Package 'allometric'

April 21, 2024

Title Structured Allometric Models for Trees

Version 2.3.0

Description Access allometric models used in forest resource analysis, such as volume equations, taper equations, biomass models, among many others. Users are able to efficiently find and select allometric models suitable for their project area and use them in analysis. Additionally, 'allometric' provides a structured framework for adding new models to an open-source models repository.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports dplyr, methods, rlang, stringr, tibble, units, RefManageR, magrittr, purrr, ISOcodes, tidyr, progress, vctrs, openssl, curl, jsonlite

Collate 'all_generics.R' 'validity_checks.R' 'AllometricModel.R' 'ModelSet.R' 'ParametricSet.R' 'ParametricModel.R' 'FixedEffectsModel.R' 'FixedEffectsSet.R' 'MixedEffectsModel.R' 'MixedEffectsSet.R' 'Publication.R' 'Taxa.R' 'Taxon.R' 'allometric.R' 'boilerplate.R' 'data.R' 'eq.R' 'fromJSON.R' 'install.R' 'load.R' 'model_tbl.R' 'publication_processing.R' 'summary.R' 'toJSON.R' 'util.R' 'utils-pipe.R' 'variable_defs.R' 'zzz.R'

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Depends R (>= 2.10)

LazyData true

BugReports https://github.com/allometric/allometric/issues

Contact bfrank70@gmail.com

NeedsCompilation no

Author Bryce Frank [aut, cre] (<https://orcid.org/0000-0003-2980-6860>), Francisco Mauro [aut] (<https://orcid.org/0000-0002-7832-6268>), Elijah Allensworth [aut]

Maintainer Bryce Frank <bfrank70@gmail.com>

Repository CRAN

Date/Publication 2024-04-21 21:12:33 UTC

R topics documented:

`allometric-package` *allometric: Structured Allometric Models for Trees*

Description

To learn more about allometric, refer to the documentation website.

Author(s)

Maintainer: Bryce Frank <bfrank70@gmail.com> (ORCID)

Authors:

- Francisco Mauro <francisco.mauro@uva.es> (ORCID)
- Elijah Allensworth <elijah.allensworth@protonmail.com>

See Also

Useful links:

- Report bugs at https://github.com/allometric/allometric/issues

---

`==,FixedEffectsModel,FixedEffectsModel-method`
Check equivalence of fixed effects models

Description

Fixed effects models are considered equal if all of the following are true:

- The model IDs are equal (or not present)
- The response unit names and units are the same
- The covariate names and units are the same and are in the same order
- The specification names and values are the same
- The predict_fn is the same
- The response definitions are the same
- The covariate definitions are the same

Usage

```
## S4 method for signature 'FixedEffectsModel,FixedEffectsModel'
e1 == e2
```

Arguments

`e1`	A `FixedEffectsModel` object
`e2`	A `FixedEffectsModel` object

==,MixedEffectsModel,MixedEffectsModel-method

Check equivalence of mixed effects models

Description

Fixed effects models are considered equal if all of the following are true:

- The model IDs are equal (or not present)
- The response unit names and units are the same
- The covariate names and units are the same and are in the same order
- The specification names and values are the same
- The predict_fn are the same
- The predict_ranef are the same
- The fixed_only slots are the same
- The response definitions are the same
- The covariate definitions are the same

Usage

```
## S4 method for signature 'MixedEffectsModel,MixedEffectsModel'
e1 == e2
```

Arguments

e1	A MixedEffectsModel object
e2	A MixedEffectsModel object

add_model

Add a model to a publication

Description

This function adds objects of class FixedEffectsModel or MixedEffectsModel to a publication. Models added in this way are added as a set containing only one model. This operation is not done in-place.

Usage

```
add_model(publication, model)
```

Arguments

publication	The publication for which a set will be added
model	The model to add to the publication

Value

A publication with the added model

---

add_set *Add a set of models to a publication*

Description

This function adds objects of class FixedEffectsSet or MixedEffectsSet to a publication. This operation is not done in-place.

Usage

```
add_set(publication, model_set)

## S4 method for signature 'Publication'
add_set(publication, model_set)
```

Arguments

publication	The publication for which a set will be added
model_set	The set of models to add to the publication

Value

A publication with the added set

---

aggregate_taxa *Aggregate family, genus, and species columns of 'tbl_df" into taxa data structure*

Description

This function facilitates aggregating family, genus, and species columns into the taxa data structure, which is a nested list composed of multiple "taxons". A taxon is a list containing family, genus, and species fields.

Usage

```
aggregate_taxa(table, grouping_col = NULL)
```

Arguments

table	The table for which the taxa will be aggregated
grouping_col	An optional column to group on when creating taxa. Rows with the same grouping_col value will be stored into the same taxa.

Value

A tibble with family, genus, and species columns added

---

brackett_acer *Brackett Acer Volume Model*

Description

An example allometric model that predicts volume for the genus *Acer*.

Usage

```
brackett_acer
```

Format

An object of class `FixedEffectsModel` of length 1.

---

brackett_rubra *An object of class* `FixedEffectsModel`

Description

Brackett Rubra Volume Model

Usage

```
brackett_rubra
```

Format

An object of class `FixedEffectsModel` of length 1.

Details

An example allometric model that predicts volume for *Alnus rubra*.

check_models_installed *Check if allometric models are currently installed*

Description

Check if allometric models are currently installed

Usage

```
check_models_installed(verbose = FALSE)
```

Arguments

verbose	Print verbose messages if TRUE

fia_trees *FIA Trees Data*

Description

A subset of data from FIA plots located in Oregon.

Usage

```
fia_trees
```

Format

`fia_trees`:
A data frame with 298 rows and 5 columns.

PLOT A plot ID
SPCD The FIA species code for the tree
DIA The diameter of the tree in inches
HT The height of the tree in feet
TPA_UNADJ The unadjusted trees per acre of the tree

Source

https://experience.arcgis.com/experience/3641cea45d614ab88791aef54f3a1849/

`FixedEffectsModel` *Create a fixed effects model*

Description

`FixedEffectsModel` represents an allometric model that only uses fixed effects.

Usage

```
FixedEffectsModel(
  response,
  covariates,
  predict_fn,
  parameters,
  descriptors = list(),
  response_definition = NA_character_,
  covariate_definitions = list()
)
```

Arguments

`response`	A named list containing one element, with a name representing the response variable and a value representing the units of the response variable using the `units::as_units` function.
`covariates`	A named list containing the covariate specifications, with names representing the covariate name and the values representing the units of the coavariate using the `units::as_units` function
`predict_fn`	A function that takes the covariate names as arguments and returns a prediction of the response variable. This function should be vectorized.
`parameters`	A named list of parameters and their values
`descriptors`	An optional named list of descriptors that describe the context of the allometric model
`response_definition`	A string containing an optional custom response definition, which is used instead of the description given by the variable naming system.
`covariate_definitions`	An optional named list of custom covariate definitions that will supersede the definitions given by the variable naming system. The names of the list must match the covariate names given in `covariates`.

Value

An object of class `FixedEffectsModel`

Slots

`response_unit` A one-element list with the name indicating the response variable and the value as the response variable units obtained using `units::as_units()`

`covariate_units` A list containing the covariate names as names and values as the values of the covariate units obtained using `units::as_units()`

`predict_fn` The prediction function, which takes covariates as arguments and returns model predictions

`descriptors` A `tibble::tbl_df` containing the model descriptors

`set_descriptors` A `tibble::tbl_df` containing the set descriptors

`pub_descriptors` A `tibble::tbl_df` containing the publication descriptors

`citation` A `RefManageR::BibEntry` object containing the reference publication

`covariate_definitions` User-provided covariate definitions

`model_type` The model type, which is parsed from the `response_unit` name

`parameters` A named list of parameters and their values

`predict_fn_populated` The prediction function populated with the parameter values

`specification` A tibble::tbl_df of the model specification, which are the parameters and the descriptors together

Examples

```
FixedEffectsModel(
  response = list(
    hst = units::as_units("m")
  ),
  covariates = list(
    dsob = units::as_units("cm")
  ),
  parameters = list(
    beta_0 = 51.9954,
    beta_1 = -0.0208,
    beta_2 = 1.0182
  ),
  predict_fn = function(dsob) {
    1.37 + beta_0 * (1 - exp(beta_1 * dsob)^beta_2)
  }
)
```

FixedEffectsSet *Create a set of fixed effects models*

Description

A `FixedEffectsSet` represents a group of fixed-effects models that all have the same functional structure. Fitting a large family of models (e.g., for many different species) using the same functional structure is a common pattern in allometric studies, and `FixedEffectsSet` facilitates the installation of these groups of models by allowing the user to specify the parameter estimates and descriptions in a dataframe.

Usage

```
FixedEffectsSet(
  response,
  covariates,
  parameter_names,
  predict_fn,
  model_specifications,
  descriptors = list(),
  response_definition = NA_character_,
  covariate_definitions = list()
)
```

Arguments

response	A named list containing one element, with a name representing the response variable and a value representing the units of the response variable using the units::as_units function.
covariates	A named list containing the covariate specifications, with names representing the covariate name and the values representing the units of the coavariate using the units::as_units function
parameter_names	A character vector naming the columns in model_specifications that represent the parameters
predict_fn	A function that takes the covariate names as arguments and returns a prediction of the response variable. This function should be vectorized.
model_specifications	A dataframe such that each row of the dataframe provides model-level descriptors and parameter estimates for that model. Models must be uniquely identifiable using the descriptors. This is usually established using the load_parameter_frame() function.
descriptors	An optional named list of descriptors that describe the context of the allometric model
response_definition	A string containing an optional custom response definition, which is used instead of the description given by the variable naming system.
covariate_definitions	An optional named list of custom covariate definitions that will supersede the definitions given by the variable naming system. The names of the list must match the covariate names given in covariates.

Value

A set of fixed effects models

Slots

response_unit A one-element list with the name indicating the response variable and the value as the response variable units obtained using units::as_units()

`covariate_units` A list containing the covariate names as names and values as the values of the covariate units obtained using `units::as_units()`

`predict_fn` The prediction function, which takes covariates as arguments and returns model predictions

`descriptors` A `tibble::tbl_df` containing the model descriptors

`set_descriptors` A `tibble::tbl_df` containing the set descriptors

`pub_descriptors` A `tibble::tbl_df` containing the publication descriptors

`citation` A `RefManageR::BibEntry` object containing the reference publication

`covariate_definitions` User-provided covariate definitions

`model_type` The model type, which is parsed from the `response_unit` name

`parameter_names` A character vector indicating the parameter names

`model_specifications` A `tibble::tbl_df` of model specifications, where each row reprents one model identified with descriptors and containing the parameter estimates.

Examples

```
fixef_set <- FixedEffectsSet(
  response = list(
    vsia = units::as_units("ft^3")
  ),
  covariates = list(
    dsob = units::as_units("in")
  ),
  predict_fn = function(dsob) {
    a * dsob^2
  },
  parameter_names = "a",
  model_specifications = tibble::tibble(mod = c(1,2), a = c(1, 2))
)
```

`get_component_defs` *Load the component definitions*

Description

Loads the component definitions from a locally stored csv file

Usage

```
get_component_defs()
```

Value

A tibble::tbl_df containing the component definitions

Examples

```
get_component_defs()
```

get_measure_defs *Load the measure definitions*

Description

Loads the measure definitions from a locally stored csv file

Usage

```
get_measure_defs()
```

Value

A tibble::tbl_df containing the measure definitions

Examples

```
get_measure_defs()
```

get_params_path *Get the parameter search path*

Description

Get the parameter search path

Usage

```
get_params_path()
```

Value

A string containing the currently set parameter search path

get_variable_def *Get the definition of a variable in the variable naming system.*

Description

When possible, variables are given standard names using the variable naming system. The definitions for a variable can be found using this function. The `search_str` argument works using partial matching of the beginning of each variable name. For example input `"d"` will return all diameter definitions but input `"dsob"` will only return the definition for diameter outside bark at breast height.

Usage

```
get_variable_def(search_str, return_exact_only = FALSE)
```

Arguments

`search_str`	The string to search with.
`return_exact_only`	Some variables are completely defined but will return "addditional" matches. For example, "hst" refers to the total height of a tree, but "hstix" refers to a site index. If this argument is false, all strings starting with "hst" will be returned. If true, then only "hst" will be returned.

Value

A data.frame containing the matched variable definitions.

ingest_models *Ingest a set of models by running the publication files*

Description

Ingest a set of models by running the publication files

Usage

```
ingest_models(verbose, pub_path = NULL, params_path = NULL)
```

Arguments

`verbose`	If TRUE, print verbose messages
`pub_path`	A path to a directory containing publication files
`params_path`	A path to a directory containing parameter files

install_models *Install allometric models from the models repository*

Description

Allometric models are stored in a remote repository located on GitHub located here. The user must install these models themselves using this function. This function clones the models repository within the allometric package directory and constructs a local dataframe containing the models. Refer to `load_models()` for information about loading the models dataframe.

Usage

```
install_models(ingest = FALSE, redownload = TRUE, verbose = TRUE)
```

Arguments

`ingest`	If `TRUE`, model publication files are run locally, otherwise a previously prepared `.RDS` file is used as the models data.
`redownload`	If `TRUE`, models are re-downloaded from the remote repository.
`verbose`	If `TRUE`, print verbose messages as models are installed.

Value

No return value, installs models into the package directory.

load_models *Load a locally installed table of allometric models*

Description

This function loads all locally installed allometric models if they are downloaded and installed, if not run the `install_models` function. The result is of class `model_tbl`, which behaves very much like a `tibble::tbl_df` or a `data.frame`.

Usage

```
load_models()
```

Details

Printing the `head` of `allometric_models`, we can see the structure of the data

```
allometric_models <- load_models()
#> Warning in readRDS(rds_path): strings not representable in native encoding will be translated to UTF-
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF

#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
```

```
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT

#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT

#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT

#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT

#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT

#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Málaga' cannot be translated to UTF-8, is it valid in 'UTF
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
#> Warning in readRDS(rds_path): input string 'Cáceres' cannot be translated to UTF-8, is it valid in 'UT
#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-

#> Warning in readRDS(rds_path): input string 'Cádiz' cannot be translated to UTF-8, is it valid in 'UTF-
head(allometric_models)
#> # A tibble: 6 x 10
#>   id     model_type  country  region   taxa  pub_id     model     family_name covt_name pub_year
#>   <chr>   <chr>     <list>   <list>   <list> <chr>     <list>    <list>    <list>     <dbl>
#> 1 76ccc16a site index  <chr [1]> <chr [2]> <Taxa> barnes_1962  <FxdEffcM> <chr [1]>  <chr [2]>    1962
#> 2 cc2078aa site index  <chr [1]> <chr [2]> <Taxa> barrett_1978 <FxdEffcM> <chr [1]>  <chr [2]>    1978
#> 3 3955ab4f stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
#> 4 48b4aecf stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
```

```
#> 5 2fa084c2 stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
#> 6 7a585d5e stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
```

The columns are:

- `id` - A unique ID for the model.
- `model_type` - The type of model (e.g., stem volume, site index, etc.)
- `country` - The country or countries from which the model data is from.
- `region` - The region or regions (e.g., state, province, etc.) from which the model data is from.
- `taxa` - The taxonomic specification of the trees that are modeled.
- `model` - The model object itself.
- `pub_id` - A unique ID representing the publication.
- `family_name` - The names of the contributing authors.
- `covt_name` - The names of the covariates used in the model.
- `pub_year` - The publication year.

Models can be searched by their attributes. Note that some of the columns are `list` columns, which contain lists as their elements. Filtering on data in these columns requires the use of `purrr::map_lgl` which is used to determine truthiness of expressions for each element in a `list` column. While this may seem complicated, we believe the nested data structures are more descriptive and concise for storing the models, and users will quickly find that searching models in this way can be very powerful.

Value

A model_tbl containing the locally installed models.

Finding Contributing Authors

Using `purr::map_lgl` to filter the `family_name` column, we are able to find publications that contain specific authors of interst. For example, we may want models only authored by `"Hann"`. This is elementary to do in `allometric`:

```
hann_models <- dplyr::filter(
 allometric_models,
 purrr::map_lgl(family_name, ~ 'Hann' %in% .)
)

head(hann_models)
#> # A tibble: 6 x 10
#>  id     model_type country  region   taxa  pub_id   model    family_name covt_name pub_year
#>  <chr>   <chr>     <list>   <list>   <list> <chr>   <list>    <list>    <list>     <dbl>
#> 1 8970949f stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>    1978
#> 2 0d53539a stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>    1978
#> 3 0d109f2c stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>    1978
#> 4 86dcc7ff stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>    1978
#> 5 037a7989 stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>    1978
```

```
#> 6 02614f74 stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>   1978
nrow(hann_models)
#> [1] 87
```

Picking apart the above code block, we see that we are using the standard `dplyr::filter` function on the `allometric_models` dataframe. The second argument is a call using `purrr:map_lgl`, which will map over each list (contained as elements in the `family_names` column). The second argument to this function, `~ 'Hann' %in% .` is itself a function that checks if `'Hann'` is in the current list. Imagine we are marching down each row of `allometric_models`, `.` represents the element of `family_names` we are considering, which is itself a list of author names.

Finding First Authors

Maybe we are only interested in models where `'Hann'` is the first author. Using a simple modification we can easily do this.

```
hann_first_author_models <- dplyr::filter(
  allometric_models,
  purrr::map_lgl(family_name, ~ 'Hann' == .[[1]])
)

head(hann_first_author_models)
#> # A tibble: 6 x 10
#>  id     model_type country  region   taxa  pub_id   model    family_name covt_name pub_year
#>  <chr>   <chr>     <list>   <list>   <list> <chr>   <list>    <list>     <list>     <dbl>
#> 1 8970949f stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>   1978
#> 2 0d53539a stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>   1978
#> 3 0d109f2c stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>   1978
#> 4 86dcc7ff stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>   1978
#> 5 037a7989 stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>   1978
#> 6 02614f74 stem volume <chr [1]> <chr [2]> <Taxa> hann_1978 <FxdEffcM> <chr [2]>  <chr [2]>   1978
nrow(hann_first_author_models)
#> [1] 50
```

We can see that `'Hann'` is the first author for 50 models in this package.

Finding Models for a Given Species

One of the most common things people need is a model for a particular species. For this, we must interact with the `taxa` column. For example, to find models for the Pinus genus we can use

```
pinus_models <- dplyr::filter(
 allometric_models,
 purrr::map_lgl(taxa, ~ "Pinus" %in% .)
)

head(pinus_models)
#> # A tibble: 6 x 10
#>  id     model_type country  region   taxa  pub_id     model    family_name covt_name pub_year
```

```
#>  <chr>   <chr>     <list>   <list>   <list> <chr>      <list>    <list>    <list>     <dbl>
#> 1 cc2078aa site index  <chr [1]> <chr [2]> <Taxa> barrett_1978 <FxdEffcM> <chr [1]>  <chr [2]>    1978
#> 2 502152d1 stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
#> 3 3fb70119 stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
#> 4 925de182 stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
#> 5 910dddb1 stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
#> 6 5b3e21e7 stem height <chr [1]> <chr [3]> <Taxa> barrett_2006 <FxdEffcM> <chr [1]>  <chr [1]>    2006
nrow(pinus_models)
#> [1] 351
```

Users can also search with a specific taxon, which allows a full specification from family to species. For example, if we want models that apply to Ponderosa pine, first declare the necessary taxon, then use it to filter as before

```
ponderosa_taxon <- Taxon(
 family = "Pinaceae", genus = "Pinus", species = "ponderosa"
)

ponderosa_models <- dplyr::filter(
 allometric_models,
 purrr::map_lgl(taxa, ~ ponderosa_taxon %in% .)
)

nrow(ponderosa_models)
#> [1] 57
```

Finding a Model with Specific Data Requirements

We can even check for models that contain certain types of data requirements. For example, the following block finds diameter-height models, specifically models that use diameter outside bark at breast height as the *only* covariate. The utility here is obvious, since many inventories are vastly limited by their available tree measurements.

```
dia_ht_models <- dplyr::filter(
    allometric_models,
    model_type == 'stem height',
    purrr::map_lgl(covt_name, ~ length(.)==1 & .[[1]] == 'dsob'),
)

nrow(dia_ht_models)
#> [1] 282
```

Breaking this down, we have the first condition `model_type=='stem_height'` selecting only models concerned with stem heights as a response variable. The second line maps over each element of the `covt_name` column, which is a character vector. The `.` represents a given character vector for that row. First, we ensure that the vector is only one element in size using `length(.)==1`, then we ensure that the first (and only) element of this vector is equal to `'dsob'`, (diameter outside bark at breast height). In this case, 282 are available in the package.

Finding a Model for a Region

By now the user should be sensing a pattern. We can apply the exact same logic as the *Finding Contributing Authors* section to find all models developed using data from `US-OR`

```
us_or_models <- dplyr::filter(
    allometric_models,
    purrr::map_lgl(region, ~ "US-OR" %in% .),
)

nrow(us_or_models)
#> [1] 537
```

We can see that 537 allometric models are defined for the state of Oregon, US.

---

load_parameter_frame *Load a parameter frame from the models/parameters directory*

Description

This is a convenience that allows a user to easily load parameter files from the models/parameters directory. It is typically used when constructing the `model_specifications` argument for `ModelSet`.

Usage

```
load_parameter_frame(name)
```

Arguments

`name`	The name of the file, excluding the extension

Value

A tibble::tbl_df of the parameter data.

---

map_publications *Iteratively process publication files*

Description

This function allows a user to flexibly extract information as it loops over the publication files. Two main internal use-cases exist for this. First, it is used to install models as is done in `insall_models()` and, second, it is used to populate the remote MongoDB. Most users will not be interested in this function, but it is exposed for usage in the `allodata` package.

Usage

```
map_publications(verbose, func, pub_path = NULL, params_path = NULL)
```

Arguments

verbose	Whether or not to print verbose messages to console
func	The publication processing function. It should take a Publication object as its only argument.
pub_path	An optional path to a publication directory, by default the internally stored set of publications is used.
params_path	An optional path to a parameters directory, by default the internally stored set of parameter files is used.

---

merge.model_tbl *Merge a* model_tbl *with another data frame.*

Description

This merge function ensures that, when model_tbl is used in a merge that the resultant dataframe is still a model_tbl.

Usage

```
## S3 method for class 'model_tbl'
merge(x, y, ...)
```

Arguments

x	A data frame or model_tbl
y	A data frame or model_tbl
...	Additional arguments passed to merge

Value

A model_tbl merged with the inputs

`MixedEffectsModel` *Create a mixed effects model*

Description

`MixedEffectsModel` represents an allometric model that uses fixed and random effects.

Usage

```
MixedEffectsModel(
  response,
  covariates,
  predict_ranef,
  predict_fn,
  parameters,
  fixed_only = FALSE,
  descriptors = list(),
  response_definition = NA_character_,
  covariate_definitions = list()
)
```

Arguments

`response`	A named list containing one element, with a name representing the response variable and a value representing the units of the response variable using the `units::as_units` function.
`covariates`	A named list containing the covariate specifications, with names representing the covariate name and the values representing the units of the coavariate using the `units::as_units` function
`predict_ranef`	A function that predicts the random effects, takes any named covariates in `covariates` as arguments
`predict_fn`	A function that takes the covariate names as arguments and returns a prediction of the response variable. This function should be vectorized.
`parameters`	A named list of parameters and their values
`fixed_only`	A boolean value indicating if the model produces predictions using only fixed effects. This is useful when publications do not provide sufficient information to predict the random effects.
`descriptors`	An optional named list of descriptors that describe the context of the allometric model
`response_definition`	A string containing an optional custom response definition, which is used instead of the description given by the variable naming system.
`covariate_definitions`	An optional named list of custom covariate definitions that will supersede the definitions given by the variable naming system. The names of the list must match the covariate names given in `covariates`.

Value

An instance of MixedEffectsModel

Slots

`parameters` A named list of parameters and their values

`predict_fn_populated` The prediction function populated with the parameter values

`specification` A tibble::tbl_df of the model specification, which are the parameters and the descriptors together

`predict_ranef` The function that predicts the random effects

`predict_ranef_populated` The function that predicts the random effects populated with the fixed effect parameter estimates

`fixed_only` A boolean value indicating if the model produces predictions using only fixed effects

Examples

```
MixedEffectsModel(
  response = list(
    hst = units::as_units("m")
  ),
  covariates = list(
    dsob = units::as_units("cm")
  ),
  parameters = list(
    beta_0 = 40.4218,
    beta_1 = -0.0276,
    beta_2 = 0.936
  ),
  predict_ranef = function() {
    list(b_0_i = 0, b_2_i = 0)
  },
  predict_fn = function(dsob) {
    1.37 + (beta_0 + b_0_i) * (1 - exp(beta_1 * dsob)^(beta_2 + b_2_i))
  },
  fixed_only = TRUE
)
```

`MixedEffectsSet` *Create a set of mixed effects models*

Description

A `MixedEffectsSet` represents a group of mixed-effects models that all have the same functional structure. Fitting a large family of models (e.g., for many different species) using the same functional structure is a common pattern in allometric studies, and `MixedEffectsSet` facilitates the installation of these groups of models by allowing the user to specify the parameter estimates and descriptions in a dataframe or spreadsheet.

Usage

```
MixedEffectsSet(
  response,
  covariates,
  parameter_names,
  predict_fn,
  model_specifications,
  predict_ranef,
  fixed_only = FALSE,
  descriptors = list(),
  response_definition = NA_character_,
  covariate_definitions = list()
)
```

Arguments

`response`	A named list containing one element, with a name representing the response variable and a value representing the units of the response variable using the `units::as_units` function.
`covariates`	A named list containing the covariate specifications, with names representing the covariate name and the values representing the units of the coavariate using the `units::as_units` function
`parameter_names`	A character vector naming the columns in `model_specifications` that represent the parameters
`predict_fn`	A function that takes the covariate names as arguments and returns a prediction of the response variable. This function should be vectorized.
`model_specifications`	A dataframe such that each row of the dataframe provides model-level descriptors and parameter estimates for that model. Models must be uniquely identifiable using the descriptors. This is usually established using the `load_parameter_frame()` function.
`predict_ranef`	A function that predicts the random effects, takes any named covariates in `covariates` as arguments
`fixed_only`	A boolean value indicating if the model produces predictions using only fixed effects. This is useful when publications do not provide sufficient information to predict the random effects.
`descriptors`	An optional named list of descriptors that describe the context of the allometric model
`response_definition`	A string containing an optional custom response definition, which is used instead of the description given by the variable naming system.
`covariate_definitions`	An optional named list of custom covariate definitions that will supersede the definitions given by the variable naming system. The names of the list must match the covariate names given in `covariates`.

Details

Because mixed-effects models already accommodate a grouping structure, `MixedEffectsSet` tends to be a much rarer occurrence than `FixedEffectsSet` and `MixedEffectsModel`.

Value

An instance of MixedEffectsSet

Slots

`parameters` A named list of parameters and their values

`predict_fn_populated` The prediction function populated with the parameter values

`specification` A tibble::tbl_df of the model specification, which are the parameters and the descriptors together

`predict_ranef` The function that predicts the random effects

`predict_ranef_populated` The function that predicts the random effects populated with the fixed effect parameter estimates

`fixed_only` A boolean value indicating if the model produces predictions using only fixed effects

`model_specifications` A `tibble::tbl_df` of model specifications, where each row reprents one model identified with descriptors and containing the parameter estimates.

Examples

```
mixed_effects_set <- MixedEffectsSet(
  response = list(
    vsia = units::as_units("ft^3")
  ),
  covariates = list(
    dsob = units::as_units("in")
  ),
  parameter_names = "a",
  predict_ranef = function(dsob, hst) {
    list(a_i = 1)
  },
  predict_fn = function(dsob) {
    (a + a_i) * dsob^2
  },
  model_specifications = tibble::tibble(a = c(1, 2))
)
```

`model_call` *Get the function call for a model*

Description

The function call is the allometric model expressed as a function of its covariates. Accessing the function call is important when determining the order of the covariates given to the prediction function.

Usage

```
model_call(object)
```

Arguments

object	The allometric model or set for which a function call will be retrieved

Value

A string of the function call

Examples

```
model_call(brackett_rubra)
```

---

predict *Predict with an allometric model*

Description

Predict with an allometric model

Usage

```
predict(model, ...)

## S4 method for signature 'FixedEffectsModel'
predict(model, ..., output_units = NULL)

## S4 method for signature 'MixedEffectsModel'
predict(model, ..., newdata = NULL, output_units = NULL)
```

Arguments

model	The allometric model used for prediction
...	Additional arguments passed to the predict_fn of the input model
output_units	Optionally specify the output units of the model as a string, e.g., "ft^3". The provided string must be compatible with the units::set_units() function.
newdata	A dataframe containing columns that match the names of the arguments given to predict_ranef. The values of this data represents information from a new group of observations for which predictions are desired (e.g., a new stand or plot).

Value

A vector of allometric model predictions

Examples

```
predict(brackett_rubra, 10, 50)
predict(brackett_rubra, 10, 50, output_units = "m^3")
```

`predict_allo` *Predict allometric attributes using a column of allometric models*

Description

A frequent pattern in forest inventory anaylsis is the need to produce predictions of models with the same functional form, but using different models. `predict_allo` enables this by allowing the user to pass a list-column of models as an argument, along with the associated covariates. This pattern plays well with `dplyr` functions such as `dplyr::mutate()`.

Usage

```
predict_allo(model_list, ...)
```

Arguments

`model_list`	A list-column of models
`...`	Additional arguments passed to each model's `predict_fn`

Value

A vector of predictions

Examples

```
tree_data <- tibble::tibble(
 dbh = c(10, 20), ht = c(50, 75), model = c(list(brackett_rubra), list(brackett_acer))
)

tree_data %>%
  dplyr::mutate(vol = predict_allo(model, dbh, ht))
```

`Publication` *Create a publication that contains allometric models*

Description

`Publication` represents a technical or scientific document that contains allometric models. Initially, publications do not contain models, and models are added using the `add_model` or `add_set` methods.

Usage

```
Publication(citation, descriptors = list())
```

Arguments

`citation`	The citation of the paper declared using the `RefManageR::BibEntry` class
`descriptors`	A named list of descriptors that are defined for all models contained in the publication.

Value

An instance of class Publication

Slots

`citation` A `RefManageR::BibEntry` of the reference publication

`response_sets` A list containing the model sets indexed by the response variable names

`descriptors` A named list containing descriptors that are defined for all models in the publication.

Examples

```
pub <- Publication(
  citation = RefManageR::BibEntry(
    key = "test_2000",
    bibtype = "article",
    author = "test",
    title = "test",
    journal = "test",
    year = 2000,
    volume = 0
  ),
  descriptors = list(
    region = "US-WA"
  )
)
```

select_model *Select an allometric model*

Description

This is a generic function used to select allometric models out of larger collections, like `model_tbl`.

Usage

```
select_model(model_tbl, id)
```

Arguments

`model_tbl`	A `model_tbl` object
`id`	The model id or index

Value

An allometric model object

set_params_path *Set the parameter search path*

Description

Set the parameter search path

Usage

```
set_params_path(params_path)
```

Arguments

`params_path`	The file path containing parameter files

Taxa *Group taxons together*

Description

Taxa represents a set of taxons. See Taxon(). These are typically used to specify species and other taxonomic groups that belong to a model.

Usage

```
Taxa(...)
```

Arguments

...	A set of Taxon objects.

Value

An instance of class Taxa

Examples

```
Taxa(
   Taxon(
      family = "Pinaceae",
      genus = "Pinus",
      species = "ponderosa"
   ),
   Taxon(
     family = "Betulaceae"
   )
)
```

Taxon *Create a taxonomic hierarchy*

Description

Taxon represents a taxonomic hierarchy (from family through species). This class represents a number of validity checks to ensure the taxon is correctly structured. A taxon must have at least a family specified, and neither genus nor species can be specified without the "shallower" layers of the hierarchy specified first. Group Taxons together with Taxa().

Usage

```
Taxon(family = NA_character_, genus = NA_character_, species = NA_character_)
```

Arguments

family	The taxonomic family
genus	The taxonomic genus
species	The taxonomic species

Value

An instance of class Taxon

Examples

```
Taxon(
  family = "Pinaceae",
  genus = "Pinus",
  species = "ponderosa"
)

Taxon(
  family = "Betulaceae"
)
```

---

toJSON *Convert a model or publication to a JSON representation*

Description

This function converts an allometric model or publication into a JSON representation. Primarily, this is used internally to populate a remotely hosted MongoDatabase.

Usage

```
toJSON(object, ...)
```

Arguments

object	An allometric model or publication
...	Additional arguments passed to jsonlite::toJSON

Value

A string containing the JSON representation of the object

Examples

```
toJSON(brackett_rubra)
```

toJSON,FixedEffectsModel-method

Convert a fixed effects model to a JSON representation

Description

This function converts a fixed effects model into a JSON representation. Primarily, this is used internally to populate a remotely hosted MongoDatabase.

Usage

```
## S4 method for signature 'FixedEffectsModel'
toJSON(object, ...)
```

Arguments

object	A fixed effects model
...	Additional arguments passed to jsonlite::toJSON

Value

A string containing the JSON representation of the object

Examples

```
toJSON(brackett_rubra, pretty = TRUE)
```

unnest_models

Unnest columns of a dataframe

Description

Unnest columns of a dataframe

Usage

```
unnest_models(data, cols)
```

Arguments

data	A dataframe
cols	A character vector indicating the columns to unnest

Value

The unnested model_tbl

unnest_taxa *Unnest the taxa column of a* model_tbl

Description

In some cases it is convenient to expand the taxonomic specifications for each model contained in the taxa column. This function achieves this, and adds family, genus, and species character columns. Models with more than one taxon are replicated as new rows.

Usage

```
unnest_taxa(data)
```

Arguments

data	A model_tbl

Value

A model_tbl with family, genus and species columns attached

unnest_taxa.model_tbl *Unnest the taxa column of a* model_tbl

Description

In some cases it is convenient to expand the taxonomic specifications for each model contained in the taxa column. This function achieves this, and adds family, genus, and species character columns. Models with more than one taxon are replicated as new rows.

Usage

```
## S3 method for class 'model_tbl'
unnest_taxa(data)
```

Arguments

data	A model_tbl

Value

A model_tbl with family, genus and species columns attached

`%in%,Taxon,character-method`
Check if a Taxon contains a character

Description

Check if a Taxon contains a character

Usage

```
## S4 method for signature 'Taxon,character'
x %in% table
```

Arguments

`x`	A Taxon object
`table`	A character vector

Value

TRUE or FALSE indicating if any of the Taxa fields appear in the character.

Index