Package: hubData (via r-universe)

November 25, 2024

Title Tools for accessing and working with hubverse data

Version 1.3.0

Description A set of utility functions for accessing and working with forecast and target data from Infectious Disease Modeling Hubs.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/testthat/edition 3

URL https://github.com/hubverse-org/hubData

BugReports https://github.com/hubverse-org/hubData/issues

Imports arrow (>= 17.0.0), checkmate, cli, dplyr (>= 1.1.0), fs, hubUtils (>= 0.3.0), lifecycle, magrittr, purrr, rlang, stringr, tibble, yaml, zoltr (>= 1.0.1)

Suggests covr, curl, dbplyr, digest, duckdb, gh, jsonlite, knitr, mockery, rmarkdown, testthat (>= 3.2.0), withr

Remotes hubverse-org/hubUtils

Config/Needs/website hubverse-org/hubStyle

Depends R (>= 2.10)

VignetteBuilder knitr

Config/pak/sysreqs cmake git make libicu-dev libssl-dev libx11-dev

Repository https://hubverse-org.r-universe.dev

RemoteUrl https://github.com/hubverse-org/hubData

RemoteRef v1.3.0

RemoteSha b7dd1686c9fb636f90434afd82c84e5275f12c0d

Contents

`coerce_to_hub_schema` *Coerce data.frame/tibble column data types to hub schema data types or character.*

Description

Coerce data.frame/tibble column data types to hub schema data types or character.

Usage

```
coerce_to_hub_schema(
  tbl,
  config_tasks,
  skip_date_coercion = FALSE,
  as_arrow_table = FALSE,
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date")
)

coerce_to_character(tbl, as_arrow_table = FALSE)
```

Arguments

`tbl`	a model output data.frame/tibble
`config_tasks`	a list version of the content's of a hub's `tasks.json` config file created using function `hubUtils::read_config()`.
`skip_date_coercion`	Logical. Whether to skip coercing dates. This can be faster, especially for larger `tbl`s.
`as_arrow_table`	Logical. Whether to return an arrow table. Defaults to `FALSE`.

output_type_id_datatype
: character string. One of "from_config", "auto", "character", "double", "integer", "logical", "Date". Defaults to "from_config" which uses the setting in the output_type_id_datatype property in the tasks.json config file if available. If the property is not set in the config, the argument falls back to "auto" which determines the output_type_id data type automatically from the tasks.json config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where output_type_ids are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.

Value

tbl with column data types coerced to hub schema data types or character. if as_arrow_table = TRUE, output is also converted to arrow table.

Functions

- coerce_to_hub_schema(): coerce columns to hub schema data types.
- coerce_to_character(): coerce all columns to character

---

collect_hub *Collect Hub model output data*

Description

collect_hub retrieves data from a <hub_connection>/<mod_out_connection> after executing any <arrow_dplyr_query> into a local tibble. The function also attempts to convert the output to a model_out_tbl class object before returning.

Usage

```
collect_hub(x, silent = FALSE, ...)
```

Arguments

x	a <hub_connection>/<mod_out_connection> or <arrow_dplyr_query> object.
silent	Logical. Whether to suppress message generated if conversion to model_out_tbl fails.
...	Further argument passed on to as_model_out_tbl().

Value

A `model_out_tbl`, unless conversion to `model_out_tbl` fails in which case a `tibble` is returned.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubUtils")
hub_con <- connect_hub(hub_path)
# Collect all data in a hub
hub_con %>% collect_hub()
# Filter data before collecting
hub_con %>%
  dplyr::filter(is.na(output_type_id)) %>%
  collect_hub()
# Pass arguments to as_model_out_tbl()
dplyr::filter(hub_con, is.na(output_type_id)) %>%
  collect_hub(remove_empty = TRUE)
```

---

collect_zoltar *Load forecasts from zoltardata.com in hubverse format*

Description

`collect_zoltar` retrieves data from a zoltardata.com project and transforms it from Zoltar's native download format into a hubverse one. Zoltar (documentation here) is a pre-hubverse research project that implements a repository of model forecast results, including tools to administer, query, and visualize uploaded data, along with R and Python APIs to access data programmatically (zoltr and zoltpy, respectively.) (This hubData function is itself implemented using the zoltr package.)

Usage

```
collect_zoltar(
  project_name,
  models = NULL,
  timezeros = NULL,
  units = NULL,
  targets = NULL,
  types = NULL,
  as_of = NULL,
  point_output_type = "median"
)
```

Arguments

`project_name`	A string naming the Zoltar project to load forecasts from. Assumes the host is zoltardata.com .

`models` A character vector that specifies the models to query. Must be model abbreviations. Defaults to NULL, which queries all models in the project.

`timezeros` A character vector that specifies the timezeros to query. Must be yyyy-mm-dd format. Defaults to NULL, which queries all timezeros in the project.

`units` A character vector that specifies the units to query. Must be unit abbreviations. Defaults to NULL, which queries all units in the project.

`targets` A character vector that specifies the targets to query. Must be target names. Defaults to NULL, which queries all targets in the project.

`types` A character vector that specifies the forecast types to query. Choices are "bin", "point", "sample", "quantile", "mean", and "median". Defaults to NULL, which queries all types in the project. Note: While Zoltar supports "named" and "mode" forecasts, this function ignores them.

`as_of` A datetime string that specifies the forecast version. The datetime must include timezone information for disambiguation, without which the query will fail. The datatime parsing function used below (`base::strftime`) is extremely lenient when it comes to formatting, so please exercise caution. Defaults to NULL to load the latest version.

`point_output_type`
A string that specifies how to convert zoltar `point` forecast data to hubverse output type. Must be either "median" or "mean". Defaults to "median".

Details

Zoltar's data model differs from that of the hubverse in a few important ways. While Zoltar's model has the concepts of unit, target, and timezero, hubverse projects have hub-configurable columns, which makes the mapping from the former to the latter imperfect. In particular, Zoltar units translate roughly to hubverse task IDs, Zoltar targets include both the target outcome and numeric horizon in the target name, and Zoltar timezeros map to round ids. Finally, Zoltar's forecast types differ from those of the hubverse. Whereas Zoltar has seven types (bin, named, point, sample, quantile, mean, median, and mode), the hubverse has six (cdf, mean, median, pmf, quantile, sample), only some of which overlap.

Additional notes:

- Requires the user to have a Zoltar account (use the Zoltar contact page to request one).
- Requires `Z_USERNAME` and `Z_PASSWORD` environment vars to be set to those of the user's Zoltar account.
- While Zoltar supports "named" and "mode" forecasts, this function ignores them.
- Rows with non-numeric values are ignored.
- This function removes numeric_horizon mentions from zoltar target names. Target names can contain a maximum of one numeric_horizon. Example: "1 wk ahead inc case" -> "wk ahead inc case".
- Querying a large number of rows may cause errors, so we recommend providing one or more filtering arguments (e.g., models, timezeros, etc.) to limit the result.

Value

A hubverse model_out_tbl containing the following columns: ”model_id”, ”timezero”, ”season”, ”unit”, ”horizon”, ”target”, ”output_type”, ”output_type_id”, and ”value”.

Examples

```
## Not run:
df <- collect_zoltar("Docs Example Project")
df <-
  collect_zoltar("Docs Example Project", models = c("docs_mod"),
                         timezeros = c("2011-10-16"), units = c("loc1", "loc3"),
                         targets = c("pct next week", "cases next week"), types = c("point"),
                         as_of = NULL, point_output_type = "mean")

## End(Not run)
```

---

connect_hub *Connect to model output data.*

Description

Connect to data in a model output directory through a Modeling Hub or directly. Data can be stored in a local directory or in the cloud on AWS or GCS.

Usage

```
connect_hub(
  hub_path,
  file_format = c("csv", "parquet", "arrow"),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  partitions = list(model_id = arrow::utf8()),
  skip_checks = FALSE
)

connect_model_output(
  model_output_dir,
  file_format = c("csv", "parquet", "arrow"),
  partition_names = "model_id",
  schema = NULL,
  skip_checks = FALSE
)
```

Arguments

hub_path	Either a character string path to a local Modeling Hub directory or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package. The hub must be fully configured with valid admin.json and tasks.json files within the hub-config directory.
file_format	The file format model output files are stored in. For connection to a fully configured hub, accessed through hub_path, file_format is inferred from the hub's file_format configuration in admin.json and is ignored by default. If supplied, it will override hub configuration setting. Multiple formats can be supplied to connect_hub but only a single file format can be supplied to connect_mod_out.
output_type_id_datatype	character string. One of "from_config", "auto", "character", "double", "integer", "logical", "Date". Defaults to "from_config" which uses the setting in the output_type_id_datatype property in the tasks.json config file if available. If the property is not set in the config, the argument falls back to "auto" which determines the output_type_id data type automatically from the tasks.json config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where output_type_ids are NA,) are being collected by a hub, the output_type_id column is assigned a character data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce output_type_id to a data type that is not valid for the data (e.g. trying to coerce"character" values to "double") will likely result in an error or potentially unexpected behaviour so use with care.
partitions	a named list specifying the arrow data types of any partitioning column.
skip_checks	Logical. If FALSE (default), check file_format parameter against the hub's model output files. Also excludes invalid model output files when opening hub datasets. Setting to TRUE will improve performance but will result in an error if the model output directory includes invalid files. **Cannot be** TRUE when there are **multiple file formats** in the hub's model output directory or when the hub's model output directory contains **files that are not model output data** (for example, a README).
model_output_dir	Either a character string path to a local directory containing model output data or an object of class <SubTreeFileSystem> created using functions s3_bucket() or gs_bucket() by providing a string S3 or GCS bucket name or path to a directory containing model output data stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the arrow package.
partition_names	character vector that defines the field names to which recursive directory names correspond to. Defaults to a single model_id field which reflects

the standard expected structure of a `model-output` directory.

`schema` An arrow::Schema object for the Dataset. If NULL (the default), the schema will be inferred from the data sources.

Value

- `connect_hub` returns an S3 object of class `<hub_connection>`.
- `connect_mod_out` returns an S3 object of class `<mod_out_connection>`.

Both objects are connected to the data in the model-output directory via an Apache arrow `FileSystemDataset` connection. The connection can be used to extract data using `dplyr` custom queries. The `<hub_connection>` class also contains modeling hub metadata.

Functions

- `connect_hub()`: connect to a fully configured Modeling Hub directory.
- `connect_model_output()`: connect directly to a `model-output` directory. This function can be used to access data directly from an appropriately set up model output directory which is not part of a fully configured hub.

Examples

```
# Connect to a local simple forecasting Hub.
hub_path <- system.file("testhubs/simple", package = "hubUtils")
hub_con <- connect_hub(hub_path)
hub_con
hub_con <- connect_hub(hub_path, output_type_id_datatype = "character")
hub_con
# Connect directly to a local `model-output` directory
mod_out_path <- system.file("testhubs/simple/model-output", package = "hubUtils")
mod_out_con <- connect_model_output(mod_out_path)
mod_out_con
# Query hub_connection for data
library(dplyr)
hub_con %>%
  filter(
    origin_date == "2022-10-08",
    horizon == 2
  ) %>%
  collect_hub()
mod_out_con %>%
  filter(
    origin_date == "2022-10-08",
    horizon == 2
  ) %>%
  collect_hub()
# Connect to a simple forecasting Hub stored in an AWS S3 bucket.
## Not run:
hub_path <- s3_bucket("hubverse/hubutils/testhubs/simple/")
hub_con <- connect_hub(hub_path)
hub_con
```

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

`create_hub_schema` *Create a Hub arrow schema*

Description

Create an arrow schema from a `tasks.json` config file. For use when opening an arrow dataset.

Usage

```
create_hub_schema(
  config_tasks,
  partitions = list(model_id = arrow::utf8()),
  output_type_id_datatype = c("from_config", "auto", "character", "double", "integer",
    "logical", "Date"),
  r_schema = FALSE
)
```

Arguments

`config_tasks`	a list version of the content's of a hub's `tasks.json` config file created using function `hubUtils::read_config()`.
`partitions`	a named list specifying the arrow data types of any partitioning column.
`output_type_id_datatype`	character string. One of `"from_config"`, `"auto"`, `"character"`, `"double"`, `"integer"`, `"logical"`, `"Date"`. Defaults to `"from_config"` which uses the setting in the `output_type_id_datatype` property in the `tasks.json` config file if available. If the property is not set in the config, the argument falls back to `"auto"` which determines the `output_type_id` data type automatically from the `tasks.json` config file as the simplest data type required to represent all output type ID values across all output types in the hub. When only point estimate output types (where `output_type_ids` are `NA`,) are being collected by a hub, the `output_type_id` column is assigned a `character` data type when auto-determined. Other data type values can be used to override automatic determination. Note that attempting to coerce `output_type_id` to a data type that is not valid for the data (e.g. trying to coerce`"character"` values to `"double"`) will likely result in an error or potentially unexpected behaviour so use with care.
`r_schema`	Logical. If `FALSE` (default), return an `arrow::schema()` object. If `TRUE`, return a character vector of R data types.

Value

an arrow schema object that can be used to define column datatypes when opening model output data. If `r_schema = TRUE`, a character vector of R data types.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubUtils")
config_tasks <- hubUtils::read_config(hub_path, "tasks")
schema <- create_hub_schema(config_tasks)
```

`create_model_out_submit_tmpl` *Create a model output submission file template*

Description

[**Defunct**] This function has been moved to the `hubValidations` package and renamed to `submission_tmpl()`.

Usage

```
create_model_out_submit_tmpl(
  hub_con,
  config_tasks,
  round_id,
  required_vals_only = FALSE,
  complete_cases_only = TRUE
)
```

Arguments

`hub_con`	A `<hub_connection>` class object.
`config_tasks`	a list version of the content's of a hub's `tasks.json` config file, accessed through the `"config_tasks"` attribute of a `<hub_connection>` object or function `hubUtils::read_config()`.
`round_id`	Character string. Round identifier. If the round is set to `round_id_from_variable: true`, IDs are values of the task ID defined in the round's `round_id` property of `config_tasks`. Otherwise should match round's `round_id` value in config. Ignored if hub contains only a single round.
`required_vals_only`	Logical. Whether to return only combinations of Task ID and related output type ID required values.
`complete_cases_only`	Logical. If `TRUE` (default) and `required_vals_only = TRUE`, only rows with complete cases of combinations of required values are returned. If `FALSE`, rows with incomplete cases of combinations of required values are included in the output.

Details

For task IDs or output_type_ids where all values are optional, by default, columns are included as columns of `NA`s when `required_vals_only = TRUE`. When such columns exist, the function returns a tibble with zero rows, as no complete cases of required value combinations exists. *(Note that determination of complete cases does excludes valid* `NA` `output_type_id` *values in* `"mean"` *and* `"median"` *output types).* To return a template of incomplete required cases, which includes `NA` columns, use `complete_cases_only = FALSE`.

When sample output types are included in the output, the `output_type_id` column contains example sample indexes which are useful for identifying the compound task ID structure of multivariate sampling distributions in particular, i.e. which combinations of task ID values represent individual samples.

When a round is set to `round_id_from_variable: true`, the value of the task ID from which round IDs are derived (i.e. the task ID specified in `round_id` property of `config_tasks`) is set to the value of the `round_id` argument in the returned output.

Value

a tibble template containing an expanded grid of valid task ID and output type ID value combinations for a given submission round and output type. If `required_vals_only = TRUE`, values are limited to the combination of required values only.

---

expand_model_out_val_grid

Create expanded grid of valid task ID and output type value combinations

---

Description

[**Defunct**] This function has been moved to the `hubValidations` package and renamed to `expand_model_out_grid()`.

Usage

```
expand_model_out_val_grid(
  config_tasks,
  round_id,
  required_vals_only = FALSE,
  all_character = FALSE,
  as_arrow_table = FALSE,
  bind_model_tasks = TRUE,
  include_sample_ids = FALSE
)
```

Arguments

config_tasks	a list version of the content's of a hub's tasks.json config file, accessed through the "config_tasks" attribute of a <hub_connection> object or function hubUtils::read_config().
round_id	Character string. Round identifier. If the round is set to round_id_from_variable: true, IDs are values of the task ID defined in the round's round_id property of config_tasks. Otherwise should match round's round_id value in config. Ignored if hub contains only a single round.
required_vals_only	Logical. Whether to return only combinations of Task ID and related output type ID required values.
all_character	Logical. Whether to return all character column.
as_arrow_table	Logical. Whether to return an arrow table. Defaults to FALSE.
bind_model_tasks	Logical. Whether to bind expanded grids of values from multiple modeling tasks into a single tibble/arrow table or return a list.
include_sample_ids	Logical. Whether to include sample identifiers in the output_type_id column.

Details

When a round is set to round_id_from_variable: true, the value of the task ID from which round IDs are derived (i.e. the task ID specified in round_id property of config_tasks) is set to the value of the round_id argument in the returned output.

When sample output types are included in the output and include_sample_ids = TRUE, the output_type_id column contains example sample indexes which are useful for identifying the compound task ID structure of multivariate sampling distributions in particular, i.e. which combinations of task ID values represent individual samples.

Value

If bind_model_tasks = TRUE (default) a tibble or arrow table containing all possible task ID and related output type ID value combinations. If bind_model_tasks = FALSE, a list containing a tibble or arrow table for each round modeling task.

Columns are coerced to data types according to the hub schema, unless all_character = TRUE. If all_character = TRUE, all columns are returned as character which can be faster when large expanded grids are expected. If required_vals_only = TRUE, values are limited to the combinations of required values only.

gs_bucket *Connect to a Google Cloud Storage (GCS) bucket*

Description

See `arrow::gs_bucket()` for details.

Value

A `SubTreeFileSystem` containing an `GcsFileSystem` and the bucket's relative path. Note that this function's success does not guarantee that you are authorized to access the bucket's contents.

Examples

```
bucket <- gs_bucket("voltrondata-labs-datasets")
```

load_model_metadata *Compile hub model metadata*

Description

Loads in hub model metadata for all models or a specified subset of models and compiles it into a tibble with one row per model.

Usage

```
load_model_metadata(hub_path, model_ids = NULL)
```

Arguments

`hub_path`	Either a character string path to a local Modeling Hub directory or an object of class `<SubTreeFileSystem>` created using functions `s3_bucket()` or `gs_bucket()` by providing a string S3 or GCS bucket name or path to a Modeling Hub directory stored in the cloud. For more details consult the Using cloud storage (S3, GCS) in the `arrow` package.
`model_ids`	A vector of character strings of models for which to load metadata. Defaults to NULL, in which case metadata for all models is loaded.

Value

`tibble` with model metadata. One row for each model, one column for each top-level field in the metadata file. For metadata files with nested structures, this tibble may contain list-columns where the entries are lists containing the nested metadata values.

Examples

```
# Load in model metadata from local hub
hub_path <- system.file("testhubs/simple", package = "hubUtils")
load_model_metadata(hub_path)
load_model_metadata(hub_path, model_ids = c("hub-baseline"))
```

---

print.hub_connection *Print a* <hub_connection> *or* <mod_out_connection> *S3 class object*

Description

Print a <hub_connection> or <mod_out_connection> S3 class object

Usage

```
## S3 method for class 'hub_connection'
print(x, verbose = FALSE, ...)

## S3 method for class 'mod_out_connection'
print(x, verbose = FALSE, ...)
```

Arguments

x	A <hub_connection> or <mod_out_connection> S3 class object.
verbose	Logical. Whether to print the full structure of the object. Defaults to FALSE.
...	Further arguments passed to or from other methods.

Functions

- print(hub_connection): print a <hub_connection> object.
- print(mod_out_connection): print a <mod_out_connection> object.

Examples

```
hub_path <- system.file("testhubs/simple", package = "hubUtils")
hub_con <- connect_hub(hub_path)
hub_con
print(hub_con)
print(hub_con, verbose = TRUE)
mod_out_path <- system.file("testhubs/simple/model-output", package = "hubUtils")
mod_out_con <- connect_model_output(mod_out_path)
print(mod_out_con)
```

s3_bucket *Connect to an AWS S3 bucket*

Description

See arrow::s3_bucket() for details.

Value

A SubTreeFileSystem containing an S3FileSystem and the bucket's relative path. Note that this function's success does not guarantee that you are authorized to access the bucket's contents.

Examples

```
bucket <- s3_bucket("voltrondata-labs-datasets")


# Turn on debug logging. The following line of code should be run in a fresh
# R session prior to any calls to `s3_bucket()` (or other S3 functions)
Sys.setenv("ARROW_S3_LOG_LEVEL"="DEBUG")
bucket <- s3_bucket("voltrondata-labs-datasets")
```

Index