# Package 'intubate'

August 29, 2016

**Type** Package

**Title** Interface to Popular R Functions for Data Science Pipelines

**Version** 1.0.0

**Date** 2016-08-28

**Author** Roberto Bertolusso

**Maintainer** Roberto Bertolusso <rbertolusso@rice.edu>

**Description** Interface to popular R functions with formulas and data, such as 'lm', so they can be included painlessly in data science pipelines implemented by 'magrittr' with the operator %>%.

**License** GPL (>= 2)

**ByteCompile** no

**Suggests** knitr, magrittr, rmarkdown

**VignetteBuilder** knitr

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2016-08-28 07:35:30

## R topics documented:

---

`intubate-package` *Interface to Popular R Functions for Data Science Pipelines.*

---

## Description

The aim of `intubate` (logo <||>) is to offer a painless way to add R functions that that are not pipe-aware to data science pipelines implemented by 'magrittr' with the operator `%>%`, without having to rely on workarounds of varying complexity. It also implements three extensions (experimental), called 'intubOrders', 'intuEnv', and 'intuBags'.

For a gentle introduction to `intubate`, please see the vignette that is included with the package.

Currently, there are 461 interfaces for:

`adabag`: Multiclass AdaBoost.M1, SAMME and Bagging

AER: Applied Econometrics with R

aod: Analysis of Overdispersed Data

ape: Analyses of Phylogenetics and Evolution

arm: Data Analysis Using Regression and Multilevel/Hierarchical Models

betareg: Beta Regression

brglm: Bias reduction in binomial-response generalized linear models

caper: Comparative Analyses of Phylogenetics and Evolution in R

car: Companion to Applied Regression

caret: Classification and Regression Training

coin: Conditional Inference Procedures in a Permutation Test Framework

CORElearn: Classification, Regression and Feature Evaluation

drc: Analysis of Dose-Response Curves

e1071: Support Vector Machines

earth: Multivariate Adaptive Regression Splines

EnvStats: Environmental Statistics, Including US EPA Guidance

fGarch: Rmetrics - Autoregressive Conditional Heteroskedastic Modelling

flexmix: Flexible Mixture Modeling

forecast: Forecasting Functions for Time Series and Linear Models

frontier: Stochastic Frontier Analysis

gam: Generalized Additive Models

gbm: Generalized Boosted Regression Models

gee: Generalized Estimation Equation Solver

glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models

glmx: Generalized Linear Models Extended

gmnl: Multinomial Logit Models with Random Parameters

gplots: Various R Programming Tools for Plotting Data

graphics: The R Graphics Package

gss: General Smoothing Splines

hdm: High-Dimensional Metrics

Hmisc: Harrell Miscellaneous

ipred: Improved Predictors

iRegression: Regression Methods for Interval-Valued Variables

ivfixed: Instrumental fixed effect panel data model

kernlab: Kernel-Based Machine Learning Lab

kknn: Weighted k-Nearest Neighbors

klaR: Classification and Visualization

lars: Least Angle Regression, Lasso and Forward Stagewise

lattice: Trellis Graphics for R

latticeExtra: Extra Graphical Utilities Based on Lattice

leaps: Regression Subset Selection

lfe: Linear Group Fixed Effects

lme4: Linear Mixed-Effects Models using 'Eigen' and S4

lmtest: Testing Linear Regression Models

MASS: Robust Regression, Linear Discriminant Analysis, Ridge Regression, Probit Regression, ...

MCMCglmm: MCMC Generalised Linear Mixed Models

mda: Mixture and Flexible Discriminant Analysis

metafor: Meta-Analysis Package for R

mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation

mhurdle: Multiple Hurdle Tobit Models

minpack.lm: R Interface to the Levenberg-Marquardt Nonlinear Least-Squares Algorithm Found in MINPACK, Plus Support for Bounds

mlogit: Multinomial logit model

mnlogit: Multinomial Logit Model

modeltools: Tools and Classes for Statistical Models

nlme: Linear and Nonlinear Mixed Effects Models

nlreg: Higher Order Inference for Nonlinear Heteroscedastic Models

nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models

ordinal: Regression Models for Ordinal Data

party: A Laboratory for Recursive Partytioning

partykit: A Toolkit for Recursive Partytioning

plotrix: Various Plotting Functions

pls: Partial Least Squares and Principal Component Regression

pROC: Display and Analyze ROC Curves

pscl: Political Science Computational Laboratory, Stanford University

psychomix: Psychometric Mixture Models

psychotools: Infrastructure for Psychometric Modeling

psychotree: Recursive Partitioning Based on Psychometric Models

quantreg: Quantile Regression

randomForest: Random Forests for Classification and Regression

Rchoice: Discrete Choice (Binary, Poisson and Ordered) Models with Random Parameters

rminer: Data Mining Classification and Regression Methods

rms: Regression Modeling Strategies

robustbase: Basic Robust Statistics

rpart: Recursive Partitioning and Regression Trees

RRF: Regularized Random Forest

RWeka: R/Weka Interface

sampleSelection: Sample Selection Models

sem: Structural Equation Models

spBayes: Univariate and Multivariate Spatial-temporal Modeling

stats: The R Stats Package (glm, lm, loess, lqs, nls, ...)

strucchange: Testing, Monitoring, and Dating Structural Changes

survey: Analysis of Complex Survey Samples

survival: Survival Analysis

SwarmSVM: Ensemble Learning Algorithms Based on Support Vector Machines

systemfit: Estimating Systems of Simultaneous Equations

tree: Classification and Regression Trees

vcd: Visualizing Categorical Data

vegan: Community Ecology Package

The aim is to provide interfaces to most methodologies used in data science.

intubate core depends only on base, stats, and utils libraries. To keep it as lean as possible, intubate will not install not load any library. You need to make sure that the library containing the functions to be interfaced are loaded (before or after intubate). Moreover, you can interface the functions of any library directly without the need to create interfaces (see ntbt) so perhaps in the future that will be the preferred way of using intubate.

intubate is still a work in progress. As such, the implementation may change in future versions until stabilization.

## Details

| | |
|---|---|
| Package: | intubate |
| Type: | Package |
| Version: | 1.0.0 |
| Date: | 2016-08-27 |
| License: | GPL (>=2) |

See examples of use below.

## Author(s)

Roberto Bertolusso

Maintainer: Roberto Bertolusso <rbertolusso@rice.edu>

## See Also

intubate

## Examples

```
## Not run:
library(intubate)
library(magrittr)

######### Interface to lm #########
## Original function to interface
lm(conc ~ uptake, CO2)

## The interface reverses the order of data and formula
ntbt_lm(CO2, conc ~ uptake)

## so it can be used easily in a pipeline.
CO2 %>%
  ntbt_lm(conc ~ uptake)

CO2 %>%
  ntbt_lm(conc ~ uptake) %>%
  summary()

######### Interface to cor.test #########
## Original function to interface
cor.test(~ CONT + INTG, data = USJudgeRatings)

## The interface reverses the order of data and formula
ntbt_cor.test(data = USJudgeRatings, ~ CONT + INTG)

## so it can be used easily in a pipeline.
USJudgeRatings %>%
  ntbt_cor.test(~ CONT + INTG)

######### Interfaces to aggregate and xtabs #########
## Original function to interface
ag <- aggregate(len ~ ., data = ToothGrowth, mean)
xtabs(len ~ ., data = ag)

## The interface reverses the order of data and formula
ag <- ntbt_aggregate(ToothGrowth, len ~ ., mean)
ntbt_xtabs(ag, len ~ .)

## so it can be used easily in a pipeline.
ToothGrowth %>%
  ntbt_aggregate(len ~ ., mean) %>%
  ntbt_xtabs(len ~ .)

## End(Not run)
```

---

adabag *Interfaces for adabag package for data science pipelines.*

---

## Description

Interfaces to adabag functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_autoprune(data, ...)
# ntbt_bagging(data, ...)  ## Already defined in ipred
ntbt_bagging.cv(data, ...)
ntbt_boosting(data, ...)
ntbt_boosting.cv(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(adabag)


## ntbt_autoprune: Builds automatically a pruned tree of class rpart
## Original function to interface
autoprune(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_autoprune(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_autoprune(Species ~ .)

## ntbt_bagging: Applies the Bagging algorithm to a data set
library(rpart)
data(iris)
```

```
## Original function to interface
bagging(Species ~ ., data = iris, mfinal = 10)

## The interface puts data as first parameter
ntbt_bagging(iris, Species ~ ., mfinal = 10)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_bagging(Species ~ ., mfinal = 10)



## Original function to interface
iris.baggingcv <- bagging.cv(Species ~ ., v = 2, data = iris, mfinal = 10,
                             control = rpart.control(cp = 0.01))
iris.baggingcv[-1]

## The interface puts data as first parameter
iris.baggingcv <- ntbt_bagging.cv(iris, Species ~ ., v = 2, mfinal = 10,
                                  control = rpart.control(cp = 0.01))
iris.baggingcv[-1]

## so it can be used easily in a pipeline.
iris.baggingcv <- iris %>%
  ntbt_bagging.cv(Species ~ ., v = 2, mfinal = 10,
                  control = rpart.control(cp = 0.01))
iris.baggingcv[-1]


## ntbt_boosting: Applies the AdaBoost.M1 and SAMME algorithms to a data set
## Original function to interface
boosting(Species ~ ., data = iris, boos = TRUE, mfinal = 5)

## The interface puts data as first parameter
ntbt_boosting(iris, Species ~ ., boos = TRUE, mfinal = 5)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_boosting(Species ~ ., boos = TRUE, mfinal = 5)


## ntbt_boosting.cv: Runs v-fold cross validation with AdaBoost.M1 or SAMME
## Original function to interface
iris.boostcv <- boosting.cv(Species ~ ., v = 2, data = iris, mfinal = 10,
                            control = rpart.control(cp = 0.01))
iris.boostcv[-1]

## The interface puts data as first parameter
iris.boostcv <- ntbt_boosting.cv(iris, Species ~ ., v = 2, mfinal = 10,
                                 control = rpart.control(cp = 0.01))
iris.boostcv[-1]

## so it can be used easily in a pipeline.
```

```
iris.boostcv <- iris %>%
  ntbt_boosting.cv(Species ~ ., v = 2, mfinal = 10,
                   control = rpart.control(cp = 0.01))
iris.boostcv[-1]

## End(Not run)
```

---

AER *Interfaces for AER package for data science pipelines.*

---

## Description

Interfaces to AER functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_ivreg(data, ...)
ntbt_tobit(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(AER)


## ntbt_ivreg: Instrumental-Variable Regression
data("CigarettesSW", package = "AER")
CigarettesSW$rprice <- with(CigarettesSW, price/cpi)
CigarettesSW$rincome <- with(CigarettesSW, income/population/cpi)
CigarettesSW$tdiff <- with(CigarettesSW, (taxs - tax)/cpi)
```

```
## Original function to interface
ivreg(log(packs) ~ log(rprice) + log(rincome) | log(rincome) + tdiff + I(tax/cpi),
      data = CigarettesSW, subset = year == "1995")

## The interface puts data as first parameter
ntbt_ivreg(CigarettesSW,
           log(packs) ~ log(rprice) + log(rincome) | log(rincome) + tdiff + I(tax/cpi),
           subset = year == "1995")

## so it can be used easily in a pipeline.
CigarettesSW %>%
  ntbt_ivreg(log(packs) ~ log(rprice) + log(rincome) | log(rincome) + tdiff + I(tax/cpi),
             subset = year == "1995")


## ntbt_tobit: Tobit Regression
data("Affairs")

## Original function to interface
tobit(affairs ~ age + yearsmarried + religiousness + occupation + rating,
      data = Affairs)

## The interface puts data as first parameter
ntbt_tobit(Affairs,
           affairs ~ age + yearsmarried + religiousness + occupation + rating)

## so it can be used easily in a pipeline.
Affairs %>%
  ntbt_tobit(affairs ~ age + yearsmarried + religiousness + occupation + rating)

## End(Not run)
```

---

aod *Interfaces for aod package for data science pipelines.*

---

## Description

Interfaces to aod functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_betabin(data, ...)
ntbt_donner(data, ...)
ntbt_negbin(data, ...)
ntbt_quasibin(data, ...)
ntbt_quasipois(data, ...)
ntbt_raoscott(data, ...)
ntbt_splitbin(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(aod)


## ntbt_betabin: beta-binomial generalized linear model accounting
##               for overdispersion in clustered binomial data (n, y)
data(orob2)
## Original function to interface
betabin(cbind(y, n - y) ~ seed, ~ 1, data = orob2)

## The interface puts data as first parameter
ntbt_betabin(orob2, cbind(y, n - y) ~ seed, ~ 1)

## so it can be used easily in a pipeline.
orob2 %>%
  ntbt_betabin(cbind(y, n - y) ~ seed, ~ 1)


## ntbt_donner: Test of Proportion Homogeneity using Donner's Adjustment
data(rats)

## Original function to interface
donner(formula = cbind(y, n - y) ~ group, data = rats)

## The interface puts data as first parameter
ntbt_donner(rats, formula = cbind(y, n - y) ~ group)

## so it can be used easily in a pipeline.
rats %>%
  ntbt_donner(formula = cbind(y, n - y) ~ group)
```

```
## ntbt_negbin: negative-binomial log linear model accounting
##              for overdispersion in counts y
data(salmonella)
## Original function to interface
negbin(y ~ log(dose + 10) + dose, ~ 1, salmonella)

## The interface puts data as first parameter
ntbt_negbin(salmonella, y ~ log(dose + 10) + dose, ~ 1)

## so it can be used easily in a pipeline.
salmonella %>%
  ntbt_negbin(y ~ log(dose + 10) + dose, ~ 1)


## ntbt_quasibin: Quasi-Likelihood Model for Proportions
data(orob2)
## Original function to interface
quasibin(cbind(y, n - y) ~ seed * root, data = orob2, phi = 0)

## The interface puts data as first parameter
ntbt_quasibin(orob2, cbind(y, n - y) ~ seed * root, phi = 0)

## so it can be used easily in a pipeline.
orob2 %>%
  ntbt_quasibin(cbind(y, n - y) ~ seed * root, phi = 0)


## ntbt_quasipois: Quasi-Likelihood Model for Counts
data(salmonella)

## Original function to interface
quasipois(y ~ log(dose + 10) + dose, data = salmonella)

## The interface puts data as first parameter
ntbt_quasipois(salmonella, y ~ log(dose + 10) + dose)

## so it can be used easily in a pipeline.
salmonella %>%
  ntbt_quasipois(y ~ log(dose + 10) + dose)


## ntbt_raoscott: Test of Proportion Homogeneity using Rao and Scott's Adjustment
data(rats)

## Original function to interface
raoscott(cbind(y, n - y) ~ group, data = rats)

## The interface puts data as first parameter
ntbt_raoscott(rats, cbind(y, n - y) ~ group)

## so it can be used easily in a pipeline.
rats %>%
```

```
  ntbt_raoscott(cbind(y, n - y) ~ group)


## ntbt_splitbin: Split Grouped Data Into Individual Data
mydata <- data.frame(
    success = c(0, 1, 0, 1),
    f1 = c("A", "A", "B", "B"),
    f2 = c("C", "D", "C", "D"),
    n = c(4, 2, 1, 3)
    )
## Original function to interface
splitbin(formula = n ~ f1 + f2 + success, data = mydata)

## The interface puts data as first parameter
ntbt_splitbin(mydata, formula = n ~ f1 + f2 + success)

## so it can be used easily in a pipeline.
mydata %>%
  ntbt_splitbin(formula = n ~ f1 + f2 + success)

## End(Not run)
```

---

ape *Interfaces for ape package for data science pipelines.*

---

## Description

Interfaces to ape functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_binaryPGLMM(data, ...)
ntbt_compar.gee(data, ...)
ntbt_correlogram.formula(data, ...)
ntbt_lmorigin(data, ...)
ntbt_yule.cov(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(ape)

## ntbt_binaryPGLMM: Phylogenetic Generalized Linear Mixed Model for Binary Data
n <- 100
phy <- compute.brlen(rtree(n=n), method = "Grafen", power = 1)
X1 <- rTraitCont(phy, model = "BM", sigma = 1)
X1 <- (X1 - mean(X1))/var(X1)
sim.dat <- data.frame(Y=array(0, dim=n), X1=X1, row.names=phy$tip.label)
sim.dat$Y <- binaryPGLMM.sim(Y ~ X1, phy = phy, data = sim.dat, s2 = .5,
                             B = matrix(c(0, .25), nrow = 2, ncol = 1), nrep = 1)$Y

## Original function to interface
binaryPGLMM(Y ~ X1, phy = phy, data = sim.dat)

## The interface puts data as first parameter
ntbt_binaryPGLMM(sim.dat, Y ~ X1, phy = phy)

## so it can be used easily in a pipeline.
sim.dat %>%
  ntbt_binaryPGLMM(Y ~ X1, phy = phy)


## ntbt_compar.gee: Comparative Analysis with GEEs
tr <- "((((Homo:0.21,Pongo:0.21):0.28,Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);"
tree.primates <- read.tree(text = tr)
dta <- data.frame(X = c(4.09434, 3.61092, 2.37024, 2.02815, -1.46968),
                  Y = c(4.74493, 3.33220, 3.36730, 2.89037, 2.30259))
rownames(dta) <- tree.primates$tip.label

## Original function to interface
compar.gee(X ~ Y, phy = tree.primates, data = dta)

## The interface puts data as first parameter
ntbt_compar.gee(dta, X ~ Y, phy = tree.primates)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_compar.gee(X ~ Y, phy = tree.primates)


## ntbt_correlogram.formula: Phylogenetic Correlogram
data(carnivora)

## Original function to interface
```

```
correlogram.formula(SW ~ Order/SuperFamily/Family/Genus,
                    data = carnivora)

## The interface puts data as first parameter
ntbt_correlogram.formula(carnivora, SW ~ Order/SuperFamily/Family/Genus)

## so it can be used easily in a pipeline.
carnivora %>%
  ntbt_correlogram.formula(SW ~ Order/SuperFamily/Family/Genus)


## ntbt_lmorigin: Multiple regression through the origin
data(lmorigin.ex1)

## Original function to interface
lmorigin(SO2 ~ ., data = lmorigin.ex1, origin = FALSE, nperm = 99)

## The interface puts data as first parameter
ntbt_lmorigin(lmorigin.ex1, SO2 ~ ., origin = FALSE, nperm = 99)

## so it can be used easily in a pipeline.
lmorigin.ex1 %>%
  ntbt_lmorigin(SO2 ~ ., origin = FALSE, nperm = 99)

## ntbt_yule.cov: Fits the Yule Model With Covariates
data(bird.orders)
dta <- data.frame (x = rnorm(45))

## Original function to interface
yule.cov(bird.orders, ~ x, data = dta)

## The interface puts data as first parameter
ntbt_yule.cov(dta, bird.orders, ~ x)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_yule.cov(bird.orders, ~ x)

## End(Not run)
```

---

arm *Interfaces for arm package for data science pipelines.*

---

## Description

Interfaces to arm functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_bayesglm(data, ...)
ntbt_bayespolr(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(arm)

## ntbt_bayesglm: Bayesian generalized linear models
n <- 100
x1 <- rnorm (n)
x2 <- rbinom (n, 1, .5)
b0 <- 1
b1 <- 1.5
b2 <- 2
y <- rbinom(n, 1, invlogit(b0+b1*x1+b2*x2))

dta <- data.frame(y, x1, x2)

## Original function to interface
bayesglm(y ~ x1 + x2, family = binomial(link="logit"), data = dta,
         prior.scale = Inf, prior.df = Inf)

## The interface puts data as first parameter
ntbt_bayesglm(dta, y ~ x1 + x2, family = binomial(link="logit"),
              prior.scale = Inf, prior.df = Inf)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_bayesglm(y ~ x1 + x2, family = binomial(link="logit"),
                prior.scale = Inf, prior.df = Inf)


## ntbt_bayespolr: Bayesian Ordered Logistic or Probit Regression
## Original function to interface
bayespolr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing,
```

```
          prior.scale = Inf, prior.df = Inf)

## The interface puts data as first parameter
ntbt_bayespolr(housing, Sat ~ Infl + Type + Cont, weights = Freq,
               prior.scale = Inf, prior.df = Inf)

## so it can be used easily in a pipeline.
housing %>%
  ntbt_bayespolr(Sat ~ Infl + Type + Cont, weights = Freq,
                 prior.scale = Inf, prior.df = Inf)

## End(Not run)
```

---

`betareg` *Interfaces for betareg package for data science pipelines.*

---

## Description

Interfaces to `betareg` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_betamix(data, ...)
ntbt_betareg(data, ...)
ntbt_betatree(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(betareg)


## ntbt_betamix: Finite Mixtures of Beta Regression for Rates and Proportions
data("ReadingSkills", package = "betareg")

## Original function to interface
set.seed(4040)
betamix(accuracy ~ iq, data = ReadingSkills, k = 3, nstart = 10,
        extra_components = extraComponent(type = "uniform", coef = 0.99, delta = 0.01))

## The interface puts data as first parameter
set.seed(4040)
ntbt_betamix(ReadingSkills, accuracy ~ iq, k = 3, nstart = 10,
          extra_components = extraComponent(type = "uniform", coef = 0.99, delta = 0.01))

## so it can be used easily in a pipeline.
ReadingSkills %>%
  ntbt_betamix(accuracy ~ iq, k = 3, nstart = 10,
           extra_components = extraComponent(type = "uniform", coef = 0.99, delta = 0.01))


## ntbt_betareg: Beta Regression for Rates and Proportions
data("GasolineYield", package = "betareg")

## Original function to interface
betareg(yield ~ batch + temp, data = GasolineYield)

## The interface puts data as first parameter
ntbt_betareg(GasolineYield, yield ~ batch + temp)

## so it can be used easily in a pipeline.
GasolineYield %>%
  ntbt_betareg(yield ~ batch + temp)


## ntbt_betatree: Beta Regression Trees
data("ReadingSkills", package = "betareg")
ReadingSkills$x1 <- rnorm(nrow(ReadingSkills))
ReadingSkills$x2 <- runif(nrow(ReadingSkills))
ReadingSkills$x3 <- factor(rnorm(nrow(ReadingSkills)) > 0)

library(partykit)
## Original function to interface
set.seed(1071)
bt <- betatree(accuracy ~ iq | iq, ~ dyslexia + x1 + x2 + x3,
               data = ReadingSkills, minsize = 10)
plot(bt)
```

```
## The interface puts data as first parameter
set.seed(1071)
bt <- ntbt_betatree(ReadingSkills, accuracy ~ iq | iq, ~ dyslexia + x1 + x2 + x3,
                    minsize = 10)
plot(bt)

## so it can be used easily in a pipeline.
set.seed(1071)
ReadingSkills %>%
  ntbt_betatree(accuracy ~ iq | iq, ~ dyslexia + x1 + x2 + x3, minsize = 10) %>%
  plot()

## End(Not run)
```

---

`brglm` *Interfaces for brglm package for data science pipelines.*

---

## Description

Interfaces to `brglm` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_brglm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(brglm)

## ntbt_brglm: Bias reduction in Binomial-response GLMs
data(lizards)

## Original function to interface
brglm(cbind(grahami, opalinus) ~ height + diameter +
        light + time, family = binomial(logit), data = lizards,
      method = "brglm.fit")
## The interface puts data as first parameter
ntbt_brglm(lizards, cbind(grahami, opalinus) ~ height + diameter +
             light + time, family = binomial(logit),
           method = "brglm.fit")

## so it can be used easily in a pipeline.
lizards %>%
  ntbt_brglm(cbind(grahami, opalinus) ~ height + diameter +
               light + time, family = binomial(logit),
             method = "brglm.fit")

## End(Not run)
```

---

caper *Interfaces for caper package for data science pipelines.*

---

## Description

Interfaces to caper functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_brunch(data, ...)
ntbt_crunch(data, ...)
ntbt_macrocaic(data, ...)
ntbt_pgls(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(caper)

## ntbt_brunch: Calculate a linear model using the brunch algorithm
data(perissodactyla)
perisso <- comparative.data(perissodactyla.tree, perissodactyla.data, Binomial)

## Original function to interface
brunch(log.female.wt ~ Territoriality, data = perisso)

## The interface puts data as first parameter
ntbt_brunch(perisso, log.female.wt ~ Territoriality)

## so it can be used easily in a pipeline.
perisso %>%
  ntbt_brunch(log.female.wt ~ Territoriality)


## ntbt_crunch: Calculate a linear model using the crunch algorithm
data(shorebird)
shorebird <- comparative.data(shorebird.tree, shorebird.data, Species)

## Original function to interface
crunch(Egg.Mass ~ F.Mass + M.Mass, data = shorebird)

## The interface puts data as first parameter
ntbt_crunch(shorebird, Egg.Mass ~ F.Mass + M.Mass)

## so it can be used easily in a pipeline.
shorebird %>%
  ntbt_crunch(Egg.Mass ~ F.Mass + M.Mass)


## ntbt_macrocaic: Comparative analysis using independent
##                 contrasts on species richness data
data(IsaacEtAl)
primates <- comparative.data(primates.tree, primates.data, binomial, na.omit=FALSE)

## Original function to interface
macrocaic(species.rich ~ body.mass, data = primates)
```

```
## The interface puts data as first parameter
ntbt_macrocaic(primates, species.rich ~ body.mass)

## so it can be used easily in a pipeline.
primates %>%
  ntbt_macrocaic(species.rich ~ body.mass)



## ntbt_pgls: Phylogenetic generalized linear models
data(shorebird)
shorebird <- comparative.data(shorebird.tree, shorebird.data, Species, vcv=TRUE, vcv.dim=3)

## Original function to interface
pgls(log(Egg.Mass) ~ log(M.Mass) * log(F.Mass), shorebird, lambda='ML')

## The interface puts data as first parameter
ntbt_pgls(shorebird, log(Egg.Mass) ~ log(M.Mass) * log(F.Mass), lambda='ML')

## so it can be used easily in a pipeline.
shorebird %>%
  ntbt_pgls(log(Egg.Mass) ~ log(M.Mass) * log(F.Mass), lambda='ML')

## End(Not run)
```

---

car *Interfaces for car package for data science pipelines.*

---

## Description

Interfaces to car functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_Boxplot(data, ...)
ntbt_boxTidwell(data, ...)
ntbt_densityPlot(data, ...)
ntbt_invTranPlot(data, ...)
ntbt_leveneTest(data, ...)
ntbt_powerTransform(data, ...)
ntbt_scatter3d(data, ...)
ntbt_scatterplot(data, ...)
ntbt_scatterplotMatrix(data, ...)
ntbt_spreadLevelPlot(data, ...)
ntbt_symbox(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(car)

## ntbt_Boxplot: Boxplots With Point Identification
## Original function to interface
Boxplot(income ~ type, data = Prestige)

## The interface puts data as first parameter
ntbt_Boxplot(Prestige, income ~ type)

## so it can be used easily in a pipeline.
Prestige %>%
  ntbt_Boxplot(income ~ type)


## ntbt_boxTidwell: Box-Tidwell Transformations
## Original function to interface
boxTidwell(prestige ~ income + education, ~ type + poly(women, 2), data = Prestige)

## The interface puts data as first parameter
ntbt_boxTidwell(Prestige, prestige ~ income + education, ~ type + poly(women, 2))

## so it can be used easily in a pipeline.
Prestige %>%
  ntbt_boxTidwell(prestige ~ income + education, ~ type + poly(women, 2))


## ntbt_densityPlot: Nonparametric Density Estimates
## Original function to interface
densityPlot(income ~ type, data = Prestige)

## The interface puts data as first parameter
ntbt_densityPlot(Prestige, income ~ type)

## so it can be used easily in a pipeline.
Prestige %>%
  ntbt_densityPlot(income ~ type)
```

```
## ntbt_invTranPlot: Choose a Predictor Transformation Visually or Numerically
## Original function to interface
invTranPlot(infant.mortality ~ gdp, data = UN)

## The interface puts data as first parameter
ntbt_invTranPlot(UN, infant.mortality ~ gdp)

## so it can be used easily in a pipeline.
UN %>%
  ntbt_invTranPlot(infant.mortality ~ gdp)

## ntbt_leveneTest: Levene's test for homogeneity of variance across groups
## Original function to interface
leveneTest(conformity ~ fcategory*partner.status, data = Moore)

## The interface puts data as first parameter
ntbt_leveneTest(Moore, conformity ~ fcategory*partner.status)

## so it can be used easily in a pipeline.
Moore %>%
  ntbt_leveneTest(conformity ~ fcategory*partner.status)

## ntbt_powerTransform: Finding Univariate or Multivariate Power Transformations
## Original function to interface
powerTransform(cycles ~ len + amp + load, Wool)

## The interface puts data as first parameter
ntbt_powerTransform(Wool, cycles ~ len + amp + load)

## so it can be used easily in a pipeline.
Wool %>%
  ntbt_powerTransform(cycles ~ len + amp + load)


## ntbt_scatter3d: Three-Dimensional Scatterplots and Point Identification
## Original function to interface
## NOTE: need rgl, mgcv, and interactive mode. Commented out.
#scatter3d(prestige ~ income + education, data = Duncan)

## The interface puts data as first parameter
#ntbt_scatter3d(Duncan, prestige ~ income + education)

## so it can be used easily in a pipeline.
#Duncan %>%
#  ntbt_scatter3d(prestige ~ income + education)


## ntbt_scatterplot: Scatterplots with Boxplots
## Original function to interface
scatterplot(prestige ~ income, data = Prestige, ellipse = TRUE)
```

```
## The interface puts data as first parameter
ntbt_scatterplot(Prestige, prestige ~ income, ellipse = TRUE)

## so it can be used easily in a pipeline.
Prestige %>%
  ntbt_scatterplot(prestige ~ income, ellipse = TRUE)

## ntbt_scatterplotMatrix: Scatterplot Matrices
## Original function to interface
scatterplotMatrix(~ income + education + prestige | type, data = Duncan)

## The interface puts data as first parameter
ntbt_scatterplotMatrix(Duncan, ~ income + education + prestige | type)

## so it can be used easily in a pipeline.
Duncan %>%
  ntbt_scatterplotMatrix(~ income + education + prestige | type)

## ntbt_spreadLevelPlot: Spread-Level Plots
## Original function to interface
spreadLevelPlot(interlocks + 1 ~ nation, data = Ornstein)

## The interface puts data as first parameter
ntbt_spreadLevelPlot(Ornstein, interlocks + 1 ~ nation)

## so it can be used easily in a pipeline.
Ornstein %>%
  ntbt_spreadLevelPlot(interlocks + 1 ~ nation)

## ntbt_symbox: Boxplots for transformations to symmetry
## Original function to interface
symbox(~ income, data = Prestige)

## The interface puts data as first parameter
ntbt_symbox(Prestige, ~ income)

## so it can be used easily in a pipeline.
Prestige %>%
  ntbt_symbox(~ income)

## End(Not run)
```

---

`caret` *Interfaces for caret package for data science pipelines.*

---

## Description

Interfaces to `caret` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_avNNet(data, ...)
ntbt_bagEarth(data, ...)
ntbt_bagFDA(data, ...)
ntbt_calibration(data, ...)
ntbt_dummyVars(data, ...)
ntbt_icr(data, ...)
ntbt_knn3(data, ...)
ntbt_lift(data, ...)
ntbt_pcaNNet(data, ...)
ntbt_sbf(data, ...)
ntbt_train(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(caret)


## ntbt_avNNet: Neural Networks Using Model Averaging
## Not found example using formula interface, and I am
## completely ignorant to construct one.
data(BloodBrain)
BB <- list(bbbDescr, logBBB)

## Original function to interface
avNNet(bbbDescr, logBBB, size = 5, linout = TRUE, trace = FALSE)

## The interface puts data as first parameter
ntbt_avNNet(BB, bbbDescr, logBBB, size = 5, linout = TRUE, trace = FALSE)
```

```
## so it can be used easily in a pipeline.
BB %>%
  ntbt_avNNet(bbbDescr, logBBB, size = 5, linout = TRUE, trace = FALSE)


## ntbt_bagEarth: Bagged Earth

## Original function to interface
bagEarth(Volume ~ ., data = trees)

## The interface puts data as first parameter
ntbt_bagEarth(trees, Volume ~ .)

## so it can be used easily in a pipeline.
trees %>%
  ntbt_bagEarth(Volume ~ .)


## ntbt_bagFDA: Bagged FDA
library(mlbench)
library(earth)
data(Glass)

set.seed(36)
inTrain <- sample(1:dim(Glass)[1], 150)

trainData <- Glass[ inTrain, ]
testData  <- Glass[-inTrain, ]
## Original function to interface
## bagFDA(Type ~ ., trainData)   ## There is an error:
## Error in requireNamespaceQuietStop("mda") : package mda is required
##                                ## even when mda is installed
## For now all of this stays commented.

## The interface puts data as first parameter
## ntbt_bagFDA(trainData, Type ~ .)

## so it can be used easily in a pipeline.
## trainData %>%
##   ntbt_bagFDA(Type ~ .)


## ntbt_calibration: Probability Calibration Plot
data(mdrr)
mdrrDescr <- mdrrDescr[, -nearZeroVar(mdrrDescr)]
mdrrDescr <- mdrrDescr[, -findCorrelation(cor(mdrrDescr), .5)]
inTrain <- createDataPartition(mdrrClass)
trainX <- mdrrDescr[inTrain[[1]], ]
trainY <- mdrrClass[inTrain[[1]]]
testX <- mdrrDescr[-inTrain[[1]], ]
testY <- mdrrClass[-inTrain[[1]]]
library(MASS)
ldaFit <- lda(trainX, trainY)
```

```
qdaFit <- qda(trainX, trainY)
testProbs <- data.frame(obs = testY,
lda <- predict(ldaFit, testX)$posterior[,1],
qda <- predict(qdaFit, testX)$posterior[,1])

## Original function to interface
calPlotData <- calibration(obs ~ lda + qda, data = testProbs)
xyplot(calPlotData, auto.key = list(columns = 2))

## The interface puts data as first parameter
calPlotData <- ntbt_calibration(testProbs, obs ~ lda + qda)
xyplot(calPlotData, auto.key = list(columns = 2))

## so it can be used easily in a pipeline.
testProbs %>%
  ntbt_calibration(obs ~ lda + qda) %>%
  xyplot(auto.key = list(columns = 2))


## ntbt_dummyVars
when <- data.frame(time = c("afternoon", "night", "afternoon",
                            "morning", "morning", "morning",
                            "morning", "afternoon", "afternoon"),
                   day = c("Mon", "Mon", "Mon",
                           "Wed", "Wed", "Fri",
                           "Sat", "Sat", "Fri"))

levels(when$time) <- list(morning="morning",
                          afternoon="afternoon",
                          night="night")
levels(when$day) <- list(Mon="Mon", Tue="Tue", Wed="Wed", Thu="Thu",
                         Fri="Fri", Sat="Sat", Sun="Sun")

## Original function to interface
mainEffects <- dummyVars(~ day + time, data = when)
mainEffects
predict(mainEffects, when[1:3,])

## The interface puts data as first parameter
mainEffects <- ntbt_dummyVars(when, ~ day + time)
mainEffects
predict(mainEffects, when[1:3,])

## so it can be used easily in a pipeline.
when %>%
  ntbt_dummyVars(~ day + time) %>%
  predict(when[1:3,])


## ntbt_icr: Independent Component Regression
## Not found example using formula interface, and I am
## completely ignorant to construct one.
data(BloodBrain)
```

```
BB <- list(bbbDescr, logBBB)

## Original function to interface
icr(bbbDescr, logBBB, n.comp = 5)

## The interface puts data as first parameter
ntbt_icr(BB, bbbDescr, logBBB, n.comp = 5)

## so it can be used easily in a pipeline.
BB %>%
  ntbt_icr(bbbDescr, logBBB, n.comp = 5)


## ntbt_knn3: k-Nearest Neighbour Classification
## Original function to interface
knn3(Species ~ ., iris)

## The interface puts data as first parameter
ntbt_knn3(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_knn3(Species ~ .)


## ntbt_lift: Lift Plot
set.seed(1)
simulated <- data.frame(obs = factor(rep(letters[1:2], each = 100)),
perfect = sort(runif(200), decreasing = TRUE),
random = runif(200))
## Original function to interface
lift1 <- lift(obs ~ random, data = simulated)
lift1
xyplot(lift1)

## The interface puts data as first parameter
lift1 <- ntbt_lift(simulated, obs ~ random)
lift1
xyplot(lift1)

## so it can be used easily in a pipeline.
simulated %>%
  ntbt_lift(obs ~ random) %>%
  xyplot()


## ntbt_pcaNNet: Neural Networks with a Principal Component Step
## Not found example using formula interface, and I am
## completely ignorant to construct one.
data(BloodBrain)
BB <- list(bbbDescr, logBBB)

## Original function to interface
```

```
pcaNNet(bbbDescr[, 1:10], logBBB, size = 5, linout = TRUE, trace = FALSE)

## The interface puts data as first parameter
ntbt_pcaNNet(BB, bbbDescr[, 1:10], logBBB, size = 5, linout = TRUE, trace = FALSE)

## so it can be used easily in a pipeline.
BB %>%
  ntbt_pcaNNet(bbbDescr[, 1:10], logBBB, size = 5, linout = TRUE, trace = FALSE)


## ntbt_sbf: Selection By Filtering (SBF)
## Not found example using formula interface, and I am
## completely ignorant to construct one.
data(BloodBrain)
BB <- list(bbbDescr, logBBB)

## Be prepared to wait...
## Original function to interface
sbf(bbbDescr, logBBB,
    sbfControl = sbfControl(functions = rfSBF,
                            verbose = FALSE,
                            method = "cv"))

## The interface puts data as first parameter
ntbt_sbf(BB, bbbDescr, logBBB,
         sbfControl = sbfControl(functions = rfSBF,
                                 verbose = FALSE,
                                 method = "cv"))

## so it can be used easily in a pipeline.
BB %>%
  ntbt_sbf(bbbDescr, logBBB,
           sbfControl = sbfControl(functions = rfSBF,
                                   verbose = FALSE,
                                   method = "cv"))


## ntbt_train: Fit Predictive Models over Different Tuning Parameters
library(mlbench)
data(BostonHousing)

## Original function to interface
train(medv ~ . + rm:lstat, data = BostonHousing, method = "lm")

## The interface puts data as first parameter
ntbt_train(BostonHousing, medv ~ . + rm:lstat, method = "lm")

## so it can be used easily in a pipeline.
BostonHousing %>%
  ntbt_train(medv ~ . + rm:lstat, method = "lm")

## End(Not run)
```

coin *Interfaces for coin package for data science pipelines.*

## Description

Interfaces to coin functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_ansari_test(data, ...)
ntbt_chisq_test(data, ...)
ntbt_cmh_test(data, ...)
ntbt_conover_test(data, ...)
ntbt_fisyat_test(data, ...)
ntbt_fligner_test(data, ...)
ntbt_friedman_test(data, ...)
ntbt_independence_test(data, ...)
ntbt_klotz_test(data, ...)
ntbt_koziol_test(data, ...)
ntbt_kruskal_test(data, ...)
ntbt_lbl_test(data, ...)
ntbt_logrank_test(data, ...)
ntbt_maxstat_test(data, ...)
ntbt_median_test(data, ...)
ntbt_mh_test(data, ...)
ntbt_mood_test(data, ...)
ntbt_normal_test(data, ...)
ntbt_oneway_test(data, ...)
ntbt_quade_test(data, ...)
ntbt_quadrant_test(data, ...)
ntbt_sign_test(data, ...)
ntbt_symmetry_test(data, ...)
ntbt_taha_test(data, ...)
ntbt_savage_test(data, ...)
ntbt_spearman_test(data, ...)
ntbt_wilcox_test(data, ...)
ntbt_wilcoxsign_test(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(coin)

## Tests of Independence in Two- or Three-Way Contingency Tables
## Please contribute better example
## Original function to interface
chisq_test(Plant ~ Type, data = CO2)
cmh_test(Plant ~ Type, data = CO2)
lbl_test(Plant ~ Type, data = CO2)

## The interface puts data as first parameter
ntbt_chisq_test(CO2, Plant ~ Type)
ntbt_cmh_test(CO2, Plant ~ Type)
ntbt_lbl_test(CO2, Plant ~ Type)

## so it can be used easily in a pipeline.
CO2 %>%
  ntbt_chisq_test(Plant ~ Type)
CO2 %>%
  ntbt_cmh_test(Plant ~ Type)
CO2 %>%
  ntbt_lbl_test(Plant ~ Type)


## Correlation Tests
## Original function to interface
## Asymptotic Spearman test
spearman_test(CONT ~ INTG, data = USJudgeRatings)
## Asymptotic Fisher-Yates test
fisyat_test(CONT ~ INTG, data = USJudgeRatings)
## Asymptotic quadrant test
quadrant_test(CONT ~ INTG, data = USJudgeRatings)
## Asymptotic Koziol-Nemec test
koziol_test(CONT ~ INTG, data = USJudgeRatings)

## The interface puts data as first parameter
## Asymptotic Spearman test
ntbt_spearman_test(USJudgeRatings, CONT ~ INTG)
## Asymptotic Fisher-Yates test
ntbt_fisyat_test(USJudgeRatings, CONT ~ INTG)
## Asymptotic quadrant test
```

```
ntbt_quadrant_test(USJudgeRatings, CONT ~ INTG)
## Asymptotic Koziol-Nemec test
ntbt_koziol_test(USJudgeRatings, CONT ~ INTG)

## so it can be used easily in a pipeline.
## Asymptotic Spearman test
USJudgeRatings %>%
  ntbt_spearman_test(CONT ~ INTG)
## Asymptotic Fisher-Yates test
USJudgeRatings %>%
  ntbt_fisyat_test(CONT ~ INTG)
## Asymptotic quadrant test
USJudgeRatings %>%
  ntbt_quadrant_test(CONT ~ INTG)
## Asymptotic Koziol-Nemec test
USJudgeRatings %>%
  ntbt_koziol_test(CONT ~ INTG)

## ntbt_independence_test: General Independence Test
## Original function to interface
independence_test(asat ~ group, data = asat, distribution = "exact",
                  alternative = "greater",
                  ytrafo = function(data)
                    trafo(data, numeric_trafo = normal_trafo),
                  xtrafo = function(data)
                      trafo(data, factor_trafo = function(x)
                          matrix(x == levels(x)[1], ncol = 1)))

## The interface puts data as first parameter
ntbt_independence_test(asat, asat ~ group, distribution = "exact",
                       alternative = "greater",
                       ytrafo = function(data)
                         trafo(data, numeric_trafo = normal_trafo),
                       xtrafo = function(data)
                         trafo(data, factor_trafo = function(x)
                           matrix(x == levels(x)[1], ncol = 1)))

## so it can be used easily in a pipeline.
asat %>%
  ntbt_independence_test(asat ~ group, distribution = "exact",
                         alternative = "greater",
                         ytrafo = function(data)
                           trafo(data, numeric_trafo = normal_trafo),
                         xtrafo = function(data)
                           trafo(data, factor_trafo = function(x)
                             matrix(x == levels(x)[1], ncol = 1)))


## Two- and K-Sample Location Tests
## Tritiated Water Diffusion Across Human Chorioamnion
## Hollander and Wolfe (1999, p. 110, Tab. 4.1)
diffusion <- data.frame(
    pd = c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46,
```

```
            1.15, 0.88, 0.90, 0.74, 1.21),
    age = factor(rep(c("At term", "12-26 Weeks"), c(10, 5)))
)
ex <- data.frame(
    y = c(3, 4, 8, 9, 1, 2, 5, 6, 7),
    x = factor(rep(c("no", "yes"), c(4, 5)))
)

## Original function to interface
kruskal_test(pd ~ age, data = diffusion, distribution = "exact")
median_test(y ~ x, data = ex, distribution = "exact")
normal_test(pd ~ age, data = diffusion, distribution = "exact", conf.int = TRUE)
oneway_test(pd ~ age, data = diffusion)
savage_test(pd ~ age, data = diffusion, distribution = "exact", conf.int = TRUE)
wilcox_test(pd ~ age, data = diffusion, distribution = "exact", conf.int = TRUE)

## The interface puts data as first parameter
ntbt_kruskal_test(diffusion, pd ~ age, distribution = "exact")
ntbt_median_test(ex, y ~ x, distribution = "exact")
ntbt_normal_test(diffusion, pd ~ age, distribution = "exact", conf.int = TRUE)
ntbt_oneway_test(diffusion, pd ~ age)
ntbt_savage_test(diffusion, pd ~ age, distribution = "exact", conf.int = TRUE)
ntbt_wilcox_test(diffusion, pd ~ age, distribution = "exact", conf.int = TRUE)

## so it can be used easily in a pipeline.
diffusion %>%
  ntbt_kruskal_test(pd ~ age, distribution = "exact")
ex %>%
  ntbt_median_test(y ~ x, distribution = "exact")
diffusion %>%
  ntbt_normal_test(pd ~ age, distribution = "exact", conf.int = TRUE)
diffusion %>%
  ntbt_oneway_test(pd ~ age)
diffusion %>%
  ntbt_savage_test(pd ~ age, distribution = "exact", conf.int = TRUE)
diffusion %>%
  ntbt_wilcox_test(pd ~ age, distribution = "exact", conf.int = TRUE)

performance <- matrix(
    c(794, 150,
       86, 570),
    nrow = 2, byrow = TRUE,
    dimnames = list(
         "First" = c("Approve", "Disprove"),
        "Second" = c("Approve", "Disprove")
    )
)

## ntbt_mh_test: Marginal Homogeneity Tests
## Effectiveness of different media for the growth of diphtheria
## Cochran (1950, Tab. 2)
cases <- c(4, 2, 3, 1, 59)
n <- sum(cases)
```

```
cochran <- data.frame(
    diphtheria = factor(
        unlist(rep(list(c(1, 1, 1, 1),
                        c(1, 1, 0, 1),
                        c(0, 1, 1, 1),
                        c(0, 1, 0, 1),
                        c(0, 0, 0, 0)),
                   cases))
    ),
    media = factor(rep(LETTERS[1:4], n)),
    case =  factor(rep(seq_len(n), each = 4))
)

## Original function to interface
mh_test(diphtheria ~ media | case, data = cochran)

## The interface puts data as first parameter
ntbt_mh_test(cochran, diphtheria ~ media | case)

## so it can be used easily in a pipeline.
cochran %>%
  ntbt_mh_test(diphtheria ~ media | case)

## ntbt_maxstat_test: Generalized Maximally Selected Statistics
## Original function to interface
maxstat_test(counts ~ coverstorey, data = treepipit)

## The interface puts data as first parameter
ntbt_maxstat_test(treepipit, counts ~ coverstorey)

## so it can be used easily in a pipeline.
treepipit %>%
  ntbt_maxstat_test(counts ~ coverstorey)


## Two- and K-Sample Scale Tests
## Serum Iron Determination Using Hyland Control Sera
## Hollander and Wolfe (1999, p. 147, Tab 5.1)
sid <- data.frame(
    serum = c(111, 107, 100, 99, 102, 106, 109, 108, 104, 99,
              101, 96, 97, 102, 107, 113, 116, 113, 110, 98,
              107, 108, 106, 98, 105, 103, 110, 105, 104,
              100, 96, 108, 103, 104, 114, 114, 113, 108, 106, 99),
    method = gl(2, 20, labels = c("Ramsay", "Jung-Parekh"))
)

## Original function to interface
ansari_test(serum ~ method, data = sid)
conover_test(serum ~ method, data = sid)
fligner_test(serum ~ method, data = sid)
klotz_test(serum ~ method, data = sid)
mood_test(serum ~ method, data = sid)
taha_test(serum ~ method, data = sid)
```

```
## The interface puts data as first parameter
ntbt_ansari_test(sid, serum ~ method)
ntbt_conover_test(sid, serum ~ method)
ntbt_fligner_test(sid, serum ~ method)
ntbt_klotz_test(sid, serum ~ method)
ntbt_mood_test(sid, serum ~ method)
ntbt_taha_test(sid, serum ~ method)

## so it can be used easily in a pipeline.
sid %>%
  ntbt_ansari_test(serum ~ method)
sid %>%
  ntbt_conover_test(serum ~ method)
sid %>%
  ntbt_fligner_test(serum ~ method)
sid %>%
  ntbt_klotz_test(serum ~ method)
sid %>%
  ntbt_mood_test(serum ~ method)
sid %>%
  ntbt_taha_test(serum ~ method)

## ntbt_logrank_test: Two- and K-Sample Tests for Censored Data
## Example data (Callaert, 2003, Tab.1)
callaert <- data.frame(
    time = c(1, 1, 5, 6, 6, 6, 6, 2, 2, 2, 3, 4, 4, 5, 5),
    group = factor(rep(0:1, c(7, 8)))
)
## Original function to interface
logrank_test(Surv(time) ~ group, data = callaert, distribution = "exact")

## The interface puts data as first parameter
ntbt_logrank_test(callaert, Surv(time) ~ group, distribution = "exact")

## so it can be used easily in a pipeline.
callaert %>%
  ntbt_logrank_test(Surv(time) ~ group, distribution = "exact")


## ntbt_symmetry_test: General Symmetry Test
## One-sided exact Fisher-Pitman test for paired observations
y1 <- c(1.83,  0.50,  1.62,  2.48, 1.68, 1.88, 1.55, 3.06, 1.30)
y2 <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)
dta <- data.frame(
    y = c(y1, y2),
    x = gl(2, length(y1)),
    block = factor(rep(seq_along(y1), 2))
)

## Original function to interface
symmetry_test(y ~ x | block, data = dta, distribution = "exact", alternative = "greater")
```

```
## The interface puts data as first parameter
ntbt_symmetry_test(dta, y ~ x | block, distribution = "exact", alternative = "greater")

## so it can be used easily in a pipeline.
dta %>%
  ntbt_symmetry_test(y ~ x | block, distribution = "exact", alternative = "greater")


## Symmetry Tests
## Data with explicit group and block information
dta <- data.frame(y = c(y1, y2), x = gl(2, length(y1)),
                  block = factor(rep(seq_along(y1), 2)))

## Original function to interface
## For two samples, the sign test is equivalent to the Friedman test...
sign_test(y ~ x | block, data = dta, distribution = "exact")
friedman_test(y ~ x | block, data = dta, distribution = "exact")
## ...and the signed-rank test is equivalent to the Quade test
wilcoxsign_test(y ~ x | block, data = dta, distribution = "exact")
quade_test(y ~ x | block, data = dta, distribution = "exact")

## The interface puts data as first parameter
ntbt_sign_test(dta, y ~ x | block, distribution = "exact")
ntbt_friedman_test(dta, y ~ x | block, distribution = "exact")
ntbt_wilcoxsign_test(dta, y ~ x | block, distribution = "exact")
ntbt_quade_test(dta, y ~ x | block, distribution = "exact")

## so it can be used easily in a pipeline.
dta %>%
  ntbt_sign_test(y ~ x | block, distribution = "exact")
dta %>%
  ntbt_friedman_test(y ~ x | block, distribution = "exact")
dta %>%
  ntbt_wilcoxsign_test(y ~ x | block, distribution = "exact")
dta %>%
  ntbt_quade_test(y ~ x | block, distribution = "exact")

## End(Not run)
```

---

`CORElearn` *Interfaces for CORElearn package for data science pipelines.*

---

## Description

Interfaces to `CORElearn` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_attrEval(data, ...)
ntbt_CoreModel(data, ...)
```

```
ntbt_discretize(data, ...)
ntbt_ordEval(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(COREElearn)

## ntbt_attrEval: Attribute evaluation
## Original function to interface
attrEval(Species ~ ., iris, estimator = "ReliefFexpRank", ReliefIterations = 30)

## The interface puts data as first parameter
ntbt_attrEval(iris, Species ~ ., estimator = "ReliefFexpRank", ReliefIterations = 30)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_attrEval(Species ~ ., estimator = "ReliefFexpRank", ReliefIterations = 30)

## ntbt_CoreModel: Build a classification or regression model
trainIdxs <- sample(x=nrow(iris), size=0.7*nrow(iris), replace=FALSE)
testIdxs <- c(1:nrow(iris))[-trainIdxs]

## Original function to interface
CoreModel(Species ~ ., iris[trainIdxs,], model = "rf",
          selectionEstimator = "MDL", minNodeWeightRF = 5,
          rfNoTrees = 100, maxThreads = 1)

## The interface puts data as first parameter
ntbt_CoreModel(iris[trainIdxs,], Species ~ ., model = "rf",
               selectionEstimator = "MDL", minNodeWeightRF = 5,
               rfNoTrees = 100, maxThreads = 1)
```

```
## so it can be used easily in a pipeline.
iris[trainIdxs,] %>%
  ntbt_CoreModel(Species ~ ., model = "rf",
                 selectionEstimator = "MDL", minNodeWeightRF = 5,
                 rfNoTrees = 100, maxThreads = 1)

## ntbt_discretize: Discretization of numeric attributes
## Original function to interface
discretize(Species ~ ., iris, method = "greedy", estimator = "ReliefFexpRank")

## The interface puts data as first parameter
ntbt_discretize(iris, Species ~ ., method = "greedy", estimator = "ReliefFexpRank")

## so it can be used easily in a pipeline.
iris %>%
  ntbt_discretize(Species ~ ., method = "greedy", estimator = "ReliefFexpRank")

## ntbt_ordEval: Evaluation of ordered attributes
dat <- ordDataGen(200)

## Original function to interface
ordEval(class ~ ., dat, ordEvalNoRandomNormalizers=100)

## The interface puts data as first parameter
ntbt_ordEval(dat, class ~ ., ordEvalNoRandomNormalizers=100)

## so it can be used easily in a pipeline.
dat %>%
  ntbt_ordEval(class ~ ., ordEvalNoRandomNormalizers=100)

## End(Not run)
```

---

`drc` *Interfaces for drc package for data science pipelines.*

---

## Description

Interfaces to `drc` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_drm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

### Details

Interfaces call their corresponding interfaced function.

### Value

Object returned by interfaced function.

### Author(s)

Roberto Bertolusso

### Examples

```
## Not run:
library(intubate)
library(magrittr)
library(drc)

## ntbt_drm: Fitting dose-response models
## Original function to interface
drm(rootl ~ conc, data = ryegrass, fct = W2.4())

## The interface puts data as first parameter
ntbt_drm(ryegrass, rootl ~ conc, fct = W2.4())

## so it can be used easily in a pipeline.
ryegrass %>%
  ntbt_drm(rootl ~ conc, fct = W2.4())

## End(Not run)
```

---

## e1071 *Interfaces for e1071 package for data science pipelines.*

---

### Description

Interfaces to `e1071` functions that can be used in a pipeline implemented by `magrittr`.

### Usage

```
ntbt_svm(data, ...)
```

### Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(e1071)

## Original function to interface
model <- svm(Species ~ ., iris)
summary(model)

## The interface reverses the order of data and formula
model <- ntbt_svm(iris, Species ~ .)
summary(model)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_svm(Species ~ .) %>%
  summary()

## End(Not run)
```

---

# earth *Interfaces for earth package for data science pipelines.*

---

## Description

Interfaces to `earth` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_earth(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(earth)

## ntbt_earth: Multivariate Adaptive Regression Splines
## Original function to interface
earth(Volume ~ ., data = trees)

## The interface puts data as first parameter
ntbt_earth(trees, Volume ~ .)

## so it can be used easily in a pipeline.
trees %>%
  ntbt_earth(Volume ~ .)

## End(Not run)
```

---

`EnvStats` *Interfaces for EnvStats package for data science pipelines.*

---

## Description

Interfaces to `EnvStats` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
#ntbt_calibrate(data, ...)   ## Not implement for 0.99.3 as needs an attach()
ntbt_gofTest(data, ...)
ntbt_gofGroupTest(data, ...)
ntbt_kendallSeasonalTrendTest(data, ...)
ntbt_kendallTrendTest(data, ...)
ntbt_stripChart(data, ...)
ntbt_summaryFull(data, ...)
ntbt_summaryStats(data, ...)
ntbt_varGroupTest(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(EnvStats)


## ntbt_calibrate: Fit a Calibration Line or Curve
#Cadmium <- EPA.97.cadmium.111.df$Cadmium
#Spike <- EPA.97.cadmium.111.df$Spike
data(EPA.97.cadmium.111.df)
## Original function to interface
calibrate(Cadmium ~ Spike, data = EPA.97.cadmium.111.df)

## NOTE: --check-as-cran does not like having an attach()
##       in the code, which is needed to interface
##       calibrate (at least for now). I will get back
##       to this after 0.99.3 is submitted. Please do not
##       report as bug (instead please find a solution to this!)

## The interface puts data as first parameter
#ntbt_calibrate(EPA.97.cadmium.111.df, Cadmium ~ Spike)

## so it can be used easily in a pipeline.
#EPA.97.cadmium.111.df %>%
#  ntbt_calibrate(Cadmium ~ Spike)


## ntbt_gofGroupTest: Goodness-of-Fit Test for a Specified Probability
##                    Distribution for Groups
## Original function to interface
gofGroupTest(Nickel.ppb ~ Well, data = EPA.09.Ex.10.1.nickel.df)

## The interface puts data as first parameter
```

```
ntbt_gofGroupTest(EPA.09.Ex.10.1.nickel.df, Nickel.ppb ~ Well)

## so it can be used easily in a pipeline.
EPA.09.Ex.10.1.nickel.df %>%
  ntbt_gofGroupTest(Nickel.ppb ~ Well)

## ntbt_gofTest: Goodness-of-Fit Test
## Original function to interface
gofTest(Nickel.ppb ~ 1, data = EPA.09.Ex.10.1.nickel.df)

## The interface puts data as first parameter
ntbt_gofTest(EPA.09.Ex.10.1.nickel.df, Nickel.ppb ~ 1)

## so it can be used easily in a pipeline.
EPA.09.Ex.10.1.nickel.df %>%
  ntbt_gofTest(Nickel.ppb ~ 1)


## ntbt_kendallSeasonalTrendTest: Nonparametric Test for Monotonic Trend Within
##                                Each Season Based on Kendall's Tau Statistic
## Original function to interface
kendallSeasonalTrendTest(Unadj.Conc ~ Month + Year, data = EPA.09.Ex.14.8.df)

## The interface puts data as first parameter
ntbt_kendallSeasonalTrendTest(EPA.09.Ex.14.8.df, Unadj.Conc ~ Month + Year)

## so it can be used easily in a pipeline.
EPA.09.Ex.14.8.df %>%
  ntbt_kendallSeasonalTrendTest(Unadj.Conc ~ Month + Year)


## ntbt_kendallTrendTest: Kendall's Nonparametric Test for Montonic Trend
## Original function to interface
kendallTrendTest(Sulfate.ppm ~ Sampling.Date, data = EPA.09.Ex.17.6.sulfate.df)

## The interface puts data as first parameter
ntbt_kendallTrendTest(EPA.09.Ex.17.6.sulfate.df, Sulfate.ppm ~ Sampling.Date)

## so it can be used easily in a pipeline.
EPA.09.Ex.17.6.sulfate.df %>%
  ntbt_kendallTrendTest(Sulfate.ppm ~ Sampling.Date)


## ntbt_stripChart: 1-D Scatter Plots with Confidence Intervals
## Original function to interface
stripChart(TcCB ~ Area, data = EPA.94b.tccb.df, col = c("red", "blue"),
           p.value = TRUE, ci.and.test = "nonparametric",
           ylab = "TcCB (ppb)")

## The interface puts data as first parameter
ntbt_stripChart(EPA.94b.tccb.df, TcCB ~ Area, col = c("red", "blue"),
                p.value = TRUE, ci.and.test = "nonparametric",
                ylab = "TcCBntbt_summaryFull (ppb)")
```

```
## so it can be used easily in a pipeline.
EPA.94b.tccb.df %>%
  ntbt_stripChart(TcCB ~ Area, col = c("red", "blue"),
                  p.value = TRUE, ci.and.test = "nonparametric",
                  ylab = "TcCB (ppb)")


## ntbt_summaryFull: Full Complement of Summary Statistics
## Original function to interface
summaryFull(TcCB ~ Area, data = EPA.94b.tccb.df)

## The interface puts data as first parameter
ntbt_summaryFull(EPA.94b.tccb.df, TcCB ~ Area)

## so it can be used easily in a pipeline.
EPA.94b.tccb.df %>%
  ntbt_summaryFull(TcCB ~ Area)


## ntbt_summaryStats: Summary Statistics
## Original function to interface
summaryStats(log10(TcCB) ~ Area, data = EPA.94b.tccb.df)

## The interface puts data as first parameter
ntbt_summaryStats(EPA.94b.tccb.df, log10(TcCB) ~ Area)

## so it can be used easily in a pipeline.
EPA.94b.tccb.df %>%
  ntbt_summaryStats(log10(TcCB) ~ Area)


## ntbt_varGroupTest: Test for Homogeneity of Variance Among Two or More Groups
## Original function to interface
varGroupTest(Arsenic.ppb ~ Well, data = EPA.09.Ex.11.1.arsenic.df)

## The interface puts data as first parameter
ntbt_varGroupTest(EPA.09.Ex.11.1.arsenic.df, Arsenic.ppb ~ Well)

## so it can be used easily in a pipeline.
EPA.09.Ex.11.1.arsenic.df %>%
  ntbt_varGroupTest(Arsenic.ppb ~ Well)

## End(Not run)
```

---

`experimental` *Experimental features.*

---

## Description

Extensions (experimental) to pipelines. Please see vignette for examples of use.

## Usage

```
as_intuBag(object)
clear_intuEnv()
intuBag(...)
is_intuBag(object)
intuEnv(...)
set_intuEnv(envir)
```

## Arguments

| | |
|---|---|
| object | Object. |
| envir | Environment. |
| ... | Objects. |

## Value

Return value of functions.

## Author(s)

Roberto Bertolusso

## Examples

```
## Please see vignette
```

---

# fGarch *Interfaces for fGarch package for data science pipelines.*

---

## Description

Interfaces to fGarch functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_garchFit(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(fGarch)


## ntbt_garchFit: Univariate GARCH Time Series Fitting
N <- 200
x.vec <- as.vector(garchSim(garchSpec(rseed = 1985), n = N)[,1])

## Original function to interface
garchFit(~ garch(1,1), data = x.vec, trace = FALSE)

## The interface puts data as first parameter
ntbt_garchFit(data = x.vec, ~ garch(1,1), trace = FALSE)

## so it can be used easily in a pipeline.
x.vec %>%
  ntbt_garchFit(~ garch(1,1), trace = FALSE)

## End(Not run)
```

---

`flexmix` *Interfaces for flexmix package for data science pipelines.*

---

## Description

Interfaces to `flexmix` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_flexmix(data, ...)
ntbt_initFlexmix(data, ...)
ntbt_stepFlexmix(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(flexmix)


## ntbt_flexmix: Flexible Mixture Modeling
data("NPreg", package = "flexmix")

## Original function to interface
set.seed(1)
ex1 <- flexmix(yn ~ x + I(x^2), data = NPreg, k = 2,
               control = list(verb = 5, iter = 100))
plot(ex1)

## The interface puts data as first parameter
set.seed(1)
ex1 <- ntbt_flexmix(NPreg, yn ~ x + I(x^2), k = 2,
                    control = list(verb = 5, iter = 100))
plot(ex1)

## so it can be used easily in a pipeline.
set.seed(1)
NPreg %>%
  ntbt_flexmix(yn ~ x + I(x^2), k = 2,
               control = list(verb = 5, iter = 100)) %>%
  plot()

## NOTE: it seems we could also add initFlexmix and stepFlexmix,
##       that call flexmix repeteadly. I will not include examples
##       for those for now.

## End(Not run)
```

`forecast` *Interfaces for forecast package for data science pipelines.*

## Description

Interfaces to `forecast` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_tslm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(forecast)

## ntbt_tslm: Fit a linear model with time series components
dta <- data.frame(y <- ts(rnorm(120,0,3) + 1:120 + 20*sin(2*pi*(1:120)/12), frequency=12))

## Original function to interface
tslm(y ~ trend + season, data = dta)

## The interface puts data as first parameter
ntbt_tslm(dta, y ~ trend + season)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_tslm(y ~ trend + season)

## End(Not run)
```

`frontier` *Interfaces for frontier package for data science pipelines.*

## Description

Interfaces to `frontier` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_sfa(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(frontier)


## ntbt_sfa: Stochastic Frontier Analysis
data(front41Data)

## Original function to interface
sfa(log(output) ~ log(capital) + log(labour), data = front41Data)

## The interface puts data as first parameter
ntbt_sfa(front41Data, log(output) ~ log(capital) + log(labour))

## so it can be used easily in a pipeline.
front41Data %>%
  ntbt_sfa(log(output) ~ log(capital) + log(labour))

## End(Not run)
```

gam *Interfaces for gam package for data science pipelines.*

## Description

Interfaces to gam functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_gam(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## See Also

gam

## Examples

```
## Not run:
library(intubate)
library(gam)
data(kyphosis)

## Original function to interface
fit <- gam(Kyphosis ~ s(Age,4) + Number, family = binomial, data = kyphosis,
    trace = TRUE)
summary(fit)
fit <- gam(Kyphosis ~ poly(Age,2) + s(Start), data = kyphosis,
           family = binomial, subset = Number > 2)
summary(fit)
fit <- gam(Ozone^(1/3) ~ lo(Solar.R) + lo(Wind, Temp),
           data = airquality, na = na.gam.replace)
summary(fit)
```

```
## The interface reverses the order of data and formula
fit <- ntbt_gam(kyphosis, Kyphosis ~ s(Age,4) + Number,
                family = binomial, trace = TRUE)
summary(fit)
fit <- ntbt_gam(data = kyphosis, Kyphosis ~ poly(Age,2) + s(Start),
                family = binomial, subset = Number > 2)
summary(fit)
fit <- ntbt_gam(data = airquality, Ozone^(1/3) ~ lo(Solar.R) + lo(Wind, Temp),
                na = na.gam.replace)
summary(fit)

## so it can be used easily in a pipeline.
library(magrittr)
kyphosis %>%
  ntbt_gam(Kyphosis ~ s(Age,4) + Number,
          family = binomial, trace = TRUE) %>%
  summary()

kyphosis %>%
  ntbt_gam(Kyphosis ~ poly(Age,2) + s(Start),
           family = binomial, subset = Number > 2) %>%
  summary()

airquality %>%
  ntbt_gam(Ozone^(1/3) ~ lo(Solar.R) + lo(Wind, Temp),
           na = na.gam.replace) %>%
  summary()

## End(Not run)
```

---

gbm *Interfaces for gbm package for data science pipelines.*

---

## Description

Interfaces to gbm functions that can be used in a pipeline implemented by magrittr.

## Arguments

| | |
|---|---|
| data | data frame, tibble, list (or object coercible by as.data.frame to a data frame) containing the variables in the model. |
| ... | Other arguments passed to gbm. |

## Details

ntbt_gbm calls gbm.

## Value

Object returned by gbm function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(gbm)

# A least squares regression example # create some data
N <- 1000
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- ordered(sample(letters[1:4],N,replace=TRUE),levels=letters[4:1])
X4 <- factor(sample(letters[1:6],N,replace=TRUE))
X5 <- factor(sample(letters[1:3],N,replace=TRUE))
X6 <- 3*runif(N)
mu <- c(-1,0,1,2)[as.numeric(X3)]

SNR <- 10 # signal-to-noise ratio
Y <- X1**1.5 + 2 * (X2**.5) + mu
sigma <- sqrt(var(Y)/SNR)
Y <- Y + rnorm(N,0,sigma)

# introduce some missing values
X1[sample(1:N,size=500)] <- NA
X4[sample(1:N,size=300)] <- NA

data <- data.frame(Y = Y, X1 = X1, X2 = X2, X3 = X3,
                   X4 = X4, X5 = X5, X6 = X6)

## Original function to interface
gbm1 <-
  gbm(Y~X1+X2+X3+X4+X5+X6,         # formula
      data=data,                   # dataset
      var.monotone=c(0,0,0,0,0,0), # -1: monotone decrease,
      # +1: monotone increase,
      #  0: no monotone restrictions
      distribution="gaussian",     # see the help for other choices
      n.trees=1000,                # number of trees
      shrinkage=0.05,              # shrinkage or learning rate,
      # 0.001 to 0.1 usually work
      interaction.depth=3,         # 1: additive model, 2: two-way interactions, etc.
      bag.fraction = 0.5,          # subsampling fraction, 0.5 is probably best
      train.fraction = 0.5,        # fraction of data for training,
      # first train.fraction*N used for training
      n.minobsinnode = 10,         # minimum total weight needed in each node
```

```
      cv.folds = 3,                # do 3-fold cross-validation
      keep.data=TRUE,              # keep a copy of the dataset with the object
      verbose=FALSE,               # don't print out progress
      n.cores=1)                   # use only a single core (detecting #cores is
# error-prone, so avoided here)
summary(gbm1)

## The interface reverses the order of data and formula
gbm1 <-
  ntbt_gbm(data=data,                   # dataset
           Y~X1+X2+X3+X4+X5+X6,         # formula
           var.monotone=c(0,0,0,0,0,0), # -1: monotone decrease,
           # +1: monotone increase,
           #  0: no monotone restrictions
           distribution="gaussian",     # see the help for other choices
           n.trees=1000,                # number of trees
           shrinkage=0.05,              # shrinkage or learning rate,
           # 0.001 to 0.1 usually work
          interaction.depth=3,        # 1: additive model, 2: two-way interactions, etc.
           bag.fraction = 0.5,          # subsampling fraction, 0.5 is probably best
           train.fraction = 0.5,        # fraction of data for training,
           # first train.fraction*N used for training
           n.minobsinnode = 10,         # minimum total weight needed in each node
           cv.folds = 3,                # do 3-fold cross-validation
           keep.data=TRUE,              # keep a copy of the dataset with the object
           verbose=FALSE,               # don't print out progress
           n.cores=1)                   # use only a single core (detecting #cores is
# error-prone, so avoided here)

## so it can be used easily in a pipeline.
data %>%
  ntbt_gbm(Y~X1+X2+X3+X4+X5+X6,         # formula
           var.monotone=c(0,0,0,0,0,0), # -1: monotone decrease,
           # +1: monotone increase,
           #  0: no monotone restrictions
           distribution="gaussian",     # see the help for other choices
           n.trees=1000,                # number of trees
           shrinkage=0.05,              # shrinkage or learning rate,
           # 0.001 to 0.1 usually work
          interaction.depth=3,        # 1: additive model, 2: two-way interactions, etc.
           bag.fraction = 0.5,          # subsampling fraction, 0.5 is probably best
           train.fraction = 0.5,        # fraction of data for training,
           # first train.fraction*N used for training
           n.minobsinnode = 10,         # minimum total weight needed in each node
           cv.folds = 3,                # do 3-fold cross-validation
           keep.data=TRUE,              # keep a copy of the dataset with the object
           verbose=FALSE,               # don't print out progress
           n.cores=1) %>%
  summary()

## End(Not run)
```

gee *Interfaces for gee package for data science pipelines.*

## Description

Interfaces to gee functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_gee(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(gee)

## ntbt_gee: solve a Generalized Estimation Equation Model
data(warpbreaks)

## Original function to interface
gee(breaks ~ tension, id=wool, data=warpbreaks, corstr="exchangeable")

## The interface puts data as first parameter
ntbt_gee(warpbreaks, breaks ~ tension, id=wool, corstr="exchangeable")

## so it can be used easily in a pipeline.
warpbreaks %>%
  ntbt_gee(breaks ~ tension, id=wool, corstr="exchangeable")

## End(Not run)
```

glmnet *Interfaces for glmnet package for data science pipelines.*

## Description

Interfaces to glmnet functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_glmnet(data, ...)
ntbt_cv.glmnet(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(glmnet)


## NOTE: glmnet package does not implement formula interface. As I need it
##       for teaching purposes, this is the first pure non-formula library
##       included, as a proof of concept that not only the functions with
##       formula + data variant can be successfully interfaced to use in
##       a pipeline.


library(ISLR)
data("Hitters")
Hitters <- na.omit(Hitters)

dta <- list(x = model.matrix(Salary ~ ., Hitters)[, -1],  ## Remove intercept
```

```
            y = model.frame(Salary ~ ., Hitters)[, 1])
grid <- 10^seq(10, -2, length = 100)

## ntbt_glmnet: fit a GLM with lasso or elasticnet regularization

## Original function to interface
attach(dta)
## Ridge Regression
ridge <- glmnet(x, y, alpha = 0, lambda = grid)
plot(ridge)

## The Lasso
lasso <- glmnet(x, y, alpha = 1, lambda = grid)
plot(lasso)
detach()

## The interface puts data as first parameter
## Ridge Regression
ridge <- ntbt_glmnet(dta, x, y, alpha = 0, lambda = grid)
plot(ridge)

## The Lasso
lasso <- ntbt_glmnet(dta, x, y, alpha = 1, lambda = grid)
plot(lasso)

## so it can be used easily in a pipeline.
## Ridge Regression
dta %>%
  ntbt_glmnet(x, y, alpha = 0, lambda = grid) %>%
  plot()

## The Lasso
dta %>%
  ntbt_glmnet(x, y, alpha = 1, lambda = grid) %>%
  plot()


## ntbt_cv.glmnet: Cross-validation for glmnet

## Original function to interface
attach(dta)
## Ridge Regression
set.seed(1)
cv.ridge <- cv.glmnet(x, y, alpha = 0)
plot(cv.ridge)

## The Lasso
cv.lasso <- cv.glmnet(x, y, alpha = 1)
plot(cv.lasso)
detach()

## The interface puts data as first parameter
## Ridge Regression
```

```
set.seed(1)
cv.ridge <- ntbt_cv.glmnet(dta, x, y, alpha = 0)
plot(cv.ridge)

## The Lasso
cv.lasso <- ntbt_cv.glmnet(dta, x, y, alpha = 1)
plot(cv.lasso)

## so it can be used easily in a pipeline.
## Ridge Regression
set.seed(1)
dta %>%
  ntbt_cv.glmnet(x, y, alpha = 0) %>%
  plot()

## The Lasso
dta %>%
  ntbt_cv.glmnet(x, y, alpha = 1) %>%
  plot()

## End(Not run)
```

---

`glmx` *Interfaces for glmx package for data science pipelines.*

---

## Description

Interfaces to `glmx` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_glmx(data, ...)
ntbt_hetglm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(glmx)

## ntbt_glmx: Generalized Linear Models with Extra Parameters
set.seed(1)
d <- data.frame(x = runif(200, -1, 1))
d$y <- rnbinom(200, mu = exp(0 + 3 * d$x), size = 1)
require("MASS")

## Original function to interface
glmx(y ~ x, data = d, family = negative.binomial, xlink = "log", xstart = 0)

## The interface puts data as first parameter
ntbt_glmx(d, y ~ x, family = negative.binomial, xlink = "log", xstart = 0)

## so it can be used easily in a pipeline.
d %>%
  ntbt_glmx(y ~ x, family = negative.binomial, xlink = "log", xstart = 0)


## ntbt_hetglm: Heteroskedastic Binary Response GLMs
n <- 200
x <- rnorm(n)
ystar <- 1 + x +  rnorm(n, sd = exp(x))
y  <- factor(ystar > 0)
dta <- data.frame(x, y)

## Original function to interface
hetglm(y ~ x | 1, data = dta)

## The interface puts data as first parameter
ntbt_hetglm(dta, y ~ x | 1)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_hetglm(y ~ x | 1)

## End(Not run)
```

---

gmnl *Interfaces for gmnl package for data science pipelines.*

---

## Description

Interfaces to gmnl functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_gmnl(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(gmnl)


## ntbt_gmnl: Estimate Multinomial Logit Models with Observed and
##             Unobserved Individual Heterogeneity
data("TravelMode", package = "AER")
library(mlogit)
TM <- mlogit.data(TravelMode, choice = "choice", shape = "long",
                  alt.levels = c("air", "train", "bus", "car"), chid.var = "individual")

## Original function to interface
gmnl(choice ~ wait + vcost + travel + gcost| 1, data = TM,
     model = "smnl", R = 100,
     notscale = c(1, 1, 1, rep(0, 4)))

## The interface puts data as first parameter
ntbt_gmnl(TM, choice ~ wait + vcost + travel + gcost| 1,
          model = "smnl", R = 100,
          notscale = c(1, 1, 1, rep(0, 4)))

## so it can be used easily in a pipeline.
TM %>%
  ntbt_gmnl(choice ~ wait + vcost + travel + gcost| 1,
            model = "smnl", R = 100,
            notscale = c(1, 1, 1, rep(0, 4)))
```

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

---

gplots *Interfaces for gplots package for data science pipelines.*

---

## Description

Interfaces to gplots functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_bandplot(data, ...)
ntbt_lowess(data, ...)
ntbt_overplot(data, ...)
ntbt_plotmeans(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(gplots)


## ntbt_bandplot: Plot x-y Points with Locally Smoothed Mean and Standard Deviation
x <- 1:1000
y <- rnorm(1000, mean=1, sd=1 + x/1000 )
dta <- data.frame(x, y)
rm(x, y)

## Original function to interface
bandplot(y ~ x, data = dta)
```

```
## The interface puts data as first parameter
ntbt_bandplot(dta, y ~ x)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_bandplot(y ~ x)


## ntbt_lowess: Scatter Plot Smoothing
## Original function to interface
lowess(dist ~ speed, data = cars)

## The interface puts data as first parameter
ntbt_lowess(cars, dist ~ speed)

## so it can be used easily in a pipeline.
cars %>%
  ntbt_lowess(dist ~ speed)

cars %>%
  ntbt_plot(dist ~ speed, main="lowess(cars)") %>%
  ntbt_lowess(dist ~ speed) %>%
  lines(col=2, lty=2)


## ntbt_overplot: Plot multiple variables on the same region,
##                with appropriate axes
data(rtPCR)

## Original function to interface
overplot(RQ ~ Conc..ug.ml. | Test.Substance,
         data = rtPCR,
         subset = Detector == "ProbeType 1" & Conc..ug.ml. > 0,
         same.scale = TRUE,
         log="xy",
         f=3/4,
         main="Detector=ProbeType 1",
         xlab="Concentration (ug/ml)",
         ylab="Relative Gene Quantification"
)## Original function to interface

## The interface puts data as first parameter
ntbt_overplot(rtPCR,
              RQ ~ Conc..ug.ml. | Test.Substance,
              subset = Detector == "ProbeType 1" & Conc..ug.ml. > 0,
              same.scale = TRUE,
              log="xy",
              f=3/4,
              main="Detector=ProbeType 1",
              xlab="Concentration (ug/ml)",
              ylab="Relative Gene Quantification"
)## Original function to interface
```

```
## so it can be used easily in a pipeline.
rtPCR %>%
  ntbt_overplot(RQ ~ Conc..ug.ml. | Test.Substance,
                subset = Detector == "ProbeType 1" & Conc..ug.ml. > 0,
                same.scale = TRUE,
                log="xy",
                f=3/4,
                main="Detector=ProbeType 1",
                xlab="Concentration (ug/ml)",
                ylab="Relative Gene Quantification"
  )## Original function to interface


## ntbt_plotmeans: Plot Group Means and Confidence Intervals
data(state)
dta <- data.frame(state.abb, state.region)

## Original function to interface
plotmeans(state.area ~ state.region, data = dta)

## The interface puts data as first parameter
ntbt_plotmeans(dta, state.area ~ state.region)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_plotmeans(state.area ~ state.region)

## End(Not run)
```

---

graphics *Interfaces for graphics package for data science pipelines.*

---

## Description

Interfaces to graphics functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_boxplot(data, ...)
ntbt_cdplot(data, ...)
ntbt_coplot(data, ...)
ntbt_mosaicplot(data, ...)
ntbt_pairs(data, ...)
ntbt_plot(data, ...)
ntbt_spineplot(data, ...)
ntbt_sunflowerplot(data, ...)
ntbt_stripchart(data, ...)
ntbt_text(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)

## boxplot
## Original function to interface
boxplot(count ~ spray, data = InsectSprays, col = "lightgray")

boxplot(len ~ dose, data = ToothGrowth,
        boxwex = 0.25, at = 1:3 - 0.2,
        subset = supp == "VC", col = "yellow",
        main = "Guinea Pigs' Tooth Growth",
        xlab = "Vitamin C dose mg",
        ylab = "tooth length",
        xlim = c(0.5, 3.5), ylim = c(0, 35), yaxs = "i")
boxplot(len ~ dose, data = ToothGrowth, add = TRUE,
        boxwex = 0.25, at = 1:3 + 0.2,
        subset = supp == "OJ", col = "orange")
legend(2, 9, c("Ascorbic acid", "Orange juice"),
       fill = c("yellow", "orange"))

## The interface reverses the order of data and formula
ntbt_boxplot(data = InsectSprays, count ~ spray, col = "lightgray")

ntbt_boxplot(data = ToothGrowth, len ~ dose,
             boxwex = 0.25, at = 1:3 - 0.2,
             subset = supp == "VC", col = "yellow",
             main = "Guinea Pigs' Tooth Growth",
             xlab = "Vitamin C dose mg",
             ylab = "tooth length",
             xlim = c(0.5, 3.5), ylim = c(0, 35), yaxs = "i")
ntbt_boxplot(data = ToothGrowth, len ~ dose,
             add = TRUE,
```

```
             boxwex = 0.25, at = 1:3 + 0.2,
             subset = supp == "OJ", col = "orange")
legend(2, 9, c("Ascorbic acid", "Orange juice"),
       fill = c("yellow", "orange"))

## so it can be used easily in a pipeline.
InsectSprays %>%
  ntbt_boxplot(count ~ spray, col = "lightgray")

ToothGrowth %T>%  ## Note the tee operator.
  ntbt_boxplot(len ~ dose,
               boxwex = 0.25, at = 1:3 - 0.2,
               subset = supp == "VC", col = "yellow",
               main = "Guinea Pigs' Tooth Growth",
               xlab = "Vitamin C dose mg",
               ylab = "tooth length",
               xlim = c(0.5, 3.5), ylim = c(0, 35),
               yaxs = "i") %>%
  ntbt_boxplot(len ~ dose,
               add = TRUE,
               boxwex = 0.25, at = 1:3 + 0.2,
               subset = supp == "OJ", col = "orange")
legend(2, 9, c("Ascorbic acid", "Orange juice"),
       fill = c("yellow", "orange"))

## cdplot
## NASA space shuttle o-ring failures
oring <- data.frame(
  fail = factor(c(2, 2, 2, 2, 1, 1, 1, 1,
                  1, 1, 2, 1, 2, 1, 1, 1,
                  1, 2, 1, 1, 1, 1, 1),
                levels = 1:2, labels = c("no", "yes")),
  temperature = c(53, 57, 58, 63, 66, 67, 67, 67,
                  68, 69, 70, 70, 70, 70, 72, 73,
                  75, 75, 76, 76, 78, 79, 81))

## Original function to interface
cdplot(fail ~ temperature, oring)
cdplot(fail ~ temperature, oring, bw = 2)
cdplot(fail ~ temperature, oring, bw = "SJ")

## The interface reverses the order of data and formula
ntbt_cdplot(oring, fail ~ temperature)
ntbt_cdplot(oring, fail ~ temperature, bw = 2)
ntbt_cdplot(oring, fail ~ temperature, bw = "SJ")

## so it can be used easily in a pipeline.
oring %>%
  ntbt_cdplot(fail ~ temperature)
oring %>%
  ntbt_cdplot(fail ~ temperature, bw = 2)
oring %>%
  ntbt_cdplot(fail ~ temperature, bw = "SJ")
```

```
## coplot
Index <- seq(length = nrow(warpbreaks))

## Original function to interface
coplot(breaks ~ Index | wool * tension, data = warpbreaks,
       show.given = 0:1)
coplot(breaks ~ Index | wool * tension, data = warpbreaks,
       col = "red", bg = "pink", pch = 21,
       bar.bg = c(fac = "light blue"))

## The interface reverses the order of data and formula
ntbt_coplot(data = warpbreaks, breaks ~ Index | wool * tension,
            show.given = 0:1)
ntbt_coplot(data = warpbreaks, breaks ~ Index | wool * tension,
            col = "red", bg = "pink", pch = 21,
            bar.bg = c(fac = "light blue"))

## so it can be used easily in a pipeline.
warpbreaks %T>%  ## Note the tee operator.
  ntbt_coplot(breaks ~ Index | wool * tension,
              show.given = 0:1) %>%
  ntbt_coplot(breaks ~ Index | wool * tension,
              col = "red", bg = "pink", pch = 21,
              bar.bg = c(fac = "light blue"))

## mosaicplot
## Original function to interface
mosaicplot(~ Sex + Age + Survived, data = Titanic, color = TRUE)

## The interface reverses the order of data and formula
ntbt_mosaicplot(data = Titanic, ~ Sex + Age + Survived, color = TRUE)

## so it can be used easily in a pipeline.
Titanic %>%
  ntbt_mosaicplot(~ Sex + Age + Survived, color = TRUE)

## pairs
## Original function to interface
pairs(~ Fertility + Education + Catholic, data = swiss,
      subset = Education < 20,
      main = "Swiss data, Education < 20")

## The interface reverses the order of data and formula
ntbt_pairs(data = swiss, ~ Fertility + Education + Catholic,
           subset = Education < 20,
           main = "Swiss data, Education < 20")

## so it can be used easily in a pipeline.
swiss %>%
  ntbt_pairs(~ Fertility + Education + Catholic,
             subset = Education < 20,
             main = "Swiss data, Education < 20")
```

```
## plot
## Original function to interface
plot(Ozone ~ Wind, data = airquality, pch = as.character(Month))

## The interface reverses the order of data and formula
ntbt_plot(data = airquality, Ozone ~ Wind, pch = as.character(Month))

## so it can be used easily in a pipeline.
airquality %>%
  ntbt_plot(Ozone ~ Wind, pch = as.character(Month))

op <- par(mfrow = c(2,1))
airquality %>%  ## Note: we are *not* using the tee operator.
  ntbt_plot(Ozone ~ Wind, pch = as.character(Month)) %>%
  ntbt_plot(Ozone ~ Wind, pch = as.character(Month),
            subset = Month != 7) %>%
  head()          ## Yes! We still have the data to do what we want!
par(op)

## text.formula() can be very natural:
within(warpbreaks, {
    time <- seq_along(breaks)
    W.T <- wool:tension
}) %>%
  ntbt_plot(breaks ~ time, type = "b") %>%
  ntbt_text(breaks ~ time, label = W.T, col = 1 + as.integer(wool))

## splineplot
## NASA space shuttle o-ring failures
oring <- data.frame(
  fail = factor(c(2, 2, 2, 2, 1, 1, 1, 1,
                  1, 1, 2, 1, 2, 1, 1, 1,
                  1, 2, 1, 1, 1, 1, 1),
                levels = 1:2, labels = c("no", "yes")),
  temperature = c(53, 57, 58, 63, 66, 67, 67, 67,
                  68, 69, 70, 70, 70, 70, 72, 73,
                  75, 75, 76, 76, 78, 79, 81))

## Original function to interface
spineplot(fail ~ temperature, oring)
spineplot(fail ~ temperature, oring, breaks = 3)

## The interface reverses the order of data and formula
ntbt_spineplot(oring, fail ~ temperature)
ntbt_spineplot(oring, fail ~ temperature, breaks = 3)

## so it can be used easily in a pipeline.
oring %>%
  ntbt_spineplot(fail ~ temperature)
oring %>%
  ntbt_spineplot(fail ~ temperature, breaks = 3)
```

```
## sunflowerplot
## Original function to interface
sunflowerplot(Petal.Width ~ Petal.Length, data = iris,
              cex = .2, cex.fact = 1, size = .035,
              seg.lwd = .8)

## The interface reverses the order of data and formula
ntbt_sunflowerplot(data = iris, Petal.Width ~ Petal.Length,
                   cex = .2, cex.fact = 1, size = .035,
                   seg.lwd = .8)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_sunflowerplot(Petal.Width ~ Petal.Length,
                     cex = .2, cex.fact = 1, size = .035,
                     seg.lwd = .8)

## stripchart
## Original function to interface
stripchart(decrease ~ treatment, data = OrchardSprays,
           main = "stripchart(OrchardSprays)",
           vertical = TRUE, log = "y")

## The interface reverses the order of data and formula
ntbt_stripchart(data = OrchardSprays, decrease ~ treatment,
                main = "stripchart(OrchardSprays)",
                vertical = TRUE, log = "y")

## so it can be used easily in a pipeline.
OrchardSprays %>%
  ntbt_stripchart(decrease ~ treatment,
                  main = "stripchart(OrchardSprays)",
                  vertical = TRUE, log = "y")

## text
data <- within(warpbreaks, {
    time <- seq_along(breaks)
    W.T <- wool:tension
})

## Original function to interface
plot(breaks ~ time, data, type = "b")
text(breaks ~ time, data, label = W.T, col = 1 + as.integer(wool))

## The interface reverses the order of data and formula
ntbt_plot(data, breaks ~ time, type = "b")
ntbt_text(data, breaks ~ time, label = W.T, col = 1 + as.integer(wool))

## so it can be used easily in a pipeline.
data %>%
  ntbt_plot(breaks ~ time, type = "b") %>%
  ntbt_text(breaks ~ time, label = W.T, col = 1 + as.integer(wool)) %>%
  head()          ## Yes! We still have the data to do what we want!
```

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

---

gss *Interfaces for gss package for data science pipelines.*

---

## Description

Interfaces to gss functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_gssanova(data, ...)
ntbt_gssanova0(data, ...)
ntbt_gssanova1(data, ...)
ntbt_ssanova(data, ...)
ntbt_ssanova0(data, ...)
ntbt_ssanova9(data, ...)
ntbt_sscden(data, ...)
ntbt_sscden1(data, ...)
ntbt_sscox(data, ...)
ntbt_ssden(data, ...)
ntbt_ssden1(data, ...)
ntbt_sshzd(data, ...)
ntbt_ssllrm(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(gss)


## ntbt_gssanova: Fitting Smoothing Spline ANOVA Models with Non-Gaussian Responses
data(bacteriuria)

## Original function to interface
gssanova(infect ~ trt + time, family="binomial", data = bacteriuria,
         id.basis = (1:820)[bacteriuria$id %in% c(3,38)], random = ~ 1 | id)
gssanova0(infect ~ trt + time, family="binomial", data = bacteriuria)
gssanova1(infect ~ trt + time, family="binomial", data = bacteriuria,
         id.basis = (1:820)[bacteriuria$id %in% c(3,38)], random = ~ 1 | id)

## The interface puts data as first parameter
ntbt_gssanova(bacteriuria, infect ~ trt + time, family="binomial",
              id.basis = (1:820)[bacteriuria$id %in% c(3,38)], random = ~ 1 | id)
ntbt_gssanova0(bacteriuria, infect ~ trt + time, family="binomial")
ntbt_gssanova1(bacteriuria, infect ~ trt + time, family="binomial",
               id.basis = (1:820)[bacteriuria$id %in% c(3,38)], random = ~ 1 | id)

## so it can be used easily in a pipeline.
bacteriuria %>%
  ntbt_gssanova(infect ~ trt + time, family="binomial",
                id.basis = (1:820)[bacteriuria$id %in% c(3,38)], random = ~ 1 | id)
bacteriuria %>%
  ntbt_gssanova0(infect ~ trt + time, family="binomial")
bacteriuria %>%
  ntbt_gssanova1(infect ~ trt + time, family="binomial",
                 id.basis = (1:820)[bacteriuria$id %in% c(3,38)], random = ~ 1 | id)


## ntbt_ssanova: Fitting Smoothing Spline ANOVA Models
data(nox)

## Original function to interface
ssanova(log10(nox) ~ comp*equi, data = nox)
ssanova0(log10(nox) ~ comp*equi, data = nox)

## The interface puts data as first parameter
ntbt_ssanova(nox, log10(nox) ~ comp*equi)
ntbt_ssanova0(nox, log10(nox) ~ comp*equi)

## so it can be used easily in a pipeline.
nox %>%
  ntbt_ssanova(log10(nox) ~ comp*equi)
nox %>%
  ntbt_ssanova0(log10(nox) ~ comp*equi)
```

```
## ntbt_ssanova9: Fitting Smoothing Spline ANOVA Models with Correlated Data
x <- runif(100); y <- 5 + 3*sin(2*pi*x) + rnorm(x)
dta <- data.frame(x, y)

## Original function to interface
ssanova9(y ~ x, data = dta, cov = list("arma", c(1, 0)))

## The interface puts data as first parameter
ntbt_ssanova9(dta, y ~ x, cov = list("arma", c(1, 0)))

## so it can be used easily in a pipeline.
dta %>%
  ntbt_ssanova9(y ~ x, cov = list("arma", c(1, 0)))


## ntbt_sscden: Estimating Conditional Probability Density Using Smoothing Splines
data(penny)

## Original function to interface
set.seed(5732)
sscden(~ year*mil, ~ mil, data = penny, ydomain = data.frame(mil=c(49, 61)))
sscden1(~ year*mil, ~ mil, data = penny, ydomain = data.frame(mil=c(49, 61)))

## The interface puts data as first parameter
set.seed(5732)
ntbt_sscden(penny, ~ year*mil, ~ mil, ydomain = data.frame(mil=c(49, 61)))
ntbt_sscden1(penny, ~ year*mil, ~ mil, ydomain = data.frame(mil=c(49, 61)))

## so it can be used easily in a pipeline.
set.seed(5732)
penny %>%
  ntbt_sscden(~ year*mil, ~ mil, ydomain = data.frame(mil=c(49, 61)))
penny %>%
  ntbt_sscden1(~ year*mil, ~ mil, ydomain = data.frame(mil=c(49, 61)))


## ntbt_sscox: Estimating Relative Risk Using Smoothing Splines
data(stan)

## Original function to interface
sscox(Surv(futime, status) ~ age, data = stan)

## The interface puts data as first parameter
ntbt_sscox(stan, Surv(futime, status) ~ age)

## so it can be used easily in a pipeline.
stan %>%
  ntbt_sscox(Surv(futime, status) ~ age)


## ntbt_ssden: Estimating Probability Density Using Smoothing Splines
data(aids)
```

```
## rectangular quadrature
quad.pt <- expand.grid(incu=((1:40)-.5)/40*100,infe=((1:40)-.5)/40*100)
quad.pt <- quad.pt[quad.pt$incu<=quad.pt$infe,]
quad.wt <- rep(1,nrow(quad.pt))
quad.wt[quad.pt$incu==quad.pt$infe] <- .5
quad.wt <- quad.wt/sum(quad.wt)*5e3

## Original function to interface
ssden(~ incu + infe, data = aids, subset = age >= 60,
      domain = data.frame(incu = c(0, 100), infe=c(0, 100)),
      quad = list(pt = quad.pt, wt = quad.wt))
ssden1(~ incu + infe, data = aids, subset = age >= 60,
       domain = data.frame(incu = c(0, 100), infe=c(0, 100)),
       quad = list(pt = quad.pt, wt = quad.wt))

## The interface puts data as first parameter
ntbt_ssden(aids, ~ incu + infe, subset = age >= 60,
           domain = data.frame(incu = c(0, 100), infe=c(0, 100)),
           quad = list(pt = quad.pt, wt = quad.wt))
ntbt_ssden1(aids, ~ incu + infe, subset = age >= 60,
            domain = data.frame(incu = c(0, 100), infe=c(0, 100)),
            quad = list(pt = quad.pt, wt = quad.wt))

## so it can be used easily in a pipeline.
aids %>%
  ntbt_ssden(~ incu + infe, subset = age >= 60,
             domain = data.frame(incu = c(0, 100), infe=c(0, 100)),
             quad = list(pt = quad.pt, wt = quad.wt))
aids %>%
  ntbt_ssden1(~ incu + infe, subset = age >= 60,
              domain = data.frame(incu = c(0, 100), infe=c(0, 100)),
              quad = list(pt = quad.pt, wt = quad.wt))


## ntbt_sshzd: Estimating Hazard Function Using Smoothing Splines
data(gastric)

## Original function to interface
sshzd(Surv(futime, status) ~ futime*trt, data = gastric)

## The interface puts data as first parameter
ntbt_sshzd(gastric, Surv(futime, status) ~ futime*trt)

## so it can be used easily in a pipeline.
gastric %>%
  ntbt_sshzd(Surv(futime, status) ~ futime*trt)


## ntbt_ssllrm: Fitting Smoothing Spline Log-Linear Regression Models
test <- function(x)
        {.3*(1e6*(x^11*(1-x)^6)+1e4*(x^3*(1-x)^10))-2}
x <- (0:100)/100
p <- 1-1/(1+exp(test(x)))
```

```
y <- rbinom(x,3,p)
y1 <- as.ordered(y)
y2 <- as.factor(rbinom(x,1,p))

dta <- data.frame(x, y1, y2)

## Original function to interface
ssllrm(~ y1*y2*x, ~ y1 + y2, data = dta)

## The interface puts data as first parameter
ntbt_ssllrm(dta, ~ y1*y2*x, ~ y1 + y2)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_ssllrm(~ y1*y2*x, ~ y1 + y2)

## End(Not run)
```

---

hdm *Interfaces for hdm package for data science pipelines.*

---

## Description

Interfaces to hdm functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_rlasso(data, ...)
ntbt_rlassoATE(data, ...)
ntbt_rlassoATET(data, ...)
ntbt_rlassoLATE(data, ...)
ntbt_rlassoLATET(data, ...)
ntbt_rlassoEffects(data, ...)
ntbt_rlassoIV(data, ...)
ntbt_rlassologit(data, ...)
# ntbt_tsls(data, ...)           ## Already defined in sem
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(hdm)


## ntbt_rlasso: rlasso: Function for Lasso estimation under homoscedastic
##                      and heteroscedastic non-Gaussian disturbances
set.seed(1)
n <- 100 #sample size
p <- 100 # number of variables
s <- 3 # nubmer of variables with non-zero coefficients
X <- Xnames <- matrix(rnorm(n*p), ncol=p)
colnames(Xnames) <- paste("V", 1:p, sep="")
beta <- c(rep(5,s), rep(0,p-s))
Y <- X %*% beta + rnorm(n)

dta <- list(Y = Y, Xnames = Xnames)
rm(Y, Xnames)

## Original function to interface
rlasso(Y ~ Xnames, data = dta)

## The interface puts data as first parameter
ntbt_rlasso(dta, Y ~ Xnames)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_rlasso(Y ~ Xnames)


## Functions for estimation of treatment effects
## ntbt_rlassoATE, ntbt_rlassoATET, ntbt_rlassoLATE, ntbt_rlassoLATET
## do not have examples of use in help.

## Original function to interface
## The interface puts data as first parameter
## so it can be used easily in a pipeline.



## ntbt_rlassoEffects: rigorous Lasso for Linear Models: Inference
set.seed(1)
n <- 100 #sample size
p <- 100 # number of variables
s <- 3 # nubmer of non-zero variables
X <- matrix(rnorm(n*p), ncol=p)
```

```
colnames(X) <- paste("X", 1:p, sep="")
beta <- c(rep(3,s), rep(0,p-s))
y <- 1 + X %*% beta + rnorm(n)
data <- data.frame(cbind(y,X))
colnames(data)[1] <- "y"
fm <- paste("y ~", paste(colnames(X), collapse="+"))
fm <- as.formula(fm)
rm(y, X)

## Original function to interface
rlassoEffects(fm, I = ~ X1 + X2 + X3 + X50, data=data)

## The interface puts data as first parameter
ntbt_rlassoEffects(data, fm, I = ~ X1 + X2 + X3 + X50)

## so it can be used easily in a pipeline.
data %>%
  ntbt_rlassoEffects(fm, I = ~ X1 + X2 + X3 + X50)



## ntbt_rlassoIV: Post-Selection and Post-Regularization Inference
##                in Linear Models with Many Controls and Instruments
## The example uses non-formula variant. Please see note below about
## possible problem.
data(EminentDomain)
dta <- list(z = EminentDomain$logGDP$z, # instruments
            x = EminentDomain$logGDP$x, # exogenous variables
            y = EminentDomain$logGDP$y, # outcome varialbe
            d = EminentDomain$logGDP$d) # treatment / endogenous variable
str(dta)
## Original function to interface
attach(dta)
rlassoIV(x=x, d=d, y=y, z=z, select.X=FALSE, select.Z=TRUE)
detach()

## The interface puts data as first parameter
## NOTE: BE CAREFUL (in general in situations as follow)
## The parameter name "d" in this function can result in a nightmare
## (it got me scratching my head for quite a bit).
## In fact, if you call with parameter names (but not naming data)
## call the following version (commented out)
# ntbt_rlassoIV(dta, x=x, d=d, y=y, z=z, select.X=FALSE, select.Z=TRUE)
## there will be an error, as R will expand "d" to "data", and use
## its info (d) instead of dta.
## Right now I am not sure how to manage this situation and avoid
## that unwanted expansion. I will get back to this later.
## To avoid problems you should specify "data" as below
ntbt_rlassoIV(data=dta, x=x, d=d, y=y, z=z, select.X=FALSE, select.Z=TRUE)
## but of course this beats the purpose (we do not want to name "data"),
## and you *cannot* do it in the pipeline version (as you do not include data
## in your call).
## SOLUTION. In cases of unfortunate parameter names: "d", "da", "dat",
```

```
## you need to make sure that that parameter is sent by position AND unnamed
ntbt_rlassoIV(dta, x=x, d, y=y, z=z, select.X=FALSE, select.Z=TRUE)
## In general, required data is sent unnamed and by position, like
ntbt_rlassoIV(dta, x, d, y, z, select.X = FALSE, select.Z = TRUE)
## and this would have been what I would have done if I would
## (have had the ability to) produce this example.
## But this is how the example was provided, giving an opportunity to
## uncover this potentially unpleasant situation.

## so it can be used easily in a pipeline.
dta %>%
  ntbt_rlassoIV(x, d, y, z, select.X = FALSE, select.Z = TRUE)


## ntbt_rlassologit: Function for logistic Lasso estimation
library(hdm)
## DGP
set.seed(2)
n <- 250
p <- 100
px <- 10
X <- matrix(rnorm(n*p), ncol=p)
beta <- c(rep(2,px), rep(0,p-px))
intercept <- 1
P <- exp(intercept + X %*% beta)/(1+exp(intercept + X %*% beta))
y <- rbinom(n, size=1, prob=P)
dta <- list(y = y, X = X)
rm(y, X)

## Original function to interface
rlassologit(y ~ X, dta)

## The interface puts data as first parameter
ntbt_rlassologit(dta, y ~ X)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_rlassologit(y ~ X)



## ntbt_tsls: Two-Stage Least Squares Estimation (TSLS)
## No example provided

## End(Not run)
```

---

Hmisc *Interfaces for Hmisc package for data science pipelines.*

---

## Description

Interfaces to Hmisc functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_areg.boot(data, ...)
ntbt_aregImpute(data, ...)
ntbt_biVar(data, ...)
ntbt_bpplotM(data, ...)
ntbt_dataRep(data, ...)
ntbt_describe(data, ...)
ntbt_Dotplot(data, ...)
ntbt_Ecdf(data, ...)
ntbt_fit.mult.impute(data, ...)
ntbt_nobsY(data, ...)
ntbt_rcorrcens(data, ...)
ntbt_redun(data, ...)
ntbt_summary(data, ...)
ntbt_summaryD(data, ...)
ntbt_summaryM(data, ...)
ntbt_summaryP(data, ...)
ntbt_summaryRc(data, ...)
ntbt_summaryS(data, ...)
ntbt_transcan(data, ...)
ntbt_varclus(data, ...)
ntbt_xYplot(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
```

```
library(Hmisc)

## ntbt_areg.boot
set.seed(171)  # to be able to reproduce example
x1 <- rnorm(200)
x2 <- runif(200)  # a variable that is really unrelated to y]
x3 <- factor(sample(c('cat','dog','cow'), 200,TRUE))  # also unrelated to y
y  <- exp(x1 + rnorm(200)/3)

data <- data.frame(y, x1, x2, x3)

## Original function to interface
f  <- areg.boot(y ~ x1 + x2 + x3, data, B = 40)
plot(f)

## The interface puts data as first parameter
f  <- ntbt_areg.boot(data, y ~ x1 + x2 + x3, B = 40)
plot(f)

## so it can be used easily in a pipeline.
data %>%
  ntbt_areg.boot(y ~ x1 + x2 + x3, B = 40) %>%
  plot()


## ntbt_aregImpute
x1 <- factor(sample(c('a','b','c'),1000,TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(1000,0,2)
x3 <- rnorm(1000)
y  <- x2 + 1*(x1=='c') + .2*x3 + rnorm(1000,0,2)
orig.x1 <- x1[1:250]
orig.x2 <- x2[251:350]
x1[1:250] <- NA
x2[251:350] <- NA
d <- data.frame(x1, x2, x3, y)

## Original function to interface
# Find value of nk that yields best validating imputation models
# tlinear=FALSE means to not force the target variable to be linear
f <- aregImpute(~y + x1 + x2 + x3, nk=c(0,3:5), tlinear=FALSE,
                data=d, B=10) # normally B=75
plot(f)
## The interface puts data as first parameter
f <- ntbt_aregImpute(d, ~y + x1 + x2 + x3, nk=c(0,3:5), tlinear=FALSE, B=10)
plot(f)

## so it can be used easily in a pipeline.
d %>%
  ntbt_aregImpute(~y + x1 + x2 + x3, nk=c(0,3:5), tlinear=FALSE, B=10) %>%
  plot()


## biVar
```

```
## NOTE: not seen any working example.
## I am too lazy (ignorant, really...) to produce one.
## Please contribute.

## Original function to interface
## The interface puts data as first parameter
## so it can be used easily in a pipeline.


## ntbt_bpplotM
set.seed(1)
n <- 800
d <- data.frame(treatment = sample(c('a','b'), n, TRUE),
                sex = sample(c('female','male'), n, TRUE),
                age = rnorm(n, 40, 10),
                bp = rnorm(n, 120, 12),
                wt = rnorm(n, 190, 30))
label(d$bp) <- 'Systolic Blood Pressure'
units(d$bp) <- 'mmHg'

## Original function to interface
bpplotM(age + bp + wt ~ treatment * sex, data=d, violin = TRUE,
        violin.opts = list(col = adjustcolor('blue', alpha.f = .15),
                           border = FALSE))

## The interface puts data as first parameter
ntbt_bpplotM(d, age + bp + wt ~ treatment * sex, violin = TRUE,
             violin.opts = list(col = adjustcolor('blue', alpha.f= .15),
                                border = FALSE))

## so it can be used easily in a pipeline.
d %>%
  ntbt_bpplotM(age + bp + wt ~ treatment * sex, violin = TRUE,
               violin.opts = list(col = adjustcolor('blue', alpha.f= .15),
                                  border = FALSE))


## dataRep
set.seed(13)
num.symptoms <- sample(1:4, 1000,TRUE)
sex <- factor(sample(c('female','male'), 1000,TRUE))
x    <- runif(1000)
x[1] <- NA
table(num.symptoms, sex, .25*round(x/.25))
data <- data.frame(num.symptoms, sex, x)

## Original function to interface
d <- dataRep(~ num.symptoms + sex + roundN(x, .25), data)
print(d, long = TRUE)

## The interface puts data as first parameter
d <- ntbt_dataRep(data, ~ num.symptoms + sex + roundN(x, .25))
print(d, long = TRUE)
```

```
## so it can be used easily in a pipeline.
data %>%
  ntbt_dataRep(~ num.symptoms + sex + roundN(x, .25)) %>%
  print(long = TRUE)


## ntbt_describe
## Original function to interface
describe(~ conc + Type, data = CO2)

## The interface puts data as first parameter
ntbt_describe(CO2, ~ conc + Type)

## so it can be used easily in a pipeline.
CO2 %>%
  ntbt_describe(~ conc + Type)


## ntbt_Dotplot
set.seed(111)
dfr <- expand.grid(month=1:12, year=c(1997,1998), reps=1:100)
month <- dfr$month; year <- dfr$year
y <- abs(month-6.5) + 2*runif(length(month)) + year-1997
s <- summarize(y, llist(month,year), smedian.hilow, conf.int=.5)

## Original function to interface
Dotplot(month ~ Cbind(y, Lower, Upper) | year, data = s)

## The interface puts data as first parameter
ntbt_Dotplot(s, month ~ Cbind(y, Lower, Upper) | year)

## so it can be used easily in a pipeline.
s %>%
  ntbt_Dotplot(month ~ Cbind(y, Lower, Upper) | year)


## ntbt_Ecdf
set.seed(135)
m <- data.frame(ch = rnorm(1000, 200, 40),
                pre.test = rnorm(100,50,10),
                post.test = rnorm(100,55,10),
                sex = sample(c('male','female'),100,TRUE),
                region = factor(sample(c('Europe','USA','Australia'),100,TRUE)),
                year = factor(sample(2001:2002,1000,TRUE)))

## Original function to interface
Ecdf(~ ch | region * year, groups = sex, m)

## The interface puts data as first parameter
ntbt_Ecdf(m, ~ ch | region * year, groups = sex)

## so it can be used easily in a pipeline.
```

```
m %>%
  ntbt_Ecdf(~ ch | region * year, groups = sex)


## ntbt_nobsY
d <- expand.grid(sex=c('female', 'male', NA),
                 country=c('US', 'Romania'),
                 reps=1:2)
d$subject.id <- c(0, 0, 3:12)
dm <- addMarginal(d, sex, country)

## Original function to interface
nobsY(sex + country ~ id(subject.id) + reps, group = 'reps', data = d)

## The interface puts data as first parameter
ntbt_nobsY(d, sex + country ~ id(subject.id) + reps, group = 'reps')

## so it can be used easily in a pipeline.
d %>%
  ntbt_nobsY(sex + country ~ id(subject.id) + reps, group = 'reps')


## ntbt_rcorrcens
set.seed(1)
x <- round(rnorm(200))
y <- rnorm(200)
rcorr.cens(x, y, outx=TRUE)   # can correlate non-censored variables
library(survival)
age <- rnorm(400, 50, 10)
bp  <- rnorm(400,120, 15)
bp[1]  <- NA
d.time <- rexp(400)
cens   <- runif(400,.5,2)
death  <- d.time <= cens
d.time <- pmin(d.time, cens)
data <- data.frame(d.time, death, age, bp)

## Original function to interface
r <- rcorrcens(Surv(d.time, death) ~ age + bp, data = data)
plot(r)

## The interface puts data as first parameter
r <- ntbt_rcorrcens(data, Surv(d.time, death) ~ age + bp)
plot(r)

## so it can be used easily in a pipeline.
data %>%
  ntbt_rcorrcens(Surv(d.time, death) ~ age + bp) %>%
  plot()


## ntbt_redun
set.seed(1)
```

```
n <- 100
x1 <- runif(n)
x2 <- runif(n)
x3 <- x1 + x2 + runif(n) / 10
x4 <- x1 + x2 + x3 + runif(n) / 10
x5 <- factor(sample(c('a','b','c'), n, replace = TRUE))
x6 <- 1 * (x5 == 'a' | x5 == 'c')
data <- data.frame(x1, x2, x3, x4, x5, x6)

## Original function to interface
redun(~ x1 + x2 + x3 + x4 + x5 + x6, data, r2 = .8, allcat = TRUE)

## The interface puts data as first parameter
ntbt_redun(data, ~ x1 + x2 + x3 + x4 + x5 + x6, r2 = .8, allcat = TRUE)

## so it can be used easily in a pipeline.
data %>%
  ntbt_redun(~ x1 + x2 + x3 + x4 + x5 + x6, r2 = .8, allcat = TRUE)


## ntbt_summary
options(digits=3)
set.seed(173)
sex <- factor(sample(c("m","f"), 500, rep=TRUE))
age <- rnorm(500, 50, 5)
treatment <- factor(sample(c("Drug","Placebo"), 500, rep=TRUE))
# Generate a 3-choice variable; each of 3 variables has 5 possible levels
symp <- c('Headache', 'Stomach Ache', 'Hangnail',
          'Muscle Ache', 'Depressed')
symptom1 <- sample(symp, 500,TRUE)
symptom2 <- sample(symp, 500,TRUE)
symptom3 <- sample(symp, 500,TRUE)
Symptoms <- mChoice(symptom1, symptom2, symptom3, label='Primary Symptoms')
data <- data.frame(sex, age, treatment, Symptoms)

## Original function to interface
summary(sex ~ treatment + Symptoms, data, fun = table)
summary(age ~ sex + treatment + Symptoms, data)

## The interface puts data as first parameter
ntbt_summary(data, sex ~ treatment + Symptoms, fun = table)
ntbt_summary(data, age ~ sex + treatment + Symptoms)

## so it can be used easily in a pipeline.
data %>%
  ntbt_summary(sex ~ treatment + Symptoms, fun = table)
data %>%
  ntbt_summary(age ~ sex + treatment + Symptoms)


## ntbt_summaryD
set.seed(135)
maj <- factor(c(rep('North',13),rep('South',13)))
```

```
g <- paste('Category',rep(letters[1:13],2))
y1 <- runif(26)
data <- data.frame(maj, g, y1)

## Original function to interface
summaryD(y1 ~ maj + g, xlab='Mean', data)

## The interface puts data as first parameter
ntbt_summaryD(data, y1 ~ maj + g, xlab='Mean')

## so it can be used easily in a pipeline.
par(mfrow=c(1,1))
data %>%
  ntbt_summaryD(y1 ~ maj + g, xlab='Mean')


## ntbt_summaryM
options(digits=3)
set.seed(173)
sex <- factor(sample(c("m","f"), 500, rep=TRUE))
country <- factor(sample(c('US', 'Canada'), 500, rep=TRUE))
age <- rnorm(500, 50, 5)
sbp <- rnorm(500, 120, 12)
label(sbp) <- 'Systolic BP'
units(sbp) <- 'mmHg'
treatment <- factor(sample(c("Drug","Placebo"), 500, rep=TRUE))
treatment[1]
sbp[1] <- NA

# Generate a 3-choice variable; each of 3 variables has 5 possible levels
symp <- c('Headache','Stomach Ache','Hangnail',
          'Muscle Ache','Depressed')
symptom1 <- sample(symp, 500,TRUE)
symptom2 <- sample(symp, 500,TRUE)
symptom3 <- sample(symp, 500,TRUE)
Symptoms <- mChoice(symptom1, symptom2, symptom3, label='Primary Symptoms')
data <- data.frame(age, sex, sbp, Symptoms, treatment)
# Note: In this example, some subjects have the same symptom checked
# multiple times; in practice these redundant selections would be NAs
# mChoice will ignore these redundant selections

## Original function to interface
f <- summaryM(age + sex + sbp + Symptoms ~ treatment, data = data, test = TRUE)
print(f, long = TRUE)

## The interface puts data as first parameter
f <- ntbt_summaryM(data, age + sex + sbp + Symptoms ~ treatment, test = TRUE)
print(f, long = TRUE)

## so it can be used easily in a pipeline.
data %>%
  ntbt_summaryM(age + sex + sbp + Symptoms ~ treatment, test = TRUE) %>%
  print(long = TRUE)
```

```
## ntbt_summaryP
n <- 100
f <- function(na=FALSE) {
  x <- sample(c('N', 'Y'), n, TRUE)
  if(na) x[runif(100) < .1] <- NA
  x
}
set.seed(1)
d <- data.frame(x1=f(), x2=f(), x3=f(), x4=f(), x5=f(), x6=f(), x7=f(TRUE),
                age=rnorm(n, 50, 10),
                race=sample(c('Asian', 'Black/AA', 'White'), n, TRUE),
                sex=sample(c('Female', 'Male'), n, TRUE),
                treat=sample(c('A', 'B'), n, TRUE),
                region=sample(c('North America','Europe'), n, TRUE))
d <- upData(d, labels=c(x1='MI', x2='Stroke', x3='AKI', x4='Migraines',
                 x5='Pregnant', x6='Other event', x7='MD withdrawal',
                 race='Race', sex='Sex'))

## Original function to interface
s <- summaryP(race + sex + ynbind(x1, x2, x3, x4, x5, x6, x7, label = 'Exclusions') ~
                region + treat, data=d)
plot(s, groups = 'treat')

## The interface puts data as first parameter
s <- ntbt_summaryP(d, race + sex + ynbind(x1, x2, x3, x4, x5, x6, x7, label = 'Exclusions') ~
                     region + treat)
plot(s, groups = 'treat')

## so it can be used easily in a pipeline.
d %>%
  ntbt_summaryP(race + sex + ynbind(x1, x2, x3, x4, x5, x6, x7, label = 'Exclusions') ~
                  region + treat) %>%
  plot(groups = 'treat')


## ntbt_summaryRc
options(digits=3)
set.seed(177)
sex <- factor(sample(c("m","f"), 500, rep=TRUE))
age <- rnorm(500, 50, 5)
bp  <- rnorm(500, 120, 7)
units(age) <- 'Years'; units(bp) <- 'mmHg'
label(bp) <- 'Systolic Blood Pressure'
L <- .5*(sex == 'm') + 0.1 * (age - 50)
y <- rbinom(500, 1, plogis(L))
data <- data.frame(y, age, bp, sex)
par(mfrow=c(1,2))

## Original function to interface
summaryRc(y ~ age + bp + stratify(sex), data,
          label.curves = list(keys = 'lines'), nloc = list(x = .1, y = .05))
```

```
## The interface puts data as first parameter
ntbt_summaryRc(data, y ~ age + bp + stratify(sex),
               label.curves = list(keys = 'lines'), nloc = list(x = .1, y = .05))

## so it can be used easily in a pipeline.
data %>%
  ntbt_summaryRc(y ~ age + bp + stratify(sex),
                 label.curves = list(keys = 'lines'), nloc = list(x = .1, y = .05))


## ntbt_summaryS
set.seed(1)
d <- data.frame(sbp=rnorm(n, 120, 10),
                dbp=rnorm(n, 80, 10),
                age=rnorm(n, 50, 10),
                days=sample(1:n, n, TRUE),
                S1=Surv(2*runif(n)), S2=Surv(runif(n)),
                race=sample(c('Asian', 'Black/AA', 'White'), n, TRUE),
                sex=sample(c('Female', 'Male'), n, TRUE),
                treat=sample(c('A', 'B'), n, TRUE),
                region=sample(c('North America','Europe'), n, TRUE),
                meda=sample(0:1, n, TRUE), medb=sample(0:1, n, TRUE))

d <- upData(d, labels=c(sbp='Systolic BP', dbp='Diastolic BP',
            race='Race', sex='Sex', treat='Treatment',
            days='Time Since Randomization',
            S1='Hospitalization', S2='Re-Operation',
            meda='Medication A', medb='Medication B'),
            units=c(sbp='mmHg', dbp='mmHg', age='Year', days='Days'))

## Original function to interface
s <- summaryS(age + sbp + dbp ~ days + region + treat,  data = d)
plot(s, groups = 'treat', panel = panel.loess, key = list(space = 'bottom', columns = 2),
     datadensity = TRUE, scat1d.opts = list(lwd = .5))

## The interface puts data as first parameter
s <- ntbt_summaryS(d, age + sbp + dbp ~ days + region + treat)
plot(s, groups = 'treat', panel = panel.loess, key = list(space = 'bottom', columns = 2),
     datadensity = TRUE, scat1d.opts = list(lwd = .5))

## so it can be used easily in a pipeline.
d %>%
  ntbt_summaryS(age + sbp + dbp ~ days + region + treat) %>%
  plot(groups = 'treat', panel = panel.loess, key = list(space = 'bottom', columns = 2),
       datadensity = TRUE, scat1d.opts = list(lwd = .5))


## ntbt_transcan, ntbt_fit.mult.impute
set.seed(1)
x1 <- factor(sample(c('a','b','c'),100,TRUE))
x2 <- (x1=='b') + 3*(x1=='c') + rnorm(100)
y  <- x2 + 1*(x1=='c') + rnorm(100)
```

```
x1[1:20] <- NA
x2[18:23] <- NA
d4 <- data.frame(x1,x2,y)

options(digits = 3)

## Original function to interface
f  <- transcan(~y + x1 + x2, n.impute = 10, shrink = TRUE, data = d4)
summary(f)
h <- fit.mult.impute(y ~ x1 + x2, lm, f, data = d4)
summary(h)

## The interface puts data as first parameter
f  <- ntbt_transcan(d4, ~y + x1 + x2, n.impute = 10, shrink = TRUE)
summary(f)
h <- ntbt_fit.mult.impute(d4, y ~ x1 + x2, lm, f)
summary(h)

## so it can be used easily in a pipeline.
d4 %>%
  ntbt_transcan(~y + x1 + x2, n.impute = 10, shrink = TRUE) %>%
  summary()

d4 %>%
  ntbt_fit.mult.impute(y ~ x1 + x2, lm, f) %>%
  summary()


## ntbt_varclus
set.seed(1)
x1 <- rnorm(200)
x2 <- rnorm(200)
x3 <- x1 + x2 + rnorm(200)
x4 <- x2 + rnorm(200)
data <- data.frame(x1, x2, x3, x4)

par(mfrow = c(1,1))

## Original function to interface
v <- varclus(~ x1 + x2 + x3 + x4, similarity = "spearman", data = data )
plot(v)

## The interface puts data as first parameter
v <- ntbt_varclus(data, ~ x1 + x2 + x3 + x4, similarity = "spearman")
plot(v)

## so it can be used easily in a pipeline.
data %>%
  ntbt_varclus(~ x1 + x2 + x3 + x4, similarity = "spearman") %>%
  plot()


## ntbt_xYplot
```

```
d <- expand.grid(x = seq(0, 2 * pi, length=150), p = 1:3, shift = c(0, pi))

## Original function to interface
xYplot(sin(x + shift)^p ~ x | shift, groups = p, data = d, type = 'l')

## The interface puts data as first parameter
ntbt_xYplot(d, sin(x + shift)^p ~ x | shift, groups = p, type = 'l')

## so it can be used easily in a pipeline.
d %>%
  ntbt_xYplot(sin(x + shift)^p ~ x | shift, groups = p, type = 'l')

## End(Not run)
```

---

`intubate` *1) Interfaces "on demand" and 2) calling of non-pipe-aware functions directly.*

---

## Description

`intubate` is a helper function used to implement "on demand" interfaces to functions you want to use in pipelines implemented by `magrittr`. Suppose intubate does not implement an interface to a fantasy function (such as `fantasy(formula, data, ...)`) that is non-pipe-aware (because `data` is not its first parameter). To create an interface you only need the line of code `ntbt_fantasy <- intubate`, after which `ntbt_fantasy(data, ...)` can be used in a data science pipeline. See examples for details and discussion.

`ntbt` lets you interface the non-pipe-aware function directly without creating an interface.

## Usage

```
## Helper function to define interfaces
intubate(data, ...)

## Function to call non-pipe-aware functions directly
ntbt(data, fti, ...)

## Deprecated helper functions
## (for compatibility with 0.99.2. Will be removed at some point)
ntbt_function_formula_data(data, ...)
ntbt_function_model_data(data, ...)
ntbt_function_object_data(data, ...)
ntbt_function_x_data(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `fti` | function to interface. |
| `...` | Other arguments passed to interfaced function. |

## Value

Object returned by interfaced function. If you call `intubate` directly it will fail.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)

## NOTE: intubate implements an interface to
##       *xyplot* (in package lattice), called *ntbt_xyplot*.
##       For the sake of argument, let's suppose the
##       interface does not exist, and you want to implement
##       it "on demand" to use it in a pipeline.

## Original function you would like to interface
library(lattice)
xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
       iris, scales = "free", layout = c(2, 2),
       auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## If you try to use *xyplot* directly in a data pipeline
## it will raise an error
library(magrittr)
try(iris %>%
    xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
           scales = "free", layout = c(2, 2),
           auto.key = list(x = .6, y = .7, corner = c(0, 0))),
    silent = TRUE)
geterrmessage()

## The error disappears if you create an interface to *xyplot*.

## Step needed to create an interface to *xyplot*.

ntbt_xyplot <- intubate

## NOTE: names of interfaces must start with
##       *ntbt_* followed by the name of the function
##       (*xyplot* in this case) you want to interface.

## Now you can use the interface instead of the original
## function. Just remember to switch the order of
## *data* and *x* (there is no need to name the parameters).
ntbt_xyplot(iris,
            Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
            scales = "free", layout = c(2, 2),
            auto.key = list(x = .6, y = .7, corner = c(0, 0)))
```

```
## The newly created interface can be used easily in a pipeline.
library(magrittr)
iris %>%
  ntbt_xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
              scales = "free", layout = c(2, 2),
              auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## Alternative: call non-pipe-aware function directly.
## You can also avoid creating an interface, by calling ntbt with the name of
## the function to interface.
ntbt(iris, xyplot,
     Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
     scales = "free", layout = c(2, 2),
     auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## In a pipeline
iris %>%
  ntbt(xyplot, Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
       scales = "free", layout = c(2, 2),
       auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## End(Not run)
```

---

`ipred` *Interfaces for ipred package for data science pipelines.*

---

## Description

Interfaces to `ipred` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_bagging(data, ...)
ntbt_errorest(data, ...)
ntbt_inbagg(data, ...)
ntbt_inclass(data, ...)
ntbt_slda(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

### Author(s)

Roberto Bertolusso

### Examples

```
## Not run:
library(intubate)
library(magrittr)
library(ipred)

## ntbt_bagging: Bagging Classification, Regression and Survival Trees
data("BreastCancer", package = "mlbench")

## Original function to interface
bagging(Class ~ Cl.thickness + Cell.size + Cell.shape + Marg.adhesion + Epith.c.size
       + Bare.nuclei + Bl.cromatin + Normal.nucleoli + Mitoses, data=BreastCancer, coob=TRUE)

## The interface puts data as first parameter
ntbt_bagging(BreastCancer,
             Class ~ Cl.thickness + Cell.size + Cell.shape + Marg.adhesion + Epith.c.size
              + Bare.nuclei + Bl.cromatin + Normal.nucleoli + Mitoses, coob=TRUE)

## so it can be used easily in a pipeline.
BreastCancer %>%
  ntbt_bagging(Class ~ Cl.thickness + Cell.size + Cell.shape + Marg.adhesion + Epith.c.size
               + Bare.nuclei + Bl.cromatin + Normal.nucleoli + Mitoses, coob=TRUE)


## ntbt_errorest: Estimators of Prediction Error
data("iris")
library("MASS")
mypredict.lda <- function(object, newdata)
  predict(object, newdata = newdata)$class

## Original function to interface
errorest(Species ~ ., data = iris, model = lda, estimator = "cv", predict = mypredict.lda)

## The interface puts data as first parameter
ntbt_errorest(iris, Species ~ ., model = lda, estimator = "cv", predict = mypredict.lda)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_errorest(Species ~ ., model = lda, estimator = "cv", predict = mypredict.lda)


## ntbt_inbagg: Indirect Bagging
library("MASS")
library("rpart")
y <- as.factor(sample(1:2, 100, replace = TRUE))
W <- mvrnorm(n = 200, mu = rep(0, 3), Sigma = diag(3))
X <- mvrnorm(n = 200, mu = rep(2, 3), Sigma = diag(3))
colnames(W) <- c("w1", "w2", "w3")
```

```
colnames(X) <- c("x1", "x2", "x3")
DATA <- data.frame(y, W, X)
pFUN <- list(list(formula = w1~x1+x2, model = lm, predict = mypredict.lm),
list(model = rpart))

## Original function to interface
inbagg(y ~ w1 + w2 + w3 ~ x1 + x2 + x3, data = DATA, pFUN = pFUN)

## The interface puts data as first parameter
ntbt_inbagg(DATA, y ~ w1 + w2 + w3 ~ x1 + x2 + x3, pFUN = pFUN)

## so it can be used easily in a pipeline.
DATA %>%
  ntbt_inbagg(y ~ w1 + w2 + w3 ~ x1 + x2 + x3, pFUN = pFUN)


## ntbt_inclass: Indirect Classification
data("Smoking", package = "ipred")
# Set three groups of variables:
# 1) explanatory variables are: TarY, NicY, COY, Sex, Age
# 2) intermediate variables are: TVPS, BPNL, COHB
# 3) response (resp) is defined by:
classify <- function(data) {
  data <- data[,c("TVPS", "BPNL", "COHB")]
  res <- t(t(data) > c(4438, 232.5, 58))
  res <- as.factor(ifelse(apply(res, 1, sum) > 2, 1, 0))
  res
}
response <- classify(Smoking[ ,c("TVPS", "BPNL", "COHB")])
smoking <- data.frame(Smoking, response)

## Original function to interface
inclass(response ~ TVPS + BPNL + COHB ~ TarY + NicY + COY + Sex + Age, data = smoking,
        pFUN = list(list(model = lm, predict = mypredict.lm)), cFUN = classify)

## The interface puts data as first parameter
ntbt_inclass(smoking, response ~ TVPS + BPNL + COHB ~ TarY + NicY + COY + Sex + Age,
             pFUN = list(list(model = lm, predict = mypredict.lm)), cFUN = classify)

## so it can be used easily in a pipeline.
smoking %>%
  ntbt_inclass(response ~ TVPS + BPNL + COHB ~ TarY + NicY + COY + Sex + Age,
               pFUN = list(list(model = lm, predict = mypredict.lm)), cFUN = classify)


## ntbt_slda: Stabilised Linear Discriminant Analysis
library("mlbench")
library("MASS")
learn <- as.data.frame(mlbench.twonorm(100))

## Original function to interface
slda(classes ~ ., data=learn)
```

```
## The interface puts data as first parameter
ntbt_slda(learn, classes ~ .)

## so it can be used easily in a pipeline.
learn %>%
  ntbt_slda(classes ~ .)

## End(Not run)
```

---

`iRegression` *Interfaces for iRegression package for data science pipelines.*

---

## Description

Interfaces to `iRegression` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_bivar(data, ...)
ntbt_ccrm(data, ...)
ntbt_cm(data, ...)
ntbt_crm(data, ...)
ntbt_MinMax(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(iRegression)

## bivar
```

```
data("soccer.bivar", package = "iRegression")

## Original function to interface
bivar(yMin ~ t1Min + t2Min, "identity", yMax ~ t1Max + t2Max, "identity", data = soccer.bivar)

## The interface puts data as first parameter
ntbt_bivar(soccer.bivar, yMin ~ t1Min + t2Min, "identity", yMax ~ t1Max + t2Max, "identity")

## so it can be used easily in a pipeline.
soccer.bivar %>%
  ntbt_bivar(yMin ~ t1Min + t2Min, "identity", yMax ~ t1Max + t2Max, "identity")

## ntbt_ccrm
data("Cardiological.CR", package = "iRegression")
## Original function to interface
ccrm(PulseC ~ SystC + DiastC, PulseR ~ SystR + DiastR, data = Cardiological.CR)

## The interface puts data as first parameter
ntbt_ccrm(Cardiological.CR, PulseC ~ SystC + DiastC, PulseR ~ SystR + DiastR)

## so it can be used easily in a pipeline.
Cardiological.CR %>%
  ntbt_ccrm(PulseC ~ SystC + DiastC, PulseR ~ SystR + DiastR)

## ntbt_cm
data("Cardiological.MinMax", package = "iRegression") ## see Billard and Diday (2000)
## Original function to interface
cm(PulseMin ~ SystMin + DiastMin, PulseMax ~ SystMax + DiastMax, data = Cardiological.MinMax)

## The interface puts data as first parameter
ntbt_cm(Cardiological.MinMax, PulseMin ~ SystMin + DiastMin, PulseMax ~ SystMax + DiastMax)

## so it can be used easily in a pipeline.
Cardiological.MinMax %>%
  ntbt_cm(PulseMin ~ SystMin + DiastMin, PulseMax ~ SystMax + DiastMax)

## ntbt_crm
data("Cardiological.CR", package = "iRegression")
## Original function to interface
crm(PulseC ~ SystC + DiastC, PulseR ~ SystR + DiastR, data = Cardiological.CR)

## The interface puts data as first parameter
ntbt_crm(Cardiological.CR, PulseC ~ SystC + DiastC, PulseR ~ SystR + DiastR)

## so it can be used easily in a pipeline.
Cardiological.CR %>%
  ntbt_crm(PulseC ~ SystC + DiastC, PulseR ~ SystR + DiastR)

## ntbt_MinMax
data("Cardiological.MinMax", package = "iRegression") ## see Billard, L. and Diday, E. (2000)
## Original function to interface
MinMax(PulseMin ~ SystMin + DiastMin, PulseMax ~ SystMax + DiastMax, data = Cardiological.MinMax)
```

```
## The interface puts data as first parameter
ntbt_MinMax(Cardiological.MinMax, PulseMin ~ SystMin + DiastMin, PulseMax ~ SystMax + DiastMax)

## so it can be used easily in a pipeline.
Cardiological.MinMax %>%
  ntbt_MinMax(PulseMin ~ SystMin + DiastMin, PulseMax ~ SystMax + DiastMax)

## End(Not run)
```

---

`ivfixed` *Interfaces for ivfixed package for data science pipelines.*

---

## Description

Interfaces to `ivfixed` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_ivFixed(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(ivfixed)


## ntbt_ivFixed: Fit an Instrumental least square dummy variable model
pib <- as.matrix(c(12,3,4,0.4,0.7,5,0.7,0.3,
                   0.6,89,7,8,45,7,4,5,0.5,5),nrows=18,ncols=1)
tir <- as.matrix(c(12,0.3,4,0.4,7,12,3.0,6.0,45,
```

```
                    7.0,0.8,44,65,23,4,6,76,9),nrows=18,ncols=1)
inf <- as.matrix(c(1.2,3.6,44,1.4,0.78,54,0.34,0.66,
                    12,0.7,8.0,12,65,43,5,76,65,8),nrows=18,ncols=1)
npl <- as.matrix(c(0.2,3.8,14,2.4,1.7,43,0.2,0.5,23,
                    7.8,88,36,65,3,44,65,7,34),nrows=18,ncols=1)
mdata <- data.frame(p = pib, t = tir, int = inf, np = npl)

## Original function to interface
ivFixed(t ~ p + int | p + np, mdata, n = 6, t = 3)

## The interface puts data as first parameter
ntbt_ivFixed(mdata, t ~ p + int | p + np, n = 6, t = 3)

## so it can be used easily in a pipeline.
mdata %>%
  ntbt_ivFixed(t ~ p + int | p + np, n = 6, t = 3)

## End(Not run)
```

---

kernlab *Interfaces for kernlab package for data science pipelines.*

---

## Description

Interfaces to kernlab functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_gausspr(data, ...)
ntbt_kfa(data, ...)
ntbt_kha(data, ...)
ntbt_kkmeans(data, ...)
ntbt_kpca(data, ...)
ntbt_kqr(data, ...)
ntbt_ksvm(data, ...)
ntbt_lssvm(data, ...)
ntbt_rvm(data, ...)
ntbt_sigest(data, ...)
ntbt_specc(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(kernlab)

##  ntbt_gausspr: Gaussian processes for regression and classification
data(iris)

## Original function to interface
gausspr(Species ~ ., data = iris, var = 2)

## The interface puts data as first parameter
ntbt_gausspr(iris, Species ~ ., var = 2)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_gausspr(Species ~ ., var = 2)


## ntbt_kfa: Kernel Feature Analysis
data(promotergene)

## Original function to interface
kfa(~ ., data = promotergene)

## The interface puts data as first parameter
ntbt_kfa(promotergene, ~ .)

## so it can be used easily in a pipeline.
promotergene %>%
  ntbt_kfa(~ .)


## ntbt_kha: Kernel Principal Components Analysis
data(iris)
test <- sample(1:150,70)

## Original function to interface
kpc <- kha(~ ., data = iris[-test, -5], kernel = "rbfdot", kpar = list(sigma=0.2),
           features = 2, eta = 0.001, maxiter = 65)
pcv(kpc)

## The interface puts data as first parameter
```

```
kpc <- ntbt_kha(iris[-test, -5], ~ ., kernel = "rbfdot", kpar = list(sigma=0.2),
                features = 2, eta = 0.001, maxiter = 65)
pcv(kpc)

## so it can be used easily in a pipeline.
iris[-test, -5] %>%
  ntbt_kha(~ ., kernel = "rbfdot", kpar = list(sigma=0.2),
           features = 2, eta = 0.001, maxiter = 65) %>%
  pcv()


## ntbt_kkmeans: Kernel k-means
## Original function to interface
sc <- kkmeans(~ ., data = iris[-test, -5], centers = 3)
centers(sc)

## The interface puts data as first parameter
sc <- ntbt_kkmeans(iris[-test, -5], ~ ., centers = 3)
centers(sc)

## so it can be used easily in a pipeline.
iris[-test, -5] %>%
  ntbt_kkmeans(~ ., centers = 3) %>%
  centers()


## ntbt_kpca: Kernel Principal Components Analysis
data(iris)
test <- sample(1:150,20)

## Original function to interface
kpc <- kpca(~ ., data = iris[-test, -5], kernel = "rbfdot",
            kpar = list(sigma = 0.2), features = 2)
pcv(kpc)

## The interface puts data as first parameter
kpc <- ntbt_kpca(iris[-test, -5], ~ ., kernel = "rbfdot",
                 kpar = list(sigma = 0.2), features = 2)
pcv(kpc)

## so it can be used easily in a pipeline.
iris[-test, -5] %>%
  ntbt_kpca(~ ., kernel = "rbfdot",
            kpar = list(sigma = 0.2), features = 2) %>%
  pcv()


## ntbt_kqr: Kernel Quantile Regression
## Not found example using formula interface, and I am
## completely ignorant to construct one.
x <- sort(runif(300))
y <- sin(pi*x) + rnorm(300,0,sd=exp(sin(2*pi*x)))
```

```
dkqr <- data.frame(x, y)

## Original function to interface
set.seed(1)
kqr(x, y, tau = 0.5, C = 0.15)

## The interface puts data as first parameter
set.seed(1)
ntbt_kqr(dkqr, x, y, tau = 0.5, C = 0.15)

## so it can be used easily in a pipeline.
set.seed(1)
dkqr %>%
  ntbt_kqr(x, y, tau = 0.5, C = 0.15)


## ntbt_ksvm: Support Vector Machines
data(spam)
index <- sample(1:dim(spam)[1])
spamtrain <- spam[index[1:floor(dim(spam)[1]/2)], ]
spamtest <- spam[index[((ceiling(dim(spam)[1]/2)) + 1):dim(spam)[1]], ]

## Original function to interface
set.seed(1)
ksvm(type ~ ., data = spamtrain, kernel = "rbfdot",
     kpar = list(sigma = 0.05), C = 5, cross = 3)

## The interface puts data as first parameter
set.seed(1)
ntbt_ksvm(spamtrain, type ~ ., kernel = "rbfdot",
          kpar = list(sigma = 0.05), C = 5, cross = 3)

## so it can be used easily in a pipeline.
set.seed(1)
spamtrain %>%
  ntbt_ksvm(type ~ ., kernel = "rbfdot",
            kpar = list(sigma = 0.05), C = 5, cross = 3)


## ntbt_lssvm: Least Squares Support Vector Machine
data(iris)

## Original function to interface
set.seed(1)
lssvm(Species ~ ., data = iris)

## The interface puts data as first parameter
set.seed(1)
ntbt_lssvm(iris, Species ~ .)

## so it can be used easily in a pipeline.
set.seed(1)
iris %>%
```

```
  ntbt_lssvm(Species ~ .)


## ntbt_rvm: Relevance Vector Machine
## Not found example using formula interface, and I am
## completely ignorant to construct one.
x <- seq(-20,20,0.1)
y <- sin(x)/x + rnorm(401,sd=0.05)

drvm <- data.frame(x, y)

## Original function to interface
set.seed(1)
rvm(x, y, tau = 0.5, C = 0.15)

## The interface puts data as first parameter
set.seed(1)
ntbt_rvm(drvm, x, y, tau = 0.5, C = 0.15)

## so it can be used easily in a pipeline.
set.seed(1)
drvm %>%
  ntbt_rvm(x, y, tau = 0.5, C = 0.15)


## ntbt_sigest: Hyperparameter estimation for the Gaussian Radial Basis kernel
data(promotergene)

## Original function to interface
set.seed(1)
sigest(Class ~ ., data = promotergene)

## The interface puts data as first parameter
set.seed(1)
ntbt_sigest(promotergene, Class ~ .)

## so it can be used easily in a pipeline.
set.seed(1)
promotergene %>%
  ntbt_sigest(Class ~ .)

## ntbt_specc: Spectral Clustering
## Not found example using formula interface, and I am
## completely ignorant to construct one.

## End(Not run)
```

---

kknn *Interfaces for kknn package for data science pipelines.*

---

## Description

Interfaces to kknn functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_train.kknn(data, ...)
ntbt_cv.kknn(data, ...)
ntbt_kknn(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(kknn)


## ntbt_train.kknn: Training kknn
## ntbt_cv.kknn:
data(miete)

## Original function to interface
train.kknn(nmqm ~ wfl + bjkat + zh, data = miete,
           kmax = 25, kernel = c("rectangular", "triangular", "epanechnikov",
                                 "gaussian", "rank", "optimal"))
cv.kknn(nmqm ~ wfl + bjkat + zh, data = miete)

## The interface puts data as first parameter
ntbt_train.kknn(miete, nmqm ~ wfl + bjkat + zh,
                kmax = 25, kernel = c("rectangular", "triangular", "epanechnikov",
                                      "gaussian", "rank", "optimal"))
ntbt_cv.kknn(miete, nmqm ~ wfl + bjkat + zh)

## so it can be used easily in a pipeline.
```

```
miete %>%
  ntbt_train.kknn(nmqm ~ wfl + bjkat + zh,
                  kmax = 25, kernel = c("rectangular", "triangular", "epanechnikov",
                                        "gaussian", "rank", "optimal"))
miete %>%
  ntbt_cv.kknn(nmqm ~ wfl + bjkat + zh)

## ntbt_kknn: Weighted k-Nearest Neighbor Classifier
m <- dim(iris)[1]
val <- sample(1:m, size = round(m/3), replace = FALSE, prob = rep(1/m, m))
iris.learn <- iris[-val,]
iris.valid <- iris[val,]

## Original function to interface
kknn(Species ~ ., iris.learn, iris.valid, distance = 1, kernel = "triangular")

## The interface puts data as first parameter
ntbt_kknn(iris.learn, Species ~ ., iris.valid, distance = 1, kernel = "triangular")

## so it can be used easily in a pipeline.
iris.learn %>%
  ntbt_kknn(Species ~ ., iris.valid, distance = 1, kernel = "triangular")
## NOTE: there is (in your face) cheating! We should be able to supply
##       both iris.learn and iris.valid. It should be possible with intuBags.

## End(Not run)
```

---

klaR *Interfaces for klaR package for data science pipelines.*

---

## Description

Interfaces to klaR functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_classscatter(data, ...)
ntbt_cond.index(data, ...)
ntbt_greedy.wilks(data, ...)
ntbt_loclda(data, ...)
ntbt_meclight(data, ...)
ntbt_NaiveBayes(data, ...)
ntbt_nm(data, ...)
ntbt_partimat(data, ...)
ntbt_plineplot(data, ...)
ntbt_pvs(data, ...)
ntbt_rda(data, ...)
ntbt_sknn(data, ...)
ntbt_stepclass(data, ...)
ntbt_woe(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(klaR)

## ntbt_classscatter: Classification scatterplot matrix
data(B3)
library(MASS)

## Original function to interface
classscatter(PHASEN ~ BSP91JW + EWAJW + LSTKJW, data = B3, method = "lda")

## The interface puts data as first parameter
ntbt_classscatter(B3, PHASEN ~ BSP91JW + EWAJW + LSTKJW, method = "lda")

## so it can be used easily in a pipeline.
B3 %>%
  ntbt_classscatter(PHASEN ~ BSP91JW + EWAJW + LSTKJW, method = "lda")


## ntbt_cond.index: Calculation of Condition Indices for Linear Regression
data(Boston)

## Original function to interface
cond.index(medv ~ ., data = Boston)

## The interface puts data as first parameter
ntbt_cond.index(Boston, medv ~ .)

## so it can be used easily in a pipeline.
Boston %>%
  ntbt_cond.index(medv ~ .)
```

```
## ntbt_greedy.wilks: Stepwise forward variable selection for classification
data(B3)

## Original function to interface
greedy.wilks(PHASEN ~ ., data = B3, niveau = 0.1)

## The interface puts data as first parameter
ntbt_greedy.wilks(B3, PHASEN ~ ., niveau = 0.1)

## so it can be used easily in a pipeline.
B3 %>%
  ntbt_greedy.wilks(PHASEN ~ ., niveau = 0.1)

## ntbt_loclda: Localized Linear Discriminant Analysis (LocLDA)
## Original function to interface
loclda(PHASEN ~ ., data = B3)

## The interface puts data as first parameter
ntbt_loclda(B3, PHASEN ~ .)

## so it can be used easily in a pipeline.
B3 %>%
  ntbt_loclda(PHASEN ~ .)


## ntbt_meclight: Minimal Error Classification
data(iris)

## Original function to interface
meclight(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_meclight(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_meclight(Species ~ .)


## ntbt_NaiveBayes: Naive Bayes Classifier
data(iris)

## Original function to interface
NaiveBayes(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_NaiveBayes(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_NaiveBayes(Species ~ .)
```

```
## ntbt_nm: Nearest Mean Classification
## Original function to interface
nm(PHASEN ~ ., data = B3)

## The interface puts data as first parameter
ntbt_nm(B3, PHASEN ~ .)

## so it can be used easily in a pipeline.
B3 %>%
  ntbt_nm(PHASEN ~ .)


## ntbt_partimat: Plotting the 2-d partitions of classification methods
## Original function to interface
partimat(Species ~ ., data = iris, method = "lda")

## The interface puts data as first parameter
ntbt_partimat(iris, Species ~ ., method = "lda")

## so it can be used easily in a pipeline.
iris %>%
  ntbt_partimat(Species ~ ., method = "lda")


## ntbt_plineplot: Plotting marginal posterior class probabilities
## Original function to interface
plineplot(PHASEN ~ ., data = B3, method = "lda", x = "EWAJW", xlab = "EWAJW")

## The interface puts data as first parameter
ntbt_plineplot(B3, PHASEN ~ ., method = "lda", x = "EWAJW", xlab = "EWAJW")

## so it can be used easily in a pipeline.
B3 %>%
  ntbt_plineplot(PHASEN ~ ., method = "lda", x = "EWAJW", xlab = "EWAJW")


## ntbt_pvs: Pairwise variable selection for classification
library("mlbench")
data("Satellite")

## Original function to interface
pvs(classes ~ ., Satellite[1:3218,], method="qda", vs.method="ks.test")

## The interface puts data as first parameter
ntbt_pvs(Satellite[1:3218,], classes ~ ., method="qda", vs.method="ks.test")

## so it can be used easily in a pipeline.
Satellite[1:3218,] %>%
  ntbt_pvs(classes ~ ., method="qda", vs.method="ks.test")


## ntbt_rda: Regularized Discriminant Analysis (RDA)
```

```
## Original function to interface
rda(Species ~ ., data = iris, gamma = 0.05, lambda = 0.2)

## The interface puts data as first parameter
ntbt_rda(iris, Species ~ ., gamma = 0.05, lambda = 0.2)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_rda(Species ~ ., gamma = 0.05, lambda = 0.2)


## ntbt_sknn: Simple k nearest Neighbours
## Original function to interface
sknn(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_sknn(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_sknn(Species ~ .)


## ntbt_stepclass: Stepwise variable selection for classification
## Original function to interface
stepclass(Species ~ ., data = iris, method = "qda",
          start.vars = "Sepal.Width", criterion = "AS")  # same as above

## The interface puts data as first parameter
ntbt_stepclass(iris, Species ~ ., method = "qda",
               start.vars = "Sepal.Width", criterion = "AS")  # same as above

## so it can be used easily in a pipeline.
iris %>%
  ntbt_stepclass(Species ~ ., method = "qda",
                 start.vars = "Sepal.Width", criterion = "AS")  # same as above


## ntbt_woe: Weights of evidence
data("GermanCredit")
set.seed(6)
train <- sample(nrow(GermanCredit), round(0.6*nrow(GermanCredit)))

## Original function to interface
woe(credit_risk ~ ., data = GermanCredit[train,], zeroadj = 0.5, applyontrain = TRUE)

## The interface puts data as first parameter
ntbt_woe(GermanCredit[train,], credit_risk ~ ., zeroadj = 0.5, applyontrain = TRUE)

## so it can be used easily in a pipeline.
GermanCredit[train,] %>%
  ntbt_woe(credit_risk ~ ., zeroadj = 0.5, applyontrain = TRUE)
```

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

---

`lars` *Interfaces for lars package for data science pipelines.*

---

## Description

Interfaces to `lars` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_lars(data, ...)
ntbt_cv.lars(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(lars)


library(ISLR)
data("Hitters")
Hitters <- na.omit(Hitters)

dta <- list(x = model.matrix(Salary ~ ., Hitters)[, -1],  ## Remove intercept
            y = model.frame(Salary ~ ., Hitters)[, 1])

## ntbt_lars: Fits Least Angle Regression, Lasso and Infinitesimal
##            Forward Stagewise regression models
```

```
## Original function to interface
attach(dta)
lasso <- lars(x, y)
plot(lasso)
detach()

## The interface puts data as first parameter
lasso <- ntbt_lars(dta, x, y)
plot(lasso)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_lars(x, y) %>%
  plot()


## ntbt_cv.lars: Computes K-fold cross-validated error curve for lars

## Original function to interface
set.seed(1)
attach(dta)
cv.lars(x, y)
detach()

## The interface puts data as first parameter
set.seed(1)
ntbt_cv.lars(dta, x, y)

## so it can be used easily in a pipeline.
set.seed(1)
dta %>%
  ntbt_cv.lars(x, y)

## End(Not run)
```

---

`lattice` *Interfaces for lattice package for data science pipelines.*

---

## Description

Interfaces to `lattice` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_barchart(data, ...)
ntbt_bwplot(data, ...)
ntbt_cloud(data, ...)
ntbt_contourplot(data, ...)
ntbt_densityplot(data, ...)
ntbt_dotplot(data, ...)
```

```
ntbt_histogram(data, ...)
ntbt_levelplot(data, ...)
ntbt_oneway(data, ...)
ntbt_parallelplot(data, ...)
ntbt_qq(data, ...)
ntbt_qqmath(data, ...)
ntbt_splom(data, ...)
ntbt_stripplot(data, ...)
ntbt_tmd(data, ...)
ntbt_wireframe(data, ...)
ntbt_xyplot(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(lattice)

## barchart
## Original function to interface
barchart(yield ~ variety | site, data = barley,
         groups = year, layout = c(1,6), stack = TRUE,
         auto.key = list(space = "right"),
         ylab = "Barley Yield (bushels/acre)",
         scales = list(x = list(rot = 45)))

## The interface reverses the order of data and formula
ntbt_barchart(data = barley, yield ~ variety | site,
              groups = year, layout = c(1,6), stack = TRUE,
              auto.key = list(space = "right"),
              ylab = "Barley Yield (bushels/acre)",
              scales = list(x = list(rot = 45)))
```

```
## so it can be used easily in a pipeline.
barley %>%
  ntbt_barchart(yield ~ variety | site,
                groups = year, layout = c(1,6), stack = TRUE,
                auto.key = list(space = "right"),
                ylab = "Barley Yield (bushels/acre)",
                scales = list(x = list(rot = 45)))

## bwplot
## Original function to interface
bwplot(voice.part ~ height, data = singer, xlab = "Height (inches)")

## The interface reverses the order of data and formula
ntbt_bwplot(data = singer, voice.part ~ height, xlab = "Height (inches)")

## so it can be used easily in a pipeline.
singer %>%
  ntbt_bwplot(voice.part ~ height, xlab = "Height (inches)")

## cloud
## Original function to interface
cloud(Sepal.Length ~ Petal.Length * Petal.Width | Species, data = iris,
      screen = list(x = -90, y = 70), distance = .4, zoom = .6)

## The interface reverses the order of data and formula
ntbt_cloud(data = iris, Sepal.Length ~ Petal.Length * Petal.Width | Species,
           screen = list(x = -90, y = 70), distance = .4, zoom = .6)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_cloud(Sepal.Length ~ Petal.Length * Petal.Width | Species,
             screen = list(x = -90, y = 70), distance = .4, zoom = .6)

## contourplot
grid <- with(
  environmental,
  {
    ozo.m <- loess((ozone^(1/3)) ~ wind * temperature * radiation,
                   parametric = c("radiation", "wind"), span = 1, degree = 2)
    w.marginal <- seq(min(wind), max(wind), length.out = 50)
    t.marginal <- seq(min(temperature), max(temperature), length.out = 50)
    r.marginal <- seq(min(radiation), max(radiation), length.out = 4)
    wtr.marginal <- list(wind = w.marginal, temperature = t.marginal,
                         radiation = r.marginal)
    ret <- expand.grid(wtr.marginal)
    ret[, "fit"] <- c(predict(ozo.m, ret))
    ret
  })

## Original function to interface
contourplot(fit ~ wind * temperature | radiation, data = grid,
            cuts = 10, region = TRUE,
            xlab = "Wind Speed (mph)",
```

```
            ylab = "Temperature (F)",
            main = "Cube Root Ozone (cube root ppb)")

## The interface reverses the order of data and formula
ntbt_contourplot(data = grid, fit ~ wind * temperature | radiation,
                 cuts = 10, region = TRUE,
                 xlab = "Wind Speed (mph)",
                 ylab = "Temperature (F)",
                 main = "Cube Root Ozone (cube root ppb)")

## so it can be used easily in a pipeline.
grid %>%
  ntbt_contourplot(fit ~ wind * temperature | radiation,
                   cuts = 10, region = TRUE,
                   xlab = "Wind Speed (mph)",
                   ylab = "Temperature (F)",
                   main = "Cube Root Ozone (cube root ppb)")

## densityplot
## Original function to interface
densityplot(~ height | voice.part, data = singer, layout = c(2, 4),
            xlab = "Height (inches)", bw = 5)

## The interface reverses the order of data and formula
ntbt_densityplot(data = singer, ~ height | voice.part, layout = c(2, 4),
                 xlab = "Height (inches)", bw = 5)

## so it can be used easily in a pipeline.
singer %>%
  ntbt_densityplot(~ height | voice.part, layout = c(2, 4),
                   xlab = "Height (inches)", bw = 5)

## dotplot
## Original function to interface
dotplot(variety ~ yield | site, data = barley, groups = year,
        key = simpleKey(levels(barley$year), space = "right"),
        xlab = "Barley Yield (bushels/acre) ",
        aspect=0.5, layout = c(1,6), ylab=NULL)

## The interface reverses the order of data and formula
ntbt_dotplot(data = barley, variety ~ yield | site, groups = year,
             key = simpleKey(levels(barley$year), space = "right"),
             xlab = "Barley Yield (bushels/acre) ",
             aspect=0.5, layout = c(1,6), ylab=NULL)

## so it can be used easily in a pipeline.
barley %>%
  ntbt_dotplot(variety ~ yield | site, groups = year,
             key = simpleKey(levels(barley$year), space = "right"),
             xlab = "Barley Yield (bushels/acre) ",
             aspect=0.5, layout = c(1,6), ylab=NULL)

## histogram
```

```
## Original function to interface
histogram(~ height | voice.part, data = singer,
          xlab = "Height (inches)", type = "density",
          panel = function(x, ...) {
            panel.histogram(x, ...)
            panel.mathdensity(dmath = dnorm, col = "black",
                              args = list(mean=mean(x),sd=sd(x)))
          })

## The interface reverses the order of data and formula
ntbt_histogram(data = singer, ~ height | voice.part,
               xlab = "Height (inches)", type = "density",
               panel = function(x, ...) {
                 panel.histogram(x, ...)
                 panel.mathdensity(dmath = dnorm, col = "black",
                                   args = list(mean=mean(x),sd=sd(x)))
               })

## so it can be used easily in a pipeline.
singer %>%
  ntbt_histogram(~ height | voice.part,
                 xlab = "Height (inches)", type = "density",
                 panel = function(x, ...) {
                   panel.histogram(x, ...)
                   panel.mathdensity(dmath = dnorm, col = "black",
                                     args = list(mean=mean(x),sd=sd(x)))
                 })

## levelplot
x <- seq(pi/4, 5 * pi, length.out = 100)
y <- seq(pi/4, 5 * pi, length.out = 100)
r <- as.vector(sqrt(outer(x^2, y^2, "+")))
grid <- expand.grid(x = x, y = y)
grid$z <- cos(r^2) * exp(-r/(pi^3))

## Original function to interface
levelplot(z ~ x*y, grid, cuts = 50, scales = list(log = "e"), xlab = "",
          ylab = "", main = "Weird Function", sub = "with log scales",
          colorkey = FALSE, region = TRUE)

## The interface reverses the order of data and formula
ntbt_levelplot(grid, z ~ x*y, cuts = 50, scales = list(log = "e"), xlab = "",
               ylab = "", main = "Weird Function", sub = "with log scales",
               colorkey = FALSE, region = TRUE)

## so it can be used easily in a pipeline.
grid %>%
  ntbt_levelplot(z ~ x*y, cuts = 50, scales = list(log = "e"), xlab = "",
                 ylab = "", main = "Weird Function", sub = "with log scales",
                 colorkey = FALSE, region = TRUE)

## oneway
## Original function to interface
```

```
fit <- oneway(height ~ voice.part, data = singer, spread = 1)
rfs(fit, aspect = 1)

## The interface reverses the order of data and formula
fit <- ntbt_oneway(data = singer, height ~ voice.part, spread = 1)
rfs(fit, aspect = 1)

## so it can be used easily in a pipeline.
singer %>%
  ntbt_oneway(height ~ voice.part, spread = 1) %>%
  rfs(aspect = 1)

## parallelplot
## Original function to interface
parallelplot(~iris[1:4], iris, groups = Species,
             horizontal.axis = FALSE, scales = list(x = list(rot = 90)))

## The interface reverses the order of data and formula
ntbt_parallelplot(iris, ~iris[1:4], groups = Species,
                  horizontal.axis = FALSE, scales = list(x = list(rot = 90)))

## so it can be used easily in a pipeline.
iris %>%
  ntbt_parallelplot(~iris[1:4], groups = Species,
                    horizontal.axis = FALSE, scales = list(x = list(rot = 90)))

## qq
## Original function to interface
qq(voice.part ~ height, data = singer, aspect = 1,
   subset = (voice.part == "Bass 2" | voice.part == "Tenor 1"))

## The interface reverses the order of data and formula
ntbt_qq(data = singer, voice.part ~ height, aspect = 1,
        subset = (voice.part == "Bass 2" | voice.part == "Tenor 1"))

## so it can be used easily in a pipeline.
singer %>%
  ntbt_qq(voice.part ~ height, aspect = 1,
          subset = (voice.part == "Bass 2" | voice.part == "Tenor 1"))

## qqmath
## Original function to interface
qqmath(~ height | voice.part, data = singer, aspect = "xy",
       prepanel = prepanel.qqmathline,
       panel = function(x, ...) {
         panel.qqmathline(x, ...)
         panel.qqmath(x, ...)
       })

## The interface reverses the order of data and formula
ntbt_qqmath(data = singer, ~ height | voice.part, aspect = "xy",
            prepanel = prepanel.qqmathline,
            panel = function(x, ...) {
```

```
              panel.qqmathline(x, ...)
              panel.qqmath(x, ...)
             })

## so it can be used easily in a pipeline.
singer %>%
  ntbt_qqmath(~ height | voice.part, aspect = "xy",
              prepanel = prepanel.qqmathline,
              panel = function(x, ...) {
                panel.qqmathline(x, ...)
                panel.qqmath(x, ...)
              })

## splom
super.sym <- trellis.par.get("superpose.symbol")

## Original function to interface
splom(~ iris[1:4], data = iris, groups = Species,
      panel = panel.superpose,
      key = list(title = "Three Varieties of Iris",
                 columns = 3,
                 points = list(pch = super.sym$pch[1:3],
                               col = super.sym$col[1:3]),
                 text = list(c("Setosa", "Versicolor", "Virginica"))))
splom(~ iris[1:3] | Species, data = iris,
      layout=c(2,2), pscales = 0,
      varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"),
      page = function(...) {
        ltext(x = seq(.6, .8, length.out = 4),
              y = seq(.9, .6, length.out = 4),
              labels = c("Three", "Varieties", "of", "Iris"),
              cex = 2)
      })

## The interface reverses the order of data and formula
ntbt_splom(data = iris, ~ iris[1:4], groups = Species,
           panel = panel.superpose,
           key = list(title = "Three Varieties of Iris",
                      columns = 3,
                      points = list(pch = super.sym$pch[1:3],
                                    col = super.sym$col[1:3]),
                      text = list(c("Setosa", "Versicolor", "Virginica"))))
ntbt_splom(data = iris, ~ iris[1:3] | Species,
           layout=c(2,2), pscales = 0,
           varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"),
           page = function(...) {
             ltext(x = seq(.6, .8, length.out = 4),
                   y = seq(.9, .6, length.out = 4),
                   labels = c("Three", "Varieties", "of", "Iris"),
                   cex = 2)
           })

## so it can be used easily in a pipeline.
```

```
iris %>%
  ntbt_splom(~ iris[1:4], groups = Species,
             panel = panel.superpose,
             key = list(title = "Three Varieties of Iris",
                        columns = 3,
                        points = list(pch = super.sym$pch[1:3],
                                      col = super.sym$col[1:3]),
                        text = list(c("Setosa", "Versicolor", "Virginica"))))
iris %>%
  ntbt_splom(~ iris[1:3] | Species,
             layout=c(2,2), pscales = 0,
             varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"),
             page = function(...) {
               ltext(x = seq(.6, .8, length.out = 4),
                     y = seq(.9, .6, length.out = 4),
                     labels = c("Three", "Varieties", "of", "Iris"),
                     cex = 2)
             })

## stripplot
## Original function to interface
stripplot(voice.part ~ jitter(height), data = singer, aspect = 1,
          jitter.data = TRUE, xlab = "Height (inches)")

## The interface reverses the order of data and formula
ntbt_stripplot(data = singer, voice.part ~ jitter(height), aspect = 1,
          jitter.data = TRUE, xlab = "Height (inches)")

## so it can be used easily in a pipeline.
singer %>%
  ntbt_stripplot(voice.part ~ jitter(height), aspect = 1,
                 jitter.data = TRUE, xlab = "Height (inches)")

## tmd
## Original function to interface
tmd(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
    data = iris, scales = "free", layout = c(2, 2),
    auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## The interface reverses the order of data and formula
ntbt_tmd(data = iris,
         Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
         scales = "free", layout = c(2, 2),
         auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## so it can be used easily in a pipeline.
iris %>%
  ntbt_tmd(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
           scales = "free", layout = c(2, 2),
           auto.key = list(x = .6, y = .7, corner = c(0, 0)))


## wireframe
```

```
g <- expand.grid(x = 1:10, y = 5:15, gr = 1:2)
g$z <- log((g$x^g$gr + g$y^2) * g$gr)

## Original function to interface
wireframe(z ~ x * y, data = g, groups = gr,
          scales = list(arrows = FALSE),
          drape = TRUE, colorkey = TRUE,
          screen = list(z = 30, x = -60))

## The interface reverses the order of data and formula
ntbt_wireframe(data = g, z ~ x * y, groups = gr,
               scales = list(arrows = FALSE),
               drape = TRUE, colorkey = TRUE,
               screen = list(z = 30, x = -60))

## so it can be used easily in a pipeline.
g %>%
  ntbt_wireframe(z ~ x * y, groups = gr,
                 scales = list(arrows = FALSE),
                 drape = TRUE, colorkey = TRUE,
                 screen = list(z = 30, x = -60))

## xyplot
## Original function to interface
xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
       data = iris, scales = "free", layout = c(2, 2),
       auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## The interface reverses the order of data and formula
ntbt_xyplot(data = iris,
            Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
            scales = "free", layout = c(2, 2),
            auto.key = list(x = .6, y = .7, corner = c(0, 0)))

## so it can be used easily in a pipeline.
iris %>%
  ntbt_xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
              scales = "free", layout = c(2, 2),
              auto.key = list(x = .6, y = .7, corner = c(0, 0)))


## End(Not run)
```

---

`latticeExtra` *Interfaces for latticeExtra package for data science pipelines.*

---

## Description

Interfaces to `latticeExtra` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_ecdfplot(data, ...)
ntbt_mapplot(data, ...)
ntbt_rootogram(data, ...)
ntbt_segplot(data, ...)
ntbt_tileplot(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(latticeExtra)


## ntbt_ecdfplot: Trellis Displays of Empirical CDF
data(singer, package = "lattice")

## Original function to interface
ecdfplot(~height | voice.part, data = singer)

## The interface puts data as first parameter
ntbt_ecdfplot(singer, ~height | voice.part)

## so it can be used easily in a pipeline.
singer %>%
  ntbt_ecdfplot(~height | voice.part)


## ntbt_mapplot: Trellis displays on Maps a.k.a. Choropleth maps
library(maps)
library(mapproj)
data(USCancerRates)
```

```
## Original function to interface
## Note: Alaska, Hawaii and others are not included in county map;
## this generates warnings with both USCancerRates and ancestry.
suppressWarnings(print(

  mapplot(rownames(USCancerRates) ~ log(rate.male) + log(rate.female),
          data = USCancerRates,
          map = map("county", plot = FALSE, fill = TRUE,
                    projection = "mercator"))

))

## The interface puts data as first parameter
suppressWarnings(print(

 ntbt_mapplot(USCancerRates, rownames(USCancerRates) ~ log(rate.male) + log(rate.female),
              map = map("county", plot = FALSE, fill = TRUE,
                        projection = "mercator"))

 ))

## so it can be used easily in a pipeline.
suppressWarnings(print(

  USCancerRates %>%
  ntbt_mapplot(rownames(USCancerRates) ~ log(rate.male) + log(rate.female),
              map = map("county", plot = FALSE, fill = TRUE,
                        projection = "mercator"))

  ))


## ntbt_rootogram: Trellis Displays of Tukey's Hanging Rootograms
library(lattice)
dta <- data.frame(x = rpois(1000, lambda = 50))

## Original function to interface
rootogram(~x, data = dta, dfun = function(x) dpois(x, lambda = 50))

## The interface puts data as first parameter
ntbt_rootogram(dta, ~x, dfun = function(x) dpois(x, lambda = 50))

## so it can be used easily in a pipeline.
dta %>%
  ntbt_rootogram(~x, dfun = function(x) dpois(x, lambda = 50))


## ntbt_segplot: Plot segments using the Trellis framework
data(USCancerRates)

## Original function to interface
segplot(reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male,
        data = subset(USCancerRates, state == "Washington"))
```

```
## The interface puts data as first parameter
ntbt_segplot(subset(USCancerRates, state == "Washington"),
             reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male)

## so it can be used easily in a pipeline.
subset(USCancerRates, state == "Washington") %>%
  ntbt_segplot(reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male)

USCancerRates %>%
  subset(state == "Washington") %>%
  ntbt_segplot(reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male)


## ntbt_tileplot: Plot a spatial mosaic from irregular 2D points
tmp <- state.center
tmp$Income <- state.x77[,"Income"]
library(deldir)

## Original function to interface
tileplot(Income ~ x * y, tmp, border = "black",
         panel = function(x, y, ...) {
           panel.voronoi(x, y, ..., points = FALSE)
           panel.text(x, y, state.abb, cex = 0.6)
         })

## The interface puts data as first parameter
ntbt_tileplot(tmp, Income ~ x * y, border = "black",
              panel = function(x, y, ...) {
                panel.voronoi(x, y, ..., points = FALSE)
                panel.text(x, y, state.abb, cex = 0.6)
              })

## so it can be used easily in a pipeline.
tmp %>%
  ntbt_tileplot(Income ~ x * y, border = "black",
                panel = function(x, y, ...) {
                  panel.voronoi(x, y, ..., points = FALSE)
                  panel.text(x, y, state.abb, cex = 0.6)
                })

## End(Not run)
```

---

leaps *Interfaces for leaps package for data science pipelines.*

---

## Description

Interfaces to `leaps` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_regsubsets(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## See Also

regsubsets

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(leaps)

## Original function to interface
fit <- regsubsets(Fertility ~ ., data = swiss, nbest = 2)
summary(fit)

## The interface reverses the order of data and formula
fit <- ntbt_regsubsets(data = swiss, Fertility ~ ., nbest = 2)
summary(fit)

## so it can be used easily in a pipeline.
swiss %>%
  ntbt_regsubsets(Fertility ~ ., nbest = 2) %>%
  summary()

## End(Not run)
```

`lfe` *Interfaces for lfe package for data science pipelines.*

## Description

Interfaces to `lfe` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_felm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## See Also

`lfe`

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(lfe)

oldopts <- options(lfe.threads=1)

set.seed(123)
## create covariates
x <- rnorm(1000)
x2 <- rnorm(length(x))
## individual and firm
id <- factor(sample(20,length(x),replace=TRUE))
firm <- factor(sample(13,length(x),replace=TRUE))
## effects for them
```

```
id.eff <- rnorm(nlevels(id))
firm.eff <- rnorm(nlevels(firm))
## left hand side
u <- rnorm(length(x))
y <- x + 0.5*x2 + id.eff[id] + firm.eff[firm] + u
data <- data.frame(x = x, x2 = x2, id = id,
                   firm = firm, u = u, y = y)

## Original function to interface
est <- felm(y ~ x + x2 | id + firm, data)
summary(est)

## The interface reverses the order of data and formula
est <- ntbt_felm(data, y ~ x + x2 | id + firm)
summary(est)

## so it can be used easily in a pipeline.
data %>%
  ntbt_felm(y ~ x + x2 | id + firm) %>%
  summary()

## End(Not run)
```

---

`lme4` *Interfaces for lme4 package for data science pipelines.*

---

## Description

Interfaces to `lme4` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_glmer(data, ...)
ntbt_glmer.nb(data, ...)
ntbt_glFormula(data, ...)
ntbt_lFormula(data, ...)
ntbt_lmer(data, ...)
# ntbt_lmList(data, ...)   ## Already defined in nlme
ntbt_nlmer(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(lme4)

## ntbt_glmer: Fitting Generalized Linear Mixed-Effects Models
## Original function to interface
glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
      data = cbpp, family = binomial)

## The interface puts data as first parameter
ntbt_glmer(cbpp, cbind(incidence, size - incidence) ~ period + (1 | herd),
           family = binomial)

## so it can be used easily in a pipeline.
cbpp %>%
  ntbt_glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             family = binomial)


## ntbt_glmer.nb: Fitting Negative Binomial GLMMs
set.seed(101)
dd <- expand.grid(f1 = factor(1:3),
                  f2 = LETTERS[1:2], g=1:9, rep=1:15,
                  KEEP.OUT.ATTRS=FALSE)
summary(mu <- 5*(-4 + with(dd, as.integer(f1) + 4*as.numeric(f2))))
dd$y <- rnbinom(nrow(dd), mu = mu, size = 0.5)

## Original function to interface
glmer.nb(y ~ f1*f2 + (1|g), data = dd, verbose = FALSE)

## The interface puts data as first parameter
ntbt_glmer.nb(dd, y ~ f1*f2 + (1|g), verbose = FALSE)

## so it can be used easily in a pipeline.
dd %>%
  ntbt_glmer.nb(y ~ f1*f2 + (1|g), verbose = FALSE)


## ntbt_lmer: Fit Linear Mixed-Effects Models

## Original function to interface
lmer(Reaction ~ Days + (Days || Subject), sleepstudy)
```

```
## The interface puts data as first parameter
ntbt_lmer(sleepstudy, Reaction ~ Days + (Days || Subject))

## so it can be used easily in a pipeline.
sleepstudy %>%
  ntbt_lmer(Reaction ~ Days + (Days || Subject))


## ntbt_lmList: Fit List of lm Objects with a Common Model
## Original function to interface
lmList(Reaction ~ Days | Subject, sleepstudy)

## The interface puts data as first parameter
ntbt_lmList(sleepstudy, Reaction ~ Days | Subject)

## so it can be used easily in a pipeline.
sleepstudy %>%
  ntbt_lmList(Reaction ~ Days | Subject)


## Original function to interface
lFormula(Reaction ~ Days + (Days|Subject), sleepstudy)
glFormula(Reaction ~ Days + (Days|Subject), sleepstudy)

## The interface puts data as first parameter
ntbt_lFormula(sleepstudy, Reaction ~ Days + (Days|Subject))
ntbt_glFormula(sleepstudy, Reaction ~ Days + (Days|Subject))

## so it can be used easily in a pipeline.
sleepstudy %>%
  ntbt_lFormula(Reaction ~ Days + (Days|Subject))
sleepstudy %>%
  ntbt_glFormula(Reaction ~ Days + (Days|Subject))


## ntbt_nlmer: Fitting Nonlinear Mixed-Effects Models
## Original function to interface
nlmer(circumference ~ SSlogis(age, Asym, xmid, scal) ~ Asym|Tree,
      Orange, start = c(Asym = 200, xmid = 725, scal = 350))

## The interface puts data as first parameter
ntbt_nlmer(Orange, circumference ~ SSlogis(age, Asym, xmid, scal) ~ Asym|Tree,
           start = c(Asym = 200, xmid = 725, scal = 350))

## so it can be used easily in a pipeline.
Orange %>%
  ntbt_nlmer(circumference ~ SSlogis(age, Asym, xmid, scal) ~ Asym|Tree,
             start = c(Asym = 200, xmid = 725, scal = 350))

## End(Not run)
```

lmtest *Interfaces for lmtest package for data science pipelines.*

## Description

Interfaces to `lmtest` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_bgtest(data, ...)
ntbt_bptest(data, ...)
ntbt_coxtest(data, ...)
ntbt_dwtest(data, ...)
ntbt_encomptest(data, ...)
ntbt_gqtest(data, ...)
ntbt_grangertest(data, ...)
ntbt_harvtest(data, ...)
ntbt_hmctest(data, ...)
ntbt_jtest(data, ...)
ntbt_raintest(data, ...)
ntbt_resettest(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(lmtest)

## ntbt_bgtest: Breusch-Godfrey Test for higher-order serial correlation
x <- rep(c(1, -1), 50)
```

```
y1 <- 1 + x + rnorm(100)
dta <- data.frame(x, y1)

## or for fourth-order serial correlation
## Original function to interface
bgtest(y1 ~ x, order = 4, data = dta)

## The interface puts data as first parameter
ntbt_bgtest(dta, y1 ~ x, order = 4)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_bgtest(y1 ~ x, order = 4)


## ntbt_bptest: Breusch-Pagan test against heteroskedasticity
## ntbt_gqtest: Goldfeld-Quandt test against heteroskedasticity
## ntbt_hmctest: Harrison-McCabe test for heteroskedasticity
x <- rep(c(-1,1), 50)
err1 <- c(rnorm(50, sd=1), rnorm(50, sd=2))
err2 <- rnorm(100)
y1 <- 1 + x + err1
y2 <- 1 + x + err2
dtah <- data.frame(x, y1, y2)

## Original function to interface
bptest(y1 ~ x, data = dtah)
gqtest(y1 ~ x, data = dtah)
hmctest(y1 ~ x, data = dtah)
bptest(y2 ~ x, data = dtah)
gqtest(y2 ~ x, data = dtah)
hmctest(y2 ~ x, data = dtah)

## The interface puts data as first parameter
ntbt_bptest(dtah, y1 ~ x)
ntbt_gqtest(dtah, y1 ~ x)
ntbt_hmctest(dtah, y1 ~ x)
ntbt_bptest(dtah, y2 ~ x)
ntbt_gqtest(dtah, y2 ~ x)
ntbt_hmctest(dtah, y2 ~ x)

## so it can be used easily in a pipeline.
dtah %>%
  ntbt_bptest(y1 ~ x)
dtah %>%
  ntbt_gqtest(y1 ~ x)
dtah %>%
  ntbt_hmctest(y1 ~ x)
dtah %>%
  ntbt_bptest(y2 ~ x)
dtah %>%
  ntbt_gqtest(y2 ~ x)
dtah %>%
```

```
  ntbt_hmctest(y2 ~ x)


## ntbt_coxtest: Cox Test for Comparing Non-Nested Models
## ntbt_encomptest: encompassing test of Davidson & MacKinnon for comparing non-nested models
## ntbt_jtest: Davidson-MacKinnon J test for comparing non-nested models
data(USDistLag)
usdl <- na.contiguous(cbind(USDistLag, lag(USDistLag, k = -1)))
colnames(usdl) <- c("con", "gnp", "con1", "gnp1")

## Original function to interface
coxtest(con ~ gnp + con1, con ~ gnp + gnp1, data = usdl)
encomptest(con ~ gnp + con1, con ~ gnp + gnp1, data = usdl)
jtest(con ~ gnp + con1, con ~ gnp + gnp1, data = usdl)

## The interface puts data as first parameter
ntbt_coxtest(usdl, con ~ gnp + con1, con ~ gnp + gnp1)
ntbt_encomptest(usdl, con ~ gnp + con1, con ~ gnp + gnp1)
ntbt_jtest(usdl, con ~ gnp + con1, con ~ gnp + gnp1)

## so it can be used easily in a pipeline.
usdl %>%
  ntbt_coxtest(con ~ gnp + con1, con ~ gnp + gnp1)
usdl %>%
  ntbt_encomptest(con ~ gnp + con1, con ~ gnp + gnp1)
usdl %>%
  ntbt_jtest(con ~ gnp + con1, con ~ gnp + gnp1)

## ntbt_dwtest: Durbin-Watson test for autocorrelation of disturbances
err1 <- rnorm(100)
x <- rep(c(-1,1), 50)
y1 <- 1 + x + err1
err2 <- filter(err1, 0.9, method="recursive")
y2 <- 1 + x + err2
dta <- data.frame(y1, y2, x)

## Original function to interface
dwtest(y1 ~ x, data = dta)
dwtest(y2 ~ x, data = dta)

## The interface puts data as first parameter
ntbt_dwtest(dta, y1 ~ x)
ntbt_dwtest(dta, y2 ~ x)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_dwtest(y1 ~ x)
dta %>%
  ntbt_dwtest(y2 ~ x)


## ntbt_grangertest: Test for Granger Causality
data(ChickEgg)
```

```
## Original function to interface
grangertest(egg ~ chicken, order = 3, data = ChickEgg)
grangertest(chicken ~ egg, order = 3, data = ChickEgg)

## The interface puts data as first parameter
ntbt_grangertest(ChickEgg, egg ~ chicken, order = 3)
ntbt_grangertest(ChickEgg, chicken ~ egg, order = 3)

## so it can be used easily in a pipeline.
ChickEgg %>%
  ntbt_grangertest(egg ~ chicken, order = 3)
ChickEgg %>%
  ntbt_grangertest(chicken ~ egg, order = 3)


## ntbt_harvtest: Harvey-Collier test for linearity
x <- 1:50
y1 <- 1 + x + rnorm(50)
y2 <- y1 + 0.3*x^2
dta <- data.frame(y1, x)

## Original function to interface
harvtest(y1 ~ x, data = dta)

## The interface puts data as first parameter
ntbt_harvtest(dta, y1 ~ x)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_harvtest(y1 ~ x)

## ntbt_raintest: Rainbow test for linearity
x <- c(1:30)
y <- x^2 + rnorm(30,0,2)
dta <- data.frame(x, y)

## Original function to interface
raintest(y ~ x, data = dta)

## The interface puts data as first parameter
ntbt_raintest(dta, y ~ x)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_raintest(y ~ x)


## ntbt_resettest: Ramsey's RESET test for functional form
x <- c(1:30)
y1 <- 1 + x + x^2 + rnorm(30)
y2 <- 1 + x + rnorm(30)
dta <- data.frame(x, y1, y2)
```

```
## Original function to interface
resettest(y1 ~ x , power=2, type="regressor", data = dta)
resettest(y2 ~ x , power=2, type="regressor", data = dta)

## The interface puts data as first parameter
ntbt_resettest(dta, y1 ~ x , power=2, type="regressor")
ntbt_resettest(dta, y2 ~ x , power=2, type="regressor")

## so it can be used easily in a pipeline.
dta %>%
  ntbt_resettest(y1 ~ x , power=2, type="regressor")
dta %>%
  ntbt_resettest(y2 ~ x , power=2, type="regressor")

## End(Not run)
```

---

MASS *Interfaces for MASS package for data science pipelines.*

---

## Description

Interfaces to MASS functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_corresp(data, ...)
ntbt_glm.nb(data, ...)
ntbt_lda(data, ...)
ntbt_lm.gls(data, ...)
ntbt_lm.ridge(data, ...)
ntbt_loglm(data, ...)
ntbt_logtrans(data, ...)
ntbt_polr(data, ...)
ntbt_qda(data, ...)
ntbt_rlm(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(MASS)

## corresp
## Original function to interface
corresp(~ Age + Eth, data = quine)

## The interface reverses the order of data and formula
ntbt_corresp(data = quine, ~ Age + Eth)

## so it can be used easily in a pipeline.
quine %>%
  ntbt_corresp(~ Age + Eth)

## glm.nb
## Original function to interface
glm.nb(Days ~ Sex/(Age + Eth*Lrn), data = quine)

## The interface reverses the order of data and formula
ntbt_glm.nb(data = quine, Days ~ Sex/(Age + Eth*Lrn))

## so it can be used easily in a pipeline.
quine %>%
  ntbt_glm.nb(Days ~ Sex/(Age + Eth*Lrn))

## lda
Iris <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
                   Sp = rep(c("s","c","v"), rep(50,3)))

## Original function to interface
lda(Sp ~ ., Iris)

## The interface reverses the order of data and formula
ntbt_lda(Iris, Sp ~ .)

## so it can be used easily in a pipeline.
Iris %>%
  ntbt_lda(Sp ~ .)

stackloss %>%
  ntbt_lda(stack.loss ~ .) %>%
  summary()

## lm.gls
## Original function to interface
```

```
lm.gls(conc ~ uptake, CO2, W = diag(nrow(CO2)))

## The interface reverses the order of data and formula
ntbt_lm.gls(CO2, conc ~ uptake, W = diag(nrow(CO2)))

## so it can be used easily in a pipeline.
CO2 %>%
  ntbt_lm.gls(conc ~ uptake, W = diag(nrow(CO2)))

## lm.ridge
## Original function to interface
lm.ridge(GNP.deflator ~ ., longley)

## The interface reverses the order of data and formula
ntbt_lm.ridge(longley, GNP.deflator ~ .)

## so it can be used easily in a pipeline.
longley %>%
  ntbt_lm.ridge(GNP.deflator ~ .)

## loglm
## Original function to interface
xtCars93 <- xtabs(~ Type + Origin, Cars93)
loglm(~ Type + Origin, xtCars93)

## The interface reverses the order of data and formula
xtCars93 <- ntbt_xtabs(Cars93, ~ Type + Origin)
ntbt_loglm(xtCars93, ~ Type + Origin)

## so it can be used easily in a pipeline.
Cars93 %>%
  ntbt_xtabs(~ Type + Origin) %>%
  ntbt_loglm(~ Type + Origin)

## logtrans
## Original function to interface
logtrans(Days ~ Age*Sex*Eth*Lrn, data = quine,
         alpha = seq(0.75, 6.5, len=20))

## The interface reverses the order of data and formula
ntbt_logtrans(data = quine, Days ~ Age*Sex*Eth*Lrn,
              alpha = seq(0.75, 6.5, len=20))

## so it can be used easily in a pipeline.
quine %>%
  ntbt_logtrans(Days ~ Age*Sex*Eth*Lrn,
                alpha = seq(0.75, 6.5, len=20))

## polr
op <- options(contrasts = c("contr.treatment", "contr.poly"))

## Original function to interface
polr(Sat ~ Infl + Type + Cont, housing)
```

```
## The interface reverses the order of data and formula
ntbt_polr(housing, Sat ~ Infl + Type + Cont)

## so it can be used easily in a pipeline.
housing %>%
  ntbt_polr(Sat ~ Infl + Type + Cont)

options(op)

## qda
set.seed(123) ## make reproducible
tr <- sample(1:50, 25)

iris3df <- data.frame(cl = factor(c(rep("s",25), rep("c",25), rep("v",25))),
                      train = rbind(iris3[tr,,1], iris3[tr,,2], iris3[tr,,3]))

## Original function to interface
qda(cl ~ ., iris3df)

## The interface reverses the order of data and formula
ntbt_qda(iris3df, cl ~ .)

## so it can be used easily in a pipeline.
iris3df %>%
  ntbt_qda(cl ~ .)

## rlm
## Original function to interface
rlm(stack.loss ~ ., stackloss)

## The interface reverses the order of data and formula
ntbt_rlm(stackloss, stack.loss ~ .)

## so it can be used easily in a pipeline.
stackloss %>%
  ntbt_rlm(stack.loss ~ .) %>%
  summary()

stackloss %>%
  ntbt_rlm(stack.loss ~ ., psi = psi.hampel, init = "lts") %>%
  summary()

stackloss %>%
  ntbt_rlm(stack.loss ~ ., psi = psi.bisquare) %>%
  summary()

## End(Not run)
```

---

MCMCglmm *Interfaces for MCMCglmm package for data science pipelines.*

---

## Description

Interfaces to MCMCglmm functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_MCMCglmm(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(MCMCglmm)


## ntbt_MCMCglmm: Multivariate Generalised Linear Mixed Models
data(PlodiaPO)

## Original function to interface
set.seed(1)
model <- MCMCglmm(PO ~ 1, random = ~ FSfamily, data = PlodiaPO, verbose = FALSE)
summary(model)

## The interface puts data as first parameter
set.seed(1)
model <- ntbt_MCMCglmm(PlodiaPO, PO ~ 1, random = ~ FSfamily, verbose = FALSE)
summary(model)

## so it can be used easily in a pipeline.
set.seed(1)
PlodiaPO %>%
  ntbt_MCMCglmm(PO ~ 1, random = ~ FSfamily, verbose = FALSE) %>%
  summary()
```

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

---

mda *Interfaces for mda package for data science pipelines.*

---

## Description

Interfaces to mda functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_fda(data, ...)
ntbt_mda(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(mda)

## ntbt_fda: Flexible Discriminant Analysis
data(iris)
## Original function to interface
fda(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_fda(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_fda(Species ~ .)
```

```
## ntbt_mda: Mixture Discriminant Analysis
data(iris)
## Original function to interface
mda(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_mda(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_mda(Species ~ .)

## End(Not run)
```

---

`metafor` *Interfaces for metafor package for data science pipelines.*

---

## Description

Interfaces to `metafor` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_escalc(data, ...)
ntbt_rma(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(metafor)


## ntbt_escalc: Calculate Effect Sizes and Outcome Measures
data(dat.bcg)
dat.long <- to.long(measure = "RR", ai = tpos, bi = tneg, ci = cpos, di = cneg,
                    data = dat.bcg, append = FALSE, vlong = TRUE)

## Original function to interface
dat <- escalc(measure="RR", outcome ~ group | study, weights = freq, data = dat.long)
rma(yi, vi, data = dat)

## The interface puts data as first parameter
dat <- ntbt_escalc(dat.long, measure="RR", outcome ~ group | study, weights = freq)
ntbt_rma(dat, yi, vi)

## so it can be used easily in a pipeline.
dat.long %>%
  ntbt_escalc(measure="RR", outcome ~ group | study, weights = freq) %>%
  ntbt_rma(yi, vi)

## End(Not run)
```

---

`mgcv` *Interfaces for mgcv package for data science pipelines.*

---

## Description

Interfaces to `mgcv` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_bam(data, ...)
ntbt_gamm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(mgcv)

## ntbt_bam: Generalized additive models for very large datasets
set.seed(3)
dat <- gamSim(1,n=25000,dist="normal",scale=20)
bs <- "cr"
k <- 12

## Original function to interface
bam(y ~ s(x0, bs=bs) + s(x1, bs=bs) + s(x2, bs=bs, k=k) + s(x3, bs=bs), data = dat)

## The interface puts data as first parameter
ntbt_bam(dat, y ~ s(x0, bs=bs) + s(x1, bs=bs) + s(x2, bs=bs, k=k) + s(x3, bs=bs))

## so it can be used easily in a pipeline.
dat %>%
  ntbt_bam(y ~ s(x0, bs=bs) + s(x1, bs=bs) + s(x2, bs=bs, k=k) + s(x3, bs=bs))


## ntbt_gam: Generalized additive models with integrated smoothness estimation
set.seed(2) ## simulate some data...
dat <- gamSim(1, n = 400, dist = "normal", scale = 2)
## Original function to interface
gam(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = dat)

## The interface puts data as first parameter
ntbt_gam(dat, y ~ s(x0) + s(x1) + s(x2) + s(x3))

## so it can be used easily in a pipeline.
dat %>%
  ntbt_gam(y ~ s(x0) + s(x1) + s(x2) + s(x3))


## ntbt_gamm: Generalized Additive Mixed Models
set.seed(0)
dat <- gamSim(1, n = 200, scale = 2)

## Original function to interface
gamm(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = dat)
```

```
## The interface puts data as first parameter
ntbt_gamm(dat, y ~ s(x0) + s(x1) + s(x2) + s(x3))

## so it can be used easily in a pipeline.
dat %>%
  ntbt_gamm(y ~ s(x0) + s(x1) + s(x2) + s(x3))

## End(Not run)
```

---

`mhurdle` *Interfaces for mhurdle package for data science pipelines.*

---

## Description

Interfaces to `mhurdle` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_mhurdle(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(mhurdle)


## ntbt_mhurdle: Estimation of limited dependent variable models
data("Interview", package = "mhurdle")

## Original function to interface
```

```
mhurdle(vacations ~ car + size | linc + linc2 | 0, Interview,
        dist = "ln", h2 = TRUE, method = "bfgs")

## The interface puts data as first parameter
ntbt_mhurdle(Interview, vacations ~ car + size | linc + linc2 | 0,
             dist = "ln", h2 = TRUE, method = "bfgs")

## so it can be used easily in a pipeline.
Interview %>%
  ntbt_mhurdle(vacations ~ car + size | linc + linc2 | 0,
               dist = "ln", h2 = TRUE, method = "bfgs")

## End(Not run)
```

---

`minpack.lm` *Interfaces for minpack.lm package for data science pipelines.*

---

## Description

Interfaces to `minpack.lm` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_nlsLM(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(minpack.lm)

## ntbt_nlsLM: Standard 'nls' framework that uses 'nls.lm' for fitting
DNase1 <- subset(DNase, Run == 1)

## Original function to interface
nlsLM(density ~ Asym/(1 + exp((xmid - log(conc))/scal)),
      data = DNase1,
      start = list(Asym = 3, xmid = 0, scal = 1))

## The interface puts data as first parameter
ntbt_nlsLM(DNase1, density ~ Asym/(1 + exp((xmid - log(conc))/scal)),
           start = list(Asym = 3, xmid = 0, scal = 1))

## so it can be used easily in a pipeline.
DNase1 %>%
  ntbt_nlsLM(density ~ Asym/(1 + exp((xmid - log(conc))/scal)),
             start = list(Asym = 3, xmid = 0, scal = 1))

## End(Not run)
```

---

`mlogit` *Interfaces for mlogit package for data science pipelines.*

---

## Description

Interfaces to `mlogit` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_mlogit(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(mlogit)


## ntbt_mlogit: Multinomial logit model
data("Fishing", package = "mlogit")
Fish <- mlogit.data(Fishing, varying = c(2:9), shape = "wide", choice = "mode")

## Original function to interface
mlogit(mode ~ price + catch, data = Fish)

## The interface puts data as first parameter
ntbt_mlogit(Fish, mode ~ price + catch)

## so it can be used easily in a pipeline.
Fish %>%
  ntbt_mlogit(mode ~ price + catch)

## End(Not run)
```

---

`mnlogit` *Interfaces for mnlogit package for data science pipelines.*

---

## Description

Interfaces to `mnlogit` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_mnlogit(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(mnlogit)


## ntbt_mnlogit: Fast estimation of multinomial logit models
library(mnlogit)
data(Fish, package = "mnlogit")

## Original function to interface
mnlogit(mode ~ price | income | catch, Fish, ncores = 2)

## The interface puts data as first parameter
ntbt_mnlogit(Fish, mode ~ price | income | catch, ncores = 2)

## so it can be used easily in a pipeline.
Fish %>%
  ntbt_mnlogit(mode ~ price | income | catch, ncores = 2)

## End(Not run)
```

---

# modeltools *Interfaces for modeltools package for data science pipelines.*

---

## Description

Interfaces to `modeltools` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_ModelEnvFormula(data, ...)
ntbt_ParseFormula(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(modeltools)

data("iris")

## Original function to interface
ModelEnvFormula(Species + Petal.Width ~ . -1, data = iris,
                subset = sample(1:150, 10))
ParseFormula(Species + Petal.Width ~ . -1, data = iris)

## The interface puts data as first parameter
ntbt_ModelEnvFormula(iris, Species + Petal.Width ~ . -1, subset = sample(1:150, 10))
ntbt_ParseFormula(iris, Species + Petal.Width ~ . -1)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_ModelEnvFormula(Species + Petal.Width ~ . -1, subset = sample(1:150, 10))

iris %>%
  ntbt_ParseFormula(Species + Petal.Width ~ . -1)

## End(Not run)
```

---

# nlme *Interfaces for nlme package for data science pipelines.*

---

## Description

Interfaces to `nlme` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_gls(data, ...)
ntbt_lme(data, ...)
ntbt_lmList(data, ...)
ntbt_nlme(data, ...)
ntbt_nlsList(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(nlme)

## gls
## Original function to interface
fm1 <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
summary(fm1)

## The interface reverses the order of data and formula
fm1 <- ntbt_gls(Ovary, follicles ~ sin(2*pi*Time) + cos(2*pi*Time),
                correlation = corAR1(form = ~ 1 | Mare))
summary(fm1)

## so it can be used easily in a pipeline.
Ovary %>%
  ntbt_gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time),
                correlation = corAR1(form = ~ 1 | Mare))  %>%
  summary()

## nlme
## Original function to interface
lme(distance ~ age, data = Orthodont) # random is ~ age
```

```
lme(distance ~ age + Sex, data = Orthodont, random = ~ 1)

## The interface reverses the order of data and formula
ntbt_lme(data = Orthodont, distance ~ age) # random is ~ age
ntbt_lme(data = Orthodont, distance ~ age + Sex, random = ~ 1)

## so it can be used easily in a pipeline.
Orthodont %>%
  ntbt_lme(distance ~ age) # random is ~ age
Orthodont %>%
  ntbt_lme(distance ~ age + Sex, random = ~ 1)

## lmList
## Original function to interface
lmList(distance ~ age | Subject, Orthodont)

## The interface reverses the order of data and formula
ntbt_lmList(Orthodont, distance ~ age | Subject)

## so it can be used easily in a pipeline.
Orthodont %>%
  ntbt_lmList(distance ~ age | Subject)

Orthodont %>%
  ntbt_lmList(distance ~ age | Subject) %>%
  summary()

## nlme
## Original function to interface
fm1 <- nlme(height ~ SSasymp(age, Asym, R0, lrc),
            data = Loblolly,
            fixed = Asym + R0 + lrc ~ 1,
            random = Asym ~ 1,
            start = c(Asym = 103, R0 = -8.5, lrc = -3.3))
summary(fm1)

## The interface reverses the order of data and formula
fm1 <- ntbt_nlme(data = Loblolly,
                 height ~ SSasymp(age, Asym, R0, lrc),
                 fixed = Asym + R0 + lrc ~ 1,
                 random = Asym ~ 1,
                 start = c(Asym = 103, R0 = -8.5, lrc = -3.3))
summary(fm1)

## so it can be used easily in a pipeline.
Loblolly %>%
  ntbt_nlme(height ~ SSasymp(age, Asym, R0, lrc),
            fixed = Asym + R0 + lrc ~ 1,
            random = Asym ~ 1,
            start = c(Asym = 103, R0 = -8.5, lrc = -3.3)) %>%
  summary()

## End(Not run)
```

nlreg *Interfaces for nlreg package for data science pipelines.*

## Description

Interfaces to nlreg functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_nlreg(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(nlreg)


## ntbt_nlreg: Fit a Nonlinear Heteroscedastic Model via Maximum Likelihood
library(boot)
data(calcium)

## Original function to interface
nlreg(cal ~ b0*(1-exp(-b1*time)), weights = ~ ( 1+time^g )^2,
      start = c(b0 = 4, b1 = 0.1, g = 1), data = calcium, hoa = TRUE)

## The interface puts data as first parameter
ntbt_nlreg(calcium, cal ~ b0*(1-exp(-b1*time)), weights = ~ ( 1+time^g )^2,
           start = c(b0 = 4, b1 = 0.1, g = 1), hoa = TRUE)

## so it can be used easily in a pipeline.
```

```
calcium %>%
  ntbt_nlreg(cal ~ b0*(1-exp(-b1*time)), weights = ~ ( 1+time^g )^2,
             start = c(b0 = 4, b1 = 0.1, g = 1), hoa = TRUE)

## End(Not run)
```

---

nnet *Interfaces for nnet package for data science pipelines.*

---

## Description

Interfaces to nnet functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_multinom(data, ...)
ntbt_nnet(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(nnet)

## multinom
options(contrasts = c("contr.treatment", "contr.poly"))
library(MASS)
example(birthwt)

## Original function to interface
multinom(low ~ ., bwt)
```

```
## The interface reverses the order of data and formula
ntbt_multinom(bwt, low ~ .)

## so it can be used easily in a pipeline.
bwt %>%
  ntbt_multinom(low ~ .)

## nnet
ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3])
targets <- class.ind( c(rep("s", 50), rep("c", 50), rep("v", 50)))
set.seed(6789) ## for reproducible results
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
                  species = factor(c(rep("s",50), rep("c", 50), rep("v", 50))))

## Original function to interface
set.seed(12345) ## for reproducible results
nnet(species ~ ., data = ird, subset = samp,
     size = 2, rang = 0.1, decay = 5e-4, maxit = 200)

## The interface reverses the order of data and formula
set.seed(12345) ## for reproducible results
ntbt_nnet(data = ird, species ~ ., subset = samp,
          size = 2, rang = 0.1, decay = 5e-4, maxit = 200)

## so it can be used easily in a pipeline.
set.seed(12345) ## for reproducible results
ird %>%
  ntbt_nnet(species ~ ., subset = samp,
            size = 2, rang = 0.1, decay = 5e-4, maxit = 200)

## End(Not run)
```

---

`ordinal` *Interfaces for ordinal package for data science pipelines.*

---

## Description

Interfaces to `ordinal` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_clm(data, ...)
ntbt_clm2(data, ...)
ntbt_clmm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(ordinal)


## ntbt_clm: Cumulative Link Models
## ntbt_clm2:
## Original function to interface
clm(rating ~ temp * contact, data = wine)
clm2(rating ~ temp * contact, data = wine)

## The interface puts data as first parameter
ntbt_clm(wine, rating ~ temp * contact)
ntbt_clm2(wine, rating ~ temp * contact)

## so it can be used easily in a pipeline.
wine %>%
  ntbt_clm(rating ~ temp * contact)
wine %>%
  ntbt_clm2(rating ~ temp * contact)


## ntbt_clmm: Cumulative Link Mixed Models
## Original function to interface
clmm(SURENESS ~ PROD + (1|RESP) + (1|RESP:PROD), data = soup,
     link = "probit", threshold = "equidistant")

## The interface puts data as first parameter
ntbt_clmm(soup, SURENESS ~ PROD + (1|RESP) + (1|RESP:PROD),
          link = "probit", threshold = "equidistant")

## so it can be used easily in a pipeline.
soup %>%
  ntbt_clmm(SURENESS ~ PROD + (1|RESP) + (1|RESP:PROD),
            link = "probit", threshold = "equidistant")

## End(Not run)
```

party *Interfaces for party package for data science pipelines.*

## Description

Interfaces to party functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_cforest(data, ...)
ntbt_ctree(data, ...)
ntbt_mob(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(party)

## ntbt_cforest: Random Forest

### honest (i.e., out-of-bag) cross-classification of
### true vs. predicted classes
data("mammoexp", package = "TH.data")
#table(mammoexp$ME, predict(cforest(ME ~ ., data = mammoexp,
#                                   control = cforest_unbiased(ntree = 50)),
#                           OOB = TRUE))

## Original function to interface
set.seed(290875)
cforest(ME ~ ., data = mammoexp, control = cforest_unbiased(ntree = 50))
```

```
## The interface puts data as first parameter
set.seed(290875)
ntbt_cforest(mammoexp, ME ~ ., control = cforest_unbiased(ntree = 50))

## so it can be used easily in a pipeline.
set.seed(290875)
mammoexp %>%
  ntbt_cforest(ME ~ ., control = cforest_unbiased(ntree = 50))

## ntbt_ctree: Conditional Inference Trees
airq <- subset(airquality, !is.na(Ozone))

## Original function to interface
set.seed(290875)
ctree(Ozone ~ ., data = airq, controls = ctree_control(maxsurrogate = 3))

## The interface puts data as first parameter
set.seed(290875)
ntbt_ctree(airq, Ozone ~ ., controls = ctree_control(maxsurrogate = 3))

## so it can be used easily in a pipeline.
set.seed(290875)
airq %>%
  ntbt_ctree(Ozone ~ ., controls = ctree_control(maxsurrogate = 3))


## ntbt_mob: Model-based Recursive Partitioning
data("BostonHousing", package = "mlbench")
## and transform variables appropriately (for a linear regression)
BostonHousing$lstat <- log(BostonHousing$lstat)
BostonHousing$rm <- BostonHousing$rm^2
## as well as partitioning variables (for fluctuation testing)
BostonHousing$chas <- factor(BostonHousing$chas, levels = 0:1,
                             labels = c("no", "yes"))
BostonHousing$rad <- factor(BostonHousing$rad, ordered = TRUE)

## Original function to interface
set.seed(290875)
mob(medv ~ lstat + rm | zn + indus + chas + nox + age + dis + rad + tax + crim + b + ptratio,
    control = mob_control(minsplit = 40), data = BostonHousing,
    model = linearModel)

## The interface puts data as first parameter
set.seed(290875)
ntbt_mob(BostonHousing,
       medv ~ lstat + rm | zn + indus + chas + nox + age + dis + rad + tax + crim + b + ptratio,
         control = mob_control(minsplit = 40), model = linearModel)

## so it can be used easily in a pipeline.
set.seed(290875)
BostonHousing %>%
  ntbt_mob(medv ~ lstat + rm | zn + indus + chas + nox + age + dis + rad + tax + crim + b + ptratio,
```

```
            control = mob_control(minsplit = 40), model = linearModel)

## End(Not run)
```

---

partykit *Interfaces for partykit package for data science pipelines.*

---

## Description

Interfaces to `partykit` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
# Commented functions already defined for package party
# ntbt_cforest(data, ...)
# ntbt_ctree(data, ...)
ntbt_glmtree(data, ...)
ntbt_lmtree(data, ...)
# ntbt_mob(data, ...)
ntbt_palmtree(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(partykit)


## ntbt_cforest: Conditional Random Forests

## Original function to interface
```

```
cf <- cforest(dist ~ speed, data = cars)

## The interface puts data as first parameter
cf <- ntbt_cforest(cars, dist ~ speed)

## so it can be used easily in a pipeline.
cf <- cars %>%
  ntbt_cforest(dist ~ speed)


## ntbt_ctree: Conditional Inference Trees
airq <- subset(airquality, !is.na(Ozone))

## Original function to interface
airct <- ctree(Ozone ~ ., data = airq)
plot(airct)

## The interface puts data as first parameter
airct <- ntbt_ctree(airq, Ozone ~ .)
plot(airct)

## so it can be used easily in a pipeline.
airq %>%
  ntbt_ctree(Ozone ~ .) %>%
  plot()


## ntbt_glmtree: Generalized Linear Model Trees
data("PimaIndiansDiabetes", package = "mlbench")

## Original function to interface
pid_tree2 <- glmtree(diabetes ~ glucose | pregnant +
                       pressure + triceps + insulin + mass + pedigree + age,
                     data = PimaIndiansDiabetes, family = binomial)
plot(pid_tree2)

## The interface puts data as first parameter
pid_tree2 <- ntbt_glmtree(PimaIndiansDiabetes, diabetes ~ glucose | pregnant +
                            pressure + triceps + insulin + mass + pedigree + age,
                          family = binomial)
plot(pid_tree2)

## so it can be used easily in a pipeline.
PimaIndiansDiabetes %>%
  ntbt_glmtree(diabetes ~ glucose | pregnant +
                 pressure + triceps + insulin + mass + pedigree + age,
               family = binomial) %>%
  plot()


## ntbt_lmtree: Linear Model Trees
data("BostonHousing", package = "mlbench")
BostonHousing <-
```

```
  transform(BostonHousing,
            chas = factor(chas, levels = 0:1, labels = c("no", "yes")),
            rad = factor(rad, ordered = TRUE))

## Original function to interface
bh_tree <- lmtree(medv ~ log(lstat) + I(rm^2) | zn + indus + chas +
                    nox + age + dis + rad + tax + crim + b + ptratio,
                  data = BostonHousing, minsize = 40)
plot(bh_tree)

## The interface puts data as first parameter
bh_tree <- ntbt_lmtree(BostonHousing,
                       medv ~ log(lstat) + I(rm^2) | zn + indus + chas +
                         nox + age + dis + rad + tax + crim + b + ptratio,
                       minsize = 40)
plot(bh_tree)

## so it can be used easily in a pipeline.
BostonHousing %>%
  ntbt_lmtree(medv ~ log(lstat) + I(rm^2) | zn + indus + chas +
                nox + age + dis + rad + tax + crim + b + ptratio,
              minsize = 40) %>%
  plot()


## ntbt_mob: Model-based Recursive Partitioning
data("PimaIndiansDiabetes", package = "mlbench")

logit <- function(y, x, start = NULL, weights = NULL, offset = NULL, ...) {
  glm(y ~ 0 + x, family = binomial, start = start, ...)
}

## Original function to interface
pid_tree <- mob(diabetes ~ glucose | pregnant + pressure + triceps + insulin +
                  mass + pedigree + age, data = PimaIndiansDiabetes, fit = logit)
plot(pid_tree)

## The interface puts data as first parameter
pid_tree <- ntbt_mob(PimaIndiansDiabetes, diabetes ~ glucose | pregnant + pressure +
                       triceps + insulin + mass + pedigree + age, fit = logit)
plot(pid_tree)

## so it can be used easily in a pipeline.
PimaIndiansDiabetes %>%
  ntbt_mob(diabetes ~ glucose | pregnant + pressure +
             triceps + insulin + mass + pedigree + age, fit = logit) %>%
  plot()


## ntbt_palmtree: Partially Additive (Generalized) Linear Model Trees
dgp <- function(nobs = 1000, nreg = 5, creg = 0.4, ptreat = 0.5, sd = 1,
  coef = c(1, 0.25, 0.25, 0, 0, -0.25), eff = 1)
{
```

```
  d <- mvtnorm::rmvnorm(nobs,
    mean = rep(0, nreg),
    sigma = diag(1 - creg, nreg) + creg)
  colnames(d) <- paste0("x", 1:nreg)
  d <- as.data.frame(d)
  d$a <- rbinom(nobs, size = 1, prob = ptreat)
  d$err <- rnorm(nobs, mean = 0, sd = sd)

  gopt <- function(d) {
    as.numeric(d$x1 > -0.545) * as.numeric(d$x2 < 0.545)
  }
  d$y <- coef[1] + drop(as.matrix(d[, paste0("x", 1:5)]) %*% coef[-1]) -
    eff * (d$a - gopt(d))^2 + d$err
  d$a <- factor(d$a)
  return(d)
}
set.seed(1)
d <- dgp()

## Original function to interface
palm <- palmtree(y ~ a | x1 + x2 + x5 | x1 + x2 + x3 + x4 + x5, data = d)
plot(palm)

## The interface puts data as first parameter
palm <- ntbt_palmtree(d, y ~ a | x1 + x2 + x5 | x1 + x2 + x3 + x4 + x5)
plot(palm)

## so it can be used easily in a pipeline.
d %>%
  ntbt_palmtree(y ~ a | x1 + x2 + x5 | x1 + x2 + x3 + x4 + x5) %>%
  plot()

## End(Not run)
```

---

plotrix *Interfaces for plotrix package for data science pipelines.*

---

## Description

Interfaces to `plotrix` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_barNest(data, ...)
ntbt_brkdn.plot(data, ...)
ntbt_brkdnNest(data, ...)
ntbt_histStack(data, ...)
ntbt_plotH(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(plotrix)

## ntbt_barNest: Display a nested breakdown of numeric values
titanic<-data.frame(
  class=c(rep("1st",325),rep("2nd",285),rep("3rd",706),rep("Crew",885)),
  age=c(rep("Adult",319),rep("Child",6),rep("Adult",261),rep("Child",24),
        rep("Adult",627),rep("Child",79),rep("Adult",885)),
  sex=c(rep("M",175),rep("F",144),rep("M",5),rep("F",1),
        rep("M",168),rep("F",93),rep("M",11),rep("F",13),
        rep("M",462),rep("F",165),rep("M",48),rep("F",31),
        rep("M",862),rep("F",23)),
  survived=c(rep("Yes",57),rep("No",118),rep("Yes",140),rep("No",4),rep("Yes",6),
             rep("Yes",14),rep("No",154),rep("Yes",80),rep("No",13),rep("Yes",24),
             rep("Yes",75),rep("No",387),rep("Yes",76),rep("No",89),
             rep("Yes",13),rep("No",35),rep("Yes",14),rep("No",17),
             rep("Yes",192),rep("No",670),rep("Yes",20),rep("No",3)))
titanic.colors<-list("gray90",c("#0000ff","#7700ee","#aa00cc","#dd00aa"),
                     c("#ddcc00","#ee9900"),c("pink","lightblue"))

## Original function to interface
barNest(survived ~ class + age + sex, titanic, col = titanic.colors,
        showall = TRUE, main = "Titanic survival by class, age and sex",
        ylab = "Proportion surviving", FUN = c("propbrk","binciWu","binciWl","valid.n"),
        shrink = 0.15, trueval = "Yes")

## The interface puts data as first parameter
ntbt_barNest(titanic, survived ~ class + age + sex, col = titanic.colors,
             showall = TRUE, main = "Titanic survival by class, age and sex",
           ylab = "Proportion surviving", FUN = c("propbrk","binciWu","binciWl","valid.n"),
             shrink = 0.15, trueval = "Yes")
```

```
## so it can be used easily in a pipeline.
titanic %>%
  ntbt_barNest(survived ~ class + age + sex, col = titanic.colors,
               showall = TRUE, main = "Titanic survival by class, age and sex",
            ylab = "Proportion surviving", FUN = c("propbrk","binciWu","binciWl","valid.n"),
               shrink = 0.15, trueval = "Yes")


## ntbt_brkdn.plot: A point/line plotting routine
test.df<-data.frame(a=rnorm(80)+4,b=rnorm(80)+4,c=rep(LETTERS[1:4],each=20),
                    d=rep(rep(letters[1:4],each=4),5))
## Original function to interface
brkdn.plot("a", "c", "d", test.df, pch = 1:4, col = 1:4)

## The interface puts data as first parameter
ntbt_brkdn.plot(test.df, "a", "c", "d", pch = 1:4, col = 1:4)

## so it can be used easily in a pipeline.
test.df %>%
  ntbt_brkdn.plot("a", "c", "d", pch = 1:4, col = 1:4)


## ntbt_brkdnNest: Perform a nested breakdown of numeric values
brkdntest <- data.frame(Age=rnorm(100,25,10),
                        Sex=sample(c("M","F"),100,TRUE),
                        Marital=sample(c("M","X","S","W"),100,TRUE),
                        Employ=sample(c("FT","PT","NO"),100,TRUE))
## Original function to interface
brkdnNest(Age ~ Sex + Marital + Employ, data = brkdntest)

## The interface puts data as first parameter
ntbt_brkdnNest(brkdntest, Age ~ Sex + Marital + Employ)

## so it can be used easily in a pipeline.
brkdntest %>%
  ntbt_brkdnNest(Age ~ Sex + Marital + Employ)


## ntbt_histStack: Histogram "stacked" by categories
set.seed(409)
df <- data.frame(len=rnorm(100)+5,
                 grp=sample(c("A","B","C","D"),100,replace=TRUE))

## Original function to interface
histStack(len ~ grp, data = df, main = "Default (rainbow) colors",
          xlab = "Length category")

## The interface puts data as first parameter
ntbt_histStack(df, len ~ grp, main = "Default (rainbow) colors",
               xlab = "Length category")

## so it can be used easily in a pipeline.
```

```
df %>%
  ntbt_histStack(len ~ grp, main = "Default (rainbow) colors",
                 xlab = "Length category")


## ntbt_plotH: Scatterplot with histogram-like bars
d <- data.frame(x=c(1,5,10:20),y=runif(13)+1,
                g=factor(sample(c("A","B","C"),13,replace=TRUE)))

## Original function to interface
plotH(y ~ x, data = d)

## The interface puts data as first parameter
ntbt_plotH(d, y ~ x)

## so it can be used easily in a pipeline.
d %>%
  ntbt_plotH(y ~ x)

## End(Not run)
```

---

`pls` *Interfaces for pls package for data science pipelines.*

---

## Description

Interfaces to `pls` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_cppls(data, ...)
ntbt_mvr(data, ...)
ntbt_pcr(data, ...)
ntbt_plsr(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(pls)

## cppls
## Original function to interface
yarn.cppls <- cppls(density ~ NIR, ncomp = 6, data = yarn, validation = "CV")
summary(yarn.cppls)

## The interface reverses the order of data and formula
yarn.cppls <- ntbt_cppls(yarn, density ~ NIR, ncomp = 6, validation = "CV")
summary(yarn.cppls)

## so it can be used easily in a pipeline.
yarn %>%
  ntbt_cppls(density ~ NIR, ncomp = 6, validation = "CV") %>%
  summary()

## mvr
## Original function to interface
yarn.mvr <- mvr(density ~ NIR, ncomp = 6, data = yarn, validation = "CV",
                method = "oscorespls")
summary(yarn.mvr)

## The interface reverses the order of data and formula
yarn.mvr <- ntbt_mvr(yarn, density ~ NIR, ncomp = 6, validation = "CV",
                     method = "oscorespls")
summary(yarn.mvr)

## so it can be used easily in a pipeline.
yarn %>%
  ntbt_mvr(density ~ NIR, ncomp = 6, validation = "CV",
           method = "oscorespls") %>%
  summary()

## pcr
## Original function to interface
yarn.pcr <- pcr(density ~ NIR, ncomp = 6, data = yarn, validation = "CV")
summary(yarn.pcr)

## The interface reverses the order of data and formula
yarn.pcr <- ntbt_pcr(yarn, density ~ NIR, ncomp = 6, validation = "CV")
summary(yarn.pcr)

## so it can be used easily in a pipeline.
yarn %>%
```

```
  ntbt_pcr(density ~ NIR, ncomp = 6, validation = "CV") %>%
  summary()

## plsr
## Original function to interface
yarn.plsr <- plsr(density ~ NIR, ncomp = 6, data = yarn, validation = "CV")
summary(yarn.plsr)

## The interface reverses the order of data and formula
yarn.plsr <- ntbt_plsr(yarn, density ~ NIR, ncomp = 6, validation = "CV")
summary(yarn.plsr)

## so it can be used easily in a pipeline.
yarn %>%
  ntbt_plsr(density ~ NIR, ncomp = 6, validation = "CV") %>%
  summary()

## End(Not run)
```

---

pROC *Interfaces for pROC package for data science pipelines.*

---

## Description

Interfaces to pROC functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_auc(data, ...)
ntbt_ci(data, ...)
ntbt_ci.auc(data, ...)
ntbt_ci.coords(data, ...)
ntbt_ci.se(data, ...)
ntbt_ci.sp(data, ...)
ntbt_ci.thresholds(data, ...)
ntbt_multiclass.roc(data, ...)
ntbt_plot.roc(data, ...)
ntbt_roc(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(pROC)

## NOTE: pROC examples below use both formula and non-formula variants.
##       In examples for other packages, almost always only
##       the formula variant is shown, but in those cases also
##       the non-formula variants should work.

## ntbt_auc: Compute the area under the ROC curve
data(aSAH)

## Original function to interface
auc(outcome ~ s100b, data = aSAH)
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
auc(outcome, s100b)
detach()
## or use $
auc(aSAH$outcome, aSAH$s100b)

## The interface puts data as first parameter
## NOTE: in this case the formula version fails, and I have found no
##       way to trick auc into accepting the formula (so far).
##       Maybe (only maybe) there is a problem with auc, as formula
##       variant may not be used, so it was probably not
##       reported as a bug before. The rest of the interfaced
##       functions seem to work fine.
## ntbt_auc(data = aSAH, outcome ~ s100baSAH)
## The non-formula variant works fine
ntbt_auc(aSAH, outcome, s100b)

## so it can be used easily in a pipeline.
#aSAH %>%
#  ntbt_auc(outcome ~ s100baSAH)
aSAH %>%
  ntbt_auc(outcome, s100b)


## ntbt_ci: Compute the confidence interval of a ROC curve
## Original function to interface
```

```
ci(outcome ~ s100b, data = aSAH)
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
ci(outcome, s100b)
detach()
## or use $
ci(aSAH$outcome, aSAH$s100b)

## The interface puts data as first parameter
ntbt_ci(aSAH, outcome ~ s100b)
ntbt_ci(aSAH, outcome, s100b)

## so it can be used easily in a pipeline.
aSAH %>%
  ntbt_ci(outcome ~ s100b)
aSAH %>%
  ntbt_ci(outcome, s100b)


## ci.auc: Compute the confidence interval of the AUC
## Original function to interface
ci.auc(outcome ~ s100b, data = aSAH)
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
ci.auc(outcome, s100b)
detach()
## or use $
ci.auc(aSAH$outcome, aSAH$s100b)

## The interface puts data as first parameter
ntbt_ci.auc(aSAH, outcome ~ s100b)
ntbt_ci.auc(aSAH, outcome, s100b)

## so it can be used easily in a pipeline.
aSAH %>%
  ntbt_ci.auc(outcome ~ s100b)
aSAH %>%
  ntbt_ci.auc(outcome, s100b)


## ntbt_ci.coords: Compute the confidence interval of arbitrary coordinates
## Original function to interface
set.seed(1)
ci.coords(outcome ~ s100b, data = aSAH, x="best", input = "threshold",
          ret=c("specificity", "ppv", "tp"))
set.seed(1)
ci.coords(aSAH$outcome, aSAH$s100b, x="best", input = "threshold",
          ret=c("specificity", "ppv", "tp"))
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
```

```
set.seed(1)
ci.coords(outcome, s100b, x="best", input = "threshold",
          ret=c("specificity", "ppv", "tp"))
detach()
## or use $
set.seed(1)
ci.coords(aSAH$outcome, aSAH$s100b, x="best", input = "threshold",
          ret=c("specificity", "ppv", "tp"))

## The interface puts data as first parameter
set.seed(1)
ntbt_ci.coords(aSAH, outcome ~ s100b, x="best", input = "threshold",
               ret=c("specificity", "ppv", "tp"))
set.seed(1)
ntbt_ci.coords(aSAH, outcome, s100b, x="best", input = "threshold",
               ret=c("specificity", "ppv", "tp"))

## so it can be used easily in a pipeline.
set.seed(1)
aSAH %>%
  ntbt_ci.coords(outcome ~ s100b, x="best", input = "threshold",
                 ret=c("specificity", "ppv", "tp"))
set.seed(1)
aSAH %>%
  ntbt_ci.coords(outcome, s100b, x="best", input = "threshold",
                 ret=c("specificity", "ppv", "tp"))


## ntbt_ci.se: Compute the confidence interval of sensitivities at given specificities
## Original function to interface
set.seed(1)
ci.se(outcome ~ s100b, data = aSAH)
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
set.seed(1)
ci.se(outcome, s100b)
detach()
## or use $
set.seed(1)
ci.se(aSAH$outcome, aSAH$s100b)

## The interface puts data as first parameter
set.seed(1)
ntbt_ci.se(aSAH, outcome ~ s100b)
set.seed(1)
ntbt_ci.se(aSAH, outcome, s100b)

## so it can be used easily in a pipeline.
set.seed(1)
aSAH %>%
  ntbt_ci.se(outcome ~ s100b)
set.seed(1)
```

```
aSAH %>%
  ntbt_ci.se(outcome, s100b)


## ntbt_ci.sp: Compute the confidence interval of specificities at given sensitivities
## Original function to interface
set.seed(1)
ci.sp(outcome ~ s100b, data = aSAH)
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
set.seed(1)
ci.sp(outcome, s100b)
detach()
## or use $
set.seed(1)
ci.sp(aSAH$outcome, aSAH$s100b)

## The interface puts data as first parameter
set.seed(1)
ntbt_ci.sp(aSAH, outcome ~ s100b)
set.seed(1)
ntbt_ci.sp(aSAH, outcome, s100b)

## so it can be used easily in a pipeline.
set.seed(1)
aSAH %>%
  ntbt_ci.sp(outcome ~ s100b, x="best", input = "threshold",
             ret=c("specificity", "ppv", "tp"))
set.seed(1)
aSAH %>%
  ntbt_ci.sp(outcome, s100b, x="best", input = "threshold",
             ret=c("specificity", "ppv", "tp"))


## ntbt_ci.thresholds: Compute the confidence interval of thresholds
## Original function to interface
set.seed(1)
ci.thresholds(outcome ~ s100b, data = aSAH)
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
set.seed(1)
ci.thresholds(outcome, s100b)
detach()
## or use $
set.seed(1)
ci.thresholds(aSAH$outcome, aSAH$s100b)

## The interface puts data as first parameter
set.seed(1)
ntbt_ci.thresholds(aSAH, outcome ~ s100b)
set.seed(1)
```

```
ntbt_ci.thresholds(aSAH, outcome, s100b)

## so it can be used easily in a pipeline.
set.seed(1)
aSAH %>%
  ntbt_ci.thresholds(outcome ~ s100b)
set.seed(1)
aSAH %>%
  ntbt_ci.thresholds(outcome, s100b)


## ntbt_multiclass.roc: Multi-clmulticlass.roc Multi-class AUCass AUC
## Original function to interface
multiclass.roc(gos6 ~ s100b, data = aSAH, levels = c(3, 4, 5))
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
multiclass.roc(gos6, s100b, levels = c(3, 4, 5))
detach()
## or use $
multiclass.roc(aSAH$gos6, aSAH$s100b, levels = c(3, 4, 5))

## The interface puts data as first parameter
ntbt_multiclass.roc(aSAH, gos6 ~ s100b, levels = c(3, 4, 5))
ntbt_multiclass.roc(aSAH, gos6, s100b, levels = c(3, 4, 5))

## so it can be used easily in a pipeline.
aSAH %>%
  ntbt_multiclass.roc(gos6 ~ s100b, levels = c(3, 4, 5))
aSAH %>%
  ntbt_multiclass.roc(gos6, s100b, levels = c(3, 4, 5))


## ntbt_plot.roc: Plot a ROC curve
## Original function to interface
plot.roc(outcome ~ s100b, data = aSAH, type="b", pch=21, col="blue", bg="grey")
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
plot.roc(outcome, s100b, type="b", pch=21, col="blue", bg="grey")
detach()
## or use $
plot.roc(aSAH$outcome, aSAH$s100b, type="b", pch=21, col="blue", bg="grey")

## The interface puts data as first parameter
ntbt_plot.roc(aSAH, outcome ~ s100b, type="b", pch=21, col="blue", bg="grey")
ntbt_plot.roc(aSAH, outcome, s100b, type="b", pch=21, col="blue", bg="grey")

## so it can be used easily in a pipeline.
aSAH %>%
  ntbt_plot.roc(outcome ~ s100b, type="b", pch=21, col="blue", bg="grey")
aSAH %>%
  ntbt_plot.roc(outcome, s100b, type="b", pch=21, col="blue", bg="grey")
```

```
## ntbt_roc: Build a ROC curve
## Original function to interface
roc(outcome ~ s100b, data = aSAH, type="b", pch=21, col="blue", bg="grey")
## For non-formula variants, either:
## 1) need to attach
attach(aSAH)
roc(outcome, s100b, type="b", pch=21, col="blue", bg="grey")
detach()
## or use $
roc(aSAH$outcome, aSAH$s100b, type="b", pch=21, col="blue", bg="grey")

## The interface puts data as first parameter
ntbt_roc(aSAH, outcome ~ s100b, type="b", pch=21, col="blue", bg="grey")
ntbt_roc(aSAH, outcome, s100b, type="b", pch=21, col="blue", bg="grey")

## so it can be used easily in a pipeline.
aSAH %>%
  ntbt_roc(outcome ~ s100b, type="b", pch=21, col="blue", bg="grey")
aSAH %>%
  ntbt_roc(outcome, s100b, type="b", pch=21, col="blue", bg="grey")

## End(Not run)
```

---

`pscl` *Interfaces for pscl package for data science pipelines.*

---

## Description

Interfaces to `pscl` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_hurdle(data, ...)
ntbt_zeroinfl(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

### Author(s)

Roberto Bertolusso

### Examples

```
## Not run:
library(intubate)
library(magrittr)
library(pscl)


## ntbt_hurdle: Hurdle Models for Count Data Regression
data("bioChemists", package = "pscl")

## Original function to interface
hurdle(art ~ ., data = bioChemists)

## The interface puts data as first parameter
ntbt_hurdle(bioChemists, art ~ .)

## so it can be used easily in a pipeline.
bioChemists %>%
  ntbt_hurdle(art ~ .)


## ntbt_zeroinfl: Zero-inflated Count Data Regression
## Original function to interface
zeroinfl(art ~ . | 1, data = bioChemists, dist = "negbin")

## The interface puts data as first parameter
ntbt_zeroinfl(bioChemists, art ~ . | 1, dist = "negbin")

## so it can be used easily in a pipeline.
bioChemists %>%
  ntbt_zeroinfl(art ~ . | 1, dist = "negbin")

## End(Not run)
```

---

## psychomix *Interfaces for psychomix package for data science pipelines.*

---

### Description

Interfaces to psychomix functions that can be used in a pipeline implemented by magrittr.

### Usage

```
ntbt_btmix(data, ...)
ntbt_raschmix(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(psychomix)


## ntbt_btmix: Finite Mixtures of Bradley-Terry Models
data("GermanParties2009", package = "psychotools")

## omit single observation with education = 1
gp <- subset(GermanParties2009, education != "1")
gp$education <- factor(gp$education)

## Original function to interface
set.seed(1)
cm <- btmix(preference ~ gender + education + age + crisis,
            data = gp, k = 1:4, nrep = 3)
plot(cm)

## The interface puts data as first parameter
set.seed(1)
cm <- ntbt_btmix(gp, preference ~ gender + education + age + crisis,
                 k = 1:4, nrep = 3)
plot(cm)

## so it can be used easily in a pipeline.
set.seed(1)
gp %>%
  ntbt_btmix(preference ~ gender + education + age + crisis, k = 1:4, nrep = 3) %>%
  plot()
```

```
## ntbt_raschmix: Finite Mixtures of Rasch Models
set.seed(1)
r2 <- simRaschmix(design = "rost2")
d <- data.frame(
  x1 = rbinom(nrow(r2), prob = c(0.4, 0.6)[attr(r2, "cluster")], size = 1),
  x2 = rnorm(nrow(r2))
)
d$resp <- r2

## Original function to interface
m1 <- raschmix(resp ~ 1, data = d, k = 1:3, score = "saturated")
plot(m1)

## The interface puts data as first parameter
m1 <- ntbt_raschmix(d, resp ~ 1, k = 1:3, score = "saturated")
plot(m1)

## so it can be used easily in a pipeline.
d %>%
  ntbt_raschmix(resp ~ 1, k = 1:3, score = "saturated") %>%
  plot()

## End(Not run)
```

---

psychotools *Interfaces for psychotools package for data science pipelines.*

---

## Description

Interfaces to psychotools functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_anchor(data, ...)
ntbt_anchortest(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(psychotools)


## ntbt_anchor: Anchor Methods for the Detection of Uniform DIF the Rasch Model
data("VerbalAggression", package = "psychotools")

## Original function to interface
anchor(resp2[, 1:12] ~ gender, data = VerbalAggression,
       class = "forward", select = "MTT", range = c(0.05, 1))

## The interface puts data as first parameter
ntbt_anchor(VerbalAggression, resp2[, 1:12] ~ gender,
            class = "forward", select = "MTT", range = c(0.05, 1))

## so it can be used easily in a pipeline.
VerbalAggression %>%
  ntbt_anchor(resp2[, 1:12] ~ gender,
              class = "forward", select = "MTT", range = c(0.05, 1))


## ntbt_anchortest: Anchor methods for the detection of uniform DIF the Rasch model

## Original function to interface
anchortest(resp2[, 1:12] ~ gender, data = VerbalAggression,
           class = "forward", select = "MTT", range = c(0.05, 1))

## The interface puts data as first parameter
ntbt_anchortest(VerbalAggression, resp2[, 1:12] ~ gender,
                class = "forward", select = "MTT", range = c(0.05, 1))

## so it can be used easily in a pipeline.
VerbalAggression %>%
  ntbt_anchortest(resp2[, 1:12] ~ gender,
                  class = "forward", select = "MTT", range = c(0.05, 1))

## End(Not run)
```

---

psychotree *Interfaces for psychotree package for data science pipelines.*

---

## Description

Interfaces to psychotree functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_bttree(data, ...)
ntbt_mpttree(data, ...)
ntbt_pctree(data, ...)
ntbt_raschtree(data, ...)
ntbt_rstree(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(psychotree)


## ntbt_bttree: Bradley-Terry Tree Models
data("Topmodel2007", package = "psychotree")

## Original function to interface
tm_tree <- bttree(preference ~ ., data = Topmodel2007,
                  minsize = 5, ref = "Barbara")
plot(tm_tree, abbreviate = 1, yscale = c(0, 0.5))

## The interface puts data as first parameter
tm_tree <- ntbt_bttree(Topmodel2007, preference ~ .,
                       minsize = 5, ref = "Barbara")
plot(tm_tree, abbreviate = 1, yscale = c(0, 0.5))

## so it can be used easily in a pipeline.
Topmodel2007 %>%
```

```
  ntbt_bttree(preference ~ ., minsize = 5, ref = "Barbara") %>%
  plot(abbreviate = 1, yscale = c(0, 0.5))



## ntbt_mpttree: MPT Tree Models
data("SourceMonitoring", package="psychotools")

## Original function to interface
sm_tree <- mpttree(y ~ sources + gender + age, data = SourceMonitoring,
                   spec = mptspec("SourceMon", .restr = list(d1 = d, d2 = d)))
plot(sm_tree, index = c("D1", "D2", "d", "b", "g"))

## The interface puts data as first parameter
sm_tree <- ntbt_mpttree(SourceMonitoring, y ~ sources + gender + age,
                        spec = mptspec("SourceMon", .restr = list(d1 = d, d2 = d)))
plot(sm_tree, index = c("D1", "D2", "d", "b", "g"))

## so it can be used easily in a pipeline.
SourceMonitoring %>%
  ntbt_mpttree(y ~ sources + gender + age,
               spec = mptspec("SourceMon", .restr = list(d1 = d, d2 = d))) %>%
  plot(index = c("D1", "D2", "d", "b", "g"))



## ntbt_pctree: Partial Credit Tree Models
data("VerbalAggression", package = "psychotools")
VerbalAggression$s2 <- VerbalAggression$resp[, 7:12]
VerbalAggression <- subset(VerbalAggression, rowSums(s2) > 0 & rowSums(s2) < 12)

## Original function to interface
pct <- pctree(s2 ~ anger + gender, data = VerbalAggression)
plot(pct, type = "profile")

## The interface puts data as first parameter
pct <- ntbt_pctree(VerbalAggression, s2 ~ anger + gender)
plot(pct, type = "profile")

## so it can be used easily in a pipeline.
VerbalAggression %>%
  ntbt_pctree(s2 ~ anger + gender) %>%
  plot(type = "profile")



## ntbt_raschtree: Rasch Tree Models
data("DIFSim", package = "psychotree")

## Original function to interface
rt <- raschtree(resp ~ age + gender + motivation, data = DIFSim)
plot(rt)
```

```
## The interface puts data as first parameter
rt <- ntbt_raschtree(DIFSim, resp ~ age + gender + motivation)
plot(rt)

## so it can be used easily in a pipeline.
DIFSim %>%
  ntbt_raschtree(resp ~ age + gender + motivation) %>%
  plot()



## ntbt_rstree: Rating Scale Tree Models
data("VerbalAggression", package = "psychotools")
VerbalAggression$s1 <- VerbalAggression$resp[, 1:6]
VerbalAggression <- subset(VerbalAggression, rowSums(s1) > 0 & rowSums(s1) < 12)

## Original function to interface
rst <- rstree(s1 ~ anger + gender, data = VerbalAggression)
plot(rst, type = "profile")

## The interface puts data as first parameter
rst <- ntbt_rstree(VerbalAggression, s1 ~ anger + gender)
plot(rst, type = "profile")

## so it can be used easily in a pipeline.
VerbalAggression %>%
  ntbt_rstree(s1 ~ anger + gender) %>%
  plot(type = "profile")

## End(Not run)
```

---

quantreg *Interfaces for quantreg package for data science pipelines.*

---

## Description

Interfaces to quantreg functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_dynrq(data, ...)
ntbt_KhmaladzeTest(data, ...)
ntbt_nlrq(data, ...)
ntbt_rq(data, ...)
ntbt_rqProcess(data, ...)
ntbt_rqss(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(quantreg)


## ntbt_dynrq: Dynamic Linear Quantile Regression
require(zoo)
data("UKDriverDeaths", package = "datasets")
dta <- data.frame(uk = log10(UKDriverDeaths))

## Original function to interface
dynrq(uk ~ L(uk, 1) + L(uk, 12), data = dta)

## The interface puts data as first parameter
ntbt_dynrq(dta, uk ~ L(uk, 1) + L(uk, 12))

## so it can be used easily in a pipeline.
dta %>%
  ntbt_dynrq(uk ~ L(uk, 1) + L(uk, 12))


## ntbt_KhmaladzeTest: Tests of Location and Location Scale Shift Hypotheses for Linear Models
data(barro)
## Original function to interface
KhmaladzeTest(y.net ~ lgdp2 + fse2 + gedy2 + Iy2 + gcony2,
              data = barro, taus = seq(.05,.95,by = .01))

## The interface puts data as first parameter
ntbt_KhmaladzeTest(barro, y.net ~ lgdp2 + fse2 + gedy2 + Iy2 + gcony2,
                   taus = seq(.05,.95,by = .01))

## so it can be used easily in a pipeline.
barro %>%
  ntbt_KhmaladzeTest(y.net ~ lgdp2 + fse2 + gedy2 + Iy2 + gcony2,
                     taus = seq(.05,.95,by = .01))


## ntbt_nlrq: Function to compute nonlinear quantile regression estimates
```

```
Dat <- NULL; Dat$x <- rep(1:25, 20)
set.seed(1)
Dat$y <- SSlogis(Dat$x, 10, 12, 2)*rnorm(500, 1, 0.1)

## Original function to interface
nlrq(y ~ SSlogis(x, Asym, mid, scal), data = Dat, tau = 0.5, trace = TRUE)

## The interface puts data as first parameter
ntbt_nlrq(Dat, y ~ SSlogis(x, Asym, mid, scal), tau = 0.5, trace = TRUE)

## so it can be used easily in a pipeline.
Dat %>%
  ntbt_nlrq(y ~ SSlogis(x, Asym, mid, scal), tau = 0.5, trace = TRUE)


## ntbt_rq: Quantile Regression
data(stackloss)
dta <- data.frame(stack.loss, stack.x)

## Original function to interface
rq(stack.loss ~ stack.x, .5, data = dta)  # median (l1) regression  fit for the stackloss data.

## The interface puts data as first parameter
ntbt_rq(dta, stack.loss ~ stack.x, .5)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_rq(stack.loss ~ stack.x, .5)


## ntbt_rqProcess: Compute Standardized Quantile Regression Process
## Original function to interface
data(barro)
rqProcess(y.net ~ lgdp2 + fse2 + gedy2 + Iy2 + gcony2,
          data = barro, taus = seq(.05,.95,by = .01))

## The interface puts data as first parameter
ntbt_rqProcess(barro, y.net ~ lgdp2 + fse2 + gedy2 + Iy2 + gcony2,
               taus = seq(.05,.95,by = .01))

## so it can be used easily in a pipeline.
barro %>%
  ntbt_rqProcess(y.net ~ lgdp2 + fse2 + gedy2 + Iy2 + gcony2,
                 taus = seq(.05,.95,by = .01))


## ntbt_rqss: Additive Quantile Regression Smoothing
n <- 200
x <- sort(rchisq(n,4))
z <- x + rnorm(n)
y <- log(x)+ .1*(log(x))^2 + log(x)*rnorm(n)/4 + z
dta <- data.frame(x, y, z)
```

```
## Original function to interface
rqss(y ~ qss(x, constraint= "N") + z, data = dta)

## The interface puts data as first parameter
ntbt_rqss(dta, y ~ qss(x, constraint= "N") + z)

## so it can be used easily in a pipeline.
dta %>%
  ntbt_rqss(y ~ qss(x, constraint= "N") + z)

## End(Not run)
```

---

`randomForest` *Interfaces for randomForest package for data science pipelines.*

---

## Description

Interfaces to `randomForest` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_randomForest(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(randomForest)

## randomForest
## Original function to interface
```

```
set.seed(71)
iris.rf <- randomForest(Species ~ ., data = iris,
                        importance = TRUE, proximity = TRUE)
print(iris.rf)
plot(iris.rf)

## The interface reverses the order of data and formula
set.seed(71)
iris.rf <- ntbt_randomForest(iris, Species ~ .,
                             importance = TRUE, proximity = TRUE)
print(iris.rf)
plot(iris.rf)

## so it can be used easily in a pipeline.
set.seed(71)
iris %>%
  ntbt_randomForest(Species ~ ., importance = TRUE,
                    proximity = TRUE) %>%
  plot()

## End(Not run)
```

---

`Rchoice` *Interfaces for Rchoice package for data science pipelines.*

---

## Description

Interfaces to `Rchoice` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_Rchoice(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(Rchoice)

## ntbt_Rchoice: Bias reduction in Binomial-response GLMs
data("Workmroz")

## Original function to interface
Rchoice(lfp ~ k5 + k618 + age + wc + hc + lwg + inc,
        data = Workmroz, family = binomial('probit'))
## The interface puts data as first parameter
ntbt_Rchoice(Workmroz, lfp ~ k5 + k618 + age + wc + hc + lwg + inc,
             family = binomial('probit'))

## so it can be used easily in a pipeline.
Workmroz %>%
  ntbt_Rchoice(lfp ~ k5 + k618 + age + wc + hc + lwg + inc,
               family = binomial('probit'))

## End(Not run)
```

---

`rminer` *Interfaces for rminer package for data science pipelines.*

---

## Description

Interfaces to `rminer` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_fit(data, ...)
ntbt_mining(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(rminer)

## ntbt_fit: Fit a supervised data mining model (classification or regression) model
x1 <- rnorm(200,100,20)
x2 <- rnorm(200,100,20)
y <- 0.7*sin(x1/(25*pi))+0.3*sin(x2/(25*pi))
dta <- data.frame(x1, x2, y)

## Original function to interface
fit(y ~ x1 + x2, data = dta, model = "mlpe")

## The interface puts data as first parameter
ntbt_fit(dta, y ~ x1 + x2, model = "mlpe")

## so it can be used easily in a pipeline.
dta %>%
  ntbt_fit(y ~ x1 + x2, model = "mlpe")


## ntbt_mining: Powerful function that trains and tests a particular fit model
##              under several runs and a given validation method
## Original function to interface
mining(y ~ x1 + x2, data = dta, model = "mlpe")

## The interface puts data as first parameter
ntbt_mining(dta, y ~ x1 + x2, model = "mlpe")

## so it can be used easily in a pipeline.
dta %>%
  ntbt_mining(y ~ x1 + x2, model = "mlpe")

## End(Not run)
```

---

`rms` *Interfaces for rms package for data science pipelines.*

---

## Description

Interfaces to `rms` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_bj(data, ...)
ntbt_cph(data, ...)
ntbt_Glm(data, ...)
ntbt_lrm(data, ...)
ntbt_npsurv(data, ...)
ntbt_ols(data, ...)
ntbt_orm(data, ...)
ntbt_psm(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(rms)

## ntbt_bj: Buckley-James Multiple Regression Model
set.seed(1)
ftime  <- 10*rexp(200)
stroke <- ifelse(ftime > 10, 0, 1)
ftime  <- pmin(ftime, 10)
units(ftime) <- "Month"
age <- rnorm(200, 70, 10)
hospital <- factor(sample(c('a','b'),200,TRUE))
dd <- datadist(age, hospital)
options(datadist = "dd")
data_bj <- data.frame(ftime, stroke, age, hospital)

## Original function to interface
bj(Surv(ftime, stroke) ~ rcs(age,5) + hospital, data_bj, x = TRUE, y = TRUE)

## The interface puts data as first parameter
f <- ntbt_bj(data_bj, Surv(ftime, stroke) ~ rcs(age,5) + hospital, x = TRUE, y = TRUE)
```

```
anova(f)

## so it can be used easily in a pipeline.
data_bj %>%
  ntbt_bj(Surv(ftime, stroke) ~ rcs(age,5) + hospital, x = TRUE, y = TRUE)


## ntbt_cph: Cox Proportional Hazards Model and Extensions
n <- 1000
set.seed(731)
age <- 50 + 12*rnorm(n)
label(age) <- "Age"
sex <- factor(sample(c('Male','Female'), n,
              rep=TRUE, prob=c(.6, .4)))
cens <- 15*runif(n)
h <- .02*exp(.04*(age-50)+.8*(sex=='Female'))
dt <- -log(runif(n))/h
label(dt) <- 'Follow-up Time'
e <- ifelse(dt <= cens,1,0)
dt <- pmin(dt, cens)
units(dt) <- "Year"
dd <- datadist(age, sex)
options(datadist='dd')
S <- Surv(dt,e)

data_cph <- data.frame(S, age, sex)

## Original function to interface
cph(S ~ rcs(age,4) + sex, data_cph, x = TRUE, y = TRUE)

## The interface puts data as first parameter
ntbt_cph(data_cph, S ~ rcs(age,4) + sex, x = TRUE, y = TRUE)

## so it can be used easily in a pipeline.
data_cph %>%
  ntbt_cph(S ~ rcs(age,4) + sex, x = TRUE, y = TRUE)


## ntbt_Glm
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
data_Glm <- data.frame(counts, outcome, treatment)

## Original function to interface
Glm(counts ~ outcome + treatment, family = poisson(), data = data_Glm)

## The interface puts data as first parameter
ntbt_Glm(data_Glm, counts ~ outcome + treatment, family = poisson())

## so it can be used easily in a pipeline.
data_Glm %>%
```

```
  ntbt_Glm(counts ~ outcome + treatment, family = poisson())


## ntbt_lrm: Logistic Regression Model
n <- 1000    # define sample size
set.seed(17) # so can reproduce the results
age            <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol    <- rnorm(n, 200, 25)
sex            <- factor(sample(c('female','male'), n,TRUE))
label(age)            <- 'Age'      # label is in Hmisc
label(cholesterol)    <- 'Total Cholesterol'
label(blood.pressure) <- 'Systolic Blood Pressure'
label(sex)            <- 'Sex'
units(cholesterol)    <- 'mg/dl'   # uses units.default in Hmisc
units(blood.pressure) <- 'mmHg'

#To use prop. odds model, avoid using a huge number of intercepts by
#grouping cholesterol into 40-tiles
ch <- cut2(cholesterol, g=40, levels.mean=TRUE) # use mean values in intervals
data_lrm <- data.frame(ch, age)

## Original function to interface
lrm(ch ~ age, data_lrm)

## The interface puts data as first parameter
ntbt_lrm(data_lrm, ch ~ age)

## so it can be used easily in a pipeline.
data_lrm %>%
  ntbt_lrm(ch ~ age)


## ntbt_npsurv: Nonparametric Survival Estimates for Censored Data
tdata <- data.frame(time   = c(1,1,1,2,2,2,3,3,3,4,4,4),
                    status = rep(c(1,0,2),4),
                    n      = c(12,3,2,6,2,4,2,0,2,3,3,5))
## Original function to interface
f <- npsurv(Surv(time, time, status, type = 'interval') ~ 1, data = tdata, weights = n)
plot(f, fun = 'event', xmax = 20, mark.time = FALSE, col = 2:3)

## The interface puts data as first parameter
f <- ntbt_npsurv(tdata, Surv(time, time, status, type = 'interval') ~ 1, weights = n)
plot(f, fun = 'event', xmax = 20, mark.time = FALSE, col = 2:3)

## so it can be used easily in a pipeline.
tdata %>%
  ntbt_npsurv(Surv(time, time, status, type = 'interval') ~ 1, weights = n) %>%
  plot(fun = 'event', xmax = 20, mark.time = FALSE, col = 2:3)


## ntbt_ols: Linear Model Estimation Using Ordinary Least Squares
set.seed(1)
```

```
x1 <- runif(200)
x2 <- sample(0:3, 200, TRUE)
distance <- (x1 + x2/3 + rnorm(200))^2
d <- datadist(x1, x2)
options(datadist="d")   # No d -> no summary, plot without giving all details
data_ols <- data.frame(distance, x1, x2)

## Original function to interface
ols(sqrt(distance) ~ rcs(x1, 4) + scored(x2), data_ols, x = TRUE)

## The interface puts data as first parameter
ntbt_ols(data_ols, sqrt(distance) ~ rcs(x1, 4) + scored(x2), x = TRUE)

## so it can be used easily in a pipeline.
data_ols %>%
  ntbt_ols(sqrt(distance) ~ rcs(x1, 4) + scored(x2), x = TRUE)


## ntbt_orm: Ordinal Regression Model
set.seed(1)
n <- 300
x1 <- c(rep(0,150), rep(1,150))
y <- rnorm(n) + 3 * x1
data_orm <- data.frame(y, x1)

## Original function to interface
orm(y ~ x1, data_orm)

## The interface puts data as first parameter
ntbt_orm(data_orm, y ~ x1)

## so it can be used easily in a pipeline.
data_orm %>%
  ntbt_orm(y ~ x1)


## ntbt_psm: Parametric Survival Model
n <- 400
set.seed(1)
age <- rnorm(n, 50, 12)
sex <- factor(sample(c('Female','Male'),n,TRUE))
dd <- datadist(age,sex)
options(datadist='dd')
# Population hazard function:
h <- .02*exp(.06*(age-50)+.8*(sex=='Female'))
d.time <- -log(runif(n))/h
cens <- 15*runif(n)
death <- ifelse(d.time <= cens,1,0)
d.time <- pmin(d.time, cens)

data_psm <- data.frame(d.time, death, sex, age)

## Original function to interface
```

```
psm(Surv(d.time, death) ~ sex * pol(age, 2), data_psm, dist = 'lognormal')
# Log-normal model is a bad fit for proportional hazards data

## The interface puts data as first parameter
ntbt_psm(data_psm, Surv(d.time, death) ~ sex * pol(age, 2), dist = 'lognormal')

## so it can be used easily in a pipeline.
data_psm %>%
  ntbt_psm(Surv(d.time, death) ~ sex * pol(age, 2), dist = 'lognormal')

## End(Not run)
```

---

`robustbase` *Interfaces for robustbase package for data science pipelines.*

---

## Description

Interfaces to `robustbase` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_adjbox(data, ...)
ntbt_glmrob(data, ...)
ntbt_lmrob(data, ...)
ntbt_ltsReg(data, ...)
ntbt_nlrob(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(robustbase)


## ntbt_adjbox: Plot an Adjusted Boxplot for Skew Distributions
## Original function to interface
adjbox(len ~ dose, data = ToothGrowth)

## The interface puts data as first parameter
ntbt_adjbox(ToothGrowth, len ~ dose)

## so it can be used easily in a pipeline.
ToothGrowth %>%
  ntbt_adjbox(len ~ dose)


## ntbt_glmrob: Robust Fitting of Generalized Linear Models
data(carrots)

## Original function to interface
glmrob(cbind(success, total-success) ~ logdose + block,
       family = binomial, data = carrots, method= "Mqle",
       control= glmrobMqle.control(tcc=1.2))

## The interface puts data as first parameter
ntbt_glmrob(carrots, cbind(success, total-success) ~ logdose + block,
            family = binomial, method= "Mqle",
            control= glmrobMqle.control(tcc=1.2))

## so it can be used easily in a pipeline.
carrots %>%
  ntbt_glmrob(cbind(success, total-success) ~ logdose + block,
              family = binomial, method= "Mqle",
              control= glmrobMqle.control(tcc=1.2))


## ntbt_lmrob: MM-type Estimators for Linear Regression
data(coleman)

## Original function to interface
set.seed(0)
lmrob(Y ~ ., data = coleman, setting = "KS2011")

## The interface puts data as first parameter
ntbt_lmrob(coleman, Y ~ ., setting = "KS2011")

## so it can be used easily in a pipeline.
coleman %>%
  ntbt_lmrob(Y ~ ., setting = "KS2011")
```

```
## ntbt_ltsReg: Least Trimmed Squares Robust (High Breakdown) Regression
data(stackloss)

## Original function to interface
ltsReg(stack.loss ~ ., data = stackloss)

## The interface puts data as first parameter
ntbt_ltsReg(stackloss, stack.loss ~ .)

## so it can be used easily in a pipeline.
stackloss %>%
  ntbt_ltsReg(stack.loss ~ .)


## ntbt_nlrob: Robust Fitting of Nonlinear Regression Models
DNase1 <- DNase[ DNase$Run == 1, ]

## Original function to interface
nlrob(density ~ Asym/(1 + exp(( xmid - log(conc) )/scal ) ),
      data = DNase1, trace = TRUE,
      start = list( Asym = 3, xmid = 0, scal = 1 ))

## The interface puts data as first parameter
ntbt_nlrob(DNase1, density ~ Asym/(1 + exp(( xmid - log(conc) )/scal ) ),
           trace = TRUE,
           start = list( Asym = 3, xmid = 0, scal = 1 ))

## so it can be used easily in a pipeline.
DNase1 %>%
  ntbt_nlrob(density ~ Asym/(1 + exp(( xmid - log(conc) )/scal ) ),
             trace = TRUE,
             start = list( Asym = 3, xmid = 0, scal = 1 ))

## End(Not run)
```

---

`rpart` *Interfaces for rpart package for data science pipelines.*

---

## Description

Interfaces to `rpart` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_rpart(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(rpart)

## rpart
## Original function to interface
fit <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis)
fit2 <- rpart(Kyphosis ~ Age + Number + Start, data = kyphosis,
              parms = list(prior = c(.65,.35), split = "information"))
par(mfrow = c(1,2), xpd = NA) # otherwise on some devices the text is clipped
plot(fit)
text(fit, use.n = TRUE)
plot(fit2)
text(fit2, use.n = TRUE)

## The interface reverses the order of data and formula
fit <- ntbt_rpart(data = kyphosis, Kyphosis ~ Age + Number + Start)
fit2 <- ntbt_rpart(data = kyphosis, Kyphosis ~ Age + Number + Start,
                   parms = list(prior = c(.65,.35), split = "information"))

par(mfrow = c(1,2), xpd = NA) # otherwise on some devices the text is clipped
plot(fit)
text(fit, use.n = TRUE)
plot(fit2)
text(fit2, use.n = TRUE)

## so it can be used easily in a pipeline.
par(mfrow = c(1,2), xpd = NA) # otherwise on some devices the text is clipped
kyphosis %>%
  ntbt_rpart(Kyphosis ~ Age + Number + Start) %T>%
  plot() %>%
  text(use.n = TRUE)
```

```
kyphosis %>%
  ntbt_rpart(Kyphosis ~ Age + Number + Start,
             parms = list(prior = c(.65,.35), split = "information")) %T>%
  plot() %>%
  text(use.n = TRUE)

## End(Not run)
```

---

RRF *Interfaces for RRF package for data science pipelines.*

---

## Description

Interfaces to RRF functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_RRF(data, ...)
ntbt_rrfImpute(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(RRF)


data(iris)
set.seed(111)
ind <- sample(2, nrow(iris), replace = TRUE, prob=c(0.8, 0.2))
```

```
## Original function to interface
RRF(Species ~ ., data=iris[ind == 1,])

## The interface puts data as first parameter
ntbt_RRF(iris[ind == 1,], Species ~ .)

## so it can be used easily in a pipeline.
iris[ind == 1,] %>%
  ntbt_RRF(Species ~ .)


## ntbt_rrfImpute: Missing Value Imputations by RRF
data(iris)
iris.na <- iris
set.seed(111)
for (i in 1:4) iris.na[sample(150, sample(20)), i] <- NA

## Original function to interface
set.seed(222)
rrfImpute(Species ~ ., iris.na)

## The interface puts data as first parameter
set.seed(222)
ntbt_rrfImpute(iris.na, Species ~ .)

## so it can be used easily in a pipeline.
set.seed(222)
iris.na %>%
  ntbt_rrfImpute(Species ~ .)

## End(Not run)
```

---

RWeka *Interfaces for RWeka package for data science pipelines.*

---

## Description

Interfaces to RWeka functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_AdaBoostM1(data, ...)
ntbt_Bagging(data, ...)
ntbt_CostSensitiveClassifier(data, ...)
ntbt_DecisionStump(data, ...)
ntbt_Discretize(data, ...)
ntbt_GainRatioAttributeEval(data, ...)
ntbt_IBk(data, ...)
ntbt_InfoGainAttributeEval(data, ...)
```

```
ntbt_J48(data, ...)
ntbt_JRip(data, ...)
ntbt_LBR(data, ...)
ntbt_LogitBoost(data, ...)
ntbt_LinearRegression(data, ...)
ntbt_LMT(data, ...)
ntbt_Logistic(data, ...)
ntbt_M5P(data, ...)
ntbt_M5Rules(data, ...)
ntbt_MultiBoostAB(data, ...)
ntbt_Normalize(data, ...)
ntbt_OneR(data, ...)
ntbt_PART(data, ...)
ntbt_SMO(data, ...)
ntbt_Stacking(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(RWeka)

## R/Weka Attribute Evaluators
## Original function to interface
GainRatioAttributeEval(Species ~ . , data = iris)
InfoGainAttributeEval(Species ~ . , data = iris)

## The interface puts data as first parameter
ntbt_GainRatioAttributeEval(iris, Species ~ .)
ntbt_InfoGainAttributeEval(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
```

```
  ntbt_GainRatioAttributeEval(Species ~ .)
iris %>%
  ntbt_InfoGainAttributeEval(Species ~ .)

## R/Weka Classifier Functions
data(infert)
infert$STATUS <- factor(infert$case, labels = c("control", "case"))

## Original function to interface
LinearRegression(weight ~ feed, data = chickwts)
Logistic(STATUS ~ spontaneous + induced, data = infert)
SMO(Species ~ ., data = iris, control = Weka_control(K = list("RBFKernel", G = 2)))

## The interface puts data as first parameter
ntbt_LinearRegression(chickwts, weight ~ feed)
ntbt_Logistic(infert, STATUS ~ spontaneous + induced)
ntbt_SMO(iris, Species ~ ., control = Weka_control(K = list("RBFKernel", G = 2)))

## so it can be used easily in a pipeline.
chickwts %>%
  ntbt_LinearRegression(weight ~ feed)
infert %>%
  ntbt_Logistic(STATUS ~ spontaneous + induced)
iris %>%
  ntbt_SMO(Species ~ ., control = Weka_control(K = list("RBFKernel", G = 2)))

## R/Weka Lazy Learners
## No examples provided. LBR seems to need 'lazyBayesianRules'
## and I am too lazy myself to install it
ntbt_IBk(chickwts, weight ~ feed)   ## Example may not make sense


## R/Weka Meta Learners
## MultiBoostAB needs Weka package 'multiBoostAB'
## CostSensitiveClassifier throws an error

## Original function to interface
AdaBoostM1(Species ~ ., data = iris, control = Weka_control(W = "DecisionStump"))
Bagging(Species ~ ., data = iris, control = Weka_control())
LogitBoost(Species ~ ., data = iris, control = Weka_control())
Stacking(Species ~ ., data = iris, control = Weka_control())

## The interface puts data as first parameter
ntbt_AdaBoostM1(iris, Species ~ ., control = Weka_control(W = "DecisionStump"))
ntbt_Bagging(iris, Species ~ ., control = Weka_control())
ntbt_LogitBoost(iris, Species ~ ., control = Weka_control())
ntbt_Stacking(iris, Species ~ ., control = Weka_control())

## so it can be used easily in a pipeline.
iris %>%
  ntbt_AdaBoostM1(Species ~ ., control = Weka_control(W = "DecisionStump"))
iris %>%
  ntbt_Bagging(Species ~ ., control = Weka_control())
```

```
iris %>%
  ntbt_LogitBoost(Species ~ ., control = Weka_control())
iris %>%
  ntbt_Stacking(Species ~ ., control = Weka_control())

## R/Weka Rule Learners
## Original function to interface
JRip(Species ~ ., data = iris)
M5Rules(mpg ~ ., data = mtcars)
OneR(Species ~ ., data = iris)
PART(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_JRip(iris, Species ~ .)
ntbt_M5Rules(mtcars, mpg ~ .)
ntbt_OneR(iris, Species ~ .)
ntbt_PART(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_JRip(Species ~ .)
mtcars %>%
  ntbt_M5Rules(mpg ~ .)
iris %>%
  ntbt_OneR(Species ~ .)
iris %>%
  ntbt_PART(Species ~ .)

## R/Weka Classifier Trees
DF3 <- read.arff(system.file("arff", "cpu.arff", package = "RWeka"))
DF4 <- read.arff(system.file("arff", "weather.arff", package = "RWeka"))

## Original function to interface
DecisionStump(play ~ ., data = DF4)
J48(Species ~ ., data = iris)
LMT(play ~ ., data = DF4)
M5P(class ~ ., data = DF3)

## The interface puts data as first parameter
ntbt_DecisionStump(DF4, play ~ .)
ntbt_J48(iris, Species ~ .)
ntbt_LMT(DF4, play ~ .)
ntbt_M5P(DF3, class ~ .)

## so it can be used easily in a pipeline.
DF4 %>%
  ntbt_DecisionStump(play ~ .)
iris %>%
  ntbt_J48(Species ~ .)
DF4 %>%
  ntbt_LMT(play ~ .)
DF3 %>%
  ntbt_M5P(class ~ .)
```

```
## R/Weka Filters
w <- read.arff(system.file("arff","weather.arff", package = "RWeka"))

## Original function to interface
Discretize(play ~., data = w)
Normalize(~., data = w)

## The interface puts data as first parameter
ntbt_Discretize(w, play ~.)
ntbt_Normalize(w, ~.)

## so it can be used easily in a pipeline.
w %>%
  ntbt_Discretize(play ~.)
w %>%
  ntbt_Normalize(~.)

## End(Not run)
```

---

`sampleSelection` *Interfaces for sampleSelection package for data science pipelines.*

---

## Description

Interfaces to `sampleSelection` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_probit(data, ...)
ntbt_binaryChoice(data, ...)
ntbt_selection(data, ...)
ntbt_heckit(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(sampleSelection)


## ntbt_probit: Binary choice models
## ntbt_binaryChoice:  (no examples found)
data(Mroz87)
Mroz87$kids <- Mroz87$kids5 > 0 | Mroz87$kids618 > 0
Mroz87$age30.39 <- Mroz87$age < 40
Mroz87$age50.60 <- Mroz87$age >= 50

## Original function to interface
probit(lfp ~ kids + age30.39 + age50.60 + educ + hushrs +
         huseduc + huswage + mtr + motheduc, data=Mroz87)

## The interface puts data as first parameter
ntbt_probit(Mroz87, lfp ~ kids + age30.39 + age50.60 + educ + hushrs +
              huseduc + huswage + mtr + motheduc)

## so it can be used easily in a pipeline.
Mroz87 %>%
  ntbt_probit(lfp ~ kids + age30.39 + age50.60 + educ + hushrs +
                huseduc + huswage + mtr + motheduc)



## ntbt_selection: Heckman-style selection models
## ntbt_heckit:
data( Mroz87 )
Mroz87$kids  <- ( Mroz87$kids5 + Mroz87$kids618 > 0 )

## Original function to interface
# Two-step estimation
heckit(lfp ~ age + I( age^2 ) + faminc + kids + educ,
        wage ~ exper + I( exper^2 ) + educ + city, Mroz87)
# ML estimation
selection(lfp ~ age + I( age^2 ) + faminc + kids + educ,
          wage ~ exper + I( exper^2 ) + educ + city, Mroz87)

## The interface puts data as first parameter
# Two-step estimation
ntbt_heckit(Mroz87, lfp ~ age + I( age^2 ) + faminc + kids + educ,
            wage ~ exper + I( exper^2 ) + educ + city)
# ML estimation
ntbt_selection(Mroz87, lfp ~ age + I( age^2 ) + faminc + kids + educ,
               wage ~ exper + I( exper^2 ) + educ + city)

## so it can be used easily in a pipeline.
# Two-step estimation
```

```
Mroz87 %>%
  ntbt_heckit(lfp ~ age + I( age^2 ) + faminc + kids + educ,
              wage ~ exper + I( exper^2 ) + educ + city)
# ML estimation
Mroz87 %>%
  ntbt_selection(lfp ~ age + I( age^2 ) + faminc + kids + educ,
                 wage ~ exper + I( exper^2 ) + educ + city)

## End(Not run)
```

---

sem *Interfaces for sem package for data science pipelines.*

---

## Description

Interfaces to sem functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_rawMoments(data, ...)
ntbt_sem(data, ...)
ntbt_tsls(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(sem)


## ntbt_rawMoments: Compute Raw Moments Matrix
```

```
## Original function to interface
rawMoments(~ Q + P + D + F + A, data = Kmenta)

## The interface puts data as first parameter
ntbt_rawMoments(Kmenta, ~ Q + P + D + F + A)

## so it can be used easily in a pipeline.
Kmenta %>%
  ntbt_rawMoments(~ Q + P + D + F + A)


## ntbt_sem: General Structural Equation Models
## NOTE: this example is NOT using the formula interface.
##       It is creating a list with the variables.
R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"),
                text="
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087
")

model.dhp.1 <- specifyEquations(covs="RGenAsp, FGenAsp", text="
RGenAsp = gam11*RParAsp + gam12*RIQ + gam13*RSES + gam14*FSES + beta12*FGenAsp
FGenAsp = gam23*RSES + gam24*FSES + gam25*FIQ + gam26*FParAsp + beta21*RGenAsp
ROccAsp = 1*RGenAsp
REdAsp = lam21(1)*RGenAsp  # to illustrate setting start values
FOccAsp = 1*FGenAsp
FEdAsp = lam42(1)*FGenAsp
")

dta <- list(R.DHP = R.DHP, model.dhp.1 = model.dhp.1)
rm(R.DHP, model.dhp.1)

## Original function to interface
attach(dta)
sem.dhp.1 <- ntbt_sem(model.dhp.1, R.DHP, 329,
                      fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp'))
summary(sem.dhp.1)
detach()

## The interface puts data as first parameter
sem.dhp.1 <- ntbt_sem(dta, model.dhp.1, R.DHP, 329,
                      fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp'))
summary(sem.dhp.1)

## so it can be used easily in a pipeline.
```

```
dta %>%
  ntbt_sem(model.dhp.1, R.DHP, 329,
           fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp')) %>%
  summary()


## ntbt_tsls: Two-Stage Least Squares
## Original function to interface
tsls(Q ~ P + D, ~ D + F + A, data = Kmenta)

## The interface puts data as first parameter
ntbt_tsls(Kmenta, Q ~ P + D, ~ D + F + A)

## so it can be used easily in a pipeline.
Kmenta %>%
  ntbt_tsls(Q ~ P + D, ~ D + F + A)

## End(Not run)
```

---

spBayes *Interfaces for spBayes package for data science pipelines.*

---

## Description

Interfaces to spBayes functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_bayesGeostatExact(data, ...)
ntbt_bayesLMConjugate(data, ...)
ntbt_spDynLM(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(spBayes)

## NOTE: Just interfacing 3 functions as a proof of concept. However,
##       there are so many things declared and used, that I am not sure if
##       everything could be passed on a pipeline (perhaps with intuBags down the line).
##       I may get back to it later.

## ntbt_bayesGeostatExact: Simple Bayesian spatial linear model with fixed semivariogram parameters
data(FORMGMT.dat)

n <- nrow(FORMGMT.dat)
p <- 5 ##an intercept an four covariates

n.samples <- 50

phi <- 0.0012

coords <- cbind(FORMGMT.dat$Longi, FORMGMT.dat$Lat)
coords <- coords*(pi/180)*6378

beta.prior.mean <- rep(0, times=p)
beta.prior.precision <- matrix(0, nrow=p, ncol=p)

alpha <- 1/1.5

sigma.sq.prior.shape <- 2.0
sigma.sq.prior.rate <- 10.0


## Original function to interface
bayesGeostatExact(Y ~ X1 + X2 + X3 + X4, data = FORMGMT.dat,
                  n.samples = n.samples,
                  beta.prior.mean = beta.prior.mean,
                  beta.prior.precision = beta.prior.precision,
                  coords = coords, phi = phi, alpha = alpha,
                  sigma.sq.prior.shape = sigma.sq.prior.shape,
                  sigma.sq.prior.rate = sigma.sq.prior.rate)

## The interface puts data as first parameter
ntbt_bayesGeostatExact(FORMGMT.dat, Y ~ X1 + X2 + X3 + X4,
                       n.samples = n.samples,
                       beta.prior.mean = beta.prior.mean,
                       beta.prior.precision = beta.prior.precision,
                       coords = coords, phi = phi, alpha = alpha,
                       sigma.sq.prior.shape = sigma.sq.prior.shape,
                       sigma.sq.prior.rate = sigma.sq.prior.rate)

## so it can be used easily in a pipeline.
```

```
FORMGMT.dat %>%
  ntbt_bayesGeostatExact(Y ~ X1 + X2 + X3 + X4,
                         n.samples = n.samples,
                         beta.prior.mean = beta.prior.mean,
                         beta.prior.precision = beta.prior.precision,
                         coords = coords, phi = phi, alpha = alpha,
                         sigma.sq.prior.shape = sigma.sq.prior.shape,
                         sigma.sq.prior.rate = sigma.sq.prior.rate)



## ntbt_bayesLMConjugate: Simple Bayesian linear model via the Normal/inverse-Gamma conjugate
n <- nrow(FORMGMT.dat)
p <- 7 ##an intercept and six covariates

n.samples <- 500

## Below we demonstrate the conjugate function in the special case
## with improper priors. The results are the same as for the above,
## up to MC error.
beta.prior.mean <- rep(0, times=p)
beta.prior.precision <- matrix(0, nrow=p, ncol=p)

prior.shape <- -p/2
prior.rate <- 0

## Original function to interface
bayesLMConjugate(Y ~ X1 + X2 + X3 + X4 + X5 + X6, data = FORMGMT.dat,
                 n.samples, beta.prior.mean,
                 beta.prior.precision,
                 prior.shape, prior.rate)

## The interface puts data as first parameter
ntbt_bayesLMConjugate(FORMGMT.dat, Y ~ X1 + X2 + X3 + X4 + X5 + X6,
                      n.samples, beta.prior.mean,
                      beta.prior.precision,
                      prior.shape, prior.rate)

## so it can be used easily in a pipeline.
FORMGMT.dat %>%
  ntbt_bayesLMConjugate(Y ~ X1 + X2 + X3 + X4 + X5 + X6,
                        n.samples, beta.prior.mean,
                        beta.prior.precision,
                        prior.shape, prior.rate)


## ntbt_spDynLM: Function for fitting univariate Bayesian dynamic
##               space-time regression models
data("NETemp.dat")
ne.temp <- NETemp.dat

set.seed(1)
```

```
##take a chunk of New England
ne.temp <- ne.temp[ne.temp[,"UTMX"] > 5500000 & ne.temp[,"UTMY"] > 3000000,]

##subset first 2 years (Jan 2000 - Dec. 2002)
y.t <- ne.temp[,4:27]
N.t <- ncol(y.t) ##number of months
n <- nrow(y.t) ##number of observation per months

##add some missing observations to illistrate prediction
miss <- sample(1:N.t, 10)
holdout.station.id <- 5
y.t.holdout <- y.t[holdout.station.id, miss]
y.t[holdout.station.id, miss] <- NA

##scale to km
coords <- as.matrix(ne.temp[,c("UTMX", "UTMY")]/1000)
max.d <- max(iDist(coords))

##set starting and priors
p <- 2 #number of regression parameters in each month

starting <- list("beta"=rep(0,N.t*p), "phi"=rep(3/(0.5*max.d), N.t),
                 "sigma.sq"=rep(2,N.t), "tau.sq"=rep(1, N.t),
                 "sigma.eta"=diag(rep(0.01, p)))

tuning <- list("phi"=rep(5, N.t))

priors <- list("beta.0.Norm"=list(rep(0,p), diag(1000,p)),
               "phi.Unif"=list(rep(3/(0.9*max.d), N.t), rep(3/(0.05*max.d), N.t)),
               "sigma.sq.IG"=list(rep(2,N.t), rep(10,N.t)),
               "tau.sq.IG"=list(rep(2,N.t), rep(5,N.t)),
               "sigma.eta.IW"=list(2, diag(0.001,p)))

##make symbolic model formula statement for each month
mods <- lapply(paste(colnames(y.t),'elev',sep='~'), as.formula)

n.samples <- 10 # in original example it is 2000.

## Original function to interface
spDynLM(mods, data=cbind(y.t,ne.temp[,"elev",drop=FALSE]), coords=coords,
        starting=starting, tuning=tuning, priors=priors, get.fitted =TRUE,
        cov.model="exponential", n.samples=n.samples, n.report=25)

## The interface puts data as first parameter
ntbt_spDynLM(cbind(y.t,ne.temp[,"elev",drop=FALSE]), mods, coords=coords,
             starting=starting, tuning=tuning, priors=priors, get.fitted =TRUE,
             cov.model="exponential", n.samples=n.samples, n.report=25)

## so it can be used easily in a pipeline.
cbind(y.t,ne.temp[,"elev",drop=FALSE]) %>%
  ntbt_spDynLM(mods, coords=coords,
               starting=starting, tuning=tuning, priors=priors, get.fitted =TRUE,
```

```
                cov.model="exponential", n.samples=n.samples, n.report=25)

## End(Not run)
```

---

stats *Interfaces for stats package for data science pipelines.*

---

## Description

Interfaces to stats functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_aggregate(data, ...)
ntbt_alias(data, ...)
ntbt_ansari.test(data, ...)
ntbt_aov(data, ...)
ntbt_bartlett.test(data, ...)
ntbt_cor.test(data, ...)
ntbt_fligner.test(data, ...)
ntbt_friedman.test(data, ...)
ntbt_ftable(data, ...)
ntbt_getInitial(data, ...)
ntbt_glm(data, ...)
ntbt_kruskal.test(data, ...)
ntbt_lm(data, ...)
ntbt_loess(data, ...)
ntbt_lqs(data, ...)
ntbt_model.frame(data, ...)
ntbt_model.matrix(data, ...)
ntbt_mood.test(data, ...)
ntbt_nls(data, ...)
ntbt_oneway.test(data, ...)
ntbt_ppr(data, ...)
ntbt_prcomp(data, ...)
ntbt_princomp(data, ...)
ntbt_quade.test(data, ...)
ntbt_replications(data, ...)
ntbt_t.test(data, ...)
ntbt_var.test(data, ...)
ntbt_wilcox.test(data, ...)
ntbt_xtabs(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)

## aggregate
## Original function to interface
ag <- aggregate(len ~ ., data = ToothGrowth, mean)
xtabs(len ~ ., data = ag)

## The interface reverses the order of data and formula
ag <- ntbt_aggregate(ToothGrowth, len ~ ., mean)
ntbt_xtabs(ag, len ~ .)

## so it can be used easily in a pipeline.
ToothGrowth %>%
  ntbt_aggregate(len ~ ., mean) %>%
  ntbt_xtabs(len ~ .)

esoph %>%
  ntbt_aggregate(cbind(ncases, ncontrols) ~ alcgp + tobgp, sum)

## alias
## Original function to interface
alias(yield ~ block + N*P*K, npk)

## The interface reverses the order of data and formula
ntbt_alias(npk, yield ~ block + N*P*K)

## so it can be used easily in a pipeline.
npk %>%
  ntbt_alias(yield ~ block + N*P*K)

## ansari.test
## Original function to interface
ansari.test(extra ~ group, data = sleep)

## The interface reverses the order of data and formula
ntbt_ansari.test(data = sleep, extra ~ group)
```

```
## so it can be used easily in a pipeline.
library(magrittr)
sleep %>%
  ntbt_ansari.test(extra ~ group)

## aov
## Original function to interface
aov(yield ~ block + N * P + K, npk)

## The interface reverses the order of data and formula
ntbt_aov(npk, yield ~ block + N * P + K)

## so it can be used easily in a pipeline.
npk %>%
  ntbt_aov(yield ~ block + N * P + K)

## bartlett.test
## Original function to interface
bartlett.test(count ~ spray, data = InsectSprays)

## The interface reverses the order of data and formula
ntbt_bartlett.test(data = InsectSprays, count ~ spray)

## so it can be used easily in a pipeline.
InsectSprays %>%
  ntbt_bartlett.test(count ~ spray)

## cor.test
## Original function to interface
cor.test(~ CONT + INTG, data = USJudgeRatings)

## The interface reverses the order of data and formula
ntbt_cor.test(data = USJudgeRatings, ~ CONT + INTG)

## so it can be used easily in a pipeline.
USJudgeRatings %>%
  ntbt_cor.test(~ CONT + INTG)

## fligner.test
## Original function to interface
fligner.test(count ~ spray, data = InsectSprays)

## The interface reverses the order of data and formula
ntbt_fligner.test(data = InsectSprays, count ~ spray)

## so it can be used easily in a pipeline.
InsectSprays %>%
  ntbt_fligner.test(count ~ spray)

## friedman.test
wb <- aggregate(warpbreaks$breaks,
                by = list(w = warpbreaks$wool,
                          t = warpbreaks$tension),
```

```
                FUN = mean)

## Original function to interface
friedman.test(x ~ w | t, data = wb)

## The interface reverses the order of data and formula
ntbt_friedman.test(data = wb, x ~ w | t)

## so it can be used easily in a pipeline.
wb %>%
  ntbt_friedman.test(x ~ w | t)

## ftable
## Original function to interface
x <- ftable(Survived ~ ., data = Titanic)
ftable(Sex ~ Class + Age, data = x)

## The interface reverses the order of data and formula
x <- ntbt_ftable(data = Titanic, Survived ~ .)
ftable(data = x, Sex ~ Class + Age)

## so it can be used easily in a pipeline.
Titanic %>%
  ntbt_ftable(Survived ~ .)

Titanic %>%
  ntbt_ftable(Survived ~ .) %>%
  ntbt_ftable(Sex ~ Class + Age)

## getInitial
PurTrt <- Puromycin[ Puromycin$state == "treated", ]

## Original function to interface
getInitial(rate ~ SSmicmen( conc, Vm, K ), PurTrt)

## The interface reverses the order of data and formula
ntbt_getInitial(PurTrt, rate ~ SSmicmen( conc, Vm, K ))

## so it can be used easily in a pipeline.
PurTrt %>%
  ntbt_getInitial(rate ~ SSmicmen( conc, Vm, K ))

## glm
utils::data(anorexia, package = "MASS")

## Original function to interface
anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                data = anorexia)
summary(anorex.1)

## The interface reverses the order of data and formula
anorex.1 <- ntbt_glm(data = anorexia,
                     formula = Postwt ~ Prewt + Treat + offset(Prewt))
```

```
summary(anorex.1)

## so it can be used easily in a pipeline.
anorexia %>%
  ntbt_glm(Postwt ~ Prewt + Treat + offset(Prewt)) %>%
  summary()

# A Gamma example, from McCullagh & Nelder (1989, pp. 300-2)
data.frame(u = c(5,10,15,20,30,40,60,80,100),
           lot1 = c(118,58,42,35,27,25,21,19,18)
           ) %>%
  ntbt_glm(lot1 ~ log(u), family = Gamma) %>%
  summary()

## kruskal.test
## Original function to interface
kruskal.test(Ozone ~ Month, airquality)

## The interface reverses the order of data and formula
ntbt_kruskal.test(airquality, Ozone ~ Month)

## so it can be used easily in a pipeline.
airquality %>%
  ntbt_kruskal.test(Ozone ~ Month)

## lm
## Original function to interface
lm(sr ~ ., LifeCycleSavings)

## The interface reverses the order of data and formula
ntbt_lm(LifeCycleSavings, sr ~ .)

## so it can be used easily in a pipeline.
library(magrittr)
LifeCycleSavings %>%
  ntbt_lm(sr ~ .)

LifeCycleSavings %>%
  ntbt_lm(sr ~ .) %>%
  summary()

## loess
## Original function to interface
loess(dist ~ speed, cars)

## The interface reverses the order of data and formula
ntbt_loess(cars, dist ~ speed)

## so it can be used easily in a pipeline.
cars %>%
  ntbt_loess(dist ~ speed)

cars %>%
```

```
  ntbt_loess(dist ~ speed,
             control = loess.control(surface = "direct"))

## lqs
library(MASS)

## Original function to interface
set.seed(123) # make reproducible
lqs(stack.loss ~ ., data = stackloss)

## The interface reverses the order of data and formula
set.seed(123) # make reproducible
ntbt_lqs(data = stackloss, stack.loss ~ .)

## so it can be used easily in a pipeline.
set.seed(123) # make reproducible
stackloss %>%
  ntbt_lqs(stack.loss ~ .)

## model.frame
## Original function to interface
model.frame(dist ~ speed, data = cars)

## The interface reverses the order of data and formula
ntbt_model.frame(data = cars, dist ~ speed)

## so it can be used easily in a pipeline.
cars %>%
  ntbt_model.frame(dist ~ speed)

## model.matrix
dd <- data.frame(a = gl(3, 4),
                 b = gl(4, 1, 12)) # balanced 2-way

## Original function to interface
model.matrix(~ a + b, dd)

## The interface reverses the order of data and formula
ntbt_model.matrix(dd, ~ a + b)

## so it can be used easily in a pipeline.
dd %>%
  ntbt_model.matrix(~ a + b)

## mood.test
## Original function to interface
mood.test(extra ~ group, data = sleep)

## The interface reverses the order of data and formula
ntbt_mood.test(data = sleep, extra ~ group)

## so it can be used easily in a pipeline.
sleep %>%
```

```
  ntbt_mood.test(extra ~ group)

## nls
## Original function to interface
nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase)

## The interface reverses the order of data and formula
ntbt_nls(data = DNase, density ~ SSlogis(log(conc), Asym, xmid, scal))

## so it can be used easily in a pipeline.
DNase %>%
  ntbt_nls(density ~ SSlogis(log(conc), Asym, xmid, scal))

## oneway.test
## Original function to interface
oneway.test(extra ~ group, data = sleep)

## The interface reverses the order of data and formula
ntbt_oneway.test(data = sleep, extra ~ group)

## so it can be used easily in a pipeline.
sleep %>%
  ntbt_oneway.test(extra ~ group)

## ppr
## Original function to interface
ppr(log(perm) ~ area + peri + shape, data = rock,
    nterms = 2, max.terms = 5)

## The interface reverses the order of data and formula
ntbt_ppr(data = rock, log(perm) ~ area + peri + shape,
         nterms = 2, max.terms = 5)

## so it can be used easily in a pipeline.
rock %>%
  ntbt_ppr(log(perm) ~ area + peri + shape,
           nterms = 2, max.terms = 5)

## prcomp
## Original function to interface
prcomp(~ Murder + Assault + Rape, data = USArrests, scale = TRUE)

## The interface reverses the order of data and formula
ntbt_prcomp(data = USArrests, ~ Murder + Assault + Rape, scale = TRUE)

## so it can be used easily in a pipeline.
USArrests %>%
  ntbt_prcomp(~ Murder + Assault + Rape, scale = TRUE)

## princomp
## Original function to interface
princomp(~ ., data = USArrests, cor = TRUE)
```

```
## The interface reverses the order of data and formula
ntbt_princomp(data = USArrests, ~ ., cor = TRUE)

## so it can be used easily in a pipeline.
USArrests %>%
  ntbt_princomp(~ ., cor = TRUE)

## quade.test
wb <- aggregate(warpbreaks$breaks,
                by = list(w = warpbreaks$wool,
                          t = warpbreaks$tension),
                FUN = mean)

## Original function to interface
quade.test(x ~ w | t, data = wb)

## The interface reverses the order of data and formula
ntbt_quade.test(data = wb, x ~ w | t)

## so it can be used easily in a pipeline.
wb %>%
  ntbt_quade.test(x ~ w | t)

## replications
## From Venables and Ripley (2002) p.165.
N <- c(0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0)
P <- c(1,1,0,0,0,1,0,1,1,1,0,0,0,1,0,1,1,0,0,1,0,1,1,0)
K <- c(1,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,1,0,1,0)
yield <- c(49.5,62.8,46.8,57.0,59.8,58.5,55.5,56.0,62.8,55.8,69.5,
           55.0, 62.0,48.8,45.5,44.2,52.0,51.5,49.8,48.8,57.2,59.0,53.2,56.0)

npk <- data.frame(block = gl(6,4), N = factor(N), P = factor(P),
                  K = factor(K), yield = yield)

## Original function to interface
replications(~ . - yield, npk)

## The interface reverses the order of data and formula
ntbt_replications(npk, ~ . - yield)

## so it can be used easily in a pipeline.
npk %>%
  ntbt_replications(~ . - yield)

## t.test
## Original function to interface
t.test(extra ~ group, data = sleep)

## The interface reverses the order of data and formula
ntbt_t.test(data = sleep, extra ~ group)

## so it can be used easily in a pipeline.
sleep %>%
```

```
  ntbt_t.test(extra ~ group)

## var.test
## Original function to interface
var.test(extra ~ group, data = sleep)

## The interface reverses the order of data and formula
ntbt_var.test(data = sleep, extra ~ group)

## so it can be used easily in a pipeline.
sleep %>%
  ntbt_var.test(extra ~ group)

## wilcox.test
## Original function to interface
wilcox.test(extra ~ group, data = sleep)

## The interface reverses the order of data and formula
ntbt_wilcox.test(data = sleep, extra ~ group)

## so it can be used easily in a pipeline.
sleep %>%
  ntbt_wilcox.test(extra ~ group)

## xtabs
## Original function to interface
ag <- aggregate(len ~ ., data = ToothGrowth, mean)
xtabs(len ~ ., data = ag)

## The interface reverses the order of data and formula
ag <- ntbt_aggregate(ToothGrowth, len ~ ., mean)
ntbt_xtabs(ag, len ~ .)

## so it can be used easily in a pipeline.
ToothGrowth %>%
  ntbt_aggregate(len ~ ., mean) %>%
  ntbt_xtabs(len ~ .)

## End(Not run)
```

---

`strucchange` *Interfaces for strucchange package for data science pipelines.*

---

## Description

Interfaces to `strucchange` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_breakpoints(data, ...)
```

```
ntbt_efp(data, ...)
ntbt_Fstats(data, ...)
ntbt_mefp(data, ...)
ntbt_recresid(data, ...)
ntbt_sctest(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(strucchange)

## ntbt_breakpoints: Dating Breaks
data("Nile")
d <- list(Nl = Nile)

## Original function to interface
breakpoints(Nl ~ 1, data = d)

## The interface puts data as first parameter
ntbt_breakpoints(d, Nl ~ 1)

## so it can be used easily in a pipeline.
d %>%
  ntbt_breakpoints(Nl ~ 1)


## ntbt_efp: Empirical Fluctuation Processes
## Original function to interface
ocus.nile <- efp(Nl ~ 1, d, type = "OLS-CUSUM")
plot(ocus.nile)

## The interface puts data as first parameter
```

```
ocus.nile <- ntbt_efp(d, Nl ~ 1, type = "OLS-CUSUM")
plot(ocus.nile)

## so it can be used easily in a pipeline.
d %>%
  ntbt_efp(Nl ~ 1, type = "OLS-CUSUM") %>%
  plot()


## ntbt_Fstats: F Statistics
## Original function to interface
fs.nile <- Fstats(Nl ~ 1, data = d)
plot(fs.nile)

## The interface puts data as first parameter
fs.nile <- ntbt_Fstats(d, Nl ~ 1)
plot(fs.nile)

## so it can be used easily in a pipeline.
d %>%
  ntbt_Fstats(Nl ~ 1) %>%
  plot()


## ntbt_mefp: Monitoring of Empirical Fluctuation Processes
df1 <- data.frame(y = rnorm(300))
df1[150:300, "y"] <- df1[150:300, "y"] + 1

## Original function to interface
mefp(y ~ 1, data = df1[1:50,, drop = FALSE], type = "ME", h = 1, alpha = 0.05)

## The interface puts data as first parameter
ntbt_mefp(df1[1:50,, drop = FALSE], y ~ 1, type = "ME", h = 1, alpha = 0.05)

## so it can be used easily in a pipeline.
df1[1:50,, drop = FALSE] %>%
  ntbt_mefp(y ~ 1, type = "ME", h = 1, alpha = 0.05)


## ntbt_recresid: Recursive Residuals
d1 <- list(x = rnorm(100) + rep(c(0, 2), each = 50))

## Original function to interface
recresid(x ~ 1, d1)

## The interface puts data as first parameter
ntbt_recresid(d1, x ~ 1)

## so it can be used easily in a pipeline.
d1 %>%
  ntbt_recresid(x ~ 1)
```

```
## ntbt_sctest: Structural Change Tests in Linear Regression Models
data("longley")
## Original function to interface
sctest(Employed ~ Year + GNP.deflator + GNP + Armed.Forces, data = longley,
       type = "Chow", point = 7)

## The interface puts data as first parameter
ntbt_sctest(longley, Employed ~ Year + GNP.deflator + GNP + Armed.Forces,
            type = "Chow", point = 7)

## so it can be used easily in a pipeline.
longley %>%
  ntbt_sctest(Employed ~ Year + GNP.deflator + GNP + Armed.Forces,
              type = "Chow", point = 7)

## End(Not run)
```

---

survey *Interfaces for survey package for data science pipelines.*

---

## Description

Interfaces to survey functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_svyby(data, ...)         ## data <-> design
ntbt_svycoxph(data, ...)      ## data <-> design
ntbt_svydesign(data, ...)
ntbt_svyglm(data, ...)        ## data <-> design
ntbt_svymean(data, ...)       ## data <-> design
ntbt_svyquantile(data, ...)   ## data <-> design
ntbt_svyratio(data, ...)      ## data <-> design
ntbt_svytotal(data, ...)      ## data <-> design
ntbt_twophase(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(survey)

## svydesign
data(api)
## Original function to interface
# stratified sample
dstrat <- svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
# one-stage cluster sample
dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
# two-stage cluster sample: weights computed from population sizes.
dclus2 <- svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)

## The interface puts data as first parameter
# stratified sample
dstrat <- ntbt_svydesign(data=apistrat, id=~1,strata=~stype, weights=~pw, fpc=~fpc)
# one-stage cluster sample
dclus1 <- ntbt_svydesign(data=apiclus1, id=~dnum, weights=~pw, fpc=~fpc)
# two-stage cluster sample: weights computed from population sizes.
dclus2 <- ntbt_svydesign(data=apiclus2, id=~dnum+snum, fpc=~fpc1+fpc2)

## so it can be used easily in a pipeline.
dstrat <- apistrat %>%
  ntbt_svydesign(id=~1,strata=~stype, weights=~pw, fpc=~fpc)
# one-stage cluster sample
dclus1 <- apiclus1 %>%
  ntbt_svydesign(id=~dnum, weights=~pw, fpc=~fpc)
# two-stage cluster sample: weights computed from population sizes.
dclus2 <- apiclus2 %>%
  ntbt_svydesign(id=~dnum+snum, fpc=~fpc1+fpc2)

## twofase
## two-phase simple random sampling.
data(pbc, package="survival")
pbc$randomized <- with(pbc, !is.na(trt) & trt>0)
pbc$id<-1:nrow(pbc)

## Original function to interface
d2pbc <- twophase(id=list(~id,~id), data=pbc, subset=~randomized)
svymean(~bili, d2pbc)

## The interface puts data as first parameter
d2pbc <- ntbt_twophase(data=pbc, id=list(~id,~id), subset=~randomized)
svymean(~bili, d2pbc)
```

```
## so it can be used easily in a pipeline.
d2pbc <- pbc %>%
  ntbt_twophase(id=list(~id,~id), subset=~randomized)
svymean(~bili, d2pbc)


## ntbt_svyby, ntbt_svyglm, ntbt_svymean,
## ntbt_svyquantile, ntbt_svyratio, ntbt_svytotal

## From vignette of survey
vars<-names(apiclus1)[c(12:13,16:23,27:37)]

## original
dclus1 <- svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
summary(dclus1)
## direct call
dclus1 <- apiclus1 %>%
  ntbt(svydesign, id = ~dnum, weights = ~pw, fpc = ~fpc)
summary(dclus1)
## interface
dclus1 <- apiclus1 %>%
  ntbt_svydesign(id = ~dnum, weights = ~pw, fpc = ~fpc)
summary(dclus1)

## original
svymean(~api00, dclus1)
## direct call
dclus1 %>%
  ntbt(svymean, x=~api00)
## interface
dclus1 %>%
  ntbt_svymean(x=~api00)

## original
svyquantile(~api00, dclus1, quantile=c(0.25,0.5,0.75), ci=TRUE)
## direct call
dclus1 %>%
  ntbt(svyquantile, ~api00, quantile=c(0.25,0.5,0.75), ci=TRUE)
## interface
dclus1 %>%
  ntbt(svyquantile, ~api00, quantile=c(0.25,0.5,0.75), ci=TRUE)

## original
svytotal(~stype, dclus1)
svytotal(~enroll, dclus1)
## direct call
dclus1 %>%
  ntbt(svytotal, ~stype)
dclus1 %>%
  ntbt(svytotal,~enroll)
## interface
dclus1 %>%
  ntbt(svytotal, ~stype)
```

```
dclus1 %>%
  ntbt(svytotal,~enroll)

## original
svyratio(~api.stu, ~enroll, dclus1)
svyratio(~api.stu, ~enroll, design=subset(dclus1, stype=="H"))
## direct call
dclus1 %>%
  ntbt(svyratio, ~api.stu, ~enroll)
dclus1 %>%
  ntbt(svyratio, ~api.stu, ~enroll, design=subset("#", stype=="H"))
## interface
dclus1 %>%
  ntbt_svyratio(~api.stu, ~enroll)
dclus1 %>%
  ntbt_svyratio(~api.stu, ~enroll, design=subset("#", stype=="H"))

## original
svymean(make.formula(vars),dclus1,na.rm=TRUE)
## direct call
dclus1 %>%
  ntbt(svymean, make.formula(vars), na.rm=TRUE)
## interface
dclus1 %>%
  ntbt_svymean(make.formula(vars), na.rm=TRUE)

## original
svyby(~ell+meals, ~stype, design=dclus1, svymean)
## direct call
dclus1 %>%
  ntbt(svyby, ~ell+meals, ~stype, svymean)
## interface
dclus1 %>%
  ntbt_svyby(~ell+meals, ~stype, svymean)

## original
regmodel <- svyglm(api00~ell+meals, design=dclus1)
summary(regmodel)
logitmodel <- svyglm(I(sch.wide=="Yes")~ell+meals, design=dclus1,
                     family=quasibinomial())
summary(logitmodel)
## direct call
dclus1 %>%
  ntbt(svyglm, api00~ell+meals) %>%
  summary()
dclus1 %>%
  ntbt(svyglm, I(sch.wide=="Yes")~ell+meals, family=quasibinomial()) %>%
  summary()
## interface
dclus1 %>%
  ntbt_svyglm(api00~ell+meals) %>%
  summary()
dclus1 %>%
```

```
  ntbt_svyglm(I(sch.wide=="Yes")~ell+meals, family=quasibinomial()) %>%
  summary()

## ntbt_svycoxph
## stratified on case status
data(nwtco)
## original
dcchs <- twophase(id=list(~seqno,~seqno), strata=list(NULL,~rel),
                  subset=~I(in.subcohort | rel), data=nwtco)
svycoxph(Surv(edrel,rel)~factor(stage)+factor(histol)+I(age/12), design=dcchs)
## direct call
nwtco %>%
  ntbt(twophase,id = list(~seqno,~seqno), strata = list(NULL,~rel),
       subset = ~I(in.subcohort | rel)) %>%
  ntbt(svycoxph, Surv(edrel,rel)~factor(stage)+factor(histol)+I(age/12))
## interface
nwtco %>%
  ntbt_twophase(id = list(~seqno,~seqno), strata = list(NULL,~rel),
       subset = ~I(in.subcohort | rel)) %>%
  ntbt_svycoxph(Surv(edrel,rel)~factor(stage)+factor(histol)+I(age/12))

## Involved example using `intubOrders`, transforming the code in:

## https://cran.r-project.org/web/packages/survey/vignettes/survey.pdf

data(api)

## First, the original code from the vignette
vars<-names(apiclus1)[c(12:13,16:23,27:37)]

dclus1 <- svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
summary(dclus1)
svymean(~api00, dclus1)
svyquantile(~api00, dclus1, quantile=c(0.25,0.5,0.75), ci=TRUE)
svytotal(~stype, dclus1)
svytotal(~enroll, dclus1)
svyratio(~api.stu,~enroll, dclus1)
svyratio(~api.stu, ~enroll, design=subset(dclus1, stype=="H"))
svymean(make.formula(vars),dclus1,na.rm=TRUE)
svyby(~ell+meals, ~stype, design=dclus1, svymean)
regmodel <- svyglm(api00~ell+meals,design=dclus1)
logitmodel <- svyglm(I(sch.wide=="Yes")~ell+meals, design=dclus1, family=quasibinomial())
summary(regmodel)
summary(logitmodel)

## Now using intubOrders and ntbt.

## Strategy 1: long pipeline, light use of intubOrders.

apiclus1 %>%
  ntbt(svydesign, id = ~dnum, weights = ~ pw, fpc = ~ fpc, "<|| summary >") %>%
  ntbt(svymean, ~ api00, "<|f| print >") %>%
  ntbt(svyquantile, ~ api00, quantile = c(0.25,0.5,0.75), ci = TRUE, "<|f| print >") %>%
```

```
  ntbt(svytotal, ~ stype, "<|f| print >") %>%
  ntbt(svytotal, ~ enroll, "<|f| print >") %>%
  ntbt(svyratio, ~ api.stu, ~ enroll, "<|f| print >") %>%
 ntbt(svyratio, ~ api.stu, ~ enroll, design = subset("#", stype == "H"), "<|f| print >") %>%
  ntbt(svymean, make.formula(vars), na.rm = TRUE, "<|f| print >") %>%
  ntbt(svyby, ~ ell + meals, ~ stype, svymean, "<|f| print >") %>%
  ntbt(svyglm, api00 ~ ell + meals, "<|f| summary >") %>%
 ntbt(svyglm, I(sch.wide == "Yes") ~ ell + meals, family = quasibinomial(), "<|f| summary >") %>%
  summary() ## We have forwarded the result from svydesign (line 2),
            ## so we could still continue using it downstream.

## Strategy 2: short pipeline, heavy use of *one* intubOrder.
apiclus1 %>%
  ntbt(svydesign, id = ~dnum, weights = ~pw, fpc = ~fpc,
       "<|f|
         summary;
         svymean(~api00, #);
         svyquantile(~api00, #, quantile = c(0.25, 0.5, 0.75), ci = TRUE);
         svytotal(~stype, #);
         svytotal(~enroll, #);
         svyratio(~api.stu,~enroll, #);
         svyratio(~api.stu, ~enroll, design = subset(#, stype == 'H'));
         svymean(make.formula(vars), #, na.rm = TRUE);
         svyby(~ell+meals, ~stype, #, svymean);
         summary(svyglm(api00~ell+meals, #));
       summary(svyglm(I(sch.wide == 'Yes')~ell+meals, #, family = quasibinomial())) >") %>%
  head()  ## We have forwarded the original dataset,
          ## so we could continue using it downstream.

## End(Not run)
```

---

survival *Interfaces to survival package for data science pipelines.*

---

## Description

Interfaces to survival functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_cch(data, ...)
ntbt_coxph(data, ...)
ntbt_pyears(data, ...)
ntbt_survConcordance(data, ...)
ntbt_survexp(data, ...)
ntbt_survfit(data, ...)
ntbt_survreg(data, ...)
ntbt_survSplit(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(survival)

## cch
subcoh <- nwtco$in.subcohort
selccoh <- with(nwtco, rel==1|subcoh==1)
ccoh.data <- nwtco[selccoh,]
ccoh.data$subcohort <- subcoh[selccoh]
ccoh.data$histol <- factor(ccoh.data$histol,labels=c("FH","UH"))
ccoh.data$stage <- factor(ccoh.data$stage,labels=c("I","II","III","IV"))
ccoh.data$age <- ccoh.data$age/12 # Age in years

## Original function to interface
cch(Surv(edrel, rel) ~ stage + histol + age, data = ccoh.data,
    subcoh = ~subcohort, id=~seqno, cohort.size=4028)

## The interface reverses the order of data and formula
ntbt_cch(data = ccoh.data, Surv(edrel, rel) ~ stage + histol + age,
    subcoh = ~subcohort, id=~seqno, cohort.size=4028)

## so it can be used easily in a pipeline.
ccoh.data %>%
  ntbt_cch(Surv(edrel, rel) ~ stage + histol + age,
    subcoh = ~subcohort, id=~seqno, cohort.size=4028)

## coxph
## Original function to interface
vet2 <- survSplit(Surv(time, status) ~., veteran,
                  cut = c(60, 120), episode = "timegroup")
coxph(Surv(tstart, time, status) ~ karno*strata(timegroup) +
                age + trt, data = vet2)
```

```
## The interface reverses the order of data and formula
vet2 <- ntbt_survSplit(veteran, Surv(time, status) ~.,
                       cut = c(60, 120), episode = "timegroup")
ntbt_coxph(data = vet2, Surv(tstart, time, status) ~
             karno*strata(timegroup) + age + trt)

## so it can be used easily in a pipeline.
veteran %>%
  ntbt_survSplit(Surv(time, status) ~.,
                 cut = c(60, 120), episode = "timegroup") %>%
  ntbt_coxph(Surv(tstart, time, status) ~
              karno*strata(timegroup) + age + trt)

## pyears
hearta <- by(heart, heart$id,
             function(x)x[x$stop == max(x$stop),])
hearta <- do.call("rbind", hearta)

## Original function to interface
pyears(Surv(stop/365.25, event) ~
         cut(age + 48, c(0,50,60,70,100)) + surgery,
       data = hearta, scale = 1)

## The interface reverses the order of data and formula
ntbt_pyears(data = hearta,
            Surv(stop/365.25, event) ~
              cut(age + 48, c(0,50,60,70,100)) + surgery,
            scale = 1)

## so it can be used easily in a pipeline.
hearta %>%
  ntbt_pyears(Surv(stop/365.25, event) ~
                cut(age + 48, c(0,50,60,70,100)) + surgery,
              scale = 1)

## survConcordance
## Original function to interface
survConcordance(Surv(time, status) ~ age, data=lung)

## The interface reverses the order of data and formula
ntbt_survConcordance(data=lung, Surv(time, status) ~ age)

## so it can be used easily in a pipeline.
lung %>%
  ntbt_survConcordance(Surv(time, status) ~ age)

## survexp
## Original function to interface
fit1 <- survexp(futime ~ 1,data=jasa,
                rmap=list(sex="male", year=accept.dt,
                          age=(accept.dt-birth.dt)),
                method='conditional')
```

```
summary(fit1, times=1:10*182.5, scale=365)

## The interface reverses the order of data and formula
fit1 <- ntbt_survexp(data=jasa, futime ~ 1,
                 rmap=list(sex="male", year=accept.dt,
                           age=(accept.dt-birth.dt)),
                 method='conditional')
summary(fit1, times=1:10*182.5, scale=365)

## so it can be used easily in a pipeline.
jasa %>%
  ntbt_survexp(futime ~ 1,
            rmap=list(sex="male", year=accept.dt,
                      age=(accept.dt-birth.dt)),
            method='conditional') %>%
  summary(times=1:10*182.5, scale=365)

## survfit
## Original function to interface
survfit(Surv(time, status) ~ x, data = aml)

## The interface reverses the order of data and formula
ntbt_survfit(data = aml, Surv(time, status) ~ x)

## so it can be used easily in a pipeline.
aml %>%
  ntbt_survfit(Surv(time, status) ~ x)

aml %>%
  ntbt_survfit(Surv(time, status) ~ x) %>%
  plot(lty = 2:3)


## survreg
## Original function to interface
survreg(Surv(time, status) ~ ph.ecog + age + strata(sex), lung)

## The interface reverses the order of data and formula
ntbt_survreg(lung, Surv(time, status) ~ ph.ecog + age + strata(sex))

## so it can be used easily in a pipeline.
lung %>%
  ntbt_survreg(Surv(time, status) ~ ph.ecog + age + strata(sex))

## survSplit
## Original function to interface
vet2 <- survSplit(Surv(time, status) ~., veteran,
                  cut = c(60, 120), episode = "timegroup")
coxph(Surv(tstart, time, status) ~ karno*strata(timegroup) +
                age + trt, data = vet2)

## The interface reverses the order of data and formula
vet2 <- ntbt_survSplit(veteran, Surv(time, status) ~.,
```

```
                        cut = c(60, 120), episode = "timegroup")
ntbt_coxph(data = vet2, Surv(tstart, time, status) ~
           karno*strata(timegroup) + age + trt)

## so it can be used easily in a pipeline.
veteran %>%
  ntbt_survSplit(Surv(time, status) ~.,
                 cut = c(60, 120),
                 episode = "timegroup") %>%
  ntbt_coxph(Surv(tstart, time, status) ~
             karno*strata(timegroup) + age + trt)

## End(Not run)
```

---

`SwarmSVM` *Interfaces for SwarmSVM package for data science pipelines.*

---

## Description

Interfaces to `SwarmSVM` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_alphasvm(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(SwarmSVM)

## ntbt_alphasvm: Support Vector Machines taking initial alpha values
data(iris)
## Original function to interface
alphasvm(Species ~ ., data = iris)

## The interface puts data as first parameter
ntbt_alphasvm(iris, Species ~ .)

## so it can be used easily in a pipeline.
iris %>%
  ntbt_alphasvm(Species ~ .)

## End(Not run)
```

---

`systemfit` *Interfaces for systemfit package for data science pipelines.*

---

## Description

Interfaces to `systemfit` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_systemfit(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(systemfit)


## ntbt_systemfit: Linear Equation System Estimation
data("Kmenta")
eqDemand <- consump ~ price + income
eqSupply <- consump ~ price + farmPrice + trend
system <- list( demand = eqDemand, supply = eqSupply )


## Original function to interface
fitols <- systemfit(system, data = Kmenta)
print(fitols)

## The interface puts data as first parameter
fitols <- ntbt_systemfit(Kmenta, system, "<|F|>")
## Need "<|F|>" (at least for now) to clarify it is a formula
print(fitols)

## so it can be used easily in a pipeline.
## Need "<|F|>" (at least for now) to clarify it is a formula
Kmenta %>%
  ntbt_systemfit(system, "<|F|>") %>%
  print()

## End(Not run)
```

---

`tree` *Interfaces for tree package for data science pipelines.*

---

## Description

Interfaces to `tree` functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_tree(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(tree)

data(cpus, package="MASS")

## Original function to interface
cpus.ltr <- tree(log10(perf) ~ syct+mmin+mmax+cach+chmin+chmax, cpus)
cpus.ltr
summary(cpus.ltr)
plot(cpus.ltr)
text(cpus.ltr)

## The interface reverses the order of data and formula
cpus.ltr <- ntbt_tree(cpus, log10(perf) ~ syct+mmin+mmax+cach+chmin+chmax)
cpus.ltr
summary(cpus.ltr)
plot(cpus.ltr);  text(cpus.ltr)

## so it can be used easily in a pipeline.
cpus %>%
  ntbt_tree(log10(perf) ~ syct + mmin + mmax + cach + chmin + chmax) %>%
  summary()

cpus %>%
  ntbt_tree(log10(perf) ~ syct + mmin + mmax + cach + chmin + chmax) %T>%
  plot() %>%
  text()

iris %>%
  ntbt_tree(Species ~.) %>%
  summary()

## End(Not run)
```

---

vcd *Interfaces for vcd package for data science pipelines.*

---

## Description

Interfaces to vcd functions that can be used in a pipeline implemented by magrittr.

## Usage

```
ntbt_agreementplot(data, ...)
ntbt_assoc(data, ...)
ntbt_cd_plot(data, ...)
ntbt_cotabplot(data, ...)
ntbt_loddsratio(data, ...)
ntbt_mosaic(data, ...)
ntbt_sieve(data, ...)
ntbt_spine(data, ...)
ntbt_structable(data, ...)
```

## Arguments

| | |
|---|---|
| data | data frame, tibble, list, ... |
| ... | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(vcd)


## ntbt_agreementplot: Bangdiwala's Observer Agreement Chart
## Original function to interface
agreementplot(Freq ~ Gender + Admit, as.data.frame(UCBAdmissions))

## The interface puts data as first parameter
ntbt_agreementplot(as.data.frame(UCBAdmissions), Freq ~ Gender + Admit)

## so it can be used easily in a pipeline.
as.data.frame(UCBAdmissions) %>%
  ntbt_agreementplot(Freq ~ Gender + Admit)
```

```
## ntbt_assoc: Extended Association Plots
## Original function to interface
assoc(Freq ~ Gender + Admit, data = as.data.frame(UCBAdmissions))

## The interface puts data as first parameter
ntbt_assoc(as.data.frame(UCBAdmissions), Freq ~ Gender + Admit)

## so it can be used easily in a pipeline.
as.data.frame(UCBAdmissions) %>%
  ntbt_assoc(Freq ~ Gender + Admit)


## ntbt_cd_plot: Conditional Density Plots
data("Arthritis")
## Original function to interface
cd_plot(Improved ~ Age, data = Arthritis)

## The interface puts data as first parameter
ntbt_cd_plot(Arthritis, Improved ~ Age)

## so it can be used easily in a pipeline.
Arthritis %>%
  ntbt_cd_plot(Improved ~ Age)


## ntbt_cotabplot: Coplot for Contingency Tables
## Original function to interface
cotabplot(~ Admit + Gender | Dept, data = UCBAdmissions)

## The interface puts data as first parameter
ntbt_cotabplot(UCBAdmissions, ~ Admit + Gender | Dept)

## so it can be used easily in a pipeline.
UCBAdmissions %>%
  ntbt_cotabplot(~ Admit + Gender | Dept)


## ntbt_loddsratio: Calculate Generalized Log Odds Ratios for Frequency Tables
data(Punishment, package = "vcd")

## Original function to interface
loddsratio(Freq ~ memory + attitude | age + education, data = Punishment)

## The interface puts data as first parameter
ntbt_loddsratio(Punishment, Freq ~ memory + attitude | age + education)

## so it can be used easily in a pipeline.
Punishment %>%
  ntbt_loddsratio(Freq ~ memory + attitude | age + education)
```

```
## ntbt_mosaic: Extended Mosaic Plots
library(MASS)
data("Titanic")
mosaic(Titanic)

## Original function to interface
mosaic(~ Sex + Age + Survived, data = Titanic,
       main = "Survival on the Titanic", shade = TRUE, legend = TRUE)

## The interface puts data as first parameter
ntbt_mosaic(Titanic, ~ Sex + Age + Survived,
            main = "Survival on the Titanic", shade = TRUE, legend = TRUE)

## so it can be used easily in a pipeline.
Titanic %>%
  ntbt_mosaic(~ Sex + Age + Survived,
              main = "Survival on the Titanic", shade = TRUE, legend = TRUE)


## ntbt_sieve: Extended Sieve Plots
data("VisualAcuity")

## Original function to interface
sieve(Freq ~ right + left,  data = VisualAcuity)

## The interface puts data as first parameter
ntbt_sieve(VisualAcuity, Freq ~ right + left)

## so it can be used easily in a pipeline.
VisualAcuity %>%
  ntbt_sieve(Freq ~ right + left)


## ntbt_spine: Spine Plots and Spinograms
data("Arthritis")

## Original function to interface
spine(Improved ~ Treatment, data = Arthritis)

## The interface puts data as first parameter
ntbt_spine(Arthritis, Improved ~ Treatment)

## so it can be used easily in a pipeline.
Arthritis %>%
  ntbt_spine(Improved ~ Treatment)


## ntbt_structable: Structured Contingency Tables
## Original function to interface
structable(Sex + Class ~ Survived + Age, data = Titanic)

## The interface puts data as first parameter
ntbt_structable(Titanic, Sex + Class ~ Survived + Age)
```

```
## so it can be used easily in a pipeline.
Titanic %>%
  ntbt_structable(Sex + Class ~ Survived + Age)

## End(Not run)
```

---

vegan *Interfaces for vegan package for data science pipelines.*

---

## Description

Interfaces to vegan functions that can be used in a pipeline implemented by `magrittr`.

## Usage

```
ntbt_adipart(data, ...)
ntbt_adonis(data, ...)
ntbt_adonis2(data, ...)
ntbt_bioenv(data, ...)
ntbt_capscale(data, ...)
ntbt_cca(data, ...)
ntbt_dbrda(data, ...)
ntbt_envfit(data, ...)
ntbt_gdispweight(data, ...)
ntbt_multipart(data, ...)
ntbt_ordicloud(data, ...)
ntbt_ordisplom(data, ...)
ntbt_ordisurf(data, ...)
ntbt_ordixyplot(data, ...)
```

## Arguments

| | |
|---|---|
| `data` | data frame, tibble, list, ... |
| `...` | Other arguments passed to the corresponding interfaced function. |

## Details

Interfaces call their corresponding interfaced function.

## Value

Object returned by interfaced function.

## Author(s)

Roberto Bertolusso

## Examples

```
## Not run:
library(intubate)
library(magrittr)
library(vegan)

## There is cheating going on on these examples,
## as the cases need two datasets, and only one
## is being piped... I may get back to this down the line.
## For now, please close an eye.

## ntbt_adipart: Additive Diversity Partitioning and Hierarchical Null Model Testing
data(mite)
data(mite.xy)
## Function to get equal area partitions of the mite data
cutter <- function (x, cut = seq(0, 10, by = 2.5)) {
    out <- rep(1, length(x))
    for (i in 2:(length(cut) - 1))
        out[which(x > cut[i] & x <= cut[(i + 1)])] <- i
    return(out)}
## The hierarchy of sample aggregation
levsm <- with(mite.xy, data.frame(
    l1=1:nrow(mite),
    l2=cutter(y, cut = seq(0, 10, by = 2.5)),
    l3=cutter(y, cut = seq(0, 10, by = 5)),
    l4=cutter(y, cut = seq(0, 10, by = 10))))

## Original function to interface
set.seed(1)
adipart(mite ~ ., levsm, index="richness", nsimul=19)

## The interface puts data as first parameter
set.seed(1)
ntbt_adipart(levsm, mite ~ ., index="richness", nsimul=19)

## so it can be used easily in a pipeline.
set.seed(1)
levsm %>%
  ntbt_adipart(mite ~ ., index="richness", nsimul=19)


## ntbt_adonis: Permutational Multivariate Analysis of Variance Using Distance Matrices
data(dune)
data(dune.env)

## Original function to interface
set.seed(1)
adonis(dune ~ Management*A1, data = dune.env)
adonis2(dune ~ Management*A1, data = dune.env)

## The interface puts data as first parameter
set.seed(1)
```

```
ntbt_adonis(dune.env, dune ~ Management*A1)
ntbt_adonis2(dune.env, dune ~ Management*A1)

## so it can be used easily in a pipeline.
set.seed(1)
dune.env %>%
  ntbt_adonis(dune ~ Management*A1)
dune.env %>%
  ntbt_adonis2(dune ~ Management*A1)


## ntbt_bioenv: Best Subset of Environmental Variables with
##              Maximum (Rank) Correlation with Community Dissimilarities
data(varespec)
data(varechem)

## Original function to interface
bioenv(wisconsin(varespec) ~ log(N) + P + K + Ca + pH + Al, varechem)

## The interface puts data as first parameter
ntbt_bioenv(varechem, wisconsin(varespec) ~ log(N) + P + K + Ca + pH + Al)

## so it can be used easily in a pipeline.
varechem %>%
  ntbt_bioenv(wisconsin(varespec) ~ log(N) + P + K + Ca + pH + Al)


## ntbt_capscale: [Partial] Distance-based Redundancy Analysis
## ntbt_dbrda:
## Original function to interface
capscale(varespec ~ N + P + K + Condition(Al), varechem,
         dist="bray")
dbrda(varespec ~ N + P + K + Condition(Al), varechem,
         dist="bray")

## The interface puts data as first parameter
ntbt_capscale(varechem, varespec ~ N + P + K + Condition(Al),
              dist="bray")
ntbt_dbrda(varechem, varespec ~ N + P + K + Condition(Al),
           dist="bray")

## so it can be used easily in a pipeline.
varechem %>%
  ntbt_capscale(varespec ~ N + P + K + Condition(Al), dist="bray")
varechem %>%
  ntbt_dbrda(varespec ~ N + P + K + Condition(Al), dist="bray")


## ntbt_cca: [Partial] [Constrained] Correspondence Analysis
##           and Redundancy Analysis

## Original function to interface
cca(varespec ~ Al + P*(K + Baresoil), data = varechem)
```

```
## The interface puts data as first parameter
ntbt_cca(varechem, varespec ~ Al + P*(K + Baresoil))

## so it can be used easily in a pipeline.
varechem %>%
  ntbt_cca(varespec ~ Al + P*(K + Baresoil))


## ntbt_gdispweight: Dispersion-based weighting of species counts
data(mite, mite.env)
## Original function to interface
gdispweight(mite ~ Shrub + WatrCont, data = mite.env)

## The interface puts data as first parameter
ntbt_gdispweight(mite.env, mite ~ Shrub + WatrCont)

## so it can be used easily in a pipeline.
mite.env %>%
  ntbt_gdispweight(mite ~ Shrub + WatrCont)


## ntbt_envfit: Fits an Environmental Vector or Factor onto an Ordination
ord <- cca(dune)

## Original function to interface
envfit(ord ~ Moisture + A1, dune.env, perm = 0)

## The interface puts data as first parameter
ntbt_envfit(dune.env, ord ~ Moisture + A1, perm = 0)

## so it can be used easily in a pipeline.
dune.env %>%
  ntbt_envfit(ord ~ Moisture + A1, perm = 0)

## ntbt_multipart: Multiplicative Diversity Partitioning
## Original function to interface
multipart(mite ~ ., levsm, index = "renyi", scales = 1, nsimul = 19)

## The interface puts data as first parameter
ntbt_multipart(levsm, mite ~ ., index = "renyi", scales = 1, nsimul = 19)

## so it can be used easily in a pipeline.
levsm %>%
  ntbt_multipart(mite ~ ., index = "renyi", scales = 1, nsimul = 19)


## ntbt_ordisurf: Fit and Plot Smooth Surfaces of Variables on Ordination.
vare.dist <- vegdist(varespec)
vare.mds <- monoMDS(vare.dist)

## Original function to interface
ordisurf(vare.mds ~ Baresoil, varechem, bubble = 5)
```

```
## The interface puts data as first parameter
ntbt_ordisurf(varechem, vare.mds ~ Baresoil, bubble = 5)

## so it can be used easily in a pipeline.
varechem %>%
  ntbt_ordisurf(vare.mds ~ Baresoil, bubble = 5)


## ntbt_ordixyplot: Trellis (Lattice) Plots for Ordination
## Original function to interface
ordicloud(ord, form = CA2 ~ CA3*CA1, groups = Manure, data = dune.env)
ordisplom(ord, data = dune.env, form = ~ . | Management, groups=Manure)
ordixyplot(ord, data=dune.env, form = CA1 ~ CA2 | Management, groups=Manure)

## The interface puts data as first parameter
ntbt_ordicloud(dune.env, ord, form = CA2 ~ CA3*CA1, groups = Manure)
ntbt_ordisplom(dune.env, ord, form = ~ . | Management, groups=Manure)
ntbt_ordixyplot(dune.env, ord, form = CA1 ~ CA2 | Management, groups=Manure)

## so it can be used easily in a pipeline.
dune.env %>%
  ntbt_ordicloud(ord, form = CA2 ~ CA3*CA1, groups = Manure)
dune.env %>%
  ntbt_ordisplom(ord, form = ~ . | Management, groups=Manure)
dune.env %>%
  ntbt_ordixyplot(ord, form = CA1 ~ CA2 | Management, groups=Manure)

## End(Not run)
```

# Index