Package 'zfit'

August 27, 2023

Type Package

Title Fit Models in a Pipe

Version 0.4.0

Author Magnus Thor Torfason

Maintainer Magnus Thor Torfason <m@zulutime.net>

Description Improve the usage of model fitting functions within a piped work flow.

License MIT + file LICENSE

URL https://torfason.github.io/zfit/, https://github.com/torfason/zfit/

Depends R (>= 3.5.0)

Suggests dplyr, estimatr, MASS, pls, testthat (>= 3.0.0), tibble

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

NeedsCompilation no

Repository CRAN

Date/Publication 2023-08-27 10:00:02 UTC

R topics documented:

zfit *zfit: Fit Models in a Pipe*

Description

Improve the usage of model fitting functions within a piped work flow.

Details

`zfit` makes it easier to use a piped workflow with functions that don't have the "correct" order of parameters (the first parameter of the function does not match the object passing through the pipe).

The issue is especially prevalent with model fitting functions, such as when passing and processing a `data.frame` (or `tibble`) before passing them to `lm()` or similar functions. The pipe passes the data object into the first parameter of the function, but the conventional estimation functions expect a formula to be the first parameter.

This package addresses the issue with three functions that make it trivial to construct a pipe-friendly version of any function:

- `zfunction()` reorders the arguments of a function. Just pass the name of a function, and the name of the parameter that should receive the piped argument, and it returns a version of the function with that parameter coming first.
- `zfold()` creates a fold (a wrapper) around a function with the reordered arguments. This is sometimes needed instead of a simple reordering, for example for achieving correct S3 dispatch, and for functions that report its name or other information in output.
- `zfitter()` takes any estimation function with the standard format of a `formula` and `data` parameter, and returns a version suitable for us in pipes (with the `data` parameter coming first). Internally, it simply calls the `zfold()` function to create a fold around the fitter function.

The package also includes ready made wrappers around the most commonly used estimation functions. `zlm()`and `zglm()` correspond to `lm()` and `glm()`, and `zlogit()`, `zprobit()`, and `zpoisson()`, use `glm()` to perform logistic or poisson regression within a pipe.

Finally, the package includes the `zprint()` function, which is intended to simplify the printing of derived results, such as `summary()`, within the pipe, without affecting the modeling result itself.

See Also

- zlm is the wrapper `lm`, probably the most common fitting function. The help file for this function includes several usage examples.
- zglm is a wrapper for `glm`, to fit generalized linear models.
- zprint is helpful for printing a `summary` of a model, but assigning the evaluated model to a variable

zfunction *Create a pipe-friendly version of a function*

Description

These functions all serve the role of rearranging the arguments of other functions, in order to create pipe-friendly versions.

`zfunction()` rearranges the arguments of any function moving the specified argument to the front of the list, so that this argument becomes the recipient of piping. It returns a copy of the input function, that is identical to the original except for the order of the arguments.

`zfold()` creates a pipe-friendly version of a function of the standard format by creating a fold (or wrapper) around it with the parameters reordered. Compared to using `zfunction()`, which makes a copy of the original function with rearranged the parameters, this creates a wrapper that in turn will call the original function with all passed parameters. This is good for making pipe-friendly versions of `S3` generics, whereas rearranging parameters directly will break the `S3` dispatch mechanism.

`zfitter()` creates a pipe-friendly version of a fitting function of the standard format — that is a function with a `formula` parameter followed by a `data` parameter. It also shortens very long data names (longer than 32 characters by default), which otherwise are a nuisance when the data comes from the pipe, because the pipeline gets converted to a very long function call.

Usage

```
zfunction(fun, x, x_not_found = c("error", "warning", "ok"))

zfold(fun, x, x_not_found = c("error", "warning", "ok"))

zfitter(fun)
```

Arguments

`fun`	The function to adapt (for `zfitter()` this should be a fitting function that takes `formula` and `data` parameters). The name should not be quoted, rather, the actual function should be passed (prefixed with package if needed).
`x`	The name of the argument that should be moved to the front of the argument list. Can be passed with or without quotes, and is processed using non-standard evaluation unless surrounded with curlies, as in `{value}`, see details below.
`x_not_found`	How to handle the case where the value of `x` is not the name of a parameter in `fun`. If `error`, abort the function. If `ok`, prepend the value to the existing parameter list. This can be useful if looking to pipe data into a parameter that is hidden by a `...`.

Details

The `x` parameter is processed using non-standard evaluation, which can be disabled using curly brackets. In other words, the following are all equivalent, and return a file renaming function with the `to` parameter as the first one:

- `zfunction(file.rename, to)`
- `zfunction(file.rename, "to")`
- `param_name <- "to"; zfunction(file.rename, {param_name})`

Examples

```
# A a grep function with x as first param is often useful
zgrep <- zfunction(grep, x)
carnames <- rownames(mtcars)
grep("ll", carnames, value=TRUE)
zgrep(carnames, "ll", value=TRUE)

# zfunction() is the best approach to wrapping functions such as
# `pls::plsr()` that hide the data parameter behind the `...`.
if (requireNamespace("pls")) {
  zplsr <- zfunction(pls::plsr, data, x_not_found = "ok")
  zplsr(cars, dist ~ speed)
}

# Curly {x} handling: These are all equivalent
param_name <- "to";
f1 <- zfunction(file.rename, to)
f2 <- zfunction(file.rename, "to")
f3 <- zfunction(file.rename, {param_name})

# Using zfold() to create a grep() wrapper with the desired arg order
zgrep <- zfold(grep, x)
carnames <- rownames(mtcars)
grep("ll", carnames, value=TRUE)
zgrep(carnames, "ll", value=TRUE)

# Using zfitter to wrap around a fitting function
# (this is the actual way zlm_robust is defined in this package)
if (requireNamespace("estimatr", quietly = TRUE)) {
  zlm_robust <- zfitter(estimatr::lm_robust)
  zlm_robust(cars, speed~dist)

  # The resulting function works well the native pipe ...
  if ( getRversion() >= "4.1.0" ) {
    cars |> zlm_robust( speed ~ dist )
  }
}

# ... or with dplyr
if ( require("dplyr", warn.conflicts=FALSE) ) {

  # Pipe cars dataset into zlm_robust for fitting
  cars %>% zlm_robust( speed ~ dist )

  # Process iris with filter() before piping. Print a summary()
  # of the fitted model using zprint() before assigning the
  # model itself (not the summary) to m.
```

```
  m <- iris %>%
    dplyr::filter(Species=="setosa") %>%
    zlm_robust(Sepal.Length ~ Sepal.Width + Petal.Width) %>%
    zprint(summary)
}
```

`zglm` *Run a glm model in a pipe*

Description

These functions are wrappers for the glm function. The `zglm` function can be used to estimate any generalized linear model in a pipe. The `zlogit`, `zprobit`, and `zpoisson` functions can be used to estimate specific models. All of these functions rely on the `glm` function for the actual estimation, they simply pass the corresponding values to the `family` parameter of the `glm` function.

Usage of these functions is very similar to the zlm function (a wrapper for lm), for detailed examples, check out the entry for that function.

The `zlogit` function calls `zglm`, specifying `family=binomial(link="logit")`.

The `zprobit` function calls `zglm`, specifying `family=binomial(link="probit")`.

The `zpoisson` function calls `zglm`, specifying `family="poisson"`.

Usage

```
zglm(
  data,
  formula,
  family = gaussian,
  weights,
  subset,
  na.action,
  start = NULL,
  etastart,
  mustart,
  offset,
  control = list(...),
  model = TRUE,
  method = "glm.fit",
  x = FALSE,
  y = TRUE,
  singular.ok = TRUE,
  contrasts = NULL,
  ...
)

zlogit(data, formula, ...)
```

```
zprobit(data, formula, ...)

zpoisson(data, formula, ...)
```

Arguments

data	A data.frame containing the model data.
formula	The formula to be fitted.
family	See the glm function.
weights	See the glm function.
subset	See the glm function.
na.action	See the glm function.
start	See the glm function.
etastart	See the glm function.
mustart	See the glm function.
offset	See the glm function.
control	See the glm function.
model	See the glm function.
method	See the glm function.
x	See the glm function.
y	See the glm function.
singular.ok	See the glm function.
contrasts	See the glm function.
...	Other arguments to be passed to the glm function.

Value

A fitted model.

See Also

- zlm is the wrapper for lm, probably the most common fitting function. The help file for zlm function includes several usage examples.

`zlm` *Run an lm model in a pipe.*

Description

This function wraps around the lm function in order to make it more friendly to pipe syntax (with the data first).

Usage

```
zlm(
  data,
  formula,
  subset,
  weights,
  na.action,
  method = "qr",
  model = TRUE,
  x = FALSE,
  y = FALSE,
  qr = TRUE,
  singular.ok = TRUE,
  contrasts = NULL,
  offset,
  ...
)
```

Arguments

`data`	A `data.frame` containing the model data.
`formula`	The `formula` to be fitted.
`subset`	See the `lm` function.
`weights`	See the `lm` function.
`na.action`	See the `lm` function.
`method`	See the `lm` function.
`model`	See the `lm` function.
`x`	See the `lm` function.
`y`	See the `lm` function.
`qr`	See the `lm` function.
`singular.ok`	See the `lm` function.
`contrasts`	See the `lm` function.
`offset`	See the `lm` function.
`...`	Other arguments to be passed to the `lm` function.

Value

A fitted model.

See Also

- zglm is a wrapper for `glm`, to fit generalized linear models.

Examples

```
# Usage is possible without pipes
zlm( cars, dist ~ speed )

# zfit works well with dplyr and magrittr pipes
if ( require("dplyr", warn.conflicts=FALSE) ) {

  # Pipe cars dataset into zlm for fitting
  cars %>% zlm(speed ~ dist)

  # Process iris with filter before piping to zlm
  iris %>%
    filter(Species == "setosa") %>%
    zlm(Sepal.Length ~ Sepal.Width + Petal.Width)
}

# zfit also works well with the native pipe
if ( require("dplyr") && getRversion() >= "4.1.0" ) {

  # Pipe cars dataset into zlm for fitting
  cars |> zlm(speed ~ dist)

  # Process iris with filter() before piping. Print a
  # summary of the fitted model using zprint() before
  # assigning the model itself (not the summary) to m.
  m <- iris |>
    filter(Species == "setosa") |>
    zlm(Sepal.Length ~ Sepal.Width + Petal.Width) |>
    zprint(summary)
}
```

`zlm_robust` *Pipe-friendly wrappers for external fitters*

Description

These functions provide pipe-friendly wrappers around model fitters provided by several external packages. The functions require the corresponding packages to be installed, if the required package is missing the functions warns with directions for how to install it.

`zlm_robust()` wraps `estimatr::lm_robust()`, which fits a linear model with a variety of options for estimating robust standard errors.

`zpolr()` wraps `MASS::polr()`, which fits an ordered logistic response for multi-value ordinal variables, using a proportional odds logistic regression.

`zplsr()` wraps `pls::plsr()`, which performs a partial least squares regression.

Examples

```
if (requireNamespace("estimatr") && getRversion() >= "4.1.0")
  zlm_robust(cars, dist ~ speed) |> summary() |> try()

if (requireNamespace("MASS") && getRversion() >= "4.1.0")
  zpolr(mtcars, ordered(gear) ~ mpg + hp) |> summary() |> try()

if (requireNamespace("pls") && getRversion() >= "4.1.0")
  zplsr(cars, dist ~ speed) |> summary() |> try()
```

---

zprint *Print the result of a function in a pipe but return original object*

Description

Given `x` and `f`, this function prints `f(x)` before returning the original `x`. It is useful in a pipe, when one wants a to print the derivative of an object in the pipe but then return or assign the original object. A common use case is printing the 'summary() of an estimated model but then assigning the original model (rather than the summary object) to a variable for further processing.

Usage

```
zprint(x, f = NULL, ...)
```

Arguments

`x`	An object, typically in a pipe.
`f`	A function to be applied to `x` before printing.
`...`	Other arguments to be passed to `f`.

Value

The original object `x`.

Examples

```
if (getRversion() >= "4.1.0" && require("dplyr")) {

  # Print summary before assigning model to variable
  m <- lm( speed ~ dist, cars) |>
    zprint(summary) # prints summary(x)
  m                 # m is the original model object

  # Print grouped data before filtering original
  cw_subset <- chickwts |>
    zprint(count, feed, sort=TRUE) |> # prints counts by feed
    filter(feed=="soybean")
  cw_subset # cw_subset is ungrouped, but filtered by feed
}
```

Index