Package: sktools (via r-universe)

August 16, 2024

Type Package

Title Helpful functions used in my courses

Version 0.05

Date 2013-04-23

Author Sebastian Kranz

Maintainer Sebastian Kranz <sebastian.kranz@uni-ulm.de>

Description Several helpful functions that I use in my courses

License GPL (>= 2)

Depends restorepoint, data.table, lubridate, stringr, R.utils

Repository https://skranz.r-universe.dev

RemoteUrl https://github.com/skranz/sktools

RemoteRef master

RemoteSha 8e629e09f0b72b1471b4a4eb89f3ada0a43e4aaf

Contents

`automatic.type.conversion`

Try to transform a vector or columns that are stored as characters or factors of a data.frame automatically in the approbriate types

---

Description

So far just attempts to convert date and date.times

Usage

```
automatic.type.conversion(x, max.failure.rate = 0.3, quiet = FALSE,
  name = "", cols = NULL, date.class = "POSIX.ct",
  date.time.class = "POSIX.ct", factorsAsStrings = FALSE,
  stringsAsFactors = FALSE, thousand.sep = NULL, ...)
```

Arguments

x	a vector or data frame that shall be converted
max.failure.rate	maximum share of rows that can be failed to convert so that conversion still takes place

Examples

```
## Not run:
  df = data.frame(text=c("c","b","a"),mixed=as.factor(c("a",NA,56)),
    dates = c("1.1.1975","1.1.2010","8.3.1999"),
    date.times = c("1.1.1975 00:10","2010-5-5 12pm","4:30:15"),
    times = c("00:10","12pm",5),
    char.num = as.factor(c("1973","1972","3")),
    half.num = c("145","..","1345")
  )
  df =automatic.type.conversion(df,quiet=FALSE)
  d = df$dates
  d[1]<"1.1.2012"

## End(Not run)
```

---

clone.list *#Set and retrieve global options for restore points # #*

Description

#Set and retrieve global options for restore points # #

Usage

```
## S3 method for class 'list'
clone(li, use.copied.ref = FALSE)
```

Arguments

options	a list of options that shall be set. Possible options are listed below #
storing	TRUE / FALSE enable or disable storing of options, setting storing = FALSE basicially turns off debugging via restore points #
use.browser	TRUE/FALSE If FALSE (default), when options are restored they are simply copied into the global environment and the R console is directly used for debugging. If TRUE a browser mode will be started when objects are restored. It is still possible to parse all R commands into the browser and to use copy and paste. To quit the browser press ESC in the R console. The advantage of the browser is that all objects are stored in a newly generated environment that mimics the environemnt of the original function, i.e. global varariables are not overwritten.

Furthermore in the browser mode, one can pass the ... object to other functions, while this does not work in the global environment. The drawback is that at least on my computer the approach does not currently work under RStudio, which crashes when it is attempted to parse a command from the console using the parse() command. #

`store` if FALSE don't store objects if restore.point or store.objects is called. May save time. If TRUE (default) turn on storage again. #

`name` key under which the objects are stored. For restore points at the beginning of a function, I would suggest the name of that function. #

`deep.copy` if TRUE (default) try to make deep copies of objects that are by default copied by reference. Works so far for environments (recursivly) and data.tables. The function will search lists whether they contain reference objects, but for reasons of speed not yet in other containers. E.g. if an evironment is stored in a data.frame, only a shallow copy will be made. Setting deep.copy = FALSE may be useful if storing takes very long and variables that are copied by reference are not used or not modified. #

`force` store even if set.storing(FALSE) has been called #

`name` key under which the objects are stored, typical the name of the calling function. If name is NULL by default the name of the calling function is chosen #

`deep.copy` if TRUE (default) variables that are copied by reference (in the moment environments and data.tables) will be stored as deep copy. May take long for large variables but ensures that the value of the stored variable do not change #

`force` store even if do.store(FALSE) has been called #

`store.if.called.from.global`
if the function is called from the global environment and store.if.called.from.global FALSE (default) does not store objects when called from the global environment but does nothing instead. #

Value

returns nothing, just called for side effects #

`cluster.vcov` *R function for computing two-way cluster-robust standard errors.*

Description

The code below was adapted by Ian Gow on May 16, 2011 using code supplied via Mitchell Petersen's website by Mahmood Arai, Jan 21, 2008.

Usage

```
cluster.vcov(data, fm, cluster1, cluster2 = NULL)
```

Details

Apart from a little cleanup of the code, the main difference between this and the earlier code is in the handling of missing values. Look at the file cluster.test.R to see example usage. Note that care should be taken to do subsetting outside of the call to lm or glm, as it is difficult to recon- struct subsetting of this kind from the fitted model. However, the code does handle transformations of variables in the model (e.g., logs). Please report any bugs, suggestions, or errors to iandgow

---

`copy.into.env` *#Restore stored objects by copying them into the global environment # #*

Description

#Restore stored objects by copying them into the global environment # #

Usage

```
copy.into.env(source = sys.frame(sys.parent(1)),
  dest = sys.frame(sys.parent(1)), names = NULL, exclude = NULL)
```

Arguments

`name`	name under which the variables have been stored #
`dest`	environment into which the stored variables shall be copied. By default the global environment. #
`was.forced`	flag whether storage of objects was forced. If FALSE (default) a warning is shown if restore.objects is called and is.storing()==FALSE, since probably no objects have been stored. #
`source`	a list or environment from which objects are copied
`the`	enviroenment into which objects shall be copied
`names`	optionally a vector of names that shall be copied. If null all objects are copied
`exclude`	optionally a vector of names that shall not be copied

Value

returns nothing but automatically copies the stored variables into the global environment #

`currentenv` *Get the environment in which this function is called*

Description

Get the environment in which this function is called

Usage

```
currentenv()
```

dyplot *Dynamic time series plots (wrapper to dygraph)*

Description

The function is a wrapper to the dygraph function in the package dygraph. It shows time series plots with interactive java script. While dygraph needs an xts time series object, dyplot works with a data frame. Missing observations can be filled if period is provided.

Usage

```
dyplot(data, xcol = colnames(data)[1], ycol = setdiff(colnames(data), xcol),
  interval = NULL)
```

Arguments

data	a data.frame
xcol	name of the column with the x-Axis variable. Ideally a datetime object
ycol	names of the columns that are shown on the yaxis
interval	if you have missing rows you can specify the interval of you data, e.g. "day" or "year" or "hour" to fill the gaps

eval.string.dplyr *Internal function used by s_filter, s_select etc.*

Description

Internal function used by s_filter, s_select etc.

Usage

```
eval.string.dplyr(.data, .fun.name, ...)
```

explore.3d.fun — *Uses Manipulate to explore the function z.fun*

Description

Uses Manipulate to explore the function z.fun

Usage

```
explore.3d.fun(z.fun, plot.type = "image", xrange, yrange = xrange,
  main = "Function Explorer", xlab = "x", ylab = "y", num.colors = 30,
  pal.colors = c("red", "white", "blue"), Vectorize.z.fun = TRUE,
  grid.length.default = 8, num.color.default = 100, image.fun = NULL,
  extra.control = list(), add.plot.fun = NULL, ...)
```

Examples

```
## Not run:
  z.fun = function(x,y,a=1,b=1,c=1,d=1,e=1,...) {
    a*x^2+b*y^2+c*x^3+d*y^3+e*x*y
  }
  explore.3d.fun(z.fun=z.fun,plot.type="image",xrange=c(-2,2),Vectorize.z.fun=F,
                 extra.control = list(a=slider(-5.0,5.0,1,step=0.01)),
                 num.color.default=30)

## End(Not run)
```

explore.image — *Uses Manipulate to interactively change some parameters of an image.plot*

Description

Uses Manipulate to interactively change some parameters of an image.plot

Usage

```
explore.image(x, y, z, xlab = "x", ylab = "y", main = "",
  num.colors = 30, add.plot.fun = NULL, pal.colors = c("red", "white",
  "blue"))
```

get.coef.and.se	*Returns estimated coefficients and standard errors from a regression model. One can also specify linear or non-linear transformations of the original coefficients, standard errors are then computed using the delta method.*

Description

Returns estimated coefficients and standard errors from a regression model. One can also specify linear or non-linear transformations of the original coefficients, standard errors are then computed using the delta method.

Usage

```
get.coef.and.se(reg, trans.coef = NULL, coef. = coef(reg),
  vcov. = vcov(reg))
```

Arguments

reg	results of any estimation (like lm) that has methods coef(reg) and vcov(reg)
trans.coef	a list

Examples

```
## Not run:
  get.coef.and.se(m1,trans.coef = list(ratio="t1/t2",prod="t1*t2",inv_t1="1/t1"))

  coef(m1)["(Intercept)"]
  se(m1)
  ?coef

  deltaMethod(m1, "b1+b2", parameterNames= paste("b", 0:2, sep=""))
  deltaMethod(m1, "t1/t2") # use names of preds. rather than coefs.
  deltaMethod(m1, "t1/t2", vcov=hccm) # use hccm function to est. vars.
  # to get the SE of 1/intercept, rename coefficients
  deltaMethod(m1, "1/b0", parameterNames= paste("b", 0:2, sep=""))
  # The next example calls the default method by extracting the
  # vector of estimates and covariance matrix explicitly
  deltaMethod(coef(m1), "t1/t2", vcov.=vcov(m1))

## End(Not run)
```

`is.false` *Checks whether val is FALSE, NULL return FALSE If val is a vector vectorized over val*

Description

Checks whether val is FALSE, NULL return FALSE If val is a vector vectorized over val

Usage

```
is.false(val)
```

`is.true` *Checks whether val is TRUE, NULL return FALSE If val is a vector vectorized over val*

Description

Checks whether val is TRUE, NULL return FALSE If val is a vector vectorized over val

Usage

```
is.true(val)
```

`ivregress` *An extended version of ivreg from AER that allows robust standard errors*

Description

An extended version of ivreg from AER that allows robust standard errors

Usage

```
ivregress(..., robust = "not", cluster1 = NULL, cluster2 = NULL)
```

make.date.time *Builds a date time object from different components*

Description

Builds a date time object from different components

Usage

```
make.date.time(year, month = 1, day = 1, hour = 0, min = 0, sec = 0,
  date, tz = "")
```

Arguments

year	an integer specifying the year
month	an integer specifying the month
day	an integer specifying the day
hour	an integer specifying the hour
min	an integer specifying the minute
sec	an integer specifying the second
date	a date object or a character that can be converted to a date. If provided then year, month and day will be extracted from this date object

Examples

```
## Not run:
  make.date.time(date="2014-01-02", hour=14)

## End(Not run)
```

matrix.to.grid *Convert data in matrix format to grid format with key columns corresponding to for row and col names and a value colum*

Description

Convert data in matrix format to grid format with key columns corresponding to for row and col names and a value colum

Usage

```
matrix.to.grid(dat, row.var = "row", col.var = "col", val.var = "value",
  row.values = rownames(dat), col.values = colnames(dat))
```

Arguments

dat	a data frame or matrix in matrix format
row.var	name of the variable corresponding to different rows
col.var	name of the variable corresponding to different columns
val.var	name of the variable corresponding to the values of the matrix
row.values	values of the row variable
col.values	values of the column variable

---

named *Set specified names for x and return the named object*

Description

Set specified names for x and return the named object

Usage

```
named(x, names, colnames, rownames)
```

Examples

```
## Not run:
  named(1:3,c("A","B","C"))

## End(Not run)
```

---

nc *Creates a vector that is named by the names of its arguments*

Description

Creates a vector that is named by the names of its arguments

Usage

```
nc(...)
```

nlist — *Creates a list that is named by the names of its arguments*

Description

Creates a list that is named by the names of its arguments

Usage

```
nlist(...)
```

plot.with.legend — *Draws a base plot with a legend to the right, must play around with width to get a good looking result*

Description

Draws a base plot with a legend to the right, must play around with width to get a good looking result

Usage

```
## S3 method for class 'with.legend'
plot(plot.expr, legend, fill, width = 5, bty = "n",
  ...)
```

print.WuHausmanTest — *This function will be called when the results of a wu.hausman.test will be printed*

Description

This function will be called when the results of a wu.hausman.test will be printed

Usage

```
## S3 method for class 'WuHausmanTest'
print(test)
```

quick.by *Quick version of by using internally data.table*

Description

Quick version of by using internally data.table

Usage

```
quick.by(df, by = NULL, expr, add.col = FALSE, as.data.table = FALSE,
  keep.row.order = add.col, ...)
```

Arguments

df	a data.frame that shall be aggregated / transformed
by	either a character vector with the columns of df over which grouping shall take place or a list vectors of size NROW(df) containing group indices
expr	a string containing an R expr operating on columns in df that shall be evaluated, corresponds to j parameter in data.table, can be a named list to generate multiple columns (see data.table introduction).
add.col	if TRUE the data generated by group-wise evaluation of expr will be cbinded to the corresponding rows in df

Value

a data.frame with values of indices in by and returning expressions

Author(s)

Sebastian Kranz

Examples

```
## Not run:
  # Simulate a data set of time needed to run a given distance
  # individuals differ by age and gender
  T <- 10
  id <- 1:T
  age    <- sample(10:12, T, replace=TRUE)
  gender <- sample(c("M","F"), T, replace=TRUE)
  time <- runif(T,10,100) - sqrt(age) - (gender=="M")*1
  df <- data.frame(id,age,gender,time)

  # Mean time for each gender group
  quick.by(df,"avgtime=mean(time)",by=c("gender"))
  # Mean time for each age / gender group
  quick.by(df,"avgtime=mean(time)",by=c("age","gender"))
```

```
# Mean time for each age / gender group and number of ind
quick.by(df,"avgtime=mean(time), groupsize=length(time)",by=c("age","gender"))


# Best time in each age / gender group and identity of winner
quick.by(df,"best.time=min(time),id.winner=id[which.min(time)]",by=c("age","gender"))

# Add best.time to original df
quick.by(df,"best.time=min(time), group.size=length(time)",by=c("age","gender"), add.col=TRUE)

# Add best time and groupsize to original df
quick.by(df,"best.time:=min(time)",by=c("age","gender"))


# Add best time within group and groupsize to original df and compute gap to best for each individual
new.df <- quick.by(df,"best.time=min(time), group.size=length(time)",by=c("age","gender"), add.col=TRUE)

# Compute gap to best.time
new.df$gap <- new.df$time - new.df$best.time
new.df


# Compare speed of quick.by with other data aggregation tools
library(plyr)

T <- 1000
ind <- sample(1:1000, T, replace=TRUE)
val <- 1:T*0.01
df <- data.frame(ind,val)
dt <- as.data.table(df)
library(rbenchmark)
benchmark(
  data.table = dt[,list(sum=sum(val)),by="ind"],
  data.table.add = dt[,sum:=sum(val),by="ind"],
  quick.by=quick.by(df,by="ind","sum=sum(val)"),
  quick.by.add=quick.by(df,by="ind","sum=sum(val)",add.col=TRUE),
  ddply = ddply(df,"ind",function(df) sum(df$val)),
  by = by(df,df$ind,function(df) sum(df$val)),
  replications=2,order="relative")
# While using data.table directly is the fastest method, quick.by is not much slower and substantially faster than

## End(Not run)
```

---

quick.df *A function that generates a data.frame more quickly from vectors of equal lengths*

Description

The function can run substantially quicker than data.frame(...), since no checks will be performed. Yet, the function only works if all vectors have the same length.

Usage

```
quick.df(...)
```

Arguments

...	vectors that have equal lengths and will be combined to a data frame

Value

a data.frame

Examples

```
## Not run:

  df = quick.df(b=1:5,c=1:5)
  df

  library(rbenchmark)

## End(Not run)
```

---

remove.global.variables *Remove all global variables (not functions)*

Description

Remove all global variables (not functions)

Usage

```
remove.global.variables(exclude = NULL, envir = .GlobalEnv)
```

---

robust.se *Computes robust standard errors for the coefficients of a fitted model*

Description

Computes robust standard errors for the coefficients of a fitted model

Usage

```
robust.se(fm, robust = c("HAC", "HC", "cluster"), ...)
```

`run.scenarios` *A helper function to simulate different scenarios. So far the function is just a simple wrapper to mapply.*

Description

A helper function to simulate different scenarios. So far the function is just a simple wrapper to mapply.

Usage

```
run.scenarios(fun, par, show.progress.bar = interactive(), other.par = NULL,
  ...)
```

Arguments

`fun`	a function that runs the simulation for given set of parameters and returns the results as a data.frame. The data.frame must have the same number of columns, for all possible parameter combinations, the number of rows can differ, however.
`par`	a list of the scalar parameters used by sim.fun. If some parameters in the list are vectors, run the simulation for every combination of parameter values.
`show.progress.bar`	= TRUE or FALSE. Shall a progress bar be shown?
`other.par`	a list of other parameters that will be used by sim.fun that do not vary across simulations and won't be specified in the results.

Value

returns a data.frame that combines the results for all scenarios replication of each dgp and each estimation procedure and each parameter constellation. The first column is an integer number specifying the number of the scenario. Then columns follow for each paramter. The remaining columns are the returned values by sim.fun. The results can be conviniently analysed graphically, e.g. with ggplot2. Aggregate statistics can for example be computed by quick.by.

Examples

```
## Not run:
  # Compute some values of a linear and quadratic function
  f = function(a,b,c) {
    x = seq(0,1,length=5)
    return(data.frame(x=x,lin=a+b*x,quad=a+b*x+c*x^2))
  }
  ret = run.scenarios(f,par=list(a=c(0,2),b=c(-1,1,2),c=c(0,1,3,4)))
  ret

  # Plot using ggplot2
  library(ggplot2)
```

```
  qplot(x=x,y=quad,data=ret, facets= a ~ b, color=as.factor(c),geom="line")


## End(Not run)
```

`simulation.study` *A helper function to conduct a simulation study for different parameter combinations*

Description

A helper function to conduct a simulation study for different parameter combinations

Usage

```
simulation.study(fun, par = NULL, repl = 1, ...,
  show.progress.bar = interactive(), add.run.id = TRUE, colnames = NULL,
  same.seeds.each.par = TRUE, seeds = floor(runif(repl, 0,
  .Machine$integer.max)), LAPPLY = lapply)
```

Arguments

`fun`	a function that returns a vector, data.frame or matrix
`par`	an optional list that specifies parameters for different scenarios. fun will be called repl times for each parameter combinantion of the parameters specified in par
`repl`	for Monte-Carlo simulation the number of times fun is called for each parameter combinantion
`...`	additional parameters that will be used by fun
`show.progress.bar`	shall a progress bar been shown?
`add.run.id`	shall a column be added that is a unique value for every unique call of fun
`colnames`	optionally a vector of colnames for the results returned by fun. It is quicker to set colnames just in the end, instead of fun setting names.
`same.seeds.each.par`	if TRUE (default) then we set for all parameter combinations with which the function is called the same random seed in the i'th replication. One effect e.g. is that if we draw some disturbances eps in our simulation the same disturbances will be drawn in the i'th repitition for all parameter values, we study. This typically facilitates the analysis of comparative statics.
`seeds`	if same.seeds.each.par = TRUE one can manually provide a vector of random seeds of length repl.
`LAPPLY`	a function that has the same behavior as lapply. One can use an different function, e.g. in order to parallelize the execution when running a simulation on a computer cluster.

Value

returns a data.frame that combines the results of all calls to fun and adds the corresponding parameter combinantion and an index for the actual replication. The data.frame can be conviniently analysed graphically, e.g. with ggplot2

Examples

```
## Not run:

 # A function that simulates demand
demand.sim = function(beta0=100,beta1=-1,sigma.eps=0.4,T=200, p.min=0, p.max= (-beta0/beta1)) {
   # Draw prices always in the same fashion, with a fixed random seed
   p=with.random.seed(runif(T,p.min,p.max), seed=123456789)
   eps = rnorm(T,0,sigma.eps) # Demand Shock for each market
   # Realized demand
   q = beta0 + beta1* p+eps # = beta0 + beta1*p + eps
   #data.frame(p=p,q=q,eps=eps)
   quick.df(p=p,q=q,eps=eps)
}
# A function that simulates data and performs OLS estimation on the simulated data
est.sim.fun = function(beta0=100,beta1=-1,...) {
  df = demand.sim(...)
  y = df$q
  X = cbind(1,df$p)
  reg = lm.fit(x=X,y=y)
 quick.df(name=c("beta0.hat","beta1.hat"),coef = coef(reg),true.coef = c(beta0,beta1),se = sqrt(diag(vcov.lm.fi
}
demand.sim(T=3)
set.seed(1234)
est.sim.fun(T=20)

simulation.study(fun=demand.sim, par=list(T=c(1,2)), repl=2, add.run.id=TRUE)


sim = simulation.study(fun=est.sim.fun, par=list(T=c(10,50,100,200), sigma.eps=c(1,3)), repl=50, add.run.id=TRU

head(sim)


library(ggplot2)

# Select only the estimate of beta1.hat
dat = sim[sim$name=="beta0.hat",]

qplot(coef,geom="density",alpha = I(0.6),data=dat, group=T, fill=as.factor(T), facets=sigma.eps~.) #+ coord_car

# Compute sample MSE and Bias
agg = quick.by(dat,list(
      mse = mean((coef-true.coef)^2),
      bias=mean(coef-true.coef)),
      by=c("T","sigma.eps"))
agg
```

```
 qplot(y=est.mse,x=T,data=agg,geom="line",group=sigma.eps, color=as.factor(sigma.eps),ylab="Estimated MSE",size
 qplot(y=est.bias,x=T,data=agg,geom="line",group=sigma.eps, color=as.factor(sigma.eps), ylab="Estimated bias",s

  ret

## End(Not run)
```

---

skMotionChart *Convenience interface to gvisMotionChart that transforms timevar into a date and allows to set default columns*

Description

Convenience interface to gvisMotionChart that transforms timevar into a date and allows to set default columns

Usage

```
skMotionChart(df, idvar = colnames(df)[1], timevar = NULL, xvar = NULL,
  yvar = NULL, colorvar = idvar, sizevar = NULL, ...)
```

---

subst.expr *Substitute some pattern by a list of alternatives*

Description

Substitute some pattern by a list of alternatives

Usage

```
subst.expr(expr.str, subst, collapse = ",", add.list = FALSE)
```

Examples

```
## Not run:
  subst.expr("mean.ATTR=mean(ATTR)",subst=list(ATTR=c("hp","li")))

## End(Not run)
```

s_arrange *Modified version of dplyr's arrange that uses string arguments*

Description

Modified version of dplyr's arrange that uses string arguments

Usage

```
s_arrange(.data, ...)
```

s_filter *Modified version of dplyr's filter that uses string arguments*

Description

Modified version of dplyr's filter that uses string arguments

Usage

```
s_filter(.data, ...)
```

Examples

```
## Not run:
  # Examples
  library(dplyr)

  # Original usage of dplyr
  mtcars
    filter(gear == 3,cyl == 8)
    select(mpg, cyl, hp:vs)

  # Select user specified cols.
  # Note that you can have a vector of strings
  # or a single string separated by ',' or a mixture of both
  cols = c("mpg","cyl, hp:vs")
  mtcars
    filter(gear == 3,cyl == 8)
    s_select(cols)

  # Filter using a string
  col = "gear"
  mtcars
    s_filter(paste0(var,"==3"), "cyl==8" )
    select(mpg, cyl, hp:vs)
    s_arrange("mpg")
```

```
  # group_by and summarise with strings
  mtcars
    s_group_by("cyl")
    s_summarise("mean(disp), max(disp)")

## End(Not run)
```

---

s_group_by *Modified version of dplyr's group_by that uses string arguments*

Description

Modified version of dplyr's group_by that uses string arguments

Usage

```
s_group_by(.data, ...)
```

---

s_mutate *Modified version of dplyr's arrange that uses string arguments*

Description

Modified version of dplyr's arrange that uses string arguments

Usage

```
s_mutate(.data, ...)
```

---

s_select *Modified version of dplyr's select that uses string arguments*

Description

Modified version of dplyr's select that uses string arguments

Usage

```
s_select(.data, ...)
```

s_summarise — *Modified version of dplyr's summarise that uses string arguments*

Description

Modified version of dplyr's summarise that uses string arguments

Usage

```
s_summarise(.data, ...)
```

to.date.time — *Try to convert a character object to a date time object, trying out several formats*

Description

Try to convert a character object to a date time object, trying out several formats

Usage

```
to.date.time(x, orders = c(c(date.orders), outer(date.orders, time.orders,
  paste, sep = "")), quiet = FALSE, tz = "UTC",
  locale = Sys.getlocale("LC_TIME"), date.orders = c("dmy", "dmY", "ymd",
  "Ymd", "mdy", "mdY"), time.orders = c("r", "R", "T", "HMSOS"))
```

to.xts — *Transform a data from to an xts object*

Description

Transform a data from to an xts object

Usage

```
to.xts(dat, time.col, interval = NULL, time = dat[[time.col]],
  fill = !is.null(interval))
```

`transform.for.growth.analysis` *Transform data set for analysis in growth regressions*

Description

Generate growth rates, absolute gains, average values, first and last values in time windows of size ahead+1

Usage

```
## S3 method for class 'for.growth.analysis'
transform(dat, ahead = 1,
  trans.vars = names(which(sapply(dat, is.numeric))),
  fixed.vars = setdiff(colnames(dat), trans.vars),
  slide = ifelse(overlapping, 1, ahead), overlapping = TRUE)
```

Arguments

`dat`	a data frame increasingly sorted in time with no gaps
`ahead`	for how many periods ahead shall growth rates etc be computed
`trans.vars`	variables that shall be transformed . By default all numeric variables in dat
`fixed.vars`	variables whose first value in each time window that shall be added to the results without any transformation.
`overlapping`	(default=TRUE) shall time windows be overlapping or not
`slide`	the distance of window starts is by default 1 if overlapping == TRUE and ahead if overlapping==FALSE

`vcov.lm.fit` *Returns the variance-covariance matrix from the return values of a call to lm.fit*

Description

Is useful in so far that calling lm.fit is much quicker than calling lm, but the normal vcov function does not work when calling with the result of lm.fit

Usage

```
## S3 method for class 'lm.fit'
vcov(lm.fit, qr = lm.fit$qr, residuals = lm.fit$residuals)
```

Arguments

`lm.fit`	The return value of a call to lm.fit

Value

the variance covariance of the ols estimate

`view.stata.var` *Views variable labels of a data.frame loaded with read.dta*

Description

Views variable labels of a data.frame loaded with read.dta

Usage

```
view.stata.var(df, View = FALSE, print = !View)
```

`with.random.seed` *Evalutes the expression expr with the specified random seed. Afterwards the function restores the original random seed.*

Description

This function can be useful in simulation studies in which some random variables, e.g. explanatory variables X shall always be drawn in the same fashion, while other data, like disturbances epsilon shall differ between runs. One can then draw the X using with.random.seed to guarantee the same X in every simulation run.

Usage

```
## S3 method for class 'random.seed'
with(expr, seed = 1234567890)
```

Arguments

`expr`	an R expression that returns some random value
`seed`	an integer number that will be used to call set.seed

Value

the value of expr evaluated under the specifed random seed

wu.hausman.test *A Wu-Hausman Test for a single endogenous variable*

Description

See e.g. Green (2003) Econometric Analysis for a description of the Wu-Hausman-Test

Usage

```
wu.hausman.test(y, X.exo, X.endo, Z)
```

Arguments

y	the vector of dependent variables
X	the matrix of explanatory variables (endogenos & exogenous)
Z	the matrix of instruments (excluded & included instruments)
endo	vector or matrix of variables that might be endogenous

Examples

```
## Not run:
  library(sktools)
  # Ice cream model
  T = 100
  beta0 = 100; beta1  = -1; beta2 = 40

  s   = rep(0:1, length.out=T) # Season: winter summer fluctuating
  eps = rnorm(T,0,2)
  c   = runif(T,10,20)

  # Optimal price if the firm knows season s and u
  p = (beta0+beta2*s+eps - beta1*c) / (2*-beta1)

  # Alternatively: prices are a random markup above cost c
  #p = c*runif(T,1,1.1)

  # Compute demand
  q = beta0 + beta1*p+ beta2*s + eps



  # Matrix of instruments
  Z = cbind(1,c,s)

  # Run the Wu-Hausman test for testing endogeniety of p
  wu.hausman.test(y=q,X.exo=cbind(1,s),X.endo=p,Z=Z)

 ##############################################################################################
  # Run a systematic simulation study of
```

```
 # the Wu-Hausman Test
#############################################################################################

 # This function simulates and estimates a demand function and
 # returns the p-value of a Wu-Hausman test for endogeniety of the price
 sim.and.test = function() {
   # Ice cream model
   T = 100
   beta0 = 100; beta1  = -1; beta2 = 40

   s   = rep(0:1, length.out=T) # Season: winter summer fluctuating
   eps = rnorm(T,0,2)
   c   = runif(T,10,20)

   # Optimal price if the firm knows season s and u
   p = (beta0+beta2*s+eps - beta1*c) / (2*-beta1)

   # Alternatively: prices are a random markup above cost c
   p = c*runif(T,1,1.1)

   # Compute demand
   q = beta0 + beta1*p+ beta2*s + eps

   # Matrix of instruments
   Z = cbind(1,c,s)

   # Run the Wu-Hausman test for testing endogeniety of p
   wu.hausman.test(y=q,X.exo=cbind(1,s),X.endo=p,Z=Z)$p.value
 }

 # Perform a simulation study of our implementation of the Wu-Hausman test
 sim = simulation.study(sim.and.test,repl=1000)

 # I just need to adapt the results (won't be neccessary in a
 # newer version of sktools)
 sim = as.data.frame(sim)
 colnames(sim)[2] = "p.value"

 # Show the results and draw a histogram
 head(sim)
 hist(sim$p.value)
 # If the data is generated such that the Null hypothesis that
 # p is exogenous holds, then the p-values should be uniformly
 # distributed.

 # Use a Kolmogorov-Smirnov Test for whether p-values are uniformely distributed
 #ks.test(sim$p.value,"punif",min=0,max=1)



## End(Not run)
```

Index