Package ‘multitool’

February 8, 2024

Title Run Multiverse Style Analyses
Version 0.1.4
Description Run the same analysis over a range of arbitrary data processing
decisions. ‘multitool’ provides an interface for creating alternative
analysis pipelines and turning them into a grid of all possible pipelines.
Using this grid as a blueprint, you can model your data across all possible
pipelines and summarize the results.
License MIT + file LICENSE
Imports clipr, correlation, DiagrammeR, dplyr, flextable, furrr,
future, ggdist, glue, ggplot2, moments, purrr, rlang, stringr,
tibble, tidyr, lme4, parameters, performance
Suggests knitr, rmarkdown, testthat (>= 3.0.0), tidyverse
Config/testthat/edition 3
Encoding UTF-8
RoxygenNote 7.3.1
VignetteBuilder knitr
URL https://ethan-young.github.io/multitool/
    https://github.com/ethan-young/multitool
BugReports https://github.com/ethan-young/multitool/issues
Depends R (>= 4.1.0)
NeedsCompilation no
Author Ethan Young [aut, cre, cph] (<https://orcid.org/0000-0002-8232-0184>),
    Stefan Vermeent [aut] (<https://orcid.org/0000-0002-9595-5373>)
Maintainer Ethan Young <young.ethan.scott@gmail.com>
Repository CRAN
Date/Publication 2024-02-08 17:40:02 UTC
R topics documented:

add_correlations .............................................................. 2
add_filters .................................................................. 4
add_model .................................................................... 5
add_parameter_keys ............................................................... 6
add_postprocess ................................................................. 7
add_preprocess ................................................................. 9
add_reliabilities ................................................................. 10
add_summary_stats .............................................................. 11
add_variables .................................................................. 12
condense ...................................................................... 14
create_blueprint_graph ....................................................... 15
detect_multiverse_n ............................................................. 16
detect_n_filters ................................................................. 17
detect_n_models ................................................................. 18
detect_n_variables ............................................................... 19
expand_decisions ................................................................. 20
reveal ........................................................................... 22
reveal_corrs .................................................................. 23
reveal_model_messages ......................................................... 24
reveal_model_parameters ....................................................... 26
reveal_model_performance ..................................................... 27
reveal_model_warnings ......................................................... 29
reveal_reliabilities ............................................................. 30
reveal_summary_stats ........................................................... 31
run_descriptives ................................................................. 33
run_multiverse ................................................................. 34
show_code_filter ............................................................... 36
summarize_filter_ns ............................................................ 37

Index ........................................................................... 38

add_correlations  Add correlations from the correlation package in easystats

Description

Add correlations from the correlation package in easystats

Usage

add_correlations(
  .df,
  var_set,
  variables,
  focus_set = NULL,
  method = "auto",
)
add_correlations

redundant = TRUE,
add_matrix = TRUE
)

Arguments

\texttt{.df} \quad \text{the original data.frame(e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.}

\texttt{var_set} \quad \text{character string. Should be a descriptive name of the correlation matrix.}

\texttt{variables} \quad \text{the variables for which you would like to correlations. These variables will be passed to \texttt{link[correlation]{correlation}}. You can also use tidyselect to select variables.}

\texttt{focus_set} \quad \text{variables to focus one in a table. This produces a table where rows are each focused variables and columns are all other variables}

\texttt{method} \quad \text{a valid method of correlation supplied to \texttt{link[correlation]{correlation}}} (e.g., 'pearson' or 'kendall'). Defaults to 'auto'. See \texttt{link[correlation]{correlation}} for more details.

\texttt{redundant} \quad \text{logical, should the result include repeated correlations? Defaults to TRUE See \texttt{link[correlation]{correlation}} for details.}

\texttt{add_matrix} \quad \text{logical, add a traditional correlation matrix to the output. Defaults to TRUE.}

Value

a data.frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

Examples

library(tidyverse)
library(multitool)

the_data <-
data.frame(
  id   = 1:500,
  iv1  = rnorm(500),
  iv2  = rnorm(500),
  iv3  = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1  = rnorm(500),
  dv2  = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
add_filters

Add filtering/exclusion criteria to a multiverse pipeline

Description

Add filtering/exclusion criteria to a multiverse pipeline

Usage

add_filters(.df, ...)

Arguments

- .df  The original data.frame (e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.
- ...  logical expressions to be used with filter separated by commas. Expressions should not be quoted.

Value

a data.frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

Examples

library(tidyverse)
nlibrary(multitool)

# Simulate some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod1 = rnorm(500),
mod2 = rnorm(500),
mod3 = rnorm(500),
da filters
```r
cov1 = rnorm(500),
cov2 = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

the_data |>
  add_filters(include1 == 0, include2 != 3, include2 != 2, include3 > -2.5)
```

---

**add_model**

Add a model and formula to a multiverse pipeline

**Description**

Add a model and formula to a multiverse pipeline

**Usage**

```r
add_model(.df, model_desc, code)
```

**Arguments**

- `.df` The original data.frame (e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.
- `model_desc` a human readable name you would like to give the model.
- `code` literal model syntax you would like to run. You can use `glue` inside formulas to dynamically generate variable names based on a variable grid. For example, if you make variable grid with two versions of your IVs (e.g., `iv1` and `iv2`), you can write your formula like so: `lm(happiness ~ {iv} + control_var)`. The only requirement is that the variables written in the formula actually exist in the underlying data. You are also responsible for loading any packages that run a particular model (e.g., `lme4` for mixed-models)

**Value**

a data.frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

**Examples**

```r
library(tidyverse)
library(multitool)
```
add_parameter_keys

Add parameter keys names for later use in summarizing model effects

Description

Add parameter keys names for later use in summarizing model effects

Usage

add_parameter_keys(.df, parameter_group, parameter_name)

Arguments

*.df* The original data.frame(e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.

*parameter_group* character, a name for the parameter of interest

*parameter_name* quoted or unquoted names of variables involved in a particular parameter of interest. Usually this is just a variable in your model (e.g., a main effect of your iv). However, it could also be an interaction term or some other term. You can use glue syntax to specify an effect that might use alternative versions of the same variable.
Value

A data frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

Examples

```r
library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
)

the_data |>
  add_variables("ivs", iv1, iv2, iv3) |>
  add_variables("dvs", dv1, dv2) |>
  add_variables("mods", starts_with("mod")) |>
  add_model("linear model", lm((dvs) ~ (ivs) * (mods))) |>
  add_parameter_keys("my_interaction", "(ivs):(mods)") |>
  add_parameter_keys("my_main_effect", (ivs))
```

Description

Add arbitrary postprocessing code to a multiverse pipeline

Usage

```
add_postprocess(.df, postprocess_name, code)
```

Arguments

- **.df**
  The original data.frame (e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.

- **postprocess_name**
  a character string. A descriptive name for what the postprocessing step accomplishes.

- **code**
  the literal code you would like to execute after each analysis.
  The code should be written to work with pipes (i.e., |> or %>%). Because the post-processing code comes last in each multiverse analysis step, the chosen model object will be passed to the post-processing code.
  For example, if you fit a simple linear model like: `lm(y ~ x1 + x2)`, and your post-processing code executes a call to `anova`, you would simply pass `anova()` to `add_postprocess()`. The underlying code would be:
  data |> filters |> lm(y ~ x1 + x2, data = _) |> anova()

Value

a data.frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

Examples

```r
library(tidyverse)
library(multitool)

the_data <- data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
)

the_data |
add_filters(include1 == 0, include2 != 3, include2 != 2, include3 > -2.5) |
add_variables("ivs", iv1, iv2, iv3) |
add_variables("dvs", dv1, dv2) |
add_variables("mods", starts_with("mod")) |
add_preprocess("scale_iv", 'mutate(ivs) = scale(ivs)') |>
add_model("linear model", lm(dvs ~ ivs * mods)) |>
add_postprocess("analysis of variance", aov())

---

### Description

Add arbitrary preprocessing code to a multiverse analysis pipeline

### Usage

```r
add_preprocess(.df, process_name, code)
```

### Arguments

- **.df**: The original `data.frame` (e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.
- **process_name**: A character string. A descriptive name for what the preprocessing step accomplishes.
- **code**: The literal code you would like to execute after data are filtered. `glue` syntax is allowed. An example might be centering or scaling a predictor after the appropriate filters are applied to the data.

The code should be written to work with pipes (i.e., `|>` or `%>%`). Pre-processing code will eventually take the base data along with any filters applied to the data. This means `mutate` calls are the most natural but other functions that take a `data.frame` as the first argument should work as well (as long as they also return a `data.frame`).

### Value

A `data.frame` with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

### Examples

```r
library(tidyverse)
library(multitool)

the_data <- data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
)
mod1 = rnorm(500),
mod2 = rnorm(500),
mod3 = rnorm(500),
cov1 = rnorm(500),
cov2 = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

the_data |> 
  add_filters(include1 == 0, include2 != 3, include2 != 2, include3 > -2.5) |> 
  add_variables("ivs", iv1, iv2, iv3) |> 
  add_variables("dvs", dv1, dv2) |> 
  add_variables("mods", starts_with("mod")) |> 
  add_preprocess("scale_iv", 'mutate(ivs) = scale(ivs))'

---

**add_reliabilities**

Add item reliabilities to a multiverse pipeline

**Description**

Add item reliabilities to a multiverse pipeline

**Usage**

```r
add_reliabilities(.df, scale_name, items)
```

**Arguments**

- `.df` the original data.frame(e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.
- `scale_name` a character string. Indicates the name of the scale or measure measured by the items or indicators in `items`.
- `items` the items (variables) that comprise a scale or measure. These variables will be passed to `link[performance]{cronbachs_alpha}, link[performance]{item_intercor}, and link[performance]{item_reliability}. You can also use tidyselect to select variables.

**Value**

a data.frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.
add_summary_stats

Examples

library(tidyverse)
library(multitool)

the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
)

the_data |>  
  add_filters(include1 == 0,include2 != 3,include2 != 2, include3 > -2.5) |>  
  add_variables("ivs", iv1, iv2, iv3) |>  
  add_variables("dvs", dv1, dv2) |>  
  add_variables("mods", starts_with("mod")) |>  
  add_reliabilities("unp_scale", c(iv1, iv2, iv3))

add_summary_stats

Add a set of descriptive statistics to compute over a set of variables

Description

Add a set of descriptive statistics to compute over a set of variables

Usage

add_summary_stats(.df, var_set, variables, stats)

Arguments

.df The original data.frame(e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.
var_set a character string. A name for the set of summary statistics
variables the variables for which you would like to compute summary statistics. You can also use tidyselect to select variables.
stats

- a character vector of stat names (e.g., `c("mean","sd")`). You are responsible for loading any packages that compute your preferred summary statistics. Summary statistic functions must work inside `summarize`.

**Value**

- a data.frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

**Examples**

```r
library(tidyverse)
library(multitool)

the_data <-
  data.frame(
    id = 1:500,
    iv1 = rnorm(500),
    iv2 = rnorm(500),
    iv3 = rnorm(500),
    mod1 = rnorm(500),
    mod2 = rnorm(500),
    mod3 = rnorm(500),
    cov1 = rnorm(500),
    cov2 = rnorm(500),
    dv1 = rnorm(500),
    dv2 = rnorm(500),
    include1 = rbinom(500, size = 1, prob = .1),
    include2 = sample(1:3, size = 500, replace = TRUE),
    include3 = rnorm(500)
  )

the_data |>  
  add_filters(include1 == 0,include2 != 3,include2 != 2, include3 > -2.5) |>  
  add_variables("ivs", iv1, iv2, iv3) |>  
  add_variables("dvs", dv1, dv2) |>  
  add_variables("mods", starts_with("mod")) |>  
  add_preprocess(process_name = "scale_iv", "mutate(ivs) = scale(ivs)") |>  
  add_preprocess(process_name = "scale_mod", mutate(mods) := scale(mods)) |>  
  add_summary_stats("iv_stats", starts_with("iv"), c("mean", "sd")) |>  
  add_summary_stats("dv_stats", starts_with("dv"), c("skewness", "kurtosis"))  
```

---

**Description**

Add a set of variable alternatives to a multiverse pipeline

Add a set of variable alternatives to a multiverse pipeline.
add_variables

Usage

add_variables(.df, var_group, ...)

Arguments

.df The original data.frame(e.g., base data set). If part of set of add_* decision functions in a pipeline, the base data will be passed along as an attribute.

var_group a character string. Indicates the name of the current set. For example, "primary_iv" could indicate this set are alternatives of the main predictor in an analysis.

... the bare unquoted names of the variables to include as alternative options for this variable set. You can also use tidyselect to select variables.

Value

a data.frame with three columns: type, group, and code. Type indicates the decision type, group is a decision, and the code is the actual code that will be executed. If part of a pipe, the current set of decisions will be appended as new rows.

Examples

library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
)

the_data |> add_variables("ivs", iv1, iv2, iv3) |> add_variables("dvs", dv1, dv2) |> add_variables("mods", starts_with("mod"))
condense

Summarize multiverse parameters

Description

Summarize multiverse parameters

Usage

condense(.unpacked, .what, .how, .group = NULL, list_cols = TRUE)

Arguments

.unpacked an unpacked (with reveal or unnest) multiverse dataset.

.what a specific column to summarize. This could be a model estimate, a summary statistic, correlation, or any other estimate computed over the multiverse.

.how a named list. The list should contain summary functions (e.g., mean or median) the user would like to compute over the individual estimates from the multiverse

.group an optional variable to group the results. This argument is passed directly to the .by argument used in dplyr::across

.list_cols logical, whether to create list columns for the raw values of any summarized columns. Useful for creating visualizations and tables. Default is TRUE.

Value

a summarized tibble containing a column for each summary method from .how

Examples

library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
)

condense
### Description

Create a Analysis Pipeline diagram

### Usage

```r
create_blueprint_graph(
  .pipeline,
  splines = "line",
  render = TRUE,
  show_code = FALSE,
  ...
)
```

### Arguments

- `.pipeline` a data frame produced by calling a series of add_* functions.
- `splines` options for how to draw edges (lines) for a grViz diagram
- `render` whether to render the graph or just output grViz code
- `show_code` whether to show the code that generated the diagram
- `...` additional options passed to DiagrammeR::grViz()
Value

grViz graph of your pipeline

Examples

library(tidyverse)
library(multitool)

# create some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# create a pipeline blueprint
full_pipeline <-
the_data |> add_filters(include1 == 0, include2 != 3, include3 > -2.5) |> add_variables(var_group = "ivs", iv1, iv2, iv3) |> add_variables(var_group = "dvs", dv1, dv2) |> add_model("linear model", lm(dv2 ~ ivs * mod))

create_blueprint_graph(full_pipeline)

detect_multiverse_n

Detect total number of analysis pipelines

Description

Detect total number of analysis pipelines

Usage

detect_multiverse_n(.pipeline, include_models = TRUE)

Arguments

.pipeline a data.frame produced by calling a series of add_* functions.
include_models Whether to count alternative models if you have more than one add_model() call.
**Value**

a numeric, the total number of unique analysis pipelines

**Examples**

```r
library(tidyverse)
library(multitool)

# create some data
the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
)

# create a pipeline blueprint
full_pipeline <-
  the_data |> 
  add_filters(include1 == 0, include2 != 3, include3 > -2.5) |> 
  add_variables(var_group = "ivs", iv1, iv2, iv3) |> 
  add_variables(var_group = "dvs", dv1, dv2) |> 
  add_model("linear model", lm({dvs} ~ {ivs} * mod))

detect_multiverse_n(full_pipeline)
```

---

**detect_n_filters**

*Detect total number of filtering expressions your pipelines*

**Description**

Detect total number of filtering expressions your pipelines

**Usage**

```r
detect_n_filters(.pipeline)
```

**Arguments**

- `.pipeline` a data.frame produced by calling a series of `add_*` functions.
detect_n_models

Value

a numeric, the total number of filtering expressions

Examples

```r
library(tidyverse)
library(multitool)

# create some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# create a pipeline blueprint
full_pipeline <-
the_data |>
add_filters(include1 == 0, include2 != 3, include3 > -2.5) |>
add_variables(var_group = "ivs", iv1, iv2, iv3) |>
add_variables(var_group = "dvs", dv1, dv2) |>
add_model("linear model", lm({dvs} ~ {ivs} * mod))

detect_n_filters(full_pipeline)
```

---

detect_n_models  Detect total number of models in your pipelines

Description

Detect total number of models in your pipelines

Usage

detect_n_models(.pipeline)

Arguments

.pipeline  a data.frame produced by calling a series of add_* functions.
detect_n_variables

Value

a numeric, the total number of unique models

Examples

library(tidyverse)
l library(multitool)

# create some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# create a pipeline blueprint
full_pipeline <-
the_data |> 
add_filters(include1 == 0, include2 != 3, include3 > -2.5) |> 
add_variables(var_group = "ivs", iv1, iv2, iv3) |> 
add_variables(var_group = "dvs", dv1, dv2) |> 
add_model("linear model", lm({dvs} ~ {ivs} * mod))

detect_n_models(full_pipeline)

detect_n_variables  
Detect total number of variable sets in your pipelines

Description

Detect total number of variable sets in your pipelines

Usage

detect_n_variables(.pipeline)

Arguments

.pipeline a data.frame produced by calling a series of add_* functions.
Value

a numeric, the total number of unique variable sets

Examples

library(tidyverse)
library(multitool)

# create some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# create a pipeline blueprint
full_pipeline <-
the_data |> 
add_filters(include1 == 0, include2 != 3, include3 > -2.5) |> 
add_variables(var_group = "ivs", iv1, iv2, iv3) |> 
add_variables(var_group = "dvs", dv1, dv2) |> 
add_model("linear model", lm(dv1 ~ ivs * mod))

detect_n_variables(full_pipeline)

---

**expand_decisions**  
*Expand a set of multiverse decisions into all possible combinations*

Description

Expand a set of multiverse decisions into all possible combinations

Usage

expand_decisions(.pipeline)

Arguments

.pipeline  
a data.frame produced by calling a series of add_* functions.
**Value**

a nested data.frame containing all combinations of arbitrary decisions for a multiverse analysis. Decision types will become list columns matching the type of decisions called along the pipeline (e.g., filters, variables, etc.). Any decisions containing glue syntax will be populated with the relevant information.

**Examples**

```r
library(tidyverse)
library(multitool)

the_data <-
data.frame(
    id = 1:500,
    iv1 = rnorm(500),
    iv2 = rnorm(500),
    iv3 = rnorm(500),
    mod1 = rnorm(500),
    mod2 = rnorm(500),
    mod3 = rnorm(500),
    cov1 = rnorm(500),
    cov2 = rnorm(500),
    dv1 = rnorm(500),
    dv2 = rnorm(500),
    include1 = rbinom(500, size = 1, prob = .1),
    include2 = sample(1:3, size = 500, replace = TRUE),
    include3 = rnorm(500)
  )

full_pipeline <-
  the_data |> 
  add_filters(include1 == 0, include2 != 3, include2 != 2, include3 > -2.5) |> 
  add_variables("ivs", iv1, iv2, iv3) |> 
  add_variables("dvs", dv1, dv2) |> 
  add_variables("mods", starts_with("mod")) |> 
  add_preprocess(process_name = "scale_iv", mutate(ivs = scale(ivs))) |> 
  add_preprocess(process_name = "scale_mod", mutate(mods := scale(mods))) |> 
  add_summary_stats("iv_stats", starts_with("iv"), c("mean", "sd")) |> 
  add_summary_stats("dv_stats", starts_with("dv"), c("skewness", "kurtosis")) |> 
  add_correlations("predictors", matches("iv|mod|cov"), focus_set = c(cov1, cov2)) |> 
  add_correlations("outcomes", matches("dv|mod"), focus_set = matches("dv")) |> 
  add_reliabilities("unp_scale", c(iv1, iv2, iv3)) |> 
  add_model("no covariates", lm(dvs ~ ivs * mods)) |> 
  add_model("with covariates", lm(dvs ~ ivs * mods + cov1)) |> 
  add_postprocess("aov", aov())

pipeline_expanded <- expand_decisions(full_pipeline)
```
reveal

Reveal the contents of a multiverse analysis

Description

Reveal the contents of a multiverse analysis

Usage

reveal(.multi, .what, .which = NULL, .unpack_specs = "no")

Arguments

.mliti a multiverse list-column tibble produced by run_multiverse.
.what the name of a list-column you would like to unpack
.which any sub-list columns you would like to unpack
.unpack_specs character, options are "no", "wide", or "long". "no" (default) keeps specifications in a list column, wide unnests specifications with each specification category as a column. "long" unnests specifications and stacks them into long format, which stacks specifications into a decision_set and alternatives columns. This is mainly useful for plotting.

Value

the unnested part of the multiverse requested. This usually contains the particular estimates or statistics you would like to analyze over the decision grid specified.

Examples

library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
)
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# Decision pipeline
full_pipeline <-
the_data |> 
  add_filters(include1 == 0, include2 != 3, include2 != 2, scale(include3) > -2.5) |> 
  add_variables("ivs", iv1, iv2, iv3) |> 
  add_variables("dvs", dv1, dv2) |> 
  add_variables("mods", starts_with("mod")) |> 
  add_model("linear_model", lm(dv1 ~ ivs * mods + cov1))

pipeline_grid <- expand_decisions(full_pipeline)

# Run the whole multiverse
the_multiverse <- run_multiverse(pipeline_grid[1:10,])

# Reveal results of the linear model
the_multiverse |> reveal(model_fitted, model_parameters)

reveal_corrs - Reveal a set of multiverse correlations

---

**Description**

Reveal a set of multiverse correlations

**Usage**

reveal_corrs(.descriptives, .which, .unpack_specs = "no")

**Arguments**

- `.descriptives` - a descriptive multiverse list-column tibble produced by `run_descriptives`.
- `.which` - the specific name of the correlations requested.
- `.unpack_specs` - character, options are "no", "wide", or "long". "no" (default) keeps specifications in a list column, wide unsets specifications with each specification category as a column. "long" unsets specifications and stacks them into long format, which stacks specifications into a decision_set and alternatives columns. This is mainly useful for plotting.

**Value**

- an unpaused set of correlations per decision from the multiverse.
Examples

```r
library(tidyverse)
library(multitool)

# create some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# create a pipeline blueprint
full_pipeline <-
the_data |> 
  add_filters(
    include1 == 0,
    include2 != 3,
    include2 != 2,
    include3 > -2.5,
    include3 < 2.5,
    between(include3, -2.5, 2.5)
  ) |> 
  add_variables(var_group = "ivs", iv1, iv2, iv3) |> 
  add_variables(var_group = "dvs", dv1, dv2) |> 
  add_correlations("predictors", starts_with("iv")) |> 
  add_summary_stats("iv_stats", starts_with("iv"), c("mean", "sd")) |> 
  add_reliabilities("vio_scale", starts_with("iv")) |> 
  add_model("linear model", lm({dvs} ~ {ivs} * mod))

my_descriptives <- run_descriptives(full_pipeline)

my_descriptives |> 
  reveal_corrs(predictors_rs)
```

---

**reveal_model_messages**  
Reveal any messages about your models during a multiverse analysis

**Description**

Reveal any messages about your models during a multiverse analysis
reveal_model_messages

Usage

reveal_model_messages(.multi, .unpackspecs = "no")

Arguments

.multi a multiverse list-column tibble produced by run_multiverse.
.unpackspecs character, options are "no", "wide", or "long". "no" (default) keeps specifications in a list column, wide unnests specifications with each specification category as a column. "long" unnests specifications and stacks them into long format, which stacks specifications into a decision_set and alternatives columns. This is mainly useful for plotting.

Value

the unnested model messages captured during analysis.

Examples

library(tidyverse)
library(multitool)

# Simulate some data
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod1 = rnorm(500),
mod2 = rnorm(500),
mod3 = rnorm(500),
cov1 = rnorm(500),
cov2 = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbetabinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# Decision pipeline
data.frame(~
add_filters(include1 == 0, include2 != 3, include2 != 2, scale(include3) > -2.5) |
add_variables("ivs", iv1, iv2, iv3) |
add_variables("dvs", dv1, dv2) |
add_variables("mods", starts_with("mod"))) |
add_model("linear_model", lm([dvs] ~ [ivs] * [mods] + cov1))

pipeline_grid <- expand_decisions(full_pipeline)
# Run the whole multiverse
the_multiverse <- run_multiverse(pipeline_grid[1:10,])

# Reveal results of the linear model
the_multiverse |> reveal_model_messages()

---

reveal_model_parameters

_Reveal the model parameters of a multiverse analysis_

**Description**

Reveal the model parameters of a multiverse analysis

**Usage**

```r
reveal_model_parameters(.multi, parameter_key = NULL, .unpack_specs = "no")
```

**Arguments**

- `.multi` a multiverse list-column tibble produced by `run_multiverse`.
- `parameter_key` character, if you added parameter keys to your pipeline, you can specify if you would like filter the parameters using one of your parameter keys. This is useful when different variables are being switched out across the multiverse but represent the same effect of interest.
- `.unpack_specs` character, options are "no", "wide", or "long". "no" (default) keeps specifications in a list column, `wide` unests specifications with each specification category as a column. "long" unests specifications and stacks them into long format, which stacks specifications into a `decision_set` and `alternatives` columns. This is mainly useful for plotting.

**Value**

the unnested model parameters from the multiverse.

**Examples**

```r
library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
  data.frame(
    id   = 1:500,
    iv1  = rnorm(500),
..."full content"
```
reveal_model_performance

Reveal the model performance/fit indices from a multiverse analysis

Description

Reveal the model performance/fit indices from a multiverse analysis

Usage

reveal_model_performance(.multi, .unpack_specs = "no")

Arguments

.multi a multiverse list-column tibble produced by run_multiverse.
Unpack specifications in a list column, wide unnests specifications with each specification category as a column. "long" unnests specifications and stacks them into long format, which stacks specifications into a decision_set and alternatives columns. This is mainly useful for plotting.

Value

The unnested model performance/fit indices from a multiverse analysis.

Examples

```r
library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod1 = rnorm(500),
mod2 = rnorm(500),
mod3 = rnorm(500),
cov1 = rnorm(500),
cov2 = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# Decision pipeline
full_pipeline <-
the_data |
  add_filters(include1 == 0, include2 != 3, include2 != 2, scale(include3) > -2.5) |
  add_variables("ivs", iv1, iv2, iv3) |
  add_variables("dvs", dv1, dv2) |
  add_variables("mods", starts_with("mod")) |
  add_model("linear_model", lm(dvs ~ {ivs} * {mods} + cov1))

pipeline_grid <- expand_decisions(full_pipeline)

# Run the whole multiverse
the_multiverse <- run_multiverse(pipeline_grid[1:10,])

# Reveal results of the linear model
the_multiverse |
  reveal_model_performance()
```
reveal_model_warnings  
Reveal any warnings about your models during a multiverse analysis

Description
Reveal any warnings about your models during a multiverse analysis

Usage
reveal_model_warnings(.multi, .unpack_specs = "no")

Arguments
- .multi  a multiverse list-column tibble produced by `run_multiverse`
- .unpack_specs character, options are "no", "wide", or "long". "no" (default) keeps specifications in a list column, wide unneests specifications with each specification category as a column. "long" unneests specifications and stacks them into long format, which stacks specifications into a `decision_set` and `alternatives` columns. This is mainly useful for plotting.

Value
the unnested model warnings captured during analysis

Examples

```r
library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod1 = rnorm(500),
  mod2 = rnorm(500),
  mod3 = rnorm(500),
  cov1 = rnorm(500),
  cov2 = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
)
```
# Decision pipeline
full_pipeline <-
  the_data |> 
  add_filters(include1 == 0, include2 != 3, include2 != 2, scale(include3) > -2.5) |> 
  add_variables("ivs", iv1, iv2, iv3) |> 
  add_variables("dvs", dv1, dv2) |> 
  add_variables("mods", starts_with("mod")) |> 
  add_model("linear_model", lm({dvs} ~ {ivs} * {mods} + cov1))

pipeline_grid <- expand_decisions(full_pipeline)

# Run the whole multiverse
the_multiverse <- run_multiverse(pipeline_grid[1:10,])

# Reveal results of the linear model
the_multiverse |> 
  reveal_model_warnings()

---

reveal_reliabilities  
_Reveal a set of multiverse cronbach’s alpha statistics_

**Description**

Reveal a set of multiverse cronbach’s alpha statistics

**Usage**

reveal_reliabilities(.descriptives, .which, .unpack_specs = "no")

**Arguments**

- `.descriptives` a descriptive multiverse list-column tibble produced by `run_descriptives`.
- `.which` the specific name of the alphas
- `.unpack_specs` character, options are "no", "wide", or "long". "no" (default) keeps specifications in a list column, wide unnests specifications with each specification category as a column. "long" unnests specifications and stacks them into long format, which stacks specifications into a decision_set and alternatives columns. This is mainly useful for plotting.

**Value**

an unnested set of correlations per decision from the multiverse.
Reveal a set of summary statistics from a multiverse analysis

**Description**

Reveal a set of summary statistics from a multiverse analysis
Usage

```r
reveal_summary_stats(.descriptives, .which, .unpack Specs = "no")
```

Arguments

- `.descriptives` a descriptive multiverse list-column tibble produced by `run_descriptives`
- `.which` the specific name of the summary statistics
- `.unpack Specs` character, options are "no", "wide", or "long". "no" (default) keeps specifications in a list column, wide unnests specifications with each specification category as a column. "long" unnests specifications and stacks them into long format, which stacks specifications into a decision_set and alternatives columns. This is mainly useful for plotting.

Value

an unnested set of summary statistics per decision from the multiverse.

Examples

```r
library(tidyverse)
library(multitool)

# create some data
the_data <-
data.frame(
  id = 1:500,
  iv1 = rnorm(500),
  iv2 = rnorm(500),
  iv3 = rnorm(500),
  mod = rnorm(500),
  dv1 = rnorm(500),
  dv2 = rnorm(500),
  include1 = rbinom(500, size = 1, prob = .1),
  include2 = sample(1:3, size = 500, replace = TRUE),
  include3 = rnorm(500)
)

# create a pipeline blueprint
full_pipeline <-
  the_data |> add_filters(
    include1 == 0,
    include2 != 3,
    include2 != 2,
    include3 > -2.5,
    include3 < 2.5,
    between(include3, -2.5, 2.5)
  ) |> add_variables(var_group = "ivs", iv1, iv2, iv3) |> add_variables(var_group = "dvs", dv1, dv2)
```
run_descriptives

```r
add_correlations("predictor correlations", starts_with("iv")) |>
add_summary_stats("iv_stats", starts_with("iv"), c("mean", "sd")) |>
add_reliabilities("vio_scale", starts_with("iv")) |>
add_model("linear model", lm({dvs} ~ {ivs} * mod))

my_descriptives <- run_descriptives(full_pipeline)

my_descriptives |>
  reveal_summary_stats(iv_stats)
```

## Description

Run a multiverse-style descriptive analysis based on a complete decision grid

## Usage

```r
run_descriptives(.pipeline, show_progress = TRUE)
```

## Arguments

- **.pipeline**: a tibble produced by a series of `add_*` calls. Importantly, this needs to be a pre-expanded pipeline because descriptive analyses only change when the underlying cases change. Thus, only filtering decisions will be used and internally expanded before calculating various descriptive analyses.

- **show_progress**: logical, whether to show a progress bar while running.

## Value

single tibble containing tidied results for all descriptive analyses specified

## Examples

```r
library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
  data.frame(
    id   = 1:500,
    iv1  = rnorm(500),
    iv2  = rnorm(500),
    iv3  = rnorm(500),
    mod1 = rnorm(500),
    mod2 = rnorm(500),
```
mod3 <- rnorm(500),
cov1 <- rnorm(500),
cov2 <- rnorm(500),
dv1 <- rnorm(500),
dv2 <- rnorm(500),
include1 <- rbinom(500, size = 1, prob = .1),
include2 <- sample(1:3, size = 500, replace = TRUE),
include3 <- rnorm(500)
)

# Decision pipeline
full_pipeline <-
  the_data |> 
  add_filters(include1 == 0, include2 != 3, include2 != 2, scale(include3) > -2.5) |> 
  add_variables("ivs", iv1, iv2, iv3) |> 
  add_variables("dvs", dv1, dv2) |> 
  add_summary_stats("iv_stats", starts_with("iv"), c("mean", "sd")) |> 
  add_summary_stats("dv_stats", starts_with("dv"), c("skewness", "kurtosis")) |> 
  add_correlations("predictors", matches("iv|mod|cov"), focus_set = c(cov1, cov2)) |> 
  add_correlations("outcomes", matches("dv|mod"), focus_set = matches("dv")) |> 
  add_reliabilities("unp_scale", c(iv1, iv2, iv3)) |> 
  add_reliabilities("vio_scale", starts_with("mod"))
run_descriptives(full_pipeline)

---

**run_multiverse**

Run a multiverse based on a complete decision grid

**Description**

Run a multiverse based on a complete decision grid

**Usage**

```r
run_multiverse(.grid, ncores = 1, save_model = FALSE, show_progress = TRUE)
```

**Arguments**

- `.grid` a tibble produced by `expand_decisions`
- `.ncores` numeric. The number of cores you want to use for parallel processing.
- `.save_model` logical, indicates whether to save the model object in its entirety. The default is FALSE because model objects are usually large and under the hood, `parameters` and `performance` is used to summarize the most useful model information.
- `.show_progress` logical, whether to show a progress bar while running.
**Value**

a single tibble containing tidied results for the model and any post-processing tests/tasks. For each unique test (e.g., an `lm` or `aov` called on an `lm`), a list column with the function name is created with `parameters` and `performance` and any warnings or messages printed while fitting the models. Internally, modeling and post-processing functions are checked to see if there are tidy or glance methods available. If not, `summary` will be called instead.

**Examples**

```r
library(tidyverse)
library(multitool)

# Simulate some data
the_data <-
data.frame(
id = 1:500,
iv1 = rnorm(500),
iv2 = rnorm(500),
iv3 = rnorm(500),
mod1 = rnorm(500),
mod2 = rnorm(500),
mod3 = rnorm(500),
cov1 = rnorm(500),
cov2 = rnorm(500),
dv1 = rnorm(500),
dv2 = rnorm(500),
include1 = rbinom(500, size = 1, prob = .1),
include2 = sample(1:3, size = 500, replace = TRUE),
include3 = rnorm(500)
)

# Decision pipeline
full_pipeline <-
the_data |>  
add_filters(include1 == 0, include2 != 3, scale(include3) > -2.5) |>  
add_variables("ivs", iv1, iv2, iv3) |>  
add_variables("dvs", dv1, dv2) |>  
add_preprocess(process_name = "scale.iv", mutate(ivs = scale(ivs))) |>  
add_preprocess(process_name = "scale.mod", mutate(mods := scale(mods))) |>  
add_model("no covariates", lm(dvs ~ ivs * mods)) |>  
add_model("covariate", lm((dvs ~ (ivs * mods) + cov1)) |>  
add_postprocess("aov", aov())

pipeline_grid <- expand_decisions(full_pipeline)

# Run the whole multiverse
the_multiverse <- run_multiverse(pipeline_grid[1:10,])
```
show_code_filter  
*Show multiverse data code pipelines*

**Description**

Each `show_code*` function should be self-explanatory - they indicate where along the multiverse pipeline to extract code. The goal of these functions is to create a window into each multiverse decision set context/results and allow the user to inspect specific decisions straight from the code that produced it.

**Usage**

```
show_code_filter(.grid, decision_num, copy = FALSE)
show_code_preprocess(.grid, decision_num, copy = FALSE)
show_code_model(.grid, decision_num, copy = FALSE)
show_code_postprocess(.grid, decision_num, copy = FALSE)
show_code_summary_stats(.grid, decision_num, copy = FALSE)
show_code_corrs(.grid, decision_num, copy = FALSE)
show_code_reliabilities(.grid, decision_num, copy = FALSE)
```

**Arguments**

- `.grid` a full decision grid created by `expand_decisions`.
- `decision_num` numeric. Indicates which 'universe' in the multiverse to show underlying code.
- `copy` logical. Whether to copy the pipeline code to the clipboard using `write.clip`. Defaults to `FALSE`.

**Value**

the code that generated results up to the specified point in an analysis pipeline. The code is printed in the console and can be optionally copied to the clipboard.

**Functions**

- `show_code_preprocess()`: Show the code up to the preprocessing stage
- `show_code_model()`: Show the code up to the modeling stage
- `show_code_postprocess()`: Show the code up to the post-processing stage
- `show_code_summary_stats()`: Show the code for computing summary statistics
- `show_code_corrs()`: Show the code for computing correlations
- `show_code_reliabilities()`: Show the code for computing scale reliability
**summarize_filter_ns**

*Summarize samples sizes for each unique filtering expression*

---

**Description**

Summarize samples sizes for each unique filtering expression

**Usage**

`summarize_filter_ns(.pipeline)`

**Arguments**

- `.pipeline` a data.frame produced by calling a series of add_* functions.

**Value**

A tibble with each row representing a filtering expression and four columns: filter_expression, variable, n_retained, and n_excluded.

**Examples**

```r
library(tidyverse)
library(multitool)

# create some data
the_data <-
  data.frame(
    id = 1:500,
    iv1 = rnorm(500),
    iv2 = rnorm(500),
    iv3 = rnorm(500),
    mod = rnorm(500),
    dv1 = rnorm(500),
    dv2 = rnorm(500),
    include1 = rbinom(500, size = 1, prob = .1),
    include2 = sample(1:3, size = 500, replace = TRUE),
    include3 = rnorm(500)
  )

# create a pipeline blueprint
full_pipeline <-
  the_data |
    add_filters(include1 == 0, include2 != 3, include3 > -2.5) |
    add_variables(var_group = "ivs", iv1, iv2, iv3) |
    add_variables(var_group = "dvs", dv1, dv2) |
    add_model("linear model", lm({dvs} ~ {ivs} * mod))

summarize_filter_ns(full_pipeline)
```
Index

add_correlations, 2
add_filters, 4
add_model, 5
add_parameter_keys, 6
add_postprocess, 7
add_preprocess, 9
add_reliabilities, 10
add_summary_stats, 11
add_variables, 12

condense, 14
create_blueprint_graph, 15

detect_multiverse_n, 16
detect_n_filters, 17
detect_n_models, 18
detect_n_variables, 19

expand_decisions, 20, 34, 36

filter, 4

glue, 9, 21

mutate, 9

parameters, 34, 35
performance, 34, 35

reveal, 14, 22
reveal_corrs, 23
reveal_model_messages, 24
reveal_model_parameters, 26
reveal_model_performance, 27
reveal_model_warnings, 29
reveal_reliabilities, 30
reveal_summary_stats, 31
run_descriptives, 23, 30, 32, 33
run_multiverse, 22, 25–27, 29, 34

show_code_corrs (show_code_filter), 36
show_code_filter, 36
show_code_model (show_code_filter), 36
show_code_postprocess
(show_code_filter), 36
show_code_preprocess
(show_code_filter), 36
show_code_reliabilities
(show_code_filter), 36
show_code_summary_stats
(show_code_filter), 36
summarize, 12
summarize_filter_ns, 37
unnest, 14

write_clip, 36