in lib/db/passwords.py Rather than load entire keychains into memory, it just quickly scans ~/.mongopw and ~/.mngdb/passwd for relevant user,passwd combo

Usage

```
getCredentials(host = getOption("httrDefaultHost"), db = NULL)
```

Arguments

`host`	(*character*)
`db`	(*mongo object*)

Value

NULL or a list with members user,passwd (and host,db if specified)

`getDB` *getDB if DB not provided already, opens a connection to mongodb via the mongolite package to the collection provided*

Description

getDB if DB not provided already, opens a connection to mongodb via the mongolite package to the collection provided

Usage

```
getDB(DB = NULL, db_host = NULL, db_name = NULL, collection = NULL)
```

Arguments

`DB`	(*mongo object*) = Opened connection to specific Mongo Database and Collection or null if not provided
`db_host:`	(*character*) string representing the db url
`db_name:`	(*character*) string representing the db name
`collection:`	(*character*) string representing the collection name we are trying to connect to

Value

(*mongo object*) a mongo ’environment’, i.e. a DB object

`getDEGs` *getDEGs Pull out DESeq2 results from httr_deg table*

Description

getDEGs Pull out DESeq2 results from httr_deg table

Usage

```
getDEGs(
  DB = NULL,
  db_host = NULL,
  db_name = NULL,
  collection = "httr_deg",
  single = F,
  flatten = F,
  warn_count = 100,
  warn_stype = T,
  warn_anl = T,
  debug = getOption("debug", default = FALSE),
  ...
)
```

Arguments

`DB`	(*mongo object*) = If specified, use this open connection to httr_deg collection, ignore db_host, db_name, collection
`db_host`	(*character*) = If DB is NULL, this specifies the host to open a connection to
`db_name`	(*character*) = If DB is NULL, this specifies the DB name to connect to
`collection`	(*character*) = If DB is NULL, this specifies the collection to connect to, default: httr_deg
`single`	(*logical*) = Whether to return a single doc - if TRUE and query matches multiple docs, then only the first match is returned
`flatten`	(*logical*) = Whether to flatten all results into a single data frame instead of returning in list structure closer to mongo doc structure
`warn_count`	(*integer*) = If query is going to return more than X documents, issue a warning (set to Inf to silence), default: 100
`warn_stype`	(*logical*) = When TRUE: If query returns documents with a mix of stype values, issue a warning
`warn_anl`	(*logical*) = When TRUE: If query returns documents with a mix of anl_name values, issue a warning
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...) ... = All other params are passed to mongoQuery to build the filtering query

Value

(*list* or *data.frame*) = When single & !flatten, this is a list matching httr_deg schema; when !single & !flatten, this is a list of lists with multiple httr_deg docs; when flatten, this is a data.frame

---

`getDocIDs` *getDocIDs*

Description

extract a certain ID field (e.g. sample_id) out of a list of documents This is designed to mimic the function of the same name from lib/db/mongo.py

Usage

```
getDocIDs(docs, id_field = "sample_id")
```

Arguments

`docs`	(*list*) = List of documents dumped out from MongoDB as a list type
`id_field`	(*character*) = Name of field to pull out of every member of docs

Value

(*vector*) = Vector of ID values from docs, atomic type depends on the type of ID (usually character) `getFCmatrix` *getFCmatrix - Pull out DESeq2 results for multiple contrasts from httr_deg collection and reshape into wide matrix*

Description

getFCmatrix - Pull out DESeq2 results for multiple contrasts from httr_deg collection and reshape into wide matrix

Usage

```
getFCmatrix(
  db_host,
  db_name,
  collections = character(0),
  anl_name = getAnlName(mean_cnt = mean_cnt, plate_effect = plate_effect, shrink
    shrinkage),
  mean_cnt = getOption("mean_cnt"),
  plate_effect = T,
  shrinkage = "normal",
  stype = "test sample",
  debug = getOption("debug", default = FALSE),
  threads = -1,
  ...
)
```

Arguments

`db_name`	(*character*) = Specifies the DB name to connect to
`collections`	(*named character vector*) = Optional vector to re-map collection names used, where name=default collection; value=new collection name
`anl_name`	(*character*) = Which analysis configuration to pull results for, default: based on mean_cnt, plate_effect, and shrinkage params, but setting here takes precedence
`mean_cnt`	(*integer*) = passed to getAnlName to set anl_name, default: 5
`plate_effect`	(*logical*) = passed to getAnlName to set anl_name, default: T
`shrinkage`	(*character*) = passed to getAnlName to set anl_name, default: "normal"
`stype`	(*character*) = Filter for sample type, set to NULL to remove this filter, default: "test sample"
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)
`threads`	(*integer*) = number of threads to use to parallelize ... = All other params are passed to mongoQuery to provide additional filtering (if chem_id is not specific enough, this will generate an error)
`host`	(*character*) = Specifies the host to open a connection to

Value

(*list of data.frame*) = First member $fc is matrix of contrast (rows) x probe (col), Second member $cmp is matrix of contrast (rows) x meta-data columns, both in same row order. Missing probes will be NA.

getProbeManifest

getProbeManifest Function to pull out all probe info as a data.frame This excludes the transcript field in order to get a flat table, but includes everything else The table is also sorted on the index field to ensure the row order of the original manifest

Description

getProbeManifest Function to pull out all probe info as a data.frame This excludes the transcript field in order to get a flat table, but includes everything else The table is also sorted on the index field to ensure the row order of the original manifest

Usage

```
getProbeManifest(
  DB = NULL,
  db_host = NULL,
  db_name = NULL,
  collection = "httr_probe",
  fields = "{\"transcripts\":0}",
  ...
)
```

Arguments

`DB`	(*mongo object*) = If specified, use this open connection to httr_probe collection, ignore db_host, db_name, collection
`db_host`	(*character*) = If DB is NULL, this specifies the host to open a connection to
`db_name`	(*character*) = If DB is NULL, this specifies the DB name to connect to
`collection`	(*character*) = If DB is NULL, this specifies the collection to connect to, default: httr_probe

Value

(*data.frame*) = Data frame with complete manifest

getQCdefault

getQCdefault Function to determine appropriate threshold for a QC metric when not specified: NOTE: When the relevant flag is not in httrQCflags, this will always return NULL

Description

getQCdefault Function to determine appropriate threshold for a QC metric when not specified: NOTE: When the relevant flag is not in httrQCflags, this will always return NULL

Usage

```
getQCdefault(qc_metric)
```

Arguments

`qc_metric`	(*character*) = Name of the QC metric to get default for

Value

(*numeric*) = Value of default threshold, or NULL if not thresholding on this metric

---

`getTrts` *getTrts*

- *Low-level function to extract all docs from httr_trt_grp_cmp that match a specific query Before returning, the ctrl_wells and trt_wells members are converted to data frames*

Description

getTrts

- Low-level function to extract all docs from httr_trt_grp_cmp that match a specific query Before returning, the ctrl_wells and trt_wells members are converted to data frames

Usage

```
getTrts(DB, query, debug = getOption("debug", default = FALSE))
```

Arguments

`DB`	(*mongo object*) = Open DB connection to httr_trt_grp_cmp collection
`query`	(*json*) = Query for returning docs from httr_trt_grp_cmp collection
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*list of lists*) = Names come from trt_grp_id field, each member is a complete httr_trt_grp_cmp doc (list)

`getWellCounts` *getWellCounts Function to pull out counts and treatment info as data.frames for a set of sample IDs*

Description

getWellCounts Function to pull out counts and treatment info as data.frames for a set of sample IDs

Usage

```
getWellCounts(
  DB = NULL,
  db_host = NULL,
  db_name = NULL,
  collection = "httr_well",
  debug = getOption("debug", default = FALSE),
  sample_id = NULL,
  ...
)
```

Arguments

`DB`	(*mongo object*) = If specified, use this open connection to httr_well collection, ignore db_host, db_name, collection
`db_host`	(*character*) = If DB is NULL, this specifies the host to open a connection to
`db_name`	(*character*) = If DB is NULL, this specifies the DB name to connect to
`collection`	(*character*) = If DB is NULL, this specifies the collection to connect to, default: httr_well
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...) ... = Additional parameters passed to mongoQuery and used to specify which wells to extract counts for
`sample_id`	(*character*) = Used to query counts for a specific subset of sample IDs, if NULL this will not be part of query

Value

(*list*) = Has two members, both data.frames: $treatments has rows = samples, $counts has rows = probes, columns = samples

`getWellInfo` *getWellInfo Function to pull out treatment info ONLY from httr_well as a data.frame for any relevant query*

Description

getWellInfo Function to pull out treatment info ONLY from httr_well as a data.frame for any relevant query

Usage

```
getWellInfo(
  DB = NULL,
  db_host = NULL,
  db_name = NULL,
  collection = "httr_well",
  debug = getOption("debug", default = FALSE),
  ...
)
```

Arguments

`DB`	(*mongo object*) = If specified, use this open connection to httr_well collection, ignore db_host, db_name, collection
`db_host`	(*character*) = If DB is NULL, this specifies the host to open a connection to
`db_name`	(*character*) = If DB is NULL, this specifies the DB name to connect to
`collection`	(*character*) = If DB is NULL, this specifies the collection to connect to, default: httr_well
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...) ... = Any additional parameters are passed to mongoQuery to constrain the query

Value

(*data.frame*) = Table of well treatment info from httr_well, currently excludes _id, probe_cnts, raw_id, counts_id, trt_id, sorts by sample_id

---

`get_global_option` *get_global_option Function to find the value of the passed global_var first checks if old name for global var exists and returns that and a warning if not, then just simply return the value corresponding to the passed global var*

Description

get_global_option Function to find the value of the passed global_var first checks if old name for global var exists and returns that and a warning if not, then just simply return the value corresponding to the passed global var

Usage

```
get_global_option(global_var)
```

Arguments

`global_var`	(*character*) = may be any of the following values: min_mapd_frac, min_n_reads_mapd, min_n_sig80, min_n_cov5, max_top10_prop, max_gini_coef, qc_flags

Value

numeric

insertByID

title insertByID - Insert a list of documents into a MongoDB collection without creating redundant entries. Takes a list of dicts and first checks whether any documents with matching IDs are already present. If so, these documents are either skipped for insert (rerun=False) or replaced (rerun=True)

Description

title insertByID - Insert a list of documents into a MongoDB collection without creating redundant entries. Takes a list of dicts and first checks whether any documents with matching IDs are already present. If so, these documents are either skipped for insert (rerun=False) or replaced (rerun=True)

Usage

```
insertByID(
  DB,
  docs,
  id_field = "sample_id",
  rerun = FALSE,
  debug = getOption("debug", default = FALSE)
)
```

Arguments

DB	(*mongo object*) = Open database/collection connection for inserting
docs	(*list*) = List of documents to insert into collection
id_field	(*character*) = The field to use for matching up documents, default: sample_id
rerun	(*logical*) = Whether to overwrite existing data with same sample_id
debug	(*logical*) = Whether to report debug messages

Value

(*integer*) = The number of documents successfully inserted, Note: Unlike the pymongo API, mongolite does not return the IDs after a successful insert NOTE: For this function in particular, it makes sense to have a custom wrapper for each collection, e.g. to make sure there are matching IDs in other collections

insertManyDEG

insertManyDEG Insert multiple docs into httr_deg collection, matched by combination of trt_grp_id and anl_name will skip docs that are already inserted if rerun = False

Description

insertManyDEG Insert multiple docs into httr_deg collection, matched by combination of trt_grp_id and anl_name will skip docs that are already inserted if rerun = False

Usage

```
insertManyDEG(
  DB = NULL,
  deg_docs = NULL,
  db_host = NULL,
  db_name = NULL,
  collection = "httr_deg",
  rerun = FALSE,
  debug = getOption("debug", default = FALSE),
  ...
)
```

Arguments

`DB`	(*mongo object*) = If specified, use this open connection to httr_deg collection, ignore db_host, db_name, collection
`deg_docs`	(*list of lists*) = Each top-level entry should be a document (list) corresponding to httr_deg schema
`db_host`	(emphcharacter) = If DB is NULL, this specifies the host to open a connection to
`db_name`	(*character*) = If DB is NULL, this specifies the DB name to connect to
`collection`	(*character*) = If DB is NULL, this specifies the collection to connect to, default: httr_deg
`rerun`	(*logical*) = Whether to re-run the insertion of this data and override existing doc with matching trt_grp_id, anl_name
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...) ... = Additional params passed to insertOneDEG

Value

(*integer*) = Number of documents successfully inserted

`insertOneDEG` *insertOneDEG - Insert a doc into httr_deg collection, matched by combination of trt_grp_id and anl_name*

Description

insertOneDEG - Insert a doc into httr_deg collection, matched by combination of trt_grp_id and anl_name

Usage

```
insertOneDEG(
  DB = NULL,
  deg_doc = NULL,
  db_host = NULL,
  db_name = NULL,
  collection = "httr_deg",
    init_note = "initial run",
    update_note = NULL,
    rerun = FALSE,
    debug = getOption("debug", default = FALSE)
  )
```

Arguments

DB	(*mongo object*) = If specified, use this open connection to httr_deg collection, ignore db_host, db_name, collection
deg_doc	(*list*) = Single document corresponding to httr_deg schema for insert
db_host	(*character*) = If DB is NULL, this specifies the host to open a connection to
db_name	(*character*) = If DB is NULL, this specifies the DB name to connect to
collection	(*character*) = If DB is NULL, this specifies the collection to connect to, default: httr_deg
init_note	(*character*) = If this is first insert into DB, set update_notes.note to this value
update_note	(*character*) = If this is subsequent insert into DB, set last update_notes.note to this value (defaults to: "update on MM/DD/YY")
rerun	(*logical*) = Whether to re-run the insertion of this data and override existing doc with matching trt_grp_id, anl_name
debug	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*logical*) = TRUE if inserted successfully, FALSE otherwise

---

insert_into_httr_study

insert_into_httr_study This function reads all inputs, validates them to create a document into httr_study

---

Description

insert_into_httr_study This function reads all inputs, validates them to create a document into httr_study

Usage

```
insert_into_httr_study(
  study_id = "httr_mcf7_pilot",
  study_name = "HTTr MCF-7 Pilot Study",
  study_desc =
    "Pilot of HTTr platform on MCF-7 cells testing multiple media types and expo
  study_probe = "httr_probe",
  study_well = "httr_well",
  study_trt_grp_cmp = "httr_trt_grp_cmp",
  study_degs = "httr_deg",
  src = "BioSpyder",
    tech = "TempoSeq",
    times_hr = list(6, 12, 24),
    max_dose_level = 8,
    media = list("DMEM", "PRF.DMEM"),
    cell_types = list("MCF-7"),
    chem_ids = list("TP0001651A01", "TP0001651A02", "TP0001651A03", "TP0001651A04"
      "TP0001651A05", "TP0001651A06", "TP0001651B01", "TP0001651B02", "TP0001651B0
      "TP0001651B04", "TP0001651B05", "TP0001651B06", "TP0001651C01", "TP0001651C0
      "TP0001651C03", "TP0001651C04", "TP0001651C05", "TP0001651C06"),
    nreps = 3,
    bs_assay_id = "",
    assay_name = "human_wt_1.2_mcf7",
    platform_name = "human_wt",
    platform_ver = "1.2",
    atten_type = "mcf7",
    probe_source_file = "180905 Human Whole Transcriptome 1.2 Manifest.xlsx",
    probe_source_date = "2018-09-05T00:00:00Z",
    probe_source_rcv = "2019-07-15T13:24:12Z",
    probe_ncct_file = "httr_mcf7_pilot_probe.csv",
    probe_ncct_date = "2019-09-23T13:44:50.419Z",
    fasta_file =
      "/share/projects/HTTr/HTTr_pipeline_dev/HumanWT_v1/AlignCount/genome/humanWT
    anno_type = "refseq",
    anno_date = "2018-09-05T00:00:00Z",
    host = NULL,
    db = NULL
  )
```

Arguments

`study_id`	(str) = short string with study ID (all collections specific to this study should have name starting with this ID, so no spaces, punctuation or special characters)
`study_name`	(str) = Name of the study in human readable format (spaces, punctuation, etc. allowed) – replaces Imran's "name" field in prev version of this collection.
`study_desc`	(str) = longer text briefly describing the study
`study_probe`	(str) = name of collection that catalogs all probes in the version of probe set used for this study
`study_well`	(str) = name of collection with the well-level data for this study
`study_trt_grp_cmp`	(str) = name of collection with the treatment groups for comparison for this study
`study_degs`	(str) = name of collection with the DEG data for this study
`src`	(str) = Source of data, e.g. "BioSpyder"
`tech`	(str) = Technology type, e.g. "TempoSeq"
`times_hr`	(list of int) = List of exposure times used (does not include Bulk Lysates or other QC samples)
`max_dose_level`	(float) = Highest dose level for any conc-response curve - this replaces Imran's concs_um field because not all concs are um or will match up across chemicals, but knowing the intended number of points in each conc-response curve is useful.

`media`	(list of str) = List of media tested (does not include Bulk Lysates or other QC samples
`cell_types`	(list of str) = List of cell-types tested (was just "cell" in previous schema; does not include Bulk Lysates or other QC samples)
`chem_ids`	(list of str) = List of chemical IDs tested (whatever IDs are used in chem_id field of httr_well_trt collection).
`nreps`	(int) = Number of replicates for primary treatment groups of interest (for screening studies, this is per concentration of test samples, also per media/time/cell-type in other study types)
`bs_assay_id`	(str) = Unique bar code or other identifier for the actual tube/batch of assay mix that BioSpyder used - we will need to get this information from them for tracking
`assay_name`	(str) = Unique identifier for this probe set version (including the combination of attenuation factors, e.g. human_wt_1.2_mcf7 to denote Human Whole Transcriptome v1.2 with MCF-7 attenuation)
`platform_name`	(str) = Just the general platform, without version number or attenuation modifiers, e.g. human_wt for human whole transcriptome, or human_s1500
`platform_ver`	(str) = Version number for the platform, e.g. 1.2 or 2.0
`atten_type`	(str) = Type of attenuation used, e.g. "none" if no attenuation, "mcf7", "heparg_super", or "broad"
`probe_source_file`	(str) = Name of the manifest file sent from BioSpyder that was used to process this data
`probe_source_date`	(date) = When did BioSpyder last update the file?
`probe_source_rcv`	(date) = When did we actually receive the manifest from BioSpyder?
`probe_ncct_file`	(str) = If NCCT performed additional clean-up of the manifest, name of the final clean file that was loaded into DB. Otherwise can be blank or same as source_file.
`probe_ncct_date`	(date) = When did NCCT finalize the probe set manifest (after validating/modifying the manifest received from BioSpyder)
`fasta_file`	(str) = Full path to fasta file used to build HISAT2 index and align fastq files
`anno_type`	(str) = What was the primary annotation used for this probe set (e.g. earlier versions are refseq, newer versions are ensembl)
`anno_date`	(date) = When was the annotation for this probe set last updated (we may update the annotations periodically to keep up with current transcriptome annotations) process: The function does validation checks on an entry into the httr_study collection and if successfull inserts it

Value

nothing explicit print the test that fail if any. If validation passes, nothing returned

`iterate`	*iterate with DB object as well as fields and potentially query named list, inquire db for matching documents*

Description

iterate with DB object as well as fields and potentially query named list, inquire db for matching documents

Usage

```
iterate(DB = NULL, query = NULL, fields = NULL)
```

Arguments

`DB`	(*mongo object*) = Opened connection to specific Mongo Database and Collection
`fields:`	(*json list*) of fields to be included in our search
`query:`	(*json query*) to be used in our search

Value

list of documents matching search specified by query/fields

`mongoQuery`	*mongoQuery Build a mongo query from an arbitrary set of params or a list Takes a list of named values/vectors and converts to a JSON-format mongo query. Each entry with length > 1 is interpreted as a set of potential matching values and converted to the $in keyword for mongo*

Description

mongoQuery Build a mongo query from an arbitrary set of params or a list Takes a list of named values/vectors and converts to a JSON-format mongo query. Each entry with length > 1 is interpreted as a set of potential matching values and converted to the $in keyword for mongo

Usage

```
mongoQuery(...)
```

Arguments

`...`	(any) = Any set of named parameters OR a single named list of all arguments

`mongoURL` *MongoURL*

Description

Just format a mongo URL based on mongoServer, username, password, and database

Usage

```
mongoURL(host, user, passwd, db, authSource = NULL, authMechanism = NULL)
```

Arguments

`host`	(*character*)
`user`	(*character*)
`passwd`	(*character*)
`db`	(*character*)
`authSource`	(*character*)
`authMechanism`	(*character*)

Value

formatted string to pass to mongolite library connect function

`openMongo` *openMongo open a connection to a specific collection in a mongo db*

Description

openMongo open a connection to a specific collection in a mongo db

Usage

```
openMongo(
  host = getOption("httrDefaultHost"),
  user = NULL,
  passwd = NULL,
  db = NULL,
  collection = NULL,
  authSource = NULL,
  authMechanism = NULL
)
```

Arguments

`host`	(*character*)
`user`	(*character*)
`passwd`	(*character*)
`db`	(*character*)
`collection`	(*character*)
`authSource`	(*character*)
`authMechanism`	(*character*)

Value

formatted string to pass to mongolite library connect function

---

`outerFences` *outerFences Function to compute "Tukey's Outer Fence" for a set of values Computes both lower and upper thresholds, which are 3*IQR beyond the lower and upper quartile respectively*

Description

outerFences Function to compute "Tukey's Outer Fence" for a set of values Computes both lower and upper thresholds, which are 3*IQR beyond the lower and upper quartile respectively

Usage

```
outerFences(x, iqr.factor = 3)
```

Arguments

`x`	(*numeric vector*) = Set of values to compute fences for
`iqr.factor`	(*numeric*) = Multiplier for IQR in the fence computation, default is 3

Value

(*numeric*, length 2) = Lower and upper fence values

`plotQCjitter` *plotQCjitter Function for drawing jitter plot of a QC metric grouped by an experimental design factor*

Description

plotQCjitter Function for drawing jitter plot of a QC metric grouped by an experimental design factor

Usage

```
plotQCjitter(
  qc_metric,
  design_fac,
  data,
  cutoff = getQCdefault(qc_metric),
  alt_cutoff = NULL,
  title = NULL,
  plot.ylim = NULL
)
```

Arguments

`qc_metric`	(*character*) = Name of the QC metric column in data to plot
`design_fac`	(*character*) = Name of the design factor column in data to group by
`data`	(*data.frame*) = Table of all QC/meta-data for plotting
`cutoff`	(*numeric*) = Single value for drawing a horizontal dashed line, denoting current threshold - if NULL, will attempt to fill in with appropriate default
`alt_cutoff`	(*numeric*) = Single value for drawing a horizontal dotted line, denoting alternate threshold - if NULL, not drawn
`title`	(*character*) = Title for the plot, optional
`plot.ylim`	(*numeric*, length 2) = Set lower and upper limits for Y axis for consistent plotting, if NULL plots will auto-decide

Value

(*ggplot2*) object that can be further modified or drawn with print

`plotQCviolin` *plotQCviolin Function for drawing violin plot of a QC metric grouped by an experimental design factor*

Description

plotQCviolin Function for drawing violin plot of a QC metric grouped by an experimental design factor

Usage

```
plotQCviolin(
  qc_metric,
  design_fac,
  data,
  cutoff = getQCdefault(qc_metric),
  alt_cutoff = NULL,
  title = NULL,
  plot.ylim = NULL
)
```

Arguments

`qc_metric`	(*character*) = Name of the QC metric column in data to plot
`design_fac`	(*character*) = Name of the design factor column in data to group by
`data`	(*data.frame*) = Table of all QC/meta-data for plotting
`cutoff`	(*numeric*) = Single value for drawing a horizontal dashed line, denoting current threshold - if NULL, will attempt to fill in with appropriate default
`alt_cutoff`	(*numeric*) = Single value for drawing a horizontal dotted line, denoting alternate threshold - if NULL, not drawn
`title`	(*character*) = Title for the plot, optional
`plot.ylim`	(*numeric*, length 2) = Set lower and upper limits for Y axis for consistent plotting, if NULL plots will auto-decide

Value

(*ggplot2*) object that can be further modified or drawn with print

---

`process_chemInfo` *process_chemInfo*

Description

process_chemInfo

Usage

```
process_chemInfo(wellTrt)
```

Arguments

`wellTrt:`	(*character*)data.table with data to check

Value

chemInfo data table a subset of the inputted data.table to be stored into httr_chem by default

`process_chem_columns` *process_chem_columns takes as input: (coming from prior manipulations)*

Description

process_chem_columns takes as input: (coming from prior manipulations)

Usage

```
process_chem_columns(wellTrt)
```

Arguments

`wellTrt:`	(*character*)data.table with data to check

Value

modified data table after removing rows with dup chems and removing chem_name, dtxsid, casrn and bottle_id columns

`qcBatch` *qcBatch Function to run QC functions on all documents in httr_counts and store results in httr_counts_qc Also pulls probe info out of httr_probe (should delegate to functions in db/probes.R), NOTE: There's no real reason here to do sample_id batching, but could still use config files to store the DB info and collection names?*

Description

qcBatch Function to run QC functions on all documents in httr_counts and store results in httr_counts_qc Also pulls probe info out of httr_probe (should delegate to functions in db/probes.R), NOTE: There's no real reason here to do sample_id batching, but could still use config files to store the DB info and collection names?

Usage

```
qcBatch(
  db_host,
  db_name,
  db_probe = "httr_probe",
  db_counts = "httr_counts",
  db_collection = "httr_counts_qc",
  keep_flags = "OK",
  calc_flag = TRUE,
  rerun = FALSE,
  min_mapd_frac = get_global_option("min_mapd_frac"),
  min_n_reads_mapd = get_global_option("min_n_reads_mapd"),
    min_n_sig80 = get_global_option("min_n_sig80"),
    min_n_cov5 = get_global_option("min_n_cov5"),
    max_top10_prop = get_global_option("max_top10_prop"),
    max_gini_coef = get_global_option("max_gini_coef"),
    qc_flags = get_global_option("qc_flags"),
    debug = getOption("debug", default = FALSE),
    ...
  )
```

Arguments

`db_host`	(*character*) = MongoDB host URL
`db_name`	(*character*) = MongoDB name
`db_probe`	(*character*) = Name of probe collection, default: httr_probe
`db_counts`	(*character*) = Name of counts collection, default: httr_counts
`db_collection`	(*character*) = Name of counts_qc collection, default: httr_counts_qc
`keep_flags`	(*character vector*) = Keep all probes with these flags in the downstream QC (everything else is considered BAD), default: "OK"
`calc_flag`	(*logical*) = Whether or not to calculate qc_flag, default True but when set to False all flags will be set to "OK"
`rerun`	(*logical*) = Whether to overwrite existing data in httr_counts_qc
`min_mapd_frac`	(n*umeric*) = minimum mapd_frac, flag with LOW_MAPD_FRAC below this cutoff
`min_n_reads_mapd`	(*integer*) = minimum n_reads_mapd, flag with LOW_READS below this cutoff
`min_n_sig80`	(*integer*) = minimum n_sig80, flag with LOW_NSIG80 below this cutoff
`min_n_cov5`	(*integer*) = minimum n_cov5, flag with LOW_NCOV5 below this cutoff
`max_top10_prop`	(*numeric*) = maximum top10_prop, flag with HIGH_TOP10 above this cutoff
`max_gini_coef`	(*numeric*) = maximum gini_coef, flag with HIGH_GINI above this cutoff in the ... param, a query can be passed that will select the documents in httr_well_trt used to further filter the sample_ids to run the qc_flag calculation and storage in the httr_counts_qc collection
`debug`	(*logical*) = Whether to report debug messages

`readWellTrtFile` *readWellTrtFile takes as input: (coming from prior manipulations)*

Description

readWellTrtFile takes as input: (coming from prior manipulations)

Usage

```
readWellTrtFile(sampleID_file)
```

Arguments

`sampleID_file:`	(*character*)file that has such list in json format

Value

data table with json data with corrected column type

`refChemGroup` *refChemGroup Construct a document for httr_trt_grp_cmp collection corresponding to a specific reference chemical treatment.*

Description

Given a specific reference chemical and plate group (or media/timeh that specify plate group), plus any optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This function is just a convenenience wrapper to chemTreatGroup with modified defaults.

Usage

```
refChemGroup(
  DB,
  trt_chem,
  pg_id = NULL,
  media = NULL,
  timeh = NULL,
  trt_dose = NULL,
  trt_type = "reference chemical",
  trt_prop_fields = c("chem_id", "conc", "conc_unit", "stype"),
  ...
)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections
`trt_chem`	(str) = Match to chem_id in httr_well
`pg_id`	(str) = Filter both trt and ctrl wells, match to pg_id in httr_well, defaults to None
`media`	(str) = Filter both trt and ctrl wells, match to media in httr_well, defaults to None
`timeh`	(str) = Filter both trt and ctrl wells, match to timeh in httr_well, defaults to None
`trt_dose`	(int) = Match to dose_level in httr_well - Excluded by default
`trt_type`	(str) = Match to stype in httr_well, defaults to "reference chemical"
`trt_prop_fields`	(vector) = Fields in httr_well collection that should match up for all trt_wells and the singular value should be propagated to field of same name in httr_trt_grp_cmp collection. Default is to propagate chem_id, conc, conc_unit, and stype fields (exclude dose_level here). ... = All additional args passed to treatGroupFromSamples

Details

Parameters:

Value

(list) = Document that was inserted into trt_grp collection Warn if neither pg_id nor media+timeh were specified:

`refRNAGroup` *refRNAGroup Construct a document for httr_trt_grp_cmp collection corresponding to a specific reference RNA QC sample comparison.*

Description

Given a specific plate group, plus any optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This function is just a convenenience wrapper to chemTreatGroup with modified defaults.

Usage

```
refRNAGroup(
  DB,
  pg_id,
  trt_type = "QC sample",
  ctrl_type = "QC sample",
  trt_src = "HBRR",
  ctrl_src = "UHRR",
  trt_prop_fields = c("stype"),
  both_prop_fields = c("pg_id", "block_id"),
  ctrl_desc_field = "trt_name",
  grp_id_opts = c("vs", "pg"),
  ...
)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections collections collections
`pg_id`	(str) = Filter both trt and ctrl wells, match to pg_id in httr_well, required for ref RNA groups
`trt_type`	(str) = Match to stype in httr_well, defaults to "QC sample"
`ctrl_type`	(str) = Match to stype in httr_well, defaults to "QC sample"
`trt_src`	(str) = Match to rna_src in httr_well, defaults to "HBRR"
`ctrl_src`	(str) = Match to rna_src in httr_well, defaults to "UHRR"
`trt_prop_fields`	(list) = Fields in httr_well collection that should match up for all trt_wells and the singular value should be propagated to field of same name in httr_trt_grp_cmp collection. Default is to propagate just the stype field (exclude all chemical and dose information).

`both_prop_fields`	(list of str) = Fields in httr_well collection that should match up for all trt_wells AND ctrl_wells, and the singular value should be propagated to to field of same name in httr_trt_grp_cmp collection. Default is to propagate pg_id and block_id only for ref RNA comparisons
`ctrl_desc_field`	(str) = Field to propagate from ctrl_wells to ctrl field in httr_trt_grp_cmp document, defaults to "trt_name" for ref RNA comparisons
`grp_id_opts`	(list) = Options to modify trt_grp_id generation, "vs" will capture the pairwise comparison type, and "pg" is usually sufficient for ref RNA to create distinct group IDs for each plate group ... = All additional args passed to treatGroupFromSamples

Details

Parameters:

Value

(list) = Document that was inserted into trt_grp collection """ Pass to chemTreatGroup:

`runDESeq2` *runDESeq2 - Core function for running DESeq2 on a set of wells This was adapted from Imran's previous DESeq2 function and I tried to keep all parameters the same Previous version can be found in httr-wf-dev repo in BitBucket, httr-ph-i branch, lib/gexp/deseq2.py, runDESeq4 function The main change here is that the default shrinkage method is now done with the lfcShrink function in DESeq2 package, instead of the old method The column used for primary treatment groups is now trt_name instead of trt_id to keep with the DB schema NOTE: This function does no sample or probe filtering - that was implemented on the python side in Imran's code, and will be handled by a separate function in the R lib NOTE: Derik's version converted conc as a factor and used that as the primary model - results should be equivalent but might be a good idea to sort samples in increasing conc order, ideally in a higher level function that's specific to handling dose-response sets*

Description

runDESeq2 - Core function for running DESeq2 on a set of wells This was adapted from Imran's previous DESeq2 function and I tried to keep all parameters the same Previous version can be found in httr-wf-dev repo in BitBucket, httr-ph-i branch, lib/gexp/deseq2.py, runDESeq4 function The main change here is that the default shrinkage method is now done with the lfcShrink function in DESeq2 package, instead of the old method The column used for primary treatment groups is now trt_name instead of trt_id to keep with the DB schema NOTE: This function does no sample or probe filtering - that was implemented on the python side in Imran's code, and will be handled by a separate function in the R lib NOTE: Derik's version converted conc as a factor and used that as the primary model - results should be equivalent but might be a good idea to sort samples in increasing conc order, ideally in a higher level function that's specific to handling dose-response sets

Usage

```
runDESeq2(
  COUNTS,
  CONDS,
  ref_level = "DMSO_0",
  plate_effect = F,
  plate_cont = F,
  shrinkage = "normal",
  threads = 1,
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`COUNTS`	(*data.frame*) = Data frame containing all counts to be analyzed, row.names = probes, colnames = samples
`CONDS`	(*data.frame*) = Data frame containing treatment data for all samples, rows = samples in same order as COUNTS, must have trt_name and plate_id columns
`ref_level`	(*character*) = Value of "reference" treatment in trt_name column (will be used as model intercept)
`plate_effect`	(*logical*) = Should the DESeq2 model include plate effects?
`plate_cont`	(*logical*) = Should the plate contrasts be returned in addition to the primary treatment contrasts? (CURRENTLY NOT SUPPORTED)
`shrinkage`	(*character*) = Type of shrinkage to use, current options are "normal" for new default shrinkage, "none" for no shrinkage
`threads`	(*integer*) = Number of threads to use, if > 1 parallelization is accomplished with BiocParallel package
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*list of data.frame*) = List with two members, first is the data.frame of all results for primary treatments, second is for plate effects Each data.frame has columns probe_id, trt_name, baseMean, log2FoldChange, lfcSE, stat, pvalue, padj

---

`runDESeq2ForChemCond`

runDESeq2ForChemCond - Run DESeq2 on a single chemical dose-response series Uses DB access functions to extract all httr_trt_grp_id entries corresponding to a dose-response series, then extracts counts for all wells, filters by min read depth and mean floor, and passes through runDESeq2. Also optionally saves the results back to the DB in addition to returning a single data.frame with all results that can be written to a TSV file or saved in Rdata format.

Description

runDESeq2ForChemCond - Run DESeq2 on a single chemical dose-response series Uses DB access functions to extract all httr_trt_grp_id entries corresponding to a dose-response series, then extracts counts for all wells, filters by min read depth and mean floor, and passes through runDESeq2. Also optionally saves the results back to the DB in addition to returning a single data.frame with all results that can be written to a TSV file or saved in Rdata format.

Usage

```
runDESeq2ForChemCond(
  chem_id,
  db_host,
  db_name,
  pg_id = NULL,
  media = NULL,
  timeh = NULL,
  max_dose_level = 8,
  min_colsum = 10^5,
  mean_cnt = getOption("mean_cnt"),
  plate_effect = F,
  shrinkage = "normal",
  threads = 1,
  collections = character(0),
  db_insert = T,
  rerun = F,
  note = NULL,
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`chem_id`	(*character*) = ID for the chemical to run, matched against chem_id field in httr_trt_grp_cmp
`db_host`	(*character*) = Host for DB connection
`db_name`	(*character*) = Name of DB
`pg_id`	(*character*) = Limit httr_trt_grp_cmp to this plate group, useful if multiple dose response series for same chem_id
`media`	(*character*) = Limit httr_trt_grp_cmp to this media type, useful if multiple dose response series for same chem_id
`timeh`	(*integer*) = Limit httr_trt_grp_cmp to this timeh, useful if multiple dose response series for same chem_id
`max_dose_level`	(*integer*) = Max number of dose levels expected
`min_colsum`	(*integer*) = Filter samples by min read total, set to NULL to skip, defaults to 100k
`mean_cnt`	(*integer*) = Filter probes by mean floor, set to NULL to skip, defaults to 5
`plate_effect`	(*logical*) = Whether or not to include plate effect in model, defaults to FALSE
`shrinkage`	(*character*) = Type of shrinkage to use, defaults to new default, a.k.a. "normal"
`threads`	(*integer*) = Number of threads to use for core DESeq2 functions, defaults to 1 (no multi-threading)

`collections`	(*named character vector*) = Optional vector to re-map collection names used, where name=default collection; value=new collection name
`db_insert`	(*logical*) = Whether to insert the results back into the DB, defaults to TRUE
`rerun`	(*logical*) = Whether to re-run the DB insertion if results are in there already (ideally this should check the DB before running...)
`note`	(*character*) = Overwrite default note fields for insert if this is specified.
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*data.frame*) = Contains DESeq2 results for all treatments in the dose-response series

---

`runDESeq2Single` *runDESeq2Single - Run DESeq2 on a single contrast (e.g. single conc reference chemical, bulk lysate, or ref RNA QC samples) Uses DB access functions to extract the httr_trt_grp_cmp entry corresponding to a specific comparison, then extracts counts for all wells, filters by min read depth and mean floor, and passes through runDESeq2. Also optionally saves the results back to the DB in addition to returning a single data.frame with all results that can be written to a TSV file or saved in Rdata format.*

Description

runDESeq2Single - Run DESeq2 on a single contrast (e.g. single conc reference chemical, bulk lysate, or ref RNA QC samples) Uses DB access functions to extract the httr_trt_grp_cmp entry corresponding to a specific comparison, then extracts counts for all wells, filters by min read depth and mean floor, and passes through runDESeq2. Also optionally saves the results back to the DB in addition to returning a single data.frame with all results that can be written to a TSV file or saved in Rdata format.

Usage

```
runDESeq2Single(
  trt_grp_id,
  db_host,
  db_name,
  min_colsum = 10^5,
  mean_cnt = getOption("mean_cnt"),
  plate_effect = F,
  shrinkage = "normal",
  threads = 1,
  collections = character(0),
  db_insert = T,
  rerun = F,
  note = NULL,
  debug = getOption("debug", default = FALSE),
  ...
)
```

Arguments

`trt_grp_id`	(*character*) = ID for the trt_grp_cmp to run, matched against trt_grp_id field in httr_trt_grp_cmp
`db_host`	(*character*) = Host for DB connection
`db_name`	(*character*) = Name of DB
`min_colsum`	(*integer*) = Filter samples by min read total, set to NULL to skip, defaults to 100k
`mean_cnt`	(*integer*) = Filter probes by mean floor, set to NULL to skip, defaults to 5
`plate_effect`	(*logical*) = Whether or not to include plate effect in model, defaults to FALSE
`shrinkage`	(*character*) = Type of shrinkage to use, defaults to new default, a.k.a. "normal"
`threads`	(*integer*) = Number of threads to use for core DESeq2 functions, defaults to 1 (no multi-threading)
`collections`	(*named character vector*) = Optional vector to re-map collection names used, where name=default collection; value=new collection name
`db_insert`	(*logical*) = Whether to insert the results back into the DB, defaults to TRUE
`rerun`	(*logical*) = Whether to re-run the DB insertion if results are in there already (ideally this should check the DB before running...)
`note`	(*character*) = Overwrite default notes with this one in update_notes field.
`debug`	(*logical*) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*data.frame*) = Contains DESeq2 results for all treatments in the dose-response series

---

`sameCtrlWells` *sameCtrlWells*

- *Function to check that a list of httr_trt_grp_cmp docs all have the same control group This is useful when extracting treatment groups that should correspond to a dose-response series*

Description

sameCtrlWells

- Function to check that a list of httr_trt_grp_cmp docs all have the same control group This is useful when extracting treatment groups that should correspond to a dose-response series

Usage

```
sameCtrlWells(trt_list, debug = getOption("debug", default = FALSE))
```

Arguments

`trt_list`	(*list of list*) = List of docs from httr_trt_grp_cmp (list) that are expected to all have the same control wells
`debug`	(logical) = Whether to print debug messages, default: FALSE, overridden by options(debug=...)

Value

(*logical*) = FALSE if there are multiple entries in trt_list with different ctrl_well sets - automatically TRUE if length(trt_list) < 2 although this will trigger a debug message

---

`sampleID_scan_and_update` *sampleID_scan_and_update This function takes two files and scan them for differences*

Description

sampleID_scan_and_update This function takes two files and scan them for differences

Usage

```
sampleID_scan_and_update(
  orig_sampleID_file = NULL,
  sampleID_file = NULL,
  targetWellTrtCol = "httr_well_trt",
  targetChem = "httr_chem",
  skipped_tests = skipped_tests,
  db_host = host,
  db_name = db,
  ...
)
```

Arguments

`orig_sampleID_file:`	(*character*) file that has already been processed prior
`sampleID_file:`	(*character*) new vs of that files with some edits - regardless where the edits took place
`targetWellTrtCol:`	(*character*) target collection where the updates will be written to - by default httr_well_trt
`targetChem:`	(*character*) the target Chem collection where some updates will be written to - by default httr_chem
`skipped_tests:`	(*character vector*) the list of tests to skip when validating the resulting set of data
`db_host:`	(*character*) mongo db server +- port
`db_name:`	(*character*) mongo db database process: The function does a diff on the two files and calls sampleID_wrapper() on that resulting list of modified sampleIds if the list validates against a list of provided tests, the function continues to replace found sampleId documents in target collections.

Value

nothing explicit print the test that fail if any. If validation passes, nothing returned `sampleID_wrapper` *sampleID_wrapper takes either as input: (coming from prior manipulations)*

Description

sampleID_wrapper takes either as input: (coming from prior manipulations)

Usage

```
sampleID_wrapper(
  wellTrt = NULL,
  chemInfo = NULL,
  sampleID_file = NULL,
  targetWellTrtCol = "httr_well_trt",
  targetChem = "httr_chem",
  status = "KEEP",
  skipped_tests = c(),
  db_host = NULL,
  db_name = NULL,
  validate = TRUE,
  max_dose_level = 8,
  required_cols = c("sample_id", "plate_id", "well_id", "trt_name", "qc_flag"),
  extra_cols = c(),
  ...
)
```

Arguments

`wellTrt:`	(*character vector*) a list named values such as: sample_id plate_id well_id chem_id chem_name dtxsid casrn conc conc_unit dose_level block_id replicate_num pg_id culture_id cell_type media stype trt_name rna_src qc_flag TC00001203_C07 TC00001203 C07 EPAPLT0593A13222 Glybenclamide DTXSID0037237 10238-21-8 100.00 uM 8 1 1 1 c2021-03-15 U-2 OS DMEM + 10% FBS test sample EPAPLT0593A13_8_100uM Plated Cells OK OR
`sampleID_file:`	(*character*)file that has such list in json format
`Target`	Chem Collection: (*character*) targetChem defaulting to httr_chem
`status:`	(*numeric*) whether of not we keep prior data if found in target collection (status="KEEP"), erase all items in collection (status="RERUN"), or just replace the sampleIDs found (status="REPLACE")
`tests:`	(*character vector*) list of tests that will be performed in the validation step, an example is provided in tests.json
`db_host:`	(*character*) mongo url with or without port
`db_name:`	(*character*) mongo database or sandbox

Value

nothing explicit

the function will process each sample ids, first making sure they pass the validation tests if they don't pass any of the validation tests declared in the relevant_test list, the offended test will be printed out and the the process is terminated if the validation tests pass, the process continues on to store (by erasing first all or some or none of potentially already present sampleIDs in the target collections) the data into two collections: the target collection and the target Chem collection as specified earlier finally, each sampleID that ended up being replaced in the target collection is removed from the httr_well collection, so that another downstream process may detect the dicrepencies between httr_well_trt and httr_well and rebuild the missing sampleIDs that were found in httr_well_trt but missing in httr_well

---

`splitIDs` *Title splitIDs - Split a list of IDs based on which are present/absent in a collection. Given a list of IDs, search against a particular collection, and return a dict with two lists keyed by "present" and "absent".*

Description

Title splitIDs - Split a list of IDs based on which are present/absent in a collection. Given a list of IDs, search against a particular collection, and return a dict with two lists keyed by "present" and "absent".

Usage

```
splitIDs(DB, ids, id_field = "sample_id")
```

Arguments

`DB`	(*mongo object*) = Open connection to a Mongo DB and Collection
`ids`	(*vector*) = Vector of IDs (typically character)
`id_field`	(*character*) = Name of ID field, default: sample_id

Value

(*list*) with two members: $present = vector of IDs that were found in the collection $absent = vector of IDs that were NOT found in collection

---

`tests` *validate_httr_well_trt_schema*

Description

validate_httr_well_trt_schema

Usage

```
tests
```

Arguments

`wellTrt:`	(*character vector*) a list of named values, such as: sample_id plate_id well_id chem_id chem_name dtxsid casrn conc conc_unit dose_level block_id replicate_num pg_id culture_id cell_type media stype trt_name rna_src qc_flag TC00001203_C07 TC00001203 C07 EPAPLT0593A13222 Glybenclamide DTXSID0037237 10238-21-8 100.00 uM 8 1 1 1 c2021-03-15 U-2 OS DMEM + 10% FBS test sample EPAPLT0593A13_8_100uM Plated Cells OK
`skipped_tests:`	(*character vector*) a list of skipped tests as defined in the tests.json file that are not used to test validate against this long function performs many tests on the data quality and integrity provided in the wellTrt list it delineates the tests that have failed

Format

An object of class `list` of length 51.

Value

nothing explicit

`treatGroupFromSamples` *treatGroupFromSamples Construct a document for httr_trt_grp_cmp collection. Given a list of treatment and control samples, construct an appropriate document and insert into httr_trt_grp_cmp Parameters:*

Description

treatGroupFromSamples Construct a document for httr_trt_grp_cmp collection. Given a list of treatment and control samples, construct an appropriate document and insert into httr_trt_grp_cmp

Parameters:

Usage

```
treatGroupFromSamples(
  DB = getOption("DB_NAME"),
  trt_wells,
  ctrl_wells,
  trt_prop_fields = c("chem_id", "conc", "conc_unit", "dose_level", "stype"),
  both_prop_fields = c("media", "timeh", "pg_id", "block_id"),
  ctrl_desc_field = "stype",
  grp_id_opts = c(),
  well = "httr_well",
  trt_grp = "httr_trt_grp_cmp",
  rerun = FALSE,
  db_insert = TRUE,
  ...
)
```

Arguments

`DB`	(string) = name of database with httr_well and httr_trt_grp_cmp collections
`trt_wells`	(vector) = Sample IDs of all treated wells in the comparison group
`ctrl_wells`	(vector) = Sample IDs of all control wells in the the comparison group
`trt_prop_fields`	(vector) = Fields in httr_well collection that should match up for all trt_wells and the singular value should be propagated to field of same name in httr_trt_grp_cmp collection. Default is to propagate chem_id, conc, conc_unit, dose_level, and stype fields.
`both_prop_fields`	(vector) = Fields in httr_well collection that should match up for all trt_wells AND ctrl_wells, and the singular value should be propagated to to field of same name in httr_trt_grp_cmp collection. Default is to propagate media, timeh, pg_id, and block_id.
`ctrl_desc_field`	(str) = Field to propagate from ctrl_wells to ctrl field in httr_trt_grp_cmp document, defaults to "stype"
`grp_id_opts`	(vector) = Additional modifiers to determine trt_grp_id, "bl" = append block ID, "pg" = append plate group ID, "pl" = append plate ID, "vs" = use trt_name for trt_wells and ctrl_wells, applied in the order specified, defaults to simple use only trt_name from trt_wells
`well`	(str) = Name of collection with individual well treatment data, default is "httr_well"
`trt_grp`	(str) = Name of collection with treatment group data, default is "httr_trt_grp_cmp"
`rerun`	(bool) = If document with same trt_grp_id exists already, should it be replaced? Default is False
`db_insert`	(bool) = Whether to insert the new document into database at all, Default is True

Value

```
(list) = Document that was inserted into trt_grp collection
```

`treatPairGroup` *treatPairGroup Construct a document for httr_trt_grp_cmp collection corresponding to a specific pair of treatment names. Given a specific pair of trt_name values, plus optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This is initially designed for the pairs of BioSpyder QC reference samples now used in newer screens, but should work for other situations as well. Note, if filtering reduces either trt or ctrl group to < min_reps samples this treatment group will be skipped.*

Description

treatPairGroup Construct a document for httr_trt_grp_cmp collection corresponding to a specific pair of treatment names.

Given a specific pair of trt_name values, plus optional filter criteria, construct an appropriate document and insert into httr_trt_grp_cmp. This is initially designed for the pairs of BioSpyder QC reference samples now used in newer screens, but should work for other situations as well. Note, if filtering reduces either trt or ctrl group to < min_reps samples this treatment group will be skipped.

Usage

```
treatPairGroup(
  DB = getOption("DB_NAME"),
  trt_name,
  ctrl_name,
  trt_type = "QC sample",
  ctrl_type = "QC sample",
  pg_id = "",
  media = "",
  timeh = 0,
  trt_src = "",
  ctrl_src = "",
  qc_flags = c("OK"),
  min_reps = 2,
  well = getOption("HTTR_WELL_NAME"),
  ...
)
```

Arguments

`DB`	name of database with httr_well and httr_trt_grp_cmp collections
`trt_name`	(string) = Match to trt_name in httr_well
`ctrl_name`	(string) = Match to trt_name in httr_well, use as control group
`trt_type`	(string) = Match to stype in httr_well, defaults to "QC sample"
`ctrl_type`	(string) = Match to stype in httr_well, defaults to "QC sample"
`pg_id`	(string) = Filter both trt and ctrl wells, match to pg_id in httr_well, defaults to None
`media`	(string) = Filter both trt and ctrl wells, match to media in httr_well, defaults to None
`timeh`	(string) = Filter both trt and ctrl wells, match to timeh in httr_well, defaults to None
`trt_src`	(string) = Filter trt wells, match to rna_src field
`ctrl_src`	(string) = Filter ctrl wells, match to rna_src field
`qc_flags`	(vector) = Filter both trt and ctrl wells, match to qc_flag field, defaults to OK only
`min_reps`	(int) = Minimum number of replicates in trt and ctrl groups, respectively - if either is less, will generate an output message and return empty dict
`well`	(string) = Name of collection with individual well treatment data, default is "httr_well"
`...`	= All additional args passed to treatGroupFromSamples

Value

(list) = Document that was inserted into trt_grp collection

`trtGrpWells` *trtGrpWells - Extract the complete set of wells involved in a series of treatment groups*

Description

trtGrpWells - Extract the complete set of wells involved in a series of treatment groups

Usage

```
trtGrpWells(trt_list, subset = "all")
```

Arguments

`trt_list`	(*list of list*) = List of docs from httr_trt_grp_cmp (list) that are expected to all have the same control wells
`subset`	(character) = Single character value indicating which subset of wells to extract. "all" (default) retrieves both ctrl and trt wells, "ctrl" and "trt" extrct just those well subsets respectively.

Value

(*character vector*) = Unique set of sample_ids for all or subset of wells in trt_list

`tss_deg_count` *tss_deg_count: Function to simply count the number or fraction of probes with |l2fc| >= cutoff (with optional pvalue cutoff)*

Description

tss_deg_count: Function to simply count the number or fraction of probes with |l2fc| >= cutoff (with optional pvalue cutoff)

Usage

```
tss_deg_count(
  degs,
  abs_l2fc,
  pval = NULL,
  l2fc_col = "log2FoldChange",
  pval_col = "pvalue",
  as_frac = FALSE
)
```

Arguments

`degs`	(*data.frame*) = Results from httr_deg for a single trt_grp_id and anl_name, required.
`abs_l2fc`	(*numeric*) = Count probes with |l2fc| >= this cutoff, required.
`pval`	(*numeric*) = Additional threshold for p-value, ignore by default.
`l2fc_col`	(*character*) = Name of l2fc column, defaults to "log2FoldChange" to match DESeq2 output format
`pval_col`	(*character*) = Name of p-value column, defaults to "pvalue" to match DESeq2 output format
`as_frac`	(*logical*) = Whether or not to return result as a fraction of total probes in table, defaults to FALSE

Value

(*numeric or integer*) = Either the total number or fraction of probes passing the threshold criteria

---

`tss_l2_norm` *tss_l2_norm: Function to compute Euclidean (L2) norm for top/bottom N probes by l2fc*

Description

tss_l2_norm: Function to compute Euclidean (L2) norm for top/bottom N probes by l2fc

Usage

```
tss_l2_norm(
  degs,
  n,
  pval = NULL,
  l2fc_col = "log2FoldChange",
  pval_col = "pvalue",
  rank_col = l2fc_col,
  rank_dir = "both",
  debug = getOption("debug", default = FALSE)
)
```

Arguments

`degs`	(*data.frame*) = Results from httr_deg for a single trt_grp_id and anl_name, required.
`n`	(*integer*) = Use the top and bottom n probes by l2fc
`pval`	(*numeric*) = If specified, change all l2fc to 0 for probes with p-values above this threshold before ranking, defaults to NULL
`l2fc_col`	(*character*) = Name of l2fc column, defaults to "log2FoldChange" to match DESeq2 output format
`pval_col`	(*character*) = Name of pvalue column, defaults to "pvalue" to match DESeq2 output format, only matters if pval is specified

`rank_col`	(*character*) = Which column to use for ranking the top/bottom N probes, defaults to l2fc_col but can rank by e.g. pval or SNR metric
`rank_dir`	(*character*) = Whether to use the top N, bottom N, or both top/bottom N when computing L2 Norm, defaults to both
`debug`	(*logical*) = Whether to print debug messages

Value

(*numeric*) = The euclidean L2 norm of the top/bottom n l2fc values

---

`unit_subtest_finish` *unit_subtest_finish generate a single row of output for the outcome of one test in a unit test and append it to current results*

Description

unit_subtest_finish generate a single row of output for the outcome of one test in a unit test and append it to current results

Usage

```
unit_subtest_finish(col = NA, row = NA, test_name, test_status, input)
```

Arguments

`col`	(*int*) = olumn number within the entire tested object that the test was run on
`row`	(*int*) = column number within the entire tested object that the test was run on
`test_name`	(*character*) = description of what was tested
`test_status`	(*character*) = either "PASS" or "FAIL" - the outcome of the test
`input`	(*data.frame*) = the growing result object report - 4 columns deep

Value

Value (*data.frame*) = returns a dataframe with 4 columns

`unit_subtest_ident` *unit_subtest_set - test whether two vectors contain the same set of elements*

Description

unit_subtest_set - test whether two vectors contain the same set of elements

Usage

```
unit_subtest_ident(result_1, result_2, running_result, row = NA, col = NA)
```

Arguments

`result_1`	(*vector*) = object to be tested
`result_2`	(*vector*) = object to be used as a reference
`running_result`	(*data.frame*) = result object to which subtest results should be appanded
`row`	(*int*) = row number within the entire tested object that the test was run on
`col`	(*int*) = column number within the entire tested object that the test was run on

Value

(*data.frame*) = returns a dataframe with 4 columns

`unit_subtest_length` *unit_subtest_length - test whether two objects are of the same length*

Description

unit_subtest_length - test whether two objects are of the same length

Usage

```
unit_subtest_length(result_1, result_2, running_result, col = NA, row = NA)
```

Arguments

`result_1`	(*vector*) = object to be tested
`result_2`	(*vector*) = object to be used as a reference
`running_result`	(*data.frame*) = result object to which subtest results should be appanded
`col`	(*int*) = column number within the entire tested object that the test was run on
`row`	(*int*) = row number within the entire tested object that the test was run on

Value

(*data.frame*) = returns a dataframe with 4 columns

`unit_subtest_set` *unit_subtest_set - test whether two vectors contain the same set of elements*

Description

unit_subtest_set - test whether two vectors contain the same set of elements

Usage

```
unit_subtest_set(result_1, result_2, running_result, col = NA, row = NA)
```

Arguments

`result_1`	(*vector*) = object to be tested
`result_2`	(*vector*) = object to be used as a reference
`running_result`	(*data.frame*) = result object to which subtest results should be appanded
`col`	(*int*) = column number within the entire tested object that the test was run on
`row`	(*int*) = row number within the entire tested object that the test was run on
`return`	(*data.frame*) = returns a dataframe with 4 columns

`unit_subtest_type` *unit_subtest_type - test whether two objects are of the same type - a wrapper around typeof()*

Description

unit_subtest_type - test whether two objects are of the same type - a wrapper around typeof()

Usage

```
unit_subtest_type(result_1, result_2, running_result)
```

Arguments

`result_1`	(any format) = object to be tested
`result_2`	(any format) = object to be used as a reference
`running_result`	(data.frame) = result object to which subtest results should be appanded

Value

(data.frame) = returns a dataframe with 4 columns

`unit_subtest_values` *unit_subtest_values - depreciated - use unit_subtest_values_1 instead*

Description

unit_subtest_values - depreciated - use unit_subtest_values_1 instead

Usage

```
unit_subtest_values(result_1, result_2, running_result, tolerance)
```

`unit_subtest_values_1` *unit_subtest_values_1 - test whether two vectors contain the same values*

Description

unit_subtest_values_1 - test whether two vectors contain the same values

Usage

```
unit_subtest_values_1(result_1, result_2, running_result, tolerance)
```

Arguments

`result_1`	(*vector*) = object to be tested
`result_2`	(*vector*) = object to be used as a reference
`running_result`	(*data.frame*) = result object to which subtest results should be appanded
`tolerance`	(*numeric*) = tolerance level expressed as a decimal for how much the results can vary without triggering a test failure

Value

(*data.frame*) = returns a dataframe with 4 columns

`unit_test` *unit_test- test whether two data frames contain the same values*

Description

unit_test- test whether two data frames contain the same values

Usage

```
unit_test(result_1, result_2, tolerance)
```

Arguments

`result_1`	(data.frame) = object to be tested
`result_2`	(data.frame) = object to be used as a reference
`tolerance`	(numeric) = tolerance level expressed as a decimal for how much the results can vary without triggering a test failure

Value

(data.frame) = returns a dataframe with 4 columns

`updateqcFlags` *updateqcFlags Function to update QC_flag for samples based on changed QC_flag threshold. Updates are saved into httr_counts_qc collection function iterates through all documents found in httr_count_qc or iterates through all documents found in httr_count_qc and the result of a passed query to httr_well_trt (query passed in the ...)*

Description

updateqcFlags Function to update QC_flag for samples based on changed QC_flag threshold. Updates are saved into httr_counts_qc collection function iterates through all documents found in httr_count_qc or iterates through all documents found in httr_count_qc and the result of a passed query to httr_well_trt (query passed in the ...)

Usage

```
updateqcFlags(
  db_host,
  db_name,
  db_collection = "httr_counts_qc",
  min_mapd_frac = get_global_option("min_mapd_frac"),
  min_n_reads_mapd = get_global_option("min_n_reads_mapd"),
  min_n_sig80 = get_global_option("min_n_sig80"),
  min_n_cov5 = get_global_option("min_n_cov5"),
  max_top10_prop = get_global_option("max_top10_prop"),
  max_gini_coef = get_global_option("max_gini_coef"),
    qc_flags = get_global_option("qc_flags"),
    debug = getOption("debug", default = FALSE),
    ...
)
```

Arguments

`db_host`	(*character*) = MongoDB host URL
`db_name`	(*character*) = MongoDB name
`db_collection`	(*character*) = Name of counts_qc collection, default: httr_counts_qc
`min_mapd_frac`	(*numeric*) = minimum mapd_frac, flag with LOW_MAPD_FRAC below this cutoff
`min_n_reads_mapd`	(*integer*) = minimum n_reads_mapd, flag with LOW_READS below this cutoff
`min_n_sig80`	(*integer*) = minimum n_sig80, flag with LOW_NSIG80 below this cutoff
`min_n_cov5`	(*integer*) = minimum n_cov5, flag with LOW_NCOV5 below this cutoff
`max_top10_prop`	(*numeric*) = maximum top10_prop, flag with HIGH_TOP10 above this cutoff
`max_gini_coef`	(*numeric*) = maximum gini_coef, flag with HIGH_GINI above this cutoff
`qc_flags`	(*character vector*) = list of flags to apply in priority order
`debug`	(*logical*) = Whether to report debug messages

Value

nothing explicitly

`update_attenuation_factors` *update_attenuation_factors Create function in httr/Rlib/db/probe.R to update/replace attenuations factors in an existing httr_probe collection*

Description

update_attenuation_factors Create function in httr/Rlib/db/probe.R to update/replace attenuations factors in an existing httr_probe collection

Usage

```
update_attenuation_factors(host, db, input_file, collection = "httr_probe")
```

Arguments

`host:`	mongo url with or without port
`db:`	mongo database or sandbox
`collection:`	mongo collection to update - default httr_probe changes attenuation values for probes found in dataframe in the collection and change to '1' any probe in collection not in the data.frame

Value

nothing explicit

`validateProbeManifest` *validateProbeManifest Function to validate a probe manifest (taken as a data.frame), so that there is a check to ensure the probe manifest is formatted correctly, is not missing and required data, and can be used.*

Description

validateProbeManifest Function to validate a probe manifest (taken as a data.frame), so that there is a check to ensure the probe manifest is formatted correctly, is not missing and required data, and can be used.

Usage

```
validateProbeManifest(input_file)
```

Arguments

`input_file`	= defaults to ../data/httrpl_automationTestData/httr_probe_data/human_wt_1.2.csv constraints are define @ https://teams.microsoft.com/l/entity/com.microsoft.teamspace.tab.wiki/tab:: f6ec-490c-87e1-685a2c988b22?context=%7B%22subEntityId%22%3A%22%7B%5C%22pageId% 6748-4867-acf9-76aacbeca6a7

Value

a list of validation errors or an empty list if all tests pass

Index