caret Package

Cheat Sheet

Specifying the Model

Possible syntaxes for specifying the variables in the model:

```
train(y ~ x1 + x2, data = dat, ...)
train(x = predictor_df, y = outcome_vector, ...)
train(recipe_object, data = dat, ...)
```

- `rfe`, `sbf`, `gafs`, and `safs` only have the `x/y` interface.
- The `train` formula method will **always** create dummy variables.
- The `x/y` interface to `train` will not create dummy variables (but the underlying model function might).

Remember to:

- Have column names in your data.
- Use factors for a classification outcome (not 0/1 or integers).
- Have valid R names for class levels (not "0"/"1")
- Set the random number seed prior to calling `train` repeatedly to get the same resamples across calls.
- Use the `train` option `na.action = na.pass` if you will being imputing missing data. Also, use this option when predicting new data containing missing values.

To pass options to the underlying model function, you can pass them to `train` via *the ellipses*:

```
train(y ~ ., data = dat, method = "rf",
      # options to `randomForest`:
      importance = TRUE)
```

Parallel Processing

The `foreach` package is used to run models in parallel. The `train` code does not change but a "`do`" package must be called first.

```
# on MacOS or Linux
library(doMC)
registerDoMC(cores=4)
```

```
# on Windows
library(doParallel)
cl <- makeCluster(2)
registerDoParallel(cl)
```

The function `parallel::detectCores` can help too.

Preprocessing

Transformations, filters, and other operations can be applied to the *predictors* with the `preProc` option.

```
train(, preProc = c("method1", "method2"), ...)
```

Methods include:

- `"center"`, `"scale"`, and `"range"` to normalize predictors.
- `"BoxCox"`, `"YeoJohnson"`, or `"expoTrans"` to transform predictors.
- `"knnImpute"`, `"bagImpute"`, or `"medianImpute"` to impute.
- `"corr"`, `"nzv"`, `"zv"`, and `"conditionalX"` to filter.
- `"pca"`, `"ica"`, or `"spatialSign"` to transform groups.

`train` determines the order of operations; the order that the methods are declared does not matter.

The `recipes` package has a more extensive list of preprocessing operations.

Adding Options

Many `train` options can be specified using the `trainControl` function:

```
train(y ~ ., data = dat, method = "cubist",
      trControl = trainControl(<options>))
```

Resampling Options

`trainControl` is used to choose a resampling method:

```
trainControl(method = <method>, <options>)
```

Methods and options are:

- `"cv"` for K-fold cross-validation (`number` sets the # folds).
- `"repeatedcv"` for repeated cross-validation (`repeats` for # repeats).
- `"boot"` for bootstrap (`number` sets the iterations).
- `"LGOCV"` for leave-group-out (`number` and `p` are options).
- `"LOO"` for leave-one-out cross-validation.
- `"oob"` for out-of-bag resampling (only for some models).
- `"timeslice"` for time-series data (options are `initialWindow`, `horizon`, `fixedWindow`, and `skip`).

Performance Metrics

To choose how to summarize a model, the `trainControl` function is used again.

```
trainControl(summaryFunction = <R function>,
             classProbs = <logical>)
```

Custom R functions can be used but `caret` includes several: `defaultSummary` (for accuracy, RMSE, etc), `twoClassSummary` (for ROC curves), and `prSummary` (for information retrieval). For the last two functions, the option `classProbs` must be set to `TRUE`.

Grid Search

To let `train` determine the values of the tuning parameter(s), the `tuneLength` option controls how many values **per tuning** parameter to evaluate.

Alternatively, specific values of the tuning parameters can be declared using the `tuneGrid` argument:

```
grid <- expand.grid(alpha = c(0.1, 0.5, 0.9),
                    lambda = c(0.001, 0.01))
train(x = x, y = y, method = "glmnet",
      preProc = c("center", "scale"),
      tuneGrid = grid)
```

Random Search

For tuning, `train` can also generate random tuning parameter combinations over a wide range. `tuneLength` controls the total number of combinations to evaluate. To use random search:

```
trainControl(search = "random")
```

Subsampling

With a large class imbalance, `train` can subsample the data to balance the classes them prior to model fitting.

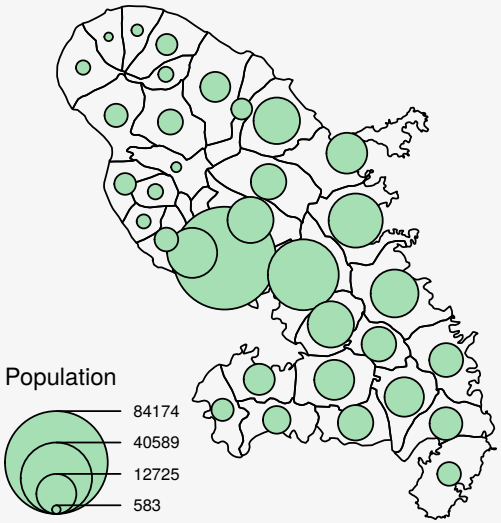
```
trainControl(sampling = "down")
```

Other values are `"up"`, `"smote"`, or `"rose"`. The latter two may require additional package installs.

Thematic maps with cartography : : **CHEAT SHEET**

Use cartography with spatial objects from sf or sp packages to create thematic maps.

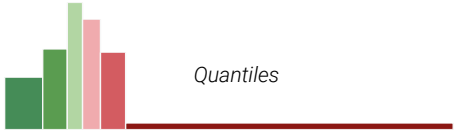
```
library(cartography)
library(sf)
mtq <- st_read("martinique.shp")
plot(st_geometry(mtq))
propSymbolsLayer(x = mtq, var = "P13_POP",
 legend.title.txt = "Population",
 col = "#a7dfb4")
```



Classification

Available methods are: quantile, equal, q6, fisher-jenks, mean-sd, sd, geometric progression...

```
bks1 <- getBreaks(v = var, nclass = 6,
 method = "quantile")
bks2 <- getBreaks(v = var, nclass = 6,
 method = "fisher-jenks")
pal <- carto.pal("green.pal",3, "wine.pal", 3)
hist(var, breaks = bks1, col = pal)
```




```
hist(var, breaks = bks2, col = pal)
```

Symbology

In most functions the x argument should be an sf object. sp objects are handled through spdf and df arguments.

Choropleth
```
choroLayer(x = mtq, var = "myvar",
 method = "quantile", nclass = 8)
```

Typology
```
typoLayer(x = mtq, var = "myvar")
```

Proportional Symbols
```
propSymbolsLayer(x = mtq, var = "myvar",
 inches = 0.1, symbols = "circle")
```

Colorized Proportional Symbols (relative data)
```
propSymbolsChoroLayer(x = mtq, var = "myvar",
 var2 = "myvar2")
```

Colorized Proportional Symbols (qualitative data)
```
propSymbolsTypoLayer(x = mtq, var = "myvar",
 var2 = "myvar2")
```

Double Proportional Symbols
```
propTrianglesLayer(x = mtq, var1 = "myvar",
 var2 = "myvar2")
```

OpenStreetMap Basemap (see rosm package)
```
tiles <- getTiles(x = mtq, type = "osm")
tilesLayer(tiles)
```

Isopleth (see SpatialPosition package)
```
smoothLayer(x = mtq, var = "myvar",
 typefct = "exponential", span = 500,
 beta = 2)
```

Discontinuities
```
discLayer(x = mtq.borders, df = mtq,
 var = "myvar", threshold = 0.5
```

Flows
```
propLinkLayer(x = mtq_link, df = mtq_df,
 var = "fij")
```

Dot Density
```
dotDensityLayer(x = mtq, var = "myvar")
```

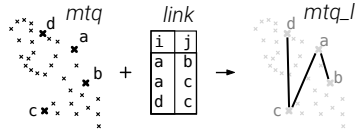
Labels
```
labelLayer(x = mtq, txt = "myvar",
 halo = TRUE, overlap = FALSE)
```

Transformations

Polygons to Grid
```
mtq_grid <- getGridLayer(x = mtq, cellsize = 3.6e+07,
  type = "hexagonal", var = "myvar")
```

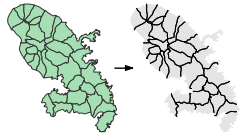
Grids layers can be used by choroLayer() or propSymbolsLayer().

Points to Links
```
mtq_link <- getLinkLayer(x = mtq, df = link)
```



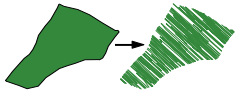
Links layers can be used by *LinkLayer().

Polygons to Borders
```
mtq_border <- getBorders(x = mtq)
```



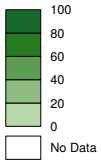
Borders layers can be used by discLayer() function

Polygons to Pencil Lines
```
mtq_pen <- getPencilLayer(x = mtq)
```



Legends

legendChoro()

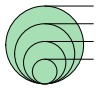



```
legendChoro(pos = "topleft",
 title.txt = "legendChoro()",
 breaks = c(0,20,40,60,80,100),
 col = carto.pal("green.pal", 5),
 nodata = TRUE, nodata.txt = "No Data")
```

legendTypo()


```
legendTypo(title.txt = "legendTypo()",
 col = c("peru", "skyblue", "gray77"),
 categ = c("type 1", "type 2", "type 3"),
 nodata = FALSE)
```

legendCirclesSymbols()




```
legendCirclesSymbols(var = c(10,100),
 title.txt = "legendCirclesSymbols()",
 col = "#a7dfb4ff", inches = 0.3)
```

See also legendSquaresSymbols(), legendBarsSymbols(), legendGradLines(), legendPropLines() and legendPropTriangles().

Map Layout

North Arrow:
```
north(pos = "topright")
```

Scale Bar:
```
barscale(size = 5)
```

Full Layout:
```
layoutLayer(
 title = "Martinique",
 tabtitle = TRUE,
 frame = TRUE,
 author = "Author",
 sources = "Sources",
 north = TRUE,
 scale = 5)
```

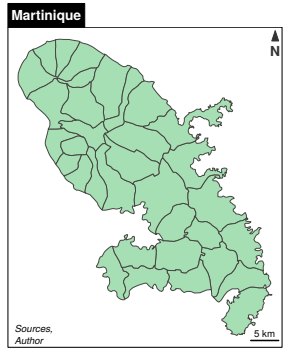
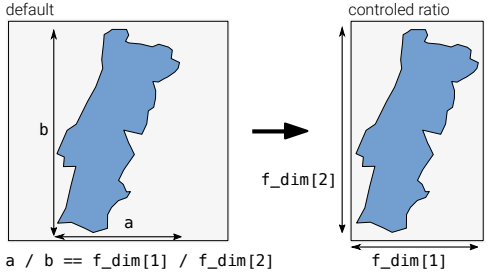


Figure Dimensions

Get figure dimensions based on the dimension ratio of a spatial object, figure margins and output resolution.

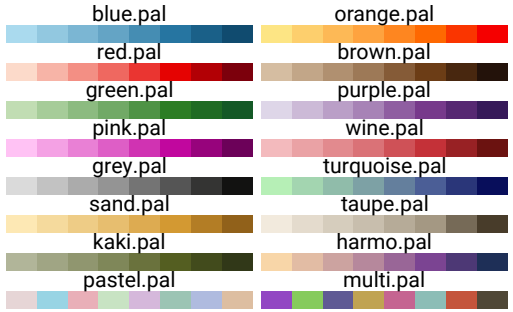
```
f_dim <- getFigDim(x = sf_obj, width = 500,
  mar = c(0,0,0,0))
png("fig.png", width = 500, height = f_dim[2])
par(mar = c(0,0,0,0))
plot(sf_obj, col = "#729fcf")
dev.off()
```



Color Palettes

```
carto.pal(pal1 = "blue.pal", n1 = 5,
 pal2 = sand.pal, n2 = 3)
```

```
display.carto.all(n = 8)
```



DeclareDesign: : CHEAT SHEET

Model

What is your model of the world, including how outcomes respond to interventions in the world?

Population

Define the size of the population, hierarchical structure (if any), and background variables.

Simple dataset with no background variables

```
pop <- declare_population(N = 100)
pop()
```

Simple dataset with background variables

```
declare_population(N = 100,
                   X = rnorm(N))
```

Two-level dataset

```
declare_population(
  schools =
    add_level(N = 10,
              funding = rnorm(N)),
  students =
    add_level(N = 100,
              scores = rnorm(N))
)
```

Outcomes

Outcomes that depend on a treatment (Z)

Using a formula

```
declare_potential_outcomes(
  Y ~ .5 * Z + rnorm(N))
```

As separate variables

```
declare_potential_outcomes(
  Y_Z_0 = rnorm(N),
  Y_Z_1 = Y_Z_0 + .5)
```

Outcomes that do not depend on treatment

```
declare_potential_outcomes(
  Y = rnorm(N))
```

Inquiry

What is the research question you want to answer?

Causal inquiries

```
declare_estimand(
  ATE = mean(Y_Z_1 - Y_Z_0))
```

Descriptive inquiries

```
declare_estimand(
  Y_median = median(Y))
```

Conditional estimands

```
declare_estimand(
  LATE = mean(Y_Z_1 - Y_Z_0),
  subset = complier == TRUE)
```

Data Strategy

How will you generate data to answer your inquiry?

Sampling

```
declare_sampling(n = 100)

declare_sampling(
  strata_n = 20,
  strata = urban_area)
```

Treatment assignment

```
declare_assignment(m = 100)

declare_assignment(
  clusters = villages,
  m = 10)
```

Answer Strategy

How will you generate an answer to your inquiry?

OLS with robust standard errors

```
declare_estimator(
  Y ~ Z, model = lm_robust)
```

2SLS instrumental variables regression with robust SEs

```
declare_estimator(
  Y ~ D | Z, model = iv_robust)
```

Difference-in-means

```
declare_estimator(
  Y ~ Z,
  model = difference_in_means)
```

DeclareDesign is a software implementation of the MIDA framework, according to which research designs have a **M**odel of the world, an **I**nquiry about that model, a **D**ata strategy that generates information about the world, and an **A**nswer strategy that uses data to make a guess about the **I**nquiry. Declared designs can be "diagnosed" to calculate the properties of the design such as power and bias using Monte Carlo simulation.

All `declare_*` functions return *functions*. Most functions take a data.frame and return a data.frame.

Design Declaration

Put together all the steps into a declared design using the + operator

```
design <-
  declare_population(N = 200, X = rnorm(N)) +
  declare_potential_outcomes(Y ~ .5 * Z + X) +
  declare_estimand(ATE = mean(Y_Z_1 - Y_Z_0)) +
  declare_sampling(n = 100) +
  declare_assignment(m = 50) +
  declare_estimator(Y ~ Z, model = lm_robust)
```

```
draw_data(design)
draw_estimates(design)
get_estimates(design, data = real_data)
draw_estimands(design)
run_design(design)
summary(design)
compare_designs(design_1, design_2)
```

Design Diagnosis

Diagnose the properties of your design

```
diagnosis <- diagnose_design(
  design, sims = 100, bootstrap_sims = 100)
```

```
summary(diagnosis)
get_diagnosands(diagnosis)
get_simulations(diagnosis)
```

Custom diagnosands

```
diagnose_design(
  design,
  diagnosands = declare_diagnosands(
    sig_pos = mean(p.value < .05 & estimate > 0)))
```

estimatr : : **CHEAT SHEET**

OLS with lm_robust()

`lm_robust()` is `lm()` with robust SEs. HC2 is the default.

```
lm_robust(mpg ~ hp, data = mtcars)
lm_robust(mpg ~ hp, se_type = "HC1",
          data = mtcars)
lm_robust(mpg ~ hp, se_type = "classical",
          data = mtcars)
```

Indicate `clusters` to get clustered SEs. CR2 is the default.

```
lm_robust(mpg ~ hp, clusters = carb,
          data = mtcars)
lm_robust(mpg ~ hp, clusters = carb,
          se_type = "stata", data = mtcars)
```

Fixed effects two ways:

```
# FEs as "dummies"
lm_robust(mpg ~ hp + as.factor(am),
          data = mtcars)

# "Absorbing" FEs (substantially faster)
lm_robust(mpg ~ hp,
          fixed_effects = ~ am,
          data = mtcars)
```

post-estimation commands:

```
fit <- lm_robust(mpg ~ hp, data = mtcars)
summary(fit)
print(fit)
tidy(fit)
vcov(fit)
confint(fit)
nobs(fit)
predict(fit, newdata = mtcars)
```

2SLS with iv_robust()

`iv_robust()` is `AER::ivreg()` with robust SEs.

```
iv_robust(mpg ~ hp | am, data = mtcars)
iv_robust(mpg ~ hp | am,
          clusters = carb, data = mtcars)
```

Two-group estimators

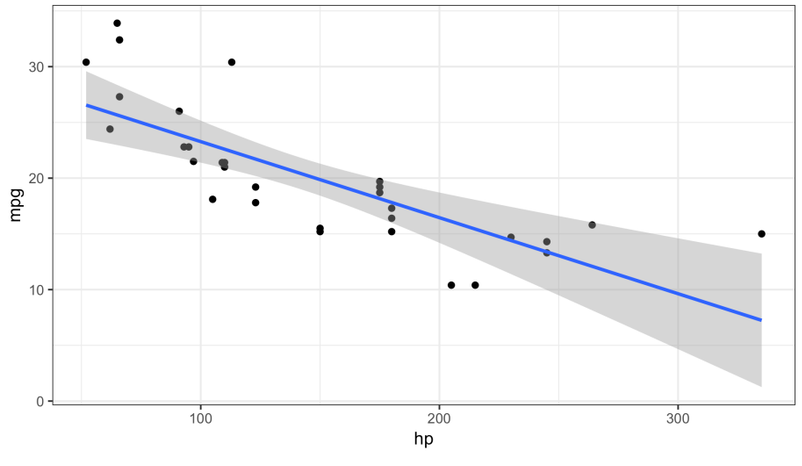
`difference_in_means()` and `horvitz_thompson()` compare two groups

```
difference_in_means(mpg ~ am, data = mtcars)
horvitz_thompson(mpg ~ am, data = mtcars)
```

ggplot2 integration

Use robust variance estimates for drawing confidence intervals:

```
library(ggplot2)
ggplot(mtcars, aes(mpg, hp)) +
  geom_point() +
  stat_smooth(method = "lm_robust") +
  theme_bw()
```



Multiple models

Same outcome, different subsets:

```
library(tidyverse)
mtcars %>%
  split(.$cyl) %>%
  map(~lm_robust(mpg ~ hp, data = .)) %>%
  map(tidy) %>%
  bind_rows(.id = "cyl")
```

Different outcomes, same subset:

```
c("mpg", "disp") %>%
  map(~formula(paste0(., " ~ hp"))) %>%
  map(~lm_robust(., data = mtcars)) %>%
  map(tidy) %>%
  bind_rows
```

Extras

```
# Lin (2013) covariate adjustment
lm_lin(mpg ~ am, covariates = ~ hp,
       data = mtcars)

# regression tables with texreg
fit <- lm_robust(mpg ~ hp, data = mtcars)
texreg::texreg(fit, include.ci = FALSE)
```

estimatr-to-Stata dictionary

estimatr	Stata
`lm_robust(y ~ z, data = dat)`	`reg y z, vce(hc2)`
`lm_robust(y ~ z, clusters = cl, se_type = "stata", data = dat)`	`reg y z, vce(cluster cl)`
`lm_robust(mpg ~ hp, fixed_effects = ~ am, se_type = "stata", data = mtcars)`	`areg mpg hp, absorb(am) vce(robust)`
`iv_robust(mpg ~ hp | am, se_type = "HC1", data = mtcars)`	`ivregress 2sls mpg (hp = am), vce(robust) small`

estimatr is part of the DeclareDesign suite of packages for designing, implementing, and analyzing social science research designs.

The *eurostat* package

R tools to access open data from Eurostat database

Search and download

Data in the Eurostat database is stored in tables. Each table has an identifier, a short table_code, and a description (e.g. *tsdtr420* - People killed in road accidents).
Key eurostat functions allow to find the table_code, download the eurostat table and polish labels in the table.

Find the table code

The **search_eurostat(pattern, ...)** function scans the directory of Eurostat tables and returns codes and descriptions of tables that match pattern.

```
library("eurostat")
query <- search_eurostat("road", type = "table")
query[1:3,1:2]
##                                 title     code
## 1           Goods transport by road ttr00005
## 2   People killed in road accidents tsdtr420
## 3 Enterprises with broadband access tin00090
```

Download the table

The **get_eurostat(id, time_format = "date", filters = "none", type = "code", cache = TRUE, ...)** function downloads the requested table from the *Eurostat bulk download facility* or from *The Eurostat Web Services JSON API* (if **filters** are defined). Downloaded data is cached (if **cache=TRUE**). Additional arguments define how to read the time column (**time_format**) and if table dimensions shall be kept as codes or converted to labels (**type**).

```
dat <- get_eurostat(id="tsdtr420", time_format="num")
head(dat)
##     unit    sex    geo  time values
## 1     NR      T     AT  1999   1079
## 2     NR      T     BE  1999   1397
## 3     NR      T     CZ  1999   1455
## 4     NR      T     DK  1999    514
## 5     NR      T     EL  1999   2116
## 6     NR      T     ES  1999   5738
```

Add labels

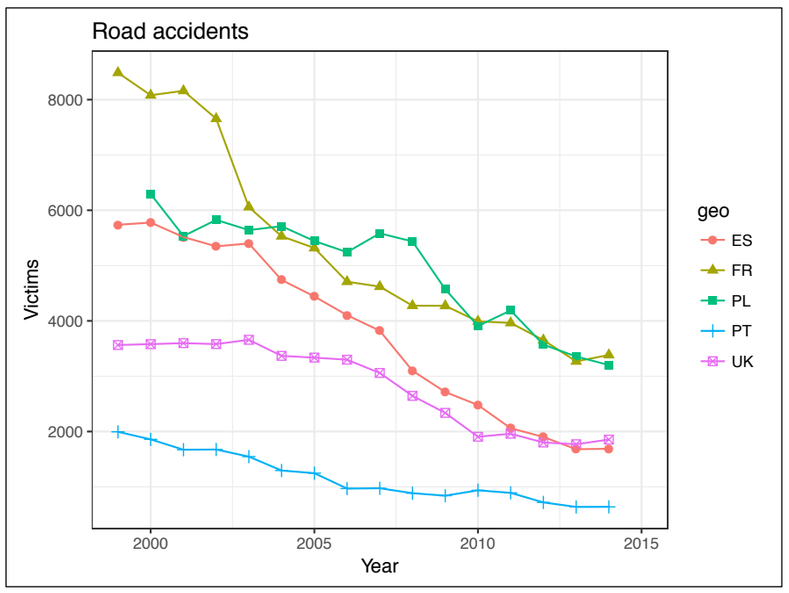
The **label_eurostat(x, lang = "en", ...)** gets definitions for Eurostat codes and replace them with labels in given language ("en", "fr" or "de").

```
dat <- label_eurostat(dat)
head(dat)
##     unit    sex            geo  time values
## 1 Number  Total        Austria  1999   1079
## 2 Number  Total        Belgium  1999   1397
## 3 Number  Total Czech Republic  1999   1455
## 4 Number  Total        Denmark  1999    514
## 5 Number  Total         Greece  1999   2116
## 6 Number  Total          Spain  1999   5738
```

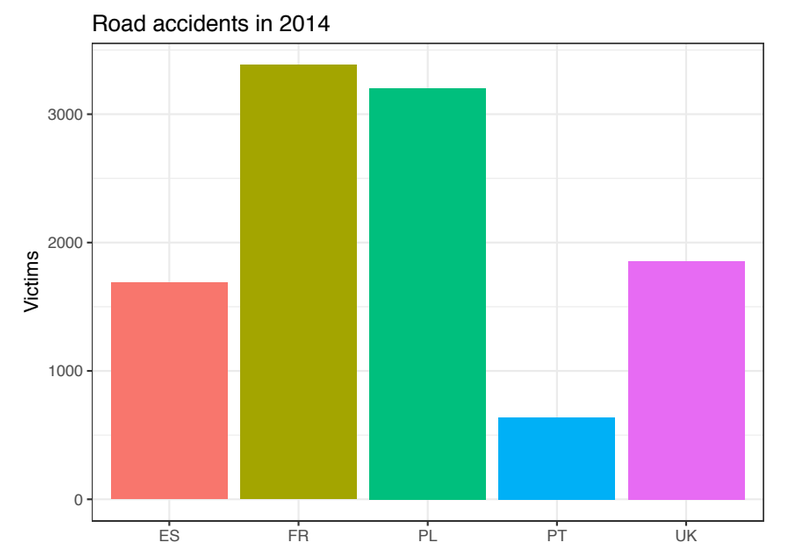
eurostat and plots

The **get_eurostat()** function returns tibbles in the long format. Packages **dplyr** and **tidyr** are well suited to transform these objects. The **ggplot2** package is well suited to plot these objects.

```
t1 <- get_eurostat("tsdtr420", filters =
            list(geo = c("UK", "FR", "PL", "ES", "PT")))

library("ggplot2")
ggplot(t1, aes(x = time, y = values, color = geo,
               group = geo, shape = geo)) +
  geom_point(size = 2) +
  geom_line() + theme_bw() +
  labs(title="Road accidents", x = "Year", y = "Victims")
```




```
library("dplyr")
t2 <- t1 %>% filter(time == "2014-01-01")
ggplot(t2, aes(geo, values, fill=geo)) +
  geom_bar(stat = "identity") + theme_bw() +
  theme(legend.position = "none")+
  labs(title="Road accidents in 2014", x="", y="Victims")
```



eurostat and maps

Fetch and process data

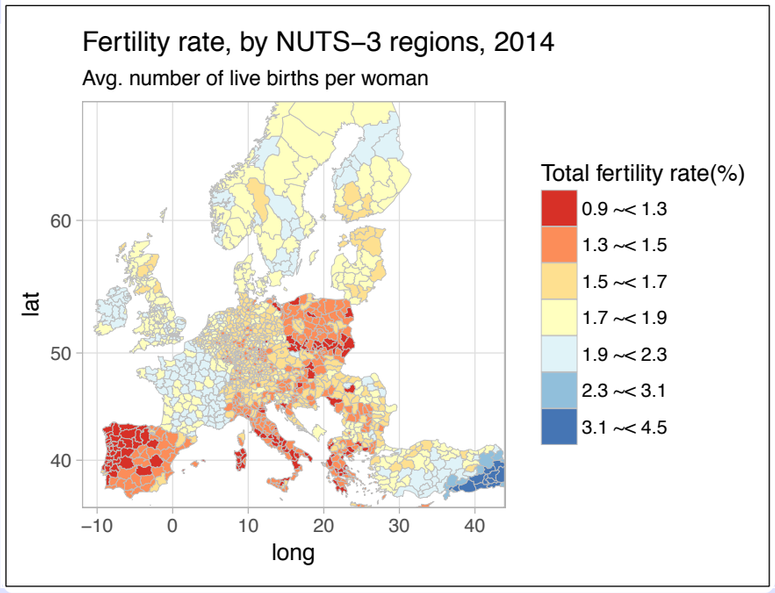
There are three function to work with geospatial data from GISCO. The **get_eurostat_geospatial()** returns preprocessed spatial data as sp-objects or as data frames. The **merge_eurostat_geospatial()** both downloads and merges the geospatial data with a preloaded tabular data. The **cut_to_classes()** is a wrapper for **cut()** - function and is used for categorizing data for maps with tidy labels.

```
library("eurostat")
library("dplyr")

fertility <- get_eurostat("demo_r_frate3")  %>%
  filter(time == "2014-01-01") %>%
  mutate(cat = cut_to_classes(values, n=7, decimals=1))

mapdata <-  merge_eurostat_geodata(fertility,
              resolution = "20")

head(select(mapdata,geo,values,cat,long,lat,order,id))
##     geo values        cat     long      lat order id
## 1 AT124   1.39 1.3 ~< 1.5 15.54245 48.90770   214 10
## 2 AT124   1.39 1.3 ~< 1.5 15.75363 48.85218   215 10
## 3 AT124   1.39 1.3 ~< 1.5 15.88763 48.78511   216 10
## 4 AT124   1.39 1.3 ~< 1.5 15.81535 48.69270   217 10
## 5 AT124   1.39 1.3 ~< 1.5 15.94094 48.67173   218 10
## 6 AT124   1.39 1.3 ~< 1.5 15.90833 48.59815   219 10
```

Draw a cartogram

The object returned by **merge_eurostat_geospatial()** are ready to be plotted with ggplot2 package. The **coord_map()** function is useful to set the projection while **labs()** adds annotations o the plot.

```
library("ggplot2")
ggplot(mapdata, aes(x = long, y = lat, group = group))+
  geom_polygon(aes(fill=cat), color="grey", size = .1)+
  scale_fill_brewer(palette = "RdYlBu") +
  labs(title="Fertility rate, by NUTS-3 regions, 2014",
       subtitle="Avg. number of live births per woman",
       fill="Total fertility rate(%)") + theme_light()+
  coord_map(xlim=c(-12,44), ylim=c(35,67))
```



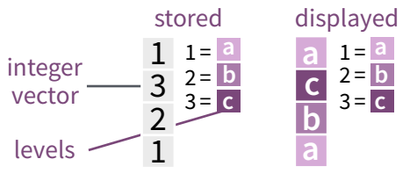
This onepager presents the **eurostat** package
Leo Lahti, Janne Huovari, Markus Kainu, Przemyslaw Biecek 2014-2017 package version 2.2.43 URL: *https://github.com/rOpenGov/eurostat*

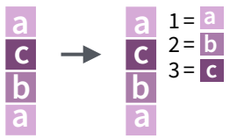
Factors with forcats : : **CHEAT SHEET**

The **forcats** package provides tools for working with factors, which are R's data structure for categorical data.

Factors

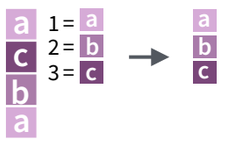
R represents categorical data with factors. A **factor** is an integer vector with a **levels** attribute that stores a set of mappings between integers and categorical values. When you view a factor, R displays not the integers, but the values associated with them.





Create a factor with factor()

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA) Convert a vector to a factor. Also **as_factor**.
f <- factor(c("a", "c", "b", "a"), levels = c("a", "b", "c"))

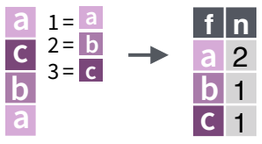


Return its levels with levels()

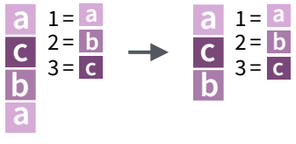
levels(x) Return/set the levels of a factor. *levels(f); levels(f) <- c("x","y","z")*

Use unclass() to see its structure

Inspect Factors

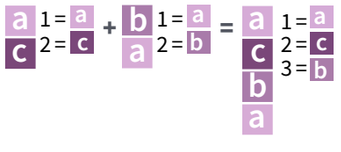


fct_count(f, sort = FALSE) Count the number of values with each level. *fct_count(f)*

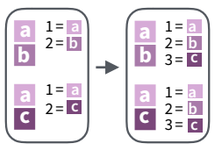


fct_unique(f) Return the unique values, removing duplicates. *fct_unique(f)*

Combine Factors

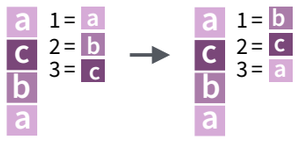


fct_c(…) Combine factors with different levels.
f1 <- factor(c("a", "c"))
f2 <- factor(c("b", "a"))
fct_c(f1, f2)

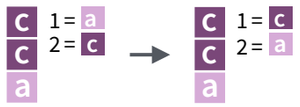


fct_unify(fs, levels = lvls_union(fs)) Standardize levels across a list of factors.
fct_unify(list(f2, f1))

Change the order of levels

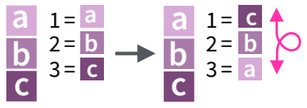


fct_relevel(.f, ..., after = 0L) Manually reorder factor levels.
fct_relevel(f, c("b", "c", "a"))

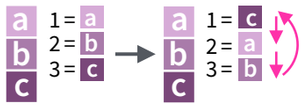


fct_infreq(f, ordered = NA) Reorder levels by the frequency in which they appear in the data (highest frequency first).
f3 <- factor(c("c", "c", "a"))
fct_infreq(f3)

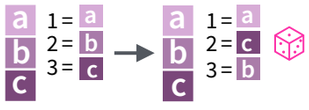
fct_inorder(f, ordered = NA) Reorder levels by order in which they appear in the data.
fct_inorder(f2)



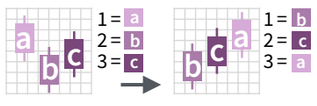
fct_rev(f) Reverse level order.
f4 <- factor(c("a","b","c"))
fct_rev(f4)



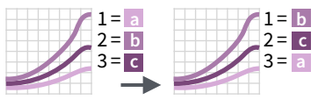
fct_shift(f) Shift levels to left or right, wrapping around end.
fct_shift(f4)



fct_shuffle(f, n = 1L) Randomly permute order of factor levels.
fct_shuffle(f4)

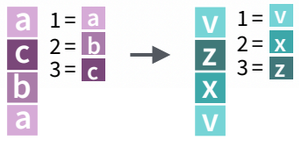


fct_reorder(.f, .x, .fun=median, ..., .desc = FALSE) Reorder levels by their relationship with another variable.
boxplot(data = iris, Sepal.Width ~ fct_reorder(Species, Sepal.Width))

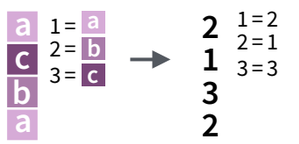


fct_reorder2(.f, .x, .y, .fun = last2, ..., .desc = TRUE) Reorder levels by their final values when plotted with two other variables.
ggplot(data = iris, aes(Sepal.Width, Sepal.Length, color = fct_reorder2(Species, Sepal.Width, Sepal.Length))) + geom_smooth()

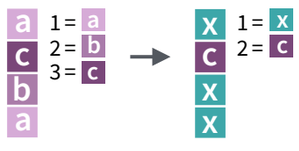
Change the value of levels



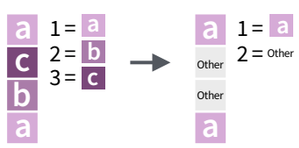
fct_recode(.f, ...) Manually change levels. Also **fct_relabel** which obeys purrr::map syntax to apply a function or expression to each level.
fct_recode(f, v = "a", x = "b", z = "c")
fct_relabel(f, ~ paste0("x", .x))



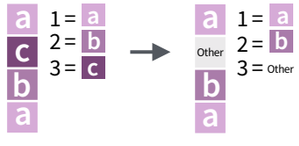
fct_anon(f, prefix = "")) Anonymize levels with random integers. *fct_anon(f)*



fct_collapse(.f, ...) Collapse levels into manually defined groups.
fct_collapse(f, x = c("a", "b"))

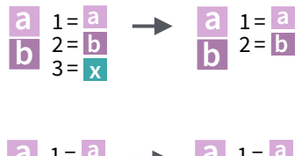


fct_lump(f, n, prop, w = NULL, other_level = "Other", ties.method = c("min", "average", "first", "last", "random", "max")) Lump together least/most common levels into a single level. Also **fct_lump_min**.
fct_lump(f, n = 1)

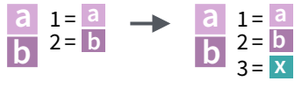


fct_other(f, keep, drop, other_level = "Other") Replace levels with "other."
fct_other(f, keep = c("a", "b"))

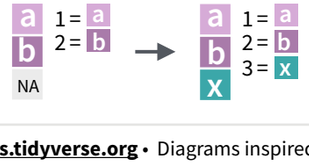
Add or drop levels



fct_drop(f, only) Drop unused levels.
f5 <- factor(c("a","b"),c("a","b","x"))
f6 <- fct_drop(f5)



fct_expand(f, …) Add levels to a factor. *fct_expand(f6, "x")*



fct_explicit_na(f, na_level="(Missing)") Assigns a level to NAs to ensure they appear in plots, etc.
fct_explicit_na(factor(c("a", "b", NA)))

GWAS Catalog access with gwasrapidd

Introduction

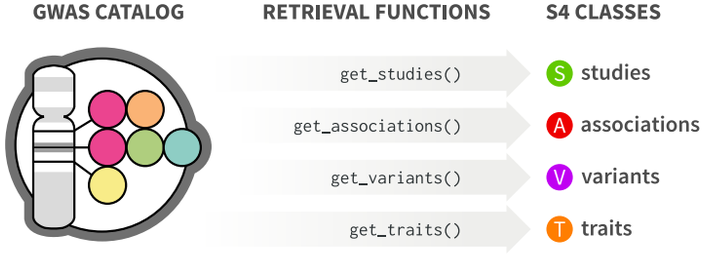
The **GWAS Catalog** is a service provided by the EMBL-EBI and NHGRI that offers a manually curated and freely available database of published genome-wide association studies (GWAS).

The GWAS Catalog data provided by the **RESTful API** is organized around four core entities:

- **studies**
- **associations**
- **variants**
- **traits**

Get GWAS Catalog Entities

gwasrapidd facilitates the access to the Catalog via the RESTful API, allowing you to programmatically retrieve data directly into R. Each of the four entities is mapped to an S4 object of a class of the same name.



Search by	Example	S	A	V	T
study_id	"GCST000858"	●	●	●	●
association_id	"24300113"	●	●	●	●
variant_id	"rs12752552"	●	●	●	
efo_id	"EFO_0005543"	●	●	●	●
pubmed_id	"21626137"	●	●	●	●
user_requested	TRUE	●			
full_pvalue_set	FALSE	●			
efo_uri	"http://www.ebi.ac.uk/efo/EFO_0004761"	●			●
genomic_range	list(chromosome = "22", start = 1L, end = 15473564L)			●	
gene_name	"BRCA1"			●	
efo_trait	"lung adenocarcinoma"	●	●	●	●
reported_trait	"Breast cancer"	●		●	
cytogenetic_band	"1p36.33"			●	

S4 Representation of GWAS Catalog Entities

S4 class studies

The **studies** object consists of eight slots, each a table (tibble). Each study is an observation (row) in the studies table — main table. All tables have the column study_id as primary key.

For details about the studies S4 class: `class?studies`.

studies
- study_id
- reported_trait
- initial_sample_size
- replication_sample_size
- gxe
- gxg
- snp_count
- qualifier
- imputed
- pooled
- study_design_comment
- full_pvalue_set
- user_requested

genotyping_techs
- study_id
- genotyping technology

platforms
- study_id
- manufacturer

ancestries
- study_id
- ancestry_id
- type
- number_of_individuals

ancestral_groups
- study_id
- ancestry_id
- ancestral_group

countries_of_recruitment
- study_id
- ancestry_id
- country_name
- major_area
- region

countries_of_origin
- study_id
- ancestry_id
- country_name
- major_area
- region

publications
- study_id
- pubmed_id
- publication_date
- publication
- title
- author_fullname
- author_orcid

S

S4 class associations

The **associations** object consists of six slots, each a table (tibble). Each association is an observation (row) in the associations table — main table. All tables have the column association_id as primary key.

For details about the associations S4 class: `class?associations`.

associations
- association_id
- pvalue
- pvalue_description
- pvalue_mantissa
- pvalue_exponent
- multiple_snp_haplotype
- snp_interaction
- snp_type
- standard_error
- range
- or_per_copy_number
- beta_number
- beta_unit
- beta_direction
- beta_description
- last_mapping_date
- last_update_date

loci
- association_id
- locus_id
- haplotype_snp_count
- description

risk_alleles
- association_id
- locus_id
- variant_id
- risk_allele
- risk_frequency
- genome_wide
- limited_list

genes
- association_id
- locus_id
- gene_name

ensembl_ids
- association_id
- locus_id
- gene_name
- ensembl_id

entrez_ids
- association_id
- locus_id
- gene_name
- entrez_id

A

S4 class variants

The **variants** object consists of four slots, each a table (tibble). Each variant is an observation (row) in the variants table — main table. All tables have the column variant_id as primary key.

For details about the variants S4 class: `class?variants`.

variants
- variant_id
- merged
- functional_class
- chromosome_name
- chromosome_position
- chromosome_region
- last_update_date

genomic_contexts
- variant_id
- gene_name
- chromosome_name
- chromosome_position
- distance
- is_closest_gene
- is_intergenic
- is_upstream
- is_downstream
- source
- mapping_method

ensembl_ids
- variant_id
- gene_name
- ensembl_id

entrez_ids
- variant_id
- gene_name
- entrez_id

V

S4 class traits

The **traits** object consists of one slot only, a table (tibble) of GWAS Catalog EFO traits. Each EFO trait is an observation (row) in the traits table — main table.

For details about the traits S4 class: `class?traits`.

traits
- efo_id
- trait
- uri

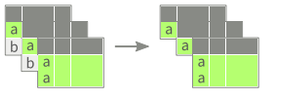
T

Manipulate Cases

Get a **studies** object **s** of two GWAS studies:

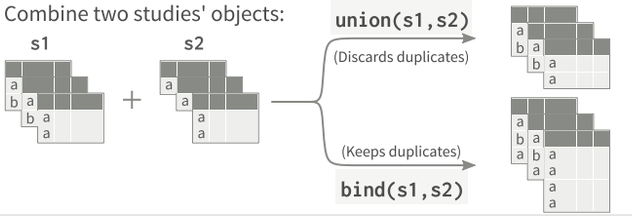
```
s <- get_studies(study_id = c('a', 'b'))
```

Subset object **s** by either identifier or position using `` `[` ``:



```
s['a'] # Subset by identifier
s[1] # Subset by position
```

Combine two studies' objects:



Leaflet Cheat Sheet

Leaflet for R

an open-source JavaScript library for mobile-friendly interactive maps

Quick Start

Installation

Use **install.packages("leaflet")** to installize the package or directly from Github *devtools::install_github("rstudio/leaflet")*.

First Map

```
m <- leaflet() %>%        # leaflet works with the pipe operator
  addTiles() %>%          # setup the default OpenStreetMap map tiles
  addMarkers(lng = 174.768, lat = -36.852, popup = "The birthplace of R")
                          # add a single point layer
m
```

Map Widget

Initialization

m <- leaflet(options = leafletOptions(...))

center	Initial geographic center of the map
zoom	Initial map zoom level
minZoom	Minimum zoom level of the map
maxZoom	Maximum zoom level of the map

Map Methods

m %>% setView(lng, lat, zoom, options = list()**)**
Set the view of the map (center and zoom level)

m %>% fitBounds(lng1, lat1, lng2, lat2**)**
Fit the view into the rectangle [lng1, lat1] - [lng2, lat2]

m %>% clearBounds()
Clear the bound, automatically determine from the map elements

Data Object

Both **leaflet()** and the **map** layers have an optional data parameter that is designed to receive spatial data with the following formats:

Base R — *The arguments of all layers take normal R objects:*
```
df <- data.frame(lat = ..., lng = ... )
leaflet(df) %>% addTiles() %>% addCircles()
```

sp *package* — library(sp) *Useful functions:*
SpatialPoints, SpatialLines, SpatialPolygons, ...

maps *package* — library(maps) *Build a map of states with colors:*
```
mapStates <- map("state", fill = TRUE, plot = FALSE)
leaflet(mapStates) %>% addTiles() %>%
  addPolygons(fillColor = topo.colors(10, alpha = NULL), stroke = FALSE)
```

Markers

Use markers to call out points, express locations with latitude/longitude coordinates, appear as icons or as circles.
Data come from vectors or assigned data frame, or **sp** package objects.

Icon Markers

Regular Icons: default and simple
addMarkers(lng, lat, popup, label**)** *add basic icon markers*
makeIcon/icons (iconUrl, iconWidth, iconHeight, iconAnchorX, iconAnchorY, shadowUrl, shadowWidth, shadowHeight, ... **)** *customize marker icons*
iconList() *create a list of icons*
Awesome Icons: customizable with colors and icons
addAwesomeMarkers, makeAwesomeIcon, awesomeIcons, awesomeIconList
Marker Clusters: option of addMarters()
clusterOptions = markerClusterOptions()
freezeAtZoom Freeze the cluster at assigned zoom level

Circle Markers

addCircleMarkers(color, radius, stroke, opacity, ... **)**
Customize their color, radius, stroke, opacity

Popups and Labels

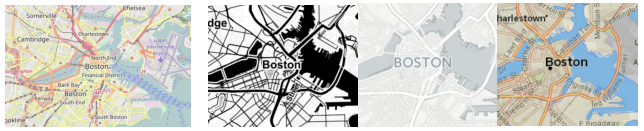
addPopups(lng, lat, ...content..., options) *Add standalone popups*
options = popupOptions(closeButton=FALSE)
addMarkers(..., popup, ...) *Show popups with markers or shapes*
addMarkers(..., label, labelOptions...) *Show labels with markers or shapes*
labelOptions = labelOptions(noHide, textOnly, textsize, direction, style)
addLabelOnlyMarkers() *Add labels without markers*

Lines and Shapes

Polygons and Polylines
addPolygons(color, weight=1, smoothFactor=0.5, opacity=1.0, fillOpacity=0.5, fillColor= ~colorQuantile("YlOrRd", ALAND)(ALAND), highlightOptions, ... **)**
highlightOptions(color, weight=2, bringToFront=TRUE**)** *highlight shapes*
Use **rmapshaper::ms_simplify** to simplify complex shapes
Circles **addCircles(**lng, lat, weight=1, radius, ... **)**
Rectangles **addRectangles(**lng1, lat1, lng2, lat2, fillColor="transparent", ... **)**

Basemaps

addTiles() providers$Stamen.Toner, CartoDB.Positron, Esri.NatGeoWorldMap



Default Tiles Third-Party Tiles **addProviderTiles()**
Use **addTiles()** to add a custom map tile URL template, use **addWMSTiles()** to add WMS (Web Map Service) tiles

GeoJSON and TopoJSON

There are two options to use the GeoJSON/TopoJSON data.
* To read into **sp** objects with the **geojsonio** or **rgdal** package:
geojsonio::geojson_read(...,what="sp") *rgdal::readOGR(...,"OGRGeoJSON")*
* Or to use the **addGeoJSON()** and **addTopoJSON()** functions:
addTopoJSON/addGeoJSON(... weight, color, fill, opacity, fillOpacity...)
Styles can also be tuned separately with a *style: {...}* object.

Other packages including **RJSONIO** and **jsonlite** can help fast parse or generate the data needed.

Shiny Integration

To integrate a Leaflet map into an app:
* In the UI, call **leafletOutput**("name")
* On the server side, assign a **renderLeaflet**(...) call to the output
* Inside the **renderLeaflet** expression, return a Leaflet map object

Modification

To modify an existing map or add incremental changes to the map, you can use **leafletProxy()**. This should be performed in an observer on the server side.

Other useful functions to edit your map:
fitBounds(0, 0, 11, 11**)** *similar to setView*
fit the view to within these bounds
addCircles(1:10, 1:10, layerId = LETTERS[1:10]**)**
create circles with layerIds of "A", "B", "C"...
removeShape(c("B", "F")**)** *remove some of the circles*
clearShapes() *clear all circles (and other shapes)*

Inputs/Events

Object Events
Object event names generally use this pattern:
input$*MAPID_OBJCATEGORY_EVENTNAME*.
Triger an event changes the value of the Shiny input at this variable.
Valid values for *OBJCATEGORY* are *marker*, *shape*, *geojson* and *topojson*.
Valid values for *EVENTNAME* are *click*, *mouseover* and *mouseout*.

All of these events are set to either *NULL* if the event has never happened, or a *list()* that includes:
* lat The latitude of the object, if available; otherwise, the mouse cursor
* lng The longitude of the object, if available; otherwise, the mouse cursor
* id The layerId, if any

GeoJSON events also include additional properties:
* featureId The feature ID, if any
* properties The feature properties

Map Events

input$*MAPID*_click	*when the map background or basemap is clicked* *value -- a list with lat and lng*
input$*MAPID*_bounds	*provide the lat/lng bounds of the visible map area* *value -- a list with north, east, south and west*
input$*MAPID*_zoom	*an integer indicates the zoom level*

Kejia Shi @ Data Science Institute, Columbia University in the City of New York, Kejia.Shi@Columbia.edu

Adapted from Rstudio materials https://rstudio.github.io/leaflet/ 2017/05/01

Machine Learning Modelling in R : : CHEAT SHEET

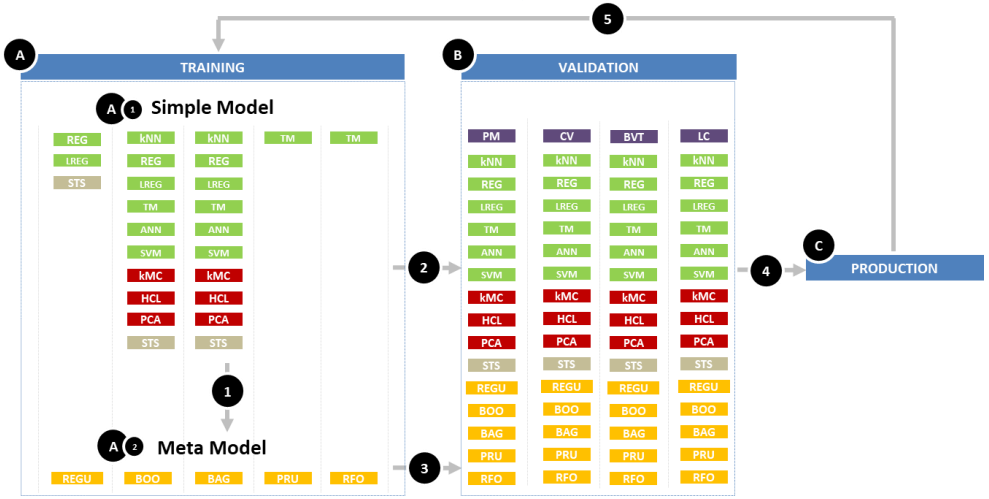
Supervised & Unsupervised Learning

	ALGORITHM	DESCRIPTION	R PACKAGE::FUNCTION	SAMPLE CODE
SUPERVISED LEARNING	NBC Naïve Bayes classifier	A classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature	e1071::naiveBayes	naiveBayes(class ~ ., data = x)
	kNN k–Nearest Neighbours	A non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression	class::knn	knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
	REG Linear Regression	Model the linear relationship between a scalar dependant variable *Y* and one or more explanatory variables (or independent variables) denoted *X*	stats::lm	lm(dist ~ speed, data=cars)
	LREG Logistic Regression	Used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.	stats::glm	glm(Y ~ ., family = binomial (link = 'logit'), data = X)
	TM Tree-Based Models	The idea is to consecutively divide (branch) the training dataset based on the input features until an assignment criterion with respect to the target variable into a "data bucket" (leaf) is reached	rpart::rpart	rpart(Kyphosis ~ Age + Number + Start, data = kyphosis)
	ANN Artificial Neural Network	Neural networks are built from units called perceptrons. Perceptrons have one or more inputs, an activation function and an output. An ANN model is built up by combining perceptrons in structured layers.	neuralnet::neuralnet	neuralnet(f,data=train_,hidden=c(5,3),linear.output=T)
	SVM Support Vector Machine	A data classification method that separates data using hyperplanes	e1071::svm	svm(formula, data = NULL, ..., subset, na.action = na.omit, scale = TRUE)
UNSUPERVISED LEARNING	PCA Principal Component Analysis	A procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.	stats::prcomp stats::princomp FactoMineR::PCA ade4::dudi.pca amap::acp	**stats** : prcomp(formula, data = NULL, subset, na.action, ...) **stats** : princomp(formula, data = NULL, subset, na.action, ...) **FactoMineR** : PCA(decathlon, quanti.sup = 11:12, quali.sup=13) **ade4 :** dudi.pca(deug$tab, center = deug$cent, scale = FALSE, scan = FALSE) **amap :** acp(lubisch)
	KMC k-Mean Clustering	Aims at partitioning *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean	stats::kmeans	kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"), trace=FALSE)
	HCL Hierarchical Clustering	An approach which builds a hierarchy from the bottom-up, and doesn't require the number of clusters to be specified beforehand.	stats::hclust	hclust(d, method = "complete", members = NULL)

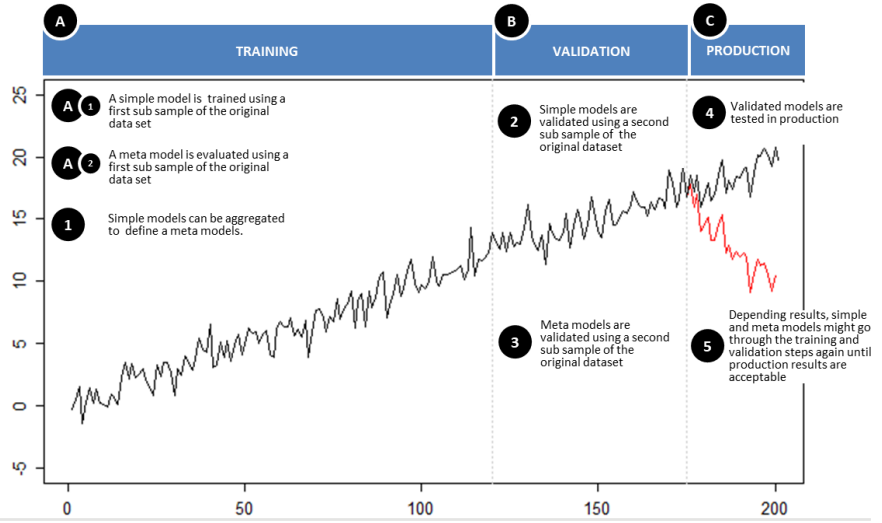
Meta-Algorithm, Time Series & Model Validation

	ALGORITHM	DESCRIPTION	R PACKAGE::FUNCTION	SAMPLE CODE
META ALGORITHM	REGU Regularisation L1 (Lasso) L2 (Ridge)	Regularization adds a penalty on the different parameters of a model to reduce the freedom of the model. Hence, the model will be less likely to fit the noise of the training data and will improve the generalization abilities of the model	glmnet::glmnet	**L1 :** glmnet(myMatrixA, myMatrixB , family = "gaussian", alpha = 1) **L2 :** glmnet(myMatrixA, myMatrixB , family = "gaussian", alpha = 0)
	BOO Boosting	A process of iteratively refining, e.g. by reweighting, of estimated regression and classification functions (though it has primarily been applied to the latter), in order to improve predictive ability.	**Parametric model** - mboost::glmboost	glmboost(Yen ~ ., data = curr1[trnidxs,])
	BAG Bagging	Bagging is a way to increase the power of a predictive statistical model by taking multiple random samples (with replacement) of the training data set, and using each of them to construct a separate model and separate predictions for the original test set	**All models**: foreach **Tree models**: ipred::bagging	**foreach :** d <- data.frame(x=1:10, y=rnorm(10)) s <- foreach(d=iter(d, by='row'), .combine=rbind) %dopar%d identical(s, d) **ipred :** bagging(formula, data, subset, na.action=na.rpart, \dots)
	PRU Pruning	Pruning is a technique that reduces the size of decision tree by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier and hence improves predictive accuracy by reducing overfitting	rpart::prune	prune(x, cp = 0.1)
	RFO Random Forrest	An ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression)	randomForest::randomForest	randomForest(X ~ . , data = Y , subset = mySub)
TIME SERIES	STS Lead-lag analysis, Auto-correlation, Spectral analysis, Time series clustering, Seasonality, Trend....	Random sampling of observations for training and testing a model can be an issue when faced with a times dimension. Random sampling may either destroy serial correlation properties in the data which we would like to exploit	stats xts forecast spectral TTR	**Auto-correlation**: acf(x, lag.max = NULL, type = c("correlation", "covariance", "partial")) **Spectral Analysis**: spec.pgram(myTs, spans = NULL) **Seasonal Decomposition of Time Series** - stl(x, s.window = 7, t.window = 50, t.jump = 1) .....
MODEL VALIDATION	PM Performance metrics	Depends on the problem: • **Regression**: squared errors, outliers, error rate... • **Classification**: Accuracy, precision, recall, F-score...	**Regression**-stats::outlierTest, stats:: qqPlot .... **Classification**-ROCR:: **Tree**: caret:: confusionMatrix	**Regresion**: fit <- lm(Y~X,data=myData) outlierTest(fit) qqPlot(fit, main="QQ Plot")
	BVT Biais-Variance Tradeoff	• Simple models with few parameters are easier to compute but may lead to poorer fits (**high bias**). • Complex models may provide more accurate fits but may over-fit the data (**high variance**)	Tailored to the analysis	Tailored to the analysis
	CV Cross validation	Cross validation compares the test performances of different model realisations with different sets or values of parameters	caret::createDataPartition caret::createFolds	createDataPartition(classes, p = 0.8, list = FALSE)
	LC Learning Curves	Learning curves plot a model's training and test errors, or the chosen performance metric, depending on the training set size	caret:: learing_curve_dat	learing_curve_dat(dat, outcome = NULL, proportion = (1:10)/10, test_prop = 0, verbose = TRUE, ...)

Standard Modelling Workflow



Time Series View



nardl Package

An R package to estimate the nonlinear cointegrating autoregressive distributed lag model

Specifying the Model

Possible syntaxes for specifying the variables in the model:

- **nardl with fixed p and q lags**

nardl(fod~inf,p,q,data=fod,ic="aic",maxlags = FALSE,graph = FALSE,case=3)

- **Auto selected lags (maxlags=TRUE)**

nardl(food~inf,data=fod,ic="aic",maxlags = TRUE,graph = FALSE,case=3)

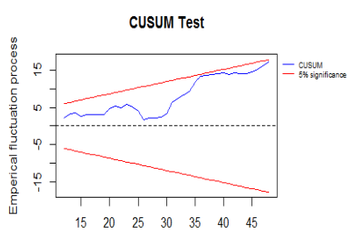
The formula:

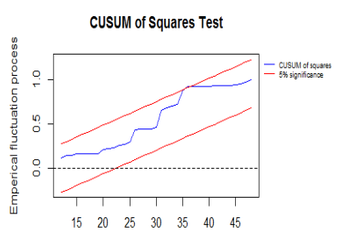
- y~x | z1+z2….
- y the dependent variable
- x the decomposed variable (this package version can't assume more than one decomposed variable)
- z1+z2+… independent variables
- Data is the dataframe
- p number of lags of the dependent variable
- q number of lags of the independent variables
- ic : c("aic","bic","ll","R2") criteria model selection
- maxlags if TRUE auto lags selection
- case case number 3 for (unrestricted intercert, no trend) and 5 (unrestricted intercept, unrestricted trend), 1 2 and 4 not supported

Cusum and CusumQ plot

Cusum and CusumQ plot (graph=TRUE)

nardl(food~inf,data=fod,ic="aic",maxlags = TRUE,graph = TRUE,case=3)





Cointegration bounds test

pssbounds(obs, fstat, tstat = NULL, case, k)

pssbounds specification include:

- Case case number 3 for (unrestricted intercert, no trend) and 5 (unrestricted intercept, unrestricted trend), 1 2 and 4 not supported
- fstat represent the value of the F-statistic
- obs represent the number of observation
- k number of regressors appearing in lag levels

Example:

reg<-nardl(food~inf,fod,ic="aic",maxlags = TRUE,graph = TRUE,case=3)

pssbounds(case=reg$case,fstat=reg$fstat,obs=reg$obs,k=reg$k)

LM test for serial correlation

LM test for serial correlation

bp2(object, nlags, fill = NULL, type = c("F", "Chi2"))

- **Methods and options are:**
- object fitted lm model
- nlags positive integer number of lags
- fill starting values for the lagged residuals in the auxiliary regression. By default 0.
- type Fisher or Chisquare statistics

Example :

reg<-nardl(food~inf,fod,ic="aic",maxlags = TRUE,graph = TRUE,case=3)

bp2(regfit,regnp,fill=0,type="F")

Lagrange multiplier test

Lagrange multiplier test for conditional heteroscedasticity of Engle (1982), as described by Tsay (2005, pp. 101-102)

ArchTest(x, lags = 12, demean = FALSE)

- **Methods and options are:**
- x numeric vector
- lags positive integer number of lags.
- demaen logical: If TRUE, remove the mean before computing the test statistic.

Example :

reg<-nardl(food~inf,fod,ic="aic",maxlags = TRUE,graph = TRUE,case=3)
x<-reg$selresidu
nlag<-reg$np
ArchTest(x,lags=nlag)

Dynamic multipliers plot

Dynamic multiplier plot

plotmplier(model, np, k, h)

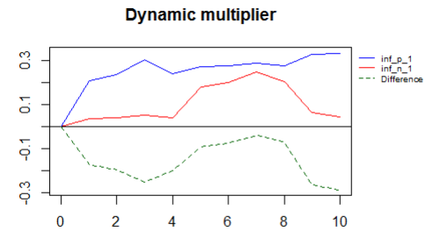
Methods and options are:

- model the fitted model
- np the selected number of lags
- k number of decomposed independent variables
- h is the horizon over which multipliers will be computed

Example

reg<-nardl(food~inf,p=4,q=4,fod,ic="aic",maxlags = FALSE,graph = TRUE,case=3)

plotmplier(reg,reg$np,1,10)



pssbounds

pssbound function display the necessary critical values to conduct the Pesaran, Shin and Smith 2001 bounds test for cointegration. See http://andyphilips.github.io/pssbounds/.

pssbounds(obs, fstat, tstat = NULL, case, k)

Methods and options are:

- obs number of observations
- fstat value of the F-statistic
- tstat value of the t-statistic
- case case number
- k number of regressors appearing in lag levels

Example

reg<-nardl(food~inf,fod,ic="aic",maxlags = TRUE,graph = TRUE,case=3)

pssbounds(case=reg$case,fstat=reg$fstat,obs=reg$obs,k=reg$k)

F-stat concludes I(1) and cointegrating, t-stat concludes I(0).

Parallel Computing :: **CHEAT SHEET**

Splitting :

Splitting a code by :

1. **Task** (different tasks on same data)
2. **Data** (one task on different data)

Hardware needs :

CPU (+2 cores)

RAM (shared memory vs distributed memory)

2 ideas in parallel computing :

1. Map-Reduced Models :
(distributed data; physically on different devices)

- **Hadoop**
- **Spark**

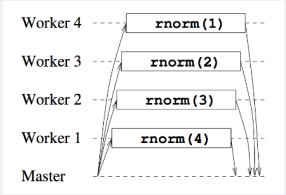
R Packages:

- sparklyr, iotools
- pbdr (programming with big data in R)

2. Master - Worker Models :
(M tasks on C cores; usually 1 < C << M)

R Packages:

- snow, snowFT, snowfal
- foreach
- future, future.apply



Not always parallel computing:

stop/start cluster takes time
overhead (communication time b/w master and workers ; not good for repeatedly sending big data!

Sequential vs Parallel:

```
library(microbenchmark)
microbenchmark( FUN1(…), FUN2(…),
times = 10)
```

parallel.R : core package

```
library(parallel)
ncores <- detectCores(logical=F) # physical cores
cl <- makeCluster(ncores)
clusterApply(cl, x = c(…), fun = FUN) # FUN(x,…)
stopCluster(cl)
```

Initialization of workers :

```
clusterCall(cl,FUN)  # calls FUN on workers
clusterEvalQ(cl, exp) # eval an exp. on workers
## clusterEvalQ(cl, library(foo))
clusterExport(cl, varlist) # varlist on workers
## clusterExport(cl, c("mean")) where mean = 10
```

Data Chunk on workers :

1. generated on workers

```
# clusterApply(cl,x, FUN) e.g FUN(){ rnorm()}
```

2. generated on master and pass to workers

```
# ind <- splitIndices(200, 5)
# clusterApply(cl, ind, FUN)
# (-) : not efficient in Big Data : heavy
```

3. chunk on workers # copy of original Data on all workers

```
# clusterExport(cl, M)        e.g. M is a matrix
# clusterApply(cl, x, FUN)    FUN contains subset M
```

foreach.R : Sequential

```
library(foreach)          # by default return a list
foreach(n = rep(5,3), m = 10^(0:2)) %do% FUN(n,m)
foreach(n, .packages = "X") %do% FUN(n)
# FUN needs package X to be run
foreach(n, .export = c("Y") ) %do% FUN(n,b=Y)
# FUN needs outside object/function "Y"
foreach(n,.combine = rbind) %do% FUN(n) #row bind
foreach(n,.combine = '+') %do% FUN(n) #rbind + colSum
foreach(n,.combine = c) %do% FUN(n) # vector
foreach(n,.combine = c) %:% when(n > 2) %do% FUN(n)
```

future.R : asynchronously

```
library(future) (variables run as soon as created)
plan(multicore)
# plans : sequential, cluster, multicore, multiprocess
x %<-% mean(rnorm(100))
y %<-% mean(rnorm(100))
```

future.apply.R : parallel _apply

```
library(future.apply) (parallel _apply functions)
plan(multicore)     # can be other plans
future_apply(n,FUN),future_lapply(…),future_sapply(…)
```

foreach.R : Parallel

```
needs backend packages support parallel computing
• doParallel(parallel.R),doFuture (future.R),doSEQ
```

doParallel.R : backend of foreach

```
library(doParallel)
cl <- makeCluster(ncores)     # ncores = 2,3,…
registerDoParallel(cl)     # register the backend
foreach(…) %dopar% FUN(…)
```

doFuture.R : backend of foreach

```
library(doFuture)
registerDoFuture()
plan(cluster , workers = 3) # can be other plans
foreach(…) %dopar% FUN(…)
```

Load Balancing: for uneven task times

```
clusterApplyLB(cl,x,FUN) # not for small task time
clusterApply(cl, x = splitIndices(10,2), FUN)
library(itertools)
foreach(s=isplitVector(1:10,chunks =2))%dopar% FUN
# e.g. FUN = sapply(s,"*",100)
future_sapply(…, future.scheduling = 1)
```

randomizr: : **CHEAT SHEET**

Two Arm Trials

Simple random assignment is like flipping coins for each unit separately.

```
simple_ra(N = 100, prob = 0.5)
```

Complete random assignment allocates a fixed number of units to each condition.

```
complete_ra(N = 100, m = 50)
complete_ra(N = 100, prob = 0.5)
```

Block random assignment conducts complete random assignment separately for groups of units.

```
blocks <- rep(c("A", "B", "C"),
              c(50, 100, 200))

# defaults to half of each block
block_ra(blocks = blocks)

# can change with block_m
block_ra(blocks = blocks,
         block_m = c(20, 30, 40))
```

Cluster random assignment allocates whole groups of units to conditions together.

```
clusters <- rep(letters, times = 1:26
cluster_ra(clusters = clusters)
```

Block and cluster random assignment conducts cluster random assignment separately for groups of clusters.

```
clusters <- rep(letters, times = 1:26)
blocks <- rep(paste0("block_", 1:5),
              c(15, 40, 65, 90, 141))
block_and_cluster_ra(blocks = blocks,
                     clusters = clusters)
```

Multi Arm Trials

Set the number of arms with `num_arms` or with `conditions`.

```
complete_ra(N = 100, num_arms = 3)
complete_ra(N = 100, conditions = c("control",
"placebo", "treatment"))
```

The `*_each` arguments in randomizr functions specify design parameters for each arm separately.

```
complete_ra(N = 100, m_each = c(20, 30, 50))
complete_ra(N = 100,
            prob_each = c(0.2, 0.3, 0.5))
```

If the design is the **same** for all blocks, use `prob_each`:

```
blocks <- rep(c("A", "B","C"),
              c(50, 100, 200))
block_ra(blocks = blocks,
         prob_each = c(.1, .1, .8))
```

If the design is **different** in different blocks, use `block_m_each` or `block_prob_each`:

```
block_m_each <- rbind(c(10, 20, 20),
                      c(30, 50, 20),
                      c(50, 75, 75))
block_ra(blocks = blocks,
         block_m_each = block_m_each)

block_prob_each <- rbind(c(.1, .1, .8),
                         c(.2, .2, .6),
                         c(.3, .3, .4))
block_ra(blocks = blocks,
         block_prob_each = block_prob_each)
```

If `conditions` is numeric, the output will be **numeric**.

If `conditions` is not numeric, the output will be a **factor** with levels in the order provided to conditions.

```
complete_ra(N = 100, conditions = -2:2)
complete_ra(N = 100, conditions = c("A", "B"))
```

Declaration

Learn about assignment procedures by "declaring" them with `declare_ra()`

```
declaration <-
  declare_ra(N = 100, m_each = c(30, 30, 40))

declaration # print design information
```

Conduct a random assignment:

```
conduct_ra(declaration)
```

Obtain observed condition probabilities (useful for inverse probability weighting if probabilities of assignment are not constant)

```
Z <- conduct_ra(declaration)
obtain_condition_probabilities(declaration, Z)
```

Sampling

All assignment functions have sampling analogues: Sampling is identical to a two arm trial where the treatment group is sampled.

Assignment	Sampling
`simple_ra()`	`simple_rs()`
`complete_ra()`	`complete_rs()`
`block_ra()`	`strata_rs()`
`cluster_ra()`	`cluster_rs()`
`block_and_cluster_ra()`	`strata_and_cluster_rs()`
`declare_ra()`	`declare_rs()`
`conduct_ra()`	`draw_rs()`

Stata

A Stata version of randomizr is available, with the same arguments but different syntax:

```
ssc install randomizr
set obs 100
complete_ra, m(50)
```

randomizr is part of the DeclareDesign suite of packages for designing, implementing, and analyzing social science research designs.

Basic Regular Expressions in R

Cheat Sheet

Character Classes

[[:digit:]] or \\d	Digits; `[0-9]`
\\D	Non-digits; `[^0-9]`
[[:lower:]]	Lower-case letters; `[a-z]`
[[:upper:]]	Upper-case letters; `[A-Z]`
[[:alpha:]]	Alphabetic characters; `[A-z]`
[[:alnum:]]	Alphanumeric characters `[A-z0-9]`
\\w	Word characters; `[A-z0-9_]`
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; `[0-9A-Fa-f]`
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; `[^[:space:]]`
[[:punct:]]	Punctuation characters; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~`
[[:graph:]]	Graphical characters; `[[:alnum:][:punct:]]`
[[:print:]]	Printable characters; `[[:alnum:][:punct:]\\s]`
[[:cntrl:]] or \\c	Control characters; `\n`, `\r` etc.

Special Metacharacters

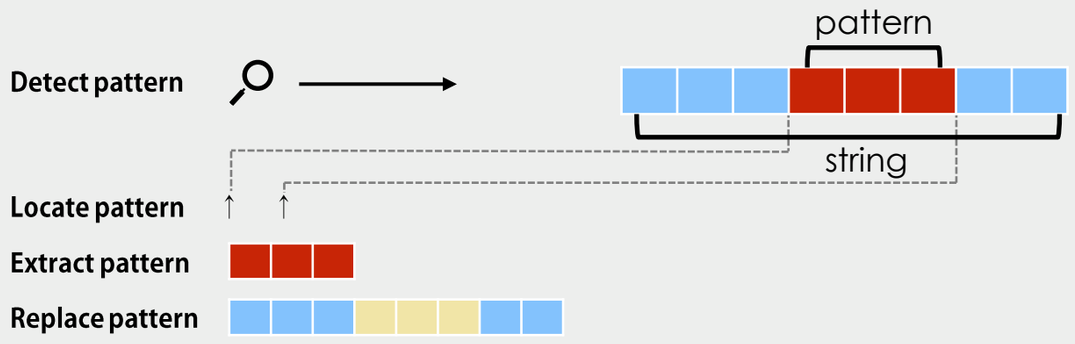
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed

Lookaraounds and Conditionals*

(?=)	Lookahead (requires PERL = TRUE), e.g. `(?=yx)`: position followed by `'xy'`
(?!)	Negative lookahead (PERL = TRUE); position NOT followed by pattern
(?<=)	Lookbehind (PERL = TRUE), e.g. `(?<=yx)`: position following `'xy'`
(?<!)	Negative lookbehind (PERL = TRUE); position NOT following pattern
?(if)then	If-then-condition (PERL = TRUE); use lookaheads, optional char. etc in if-clause
?(if)then|else	If-then-else-condition (PERL = TRUE)

*see, e.g. http://www.regular-expressions.info/lookaround.html
http://www.regular-expressions.info/conditional.html

Functions for Pattern Matching




```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

Detect Patterns

grep(pattern, string)
```
[1] 1 3
```

grep(pattern, string, value = TRUE)
```
[1] "Hiphopopotamus"
[2] "time for bottomless lyrics“
```

grepl(pattern, string)
```
[1]  TRUE FALSE  TRUE
```

stringr::str_detect(string, pattern)
```
[1]  TRUE FALSE  TRUE
```

Locate Patterns

regexpr(pattern, string)
find starting position and length of first match

gregexpr(pattern, string)
find starting position and length of all matches

stringr::**str_locate(string, pattern)**
find starting and end position of first match

stringr::**str_locate_all(string, pattern)**
find starting and end position of all matches

Split a String using a Pattern

strsplit(string, pattern) or stringr::**str_split(string, pattern)**

Extract Patterns

regmatches(string, regexpr(pattern, string))
extract first match `[1] "tam" "tim"`

regmatches(string, gregexpr(pattern, string))
extract all matches, outputs a list
`[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"`

stringr::**str_extract(string, pattern)**
extract first match `[1] "tam" NA "tim"`

stringr::**str_extract_all(string, pattern)**
extract all matches, outputs a list

stringr::**str_extract_all(string, pattern, simplify = TRUE)**
extract all matches, outputs a matrix

stringr::**str_match(string, pattern)**
extract first match + individual character groups

stringr::**str_match_all(string, pattern)**
extract all matches + individual character groups

Replace Patterns

sub(pattern, replacement, string)
replace first match

gsub(pattern, replacement, string)
replace all matches

stringr::**str_replace(string, pattern, replacement)**
replace first match

stringr::**str_replace_all(string, pattern, replacement)**
replace all matches

Character Classes and Groups

.	Any character except \n
|	Or, e.g. `(a|b)`
[…]	List permitted characters, e.g. `[abc]`
[a-z]	Specify character ranges
[^…]	List excluded characters
(…)	Grouping, enables back referencing using `\\N` where `N` is an integer

Anchors

^	Start of the string
$	End of the string
\\b	Empty string at either edge of a word
\\B	NOT the edge of a word
\\<	Beginning of a word
\\>	End of a word

Quantifiers

*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{n,m}	Matches between n and m times

General Modes

By default R uses *extended* regular expressions. You can switch to *PCRE regular expressions* using `PERL = TRUE` for base or by wrapping patterns with `perl()` for stringr.

All functions can be used with literal searches using `fixed = TRUE` for base or by wrapping patterns with `fixed()` for stringr.

All base functions can be made case insensitive by specifying `ignore.cases = TRUE`.

Escaping Characters

Metacharacters (`. * +` etc.) can be used as literal characters by escaping them. Characters can be escaped using `\\` or by enclosing them in `\\Q...\\E`.

Case Conversions

Regular expressions can be made case insensitive using `(?i)`. In backreferences, the strings can be converted to lower or upper case using `\\L` or `\\U` (e.g. `\\L\\1`). This requires `PERL = TRUE`.

Greedy Matching

By default the asterisk `*` is greedy, i.e. it always matches the longest possible string. It can be used in lazy mode by adding `?`, i.e. `*?`.

Greedy mode can be turned off using `(?U)`. This switches the syntax, so that `(?U)a*` is lazy and `(?U)a*?` is greedy.

Note

Regular expressions can conveniently be created using e.g. the packages rex or rebus.

 Ian Kopacka • ian.kopacka@ages.at

Updated: 10/18

Data & Variable Transformation with sjmisc Cheat Sheet

sjmisc complements dplyr, and helps with data transformation tasks and recoding *variables*.

sjmisc works together seamlessly with dplyr and pipes. All functions are designed to support labelled data.

Design Philosophy

The design of sjmisc functions follows the tidyverse-approach: first argument is always the data (either a *data frame* **or** *vector*), followed by variable names to be processed by the functions.

The returned object for each function *equals the type of the data-argument*.

Vector input

- If the data-argument is a *vector*, functions return a *vector*.

rec(mtcars$carb, rec = "1,2=1; 3,4=2; else=3")

Data frame input

- If the data-argument is a *data frame*, functions return a *data frame*.

rec(mtcars, carb, rec = "1,2=1; 3,4=2; else=3")

The …-ellipses Argument

Apply functions to a single variable, selected variables or to a complete data frame.

Variable selection is powered by dplyr's select(): Separate variables with comma, or use dplyr's select-helpers to select variables, e.g. *?rec*:

```
rec(mtcars, one_of(c("gear", "carb")),
    rec = "min:3=1; 4:max=2")

rec(mtcars, gear, carb, rec = "min:3=1; 4:max=2")
```

Descriptives and Summaries

Most of the sjmisc functions (including recode-functions) also work on grouped data frames:

```
library(dplyr)
efc %>%
  group_by(e16sex, c172code) %>%
  frq(e42dep)
```

Frequency Tables

frq(x, ..., sort.frq = c("none", "asc", "desc"), weight.by = NULL, auto.grp = NULL, ...)

Print frequency tables of (labelled) vectors. Uses variable labels as table header.

data(efc); frq(efc, e42dep, c161sex)

Use this data set in examples!

flat_table(data, ..., margin = c("counts", "cell", "row", "col"), digits = 2, show.values = FALSE)

Print contingency tables of (labelled) vectors. Uses value labels.

flat_table(efc, e42dep, c172code, e16sex)

count_na(x, ...)

Print frequency table of tagged NA values.

```
library(haven); x <- labelled(c(1:3,
tagged_na("a",  "a", "z")), labels =
c("Refused" = tagged_na("a"), "N/A" =
tagged_na("z")))
count_na(x)
```

Descriptive Summary

descr(x, ..., max.length = NULL)

Descriptive summary of data frames, including variable labels in output.

descr(efc, contains("cop"), max.length = 20)

Finding Variables in a Data Frame

Use find_var() to search for variables by names, value or variable labels. Returns vector/data frame.

```
# variables with "cop" in names and variable labels
find_var(efc, pattern = "cop", out = "df" )

# variables with "level" in names and value labels
find_var(efc, "level", search = "name_value")
```

Recode and Transform Variables

Recode functions add a *suffix* to new variables, so original variables are preserved.

By default, original input data frame and new created variables are returned. Use *append = FALSE* to return the recoded variables only.

rec(x, ..., rec, as.num = TRUE, var.label = NULL, val.labels = NULL, append = TRUE, suffix = "_r")

Recode values, return result as numeric, character or categorical (factor).

rec(mtcars, carb, rec = "1,2=1; 3,4=2; else=3")

dicho(x, ..., dich.by = "median", as.num = FALSE, var.label = NULL, val.labels = NULL, append = TRUE, suffix = "_d")

Dichotomise variable by median, mean or specific value.

dicho(mtcars, disp)

split_var(x, ..., n, as.num = FALSE, val.labels = NULL, var.label = NULL, inclusive = FALSE, append = TRUE, suffix = "_g")

Split variable into equal sized groups. Unlike dplyr::ntile(), does not split original categories into different values (see examples in ?split_var).

split_var(mtcars, mpg, disp, n = 3)

group_var(x, ..., size = 5, as.num = TRUE, right.interval = FALSE, n = 30, append = TRUE, suffix = "_gr")

Split variable into groups with equal value range, or into a max. # of groups (value range per group is adjusted to match # of groups).

group_var(mtcars, mpg, disp, size = 5)
group_var(mtcars, mpg, size = "auto", n = 4)

std(x, ..., robust = "sd", include.fac = FALSE, append = TRUE, suffix = "_z")

Z-standardise variables. Also center().

std(efc, e17age, c160age)

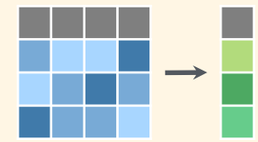
recode_to(x, ..., lowest = 0, highest = -1, append = TRUE, suffix = "_r0")

Shift ("renumber") categories or values.

recode_to(mtcars$gear)

Summarise Variables and Cases

The summary functions mostly mimic base R equivalents, but are designed to work together with pipes and dplyr.



row_sums(x, ..., na.rm = TRUE, var = "rowsums", append = FALSE)

Row sums of data frames.

row_sums(efc, c82cop1:c90cop9)

row_means(x, ..., n, var = "rowmeans", append = FALSE)

Row means, for at least n valid (non-NA) values.

row_means(efc, c82cop1:c90cop9, n = 7)

row_count(x, ..., count, var = "rowcount", append = FALSE)

Row-wise count # of values in data frames. Also col_count().

row_count(efc, c82cop1:c90cop9, count = 2)

Other Useful Functions

- add_columns() and replace_columns() to combine data frames, but either replace or preserve existing columns.
- set_na() and replace_na() to convert regular into missing values, or vice versa. replace_na() also replaces specific tagged NA values only.
- remove_var() and var_rename() to remove variables from data frames, or rename variables.
- group_str() to group similar string values. Useful for variables with similar, but not identically spelled string values that should be "merged".
- merge_df() to full join data frames and preserve value and variable labels.
- to_long() to gather multiple columns in data frames from wide into long format.

Use with %>% and dplyr

```
# use sjmisc-functions in pipes
mtcars %>% select(gear, carb) %>%
  rec(rec = "min:3=1; 4:max=2")

# use sjmisc-function inside mutate
mtcars %>% select(gear, carb) %>% mutate(
  carb2 = rec(carb, rec = "1,2=0;3:8=1"),
  gear2 = rec(gear, rec = "3=1;4:max=2"))
```

 Daniel Lüdecke • d.luedecke@uke.de • github.com/strengejacke

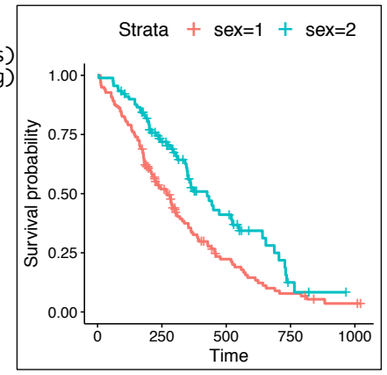
Creating Survival Plots

Informative and Elegant with *survminer*

Survival Curves

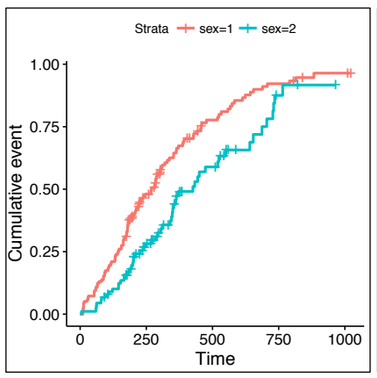
The **ggsurvplot()** function creates **ggplot2** plots from **survfit** objects.

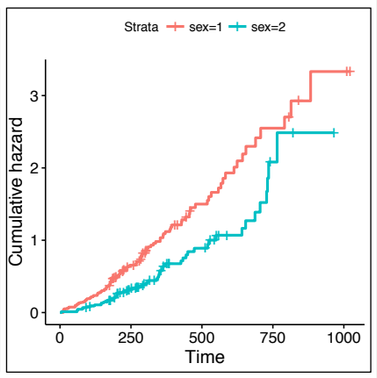
```
library("survival")
fit <- survfit(Surv(time,status)
            ~ sex, data = lung)

class(fit)
## [1] "survfit"

library("survminer")
ggsurvplot(fit, data = lung)
```



Use the **fun** argument to set the transformation of the survival curve. E.g. **"event"** for cumulative events, **"cumhaz"** for the cumulative hazard function or **"pct"** for survival probability in percentage.

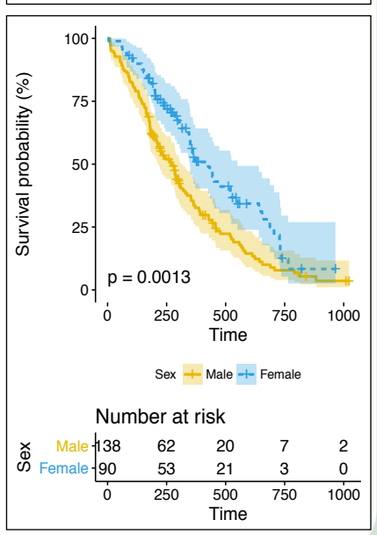
```
ggsurvplot(fit, data = lung, fun = "event")
ggsurvplot(fit, data = lung, fun = "cumhaz")
```





With lots of graphical parameters you have full control over look and feel of the survival plots; position and content of the legend; additional annotations like p-value, title, subtitle.

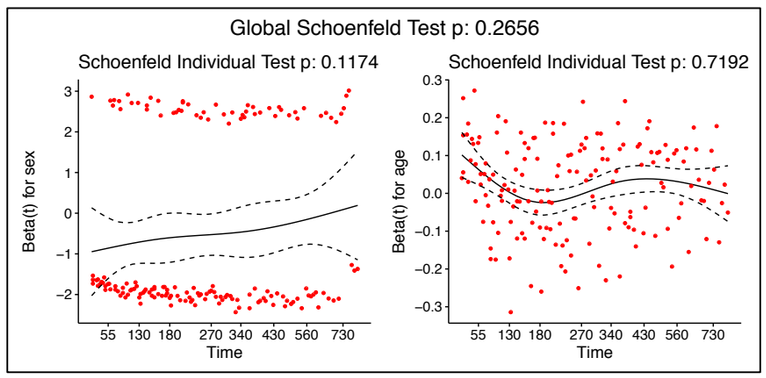
```
ggsurvplot(fit, data = lung,
     conf.int = TRUE,
     pval = TRUE,
     fun = "pct",
     risk.table = TRUE,
     size = 1,
     linetype = "strata",
     palette = c("#E7B800",
                 "#2E9FDF"),
     legend = "bottom",
     legend.title = "Sex",
     legend.labs = c("Male",
                     "Female"))
```



Diagnostics of Cox Model

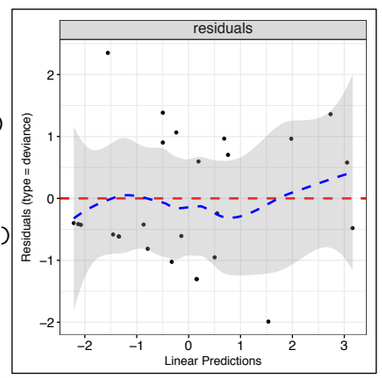
The function **cox.zph()** from **survival** package may be used to test the proportional hazards assumption for a Cox regression model fit. The graphical verification of this assumption may be performed with the function **ggcoxzph()** from the **survminer** package. For each covariate it produces plots with scaled Schoenfeld residuals against the time.

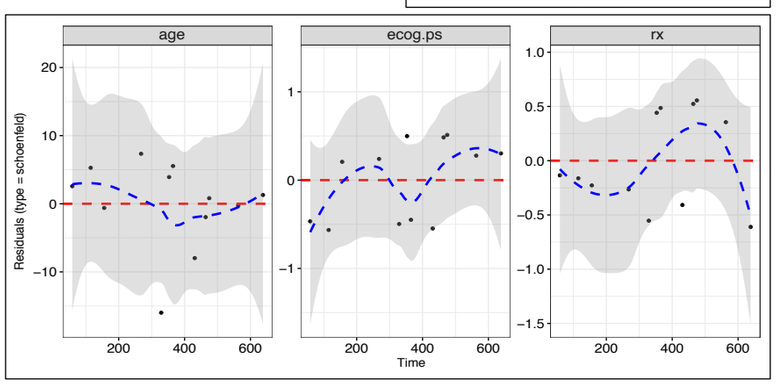
```
library("survival")
fit <- coxph(Surv(time, status) ~ sex + age, data = lung)
ftest <- cox.zph(fit)
ftest
##            rho chisq     p
## sex     0.1236 2.452 0.117
## age    -0.0275 0.129 0.719
## GLOBAL      NA 2.651 0.266
library("survminer")
ggcoxzph(ftest)
```



The function **ggcoxdiagnostics()** plots different types of residuals as a function of time, linear predictor or observation id. The type of residual is selected with **type** argument. Possible values are "martingale", "deviance", "score", "schoenfeld", "dfbeta"', "dfbetas", and "scaledsch".
The **ox.scale** argument defines what shall be plotted on the OX axis. Possible values are "linear.predictions", "observation.id", "time".
Logical arguments **hline** and **sline** may be used to add horizontal line or smooth line to the plot.

```
library("survival")
library("survminer")
fit <- coxph(Surv(time, status) ~
           sex + age, data = lung)


ggcoxdiagnostics(fit,
     type = "deviance",
     ox.scale = "linear.predictions")


ggcoxdiagnostics(fit,
     type = "schoenfeld",
     ox.scale = "time")
```

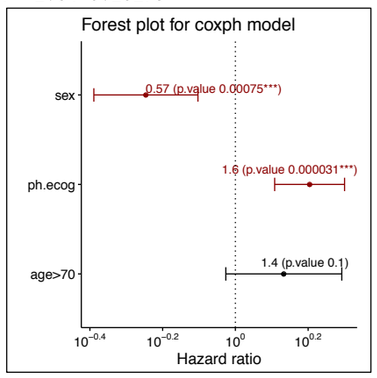




Summary of Cox Model

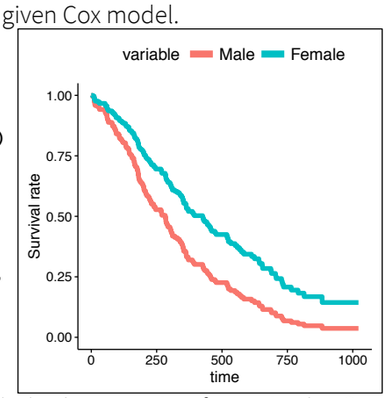
The function **ggforest()** from the **survminer** package creates a forest plot for a Cox regression model fit. Hazard ratio estimates along with confidence intervals and p-values are plotter for each variable.

```
library("survival")
library("survminer")
lung$age <- ifelse(lung$age > 70, ">70","<= 70")
fit <- coxph( Surv(time, status) ~ sex + ph.ecog + age, data = lung)
fit

## Call:
## coxph(formula = Surv(time, status) ~ sex+ph.ecog+age, data=lung)
##
##            coef exp(coef) se(coef)     z       p
## sex     -0.567     0.567    0.168 -3.37 0.00075
## ph.ecog  0.470     1.600    0.113  4.16 3.1e-05
## age>70   0.307     1.359    0.187  1.64 0.10175
##
## Likelihood ratio test=31.6  on
## n= 227, number of events= 164

ggforest(fit)
```



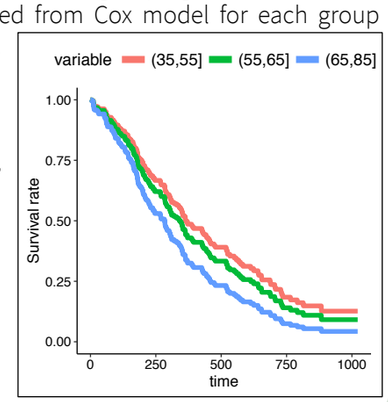
The function **ggcoxadjustedcurves()** from the **survminer** package plots Adjusted Survival Curves for Cox Proportional Hazards Model. Adjusted Survival Curves show how a selected factor influences survival estimated from a Cox model.
Note that these curves differ from Kaplan Meier estimates since they present expected ssurvival based on given Cox model.

```
library("survival")
library("survminer")

lung$sex <- ifelse(lung$sex == 1,
                   "Male", "Female")

fit <- coxph(Surv(time, status) ~
             sex + ph.ecog + age,
             data = lung)

ggcoxadjustedcurves(fit, data=lung,
             variable=lung$sex)
```



Note that it is not necessary to include the grouping factor in the Cox model. Survival curves are estimated from Cox model for each group defined by the factor independently.

```
lung$age3 <- cut(lung$age,
                 c(35,55,65,85))

ggcoxadjustedcurves(fit, data=lung,
             variable=lung$age3)
```



This onepager presents the **survminer** package [Alboukadel Kassambara, Marcin Kosinski 2017] in version 0.3.
See *https://github.com/kassambara/survminer/* for more details.

The teachR's : : CHEAT SHEET

Getting ready to teach some R? Use our cheat sheet to **prepare**, **teach** and **debrief**

DRAFT v0.1

Before the course (design)

Use these to prepare your lecture/course:

Who are your learners? (Persona Analysis)
(change according to requirements…[1])

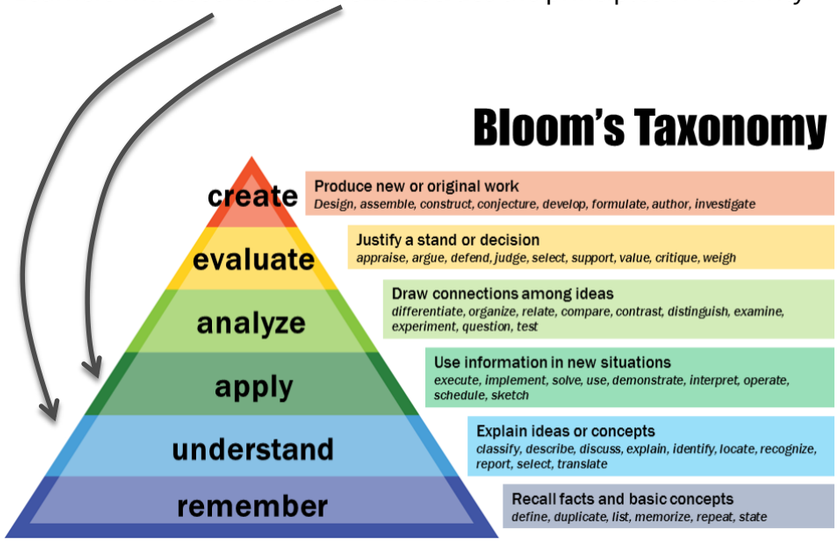
The R novice
Background: some statistics, some programming
Prior knowledge: basic R course, base R syntax
Goals: understand tidy concepts,
expose to tidyverse practices
Special needs: First successes, mitigate fears, encourage learning

The R "false expert"
Background: working with R for some time, but doesn't keep-up
Prior knowledge: been using base R syntax, loops, and functions
Goals: strengthen tidyverse familiarity, apply dplyr workflow
Special needs: switch from obsolete methods to state-of-the-art R

Define goals using Bloom's Taxonomy [2]

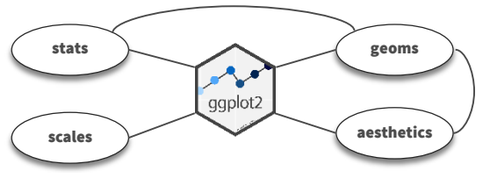
Design your classes to move your learners "up the pyramid"
Keep "realistic goals" for each persona

For example (R shiny – novice):
Learners will **describe** and **demonstrate** the principles of reactivity



Design your lecture using Conceptual maps

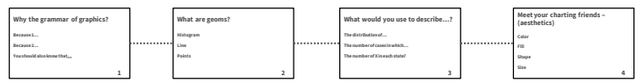
Keep the number of elements small (up to ~7 items), e.g.:



Write the "final exam"

How are you going to test knowledge after the lecture?
What should learners be able to answer?

Turn the concepts into slides



Add faded examples (exercises) and check-in slides

Check-ins, e.g.: multiple choice quick questions"
Faded examples = fill in the blanks, e.g.:

```
ggplot(data = ___, mapping = aes(x = ___, y = ___)) +
  geom_*() +
```

During the course (implement)

Things you can implement to improve the lecture workflow

Learners use **sticky notes** to indicate status:
Green = everything is OK/Exercise completed.
Red = help me! I ran into a problem.
Blue = I need a break.

Engage with online mini-polls during lecture
Use regular check-ins during your lecture (3-4 check-ins per hour). No one "opts-out" (everyone answers once in a while – you choose)

After the course (learn/improve)

Make sure you make the most to improve your next lecture

Use feedback to understand what went well, and what you need to improve.

Measure the time each lecture takes you (or where did you get to), so that next time your time estimates will be better

Useful tips and tricks

Useful tips for preparations

Use github to upload course materials

RMarkdown for exercises

Recommended reading materials/references for R courses:

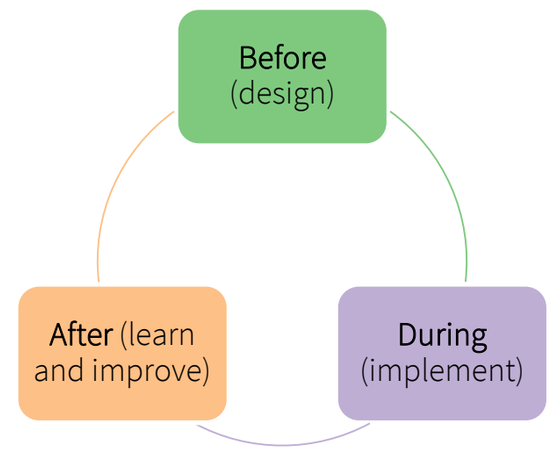
R for Data Science / Garrett Grolemund and Hadley Wickham (r4ds.had.co.nz)

Advanced R/ Hadley Wickham (adv-r.had.co.nz)

RStudio **Cheat sheets**:

https://www.rstudio.com/resources/cheatsheets/

Iterative work flow



Additional sources

[1] Dreyfus, Stuart E., and Hubert L. Dreyfus. *A five stage model of the mental activities involved in directed skill acquisition.* No. ORC-80-2. California Univ Berkeley Operations Research Center, 1980.

[2] Content downloaded from https://cft.vanderbilt.edu/guides-sub-pages/blooms-taxonomy/ (CC-BY-SA Vanderbilt University Center for Teaching)

Sarid Research Services

Class Agnostic Time Series with tsbox : : **CHEAT SHEET**

Basics

IDEA

tsbox provides a time series toolkit which:

1. works identically with most time series **classes**
2. handles regular and irregular **frequencies**
3. **converts** between classes and frequencies

Most functions in tsbox have the same structure:

function starts with ts_

first argument is any ts-boxable object

```
a <- ts_pc(AirPassengers)
```

returns a ts-boxable obect of the same class as input

COMBINE TIME SERIES

collect time series of **all classes** and **frequencies** as multiple time series

```
ts_c(mdeaths, austres)
```

combine time series to a new, single time series (first series wins if overlapping)

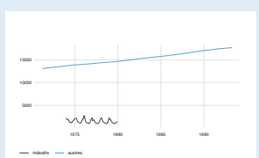
```
ts_bind(mdeaths, austres)
```

like ts_bind, but extra- and retropolate, using growth rates

```
ts_chain(mdeaths, austres)
```

PLOT AND SUMMARIZE

Plot time series of **all classes** and **frequencies**



```
ts_plot(mdeaths, austres)
ts_ggplot(mdeaths, austres)
```

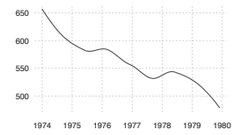
```
ts_summary(ts_c(mdeaths, austres))
```

```
      id obs    diff freq      start        end
1 mdeaths  72 1 month   12 1974-01-01 1979-12-01
2 austres  89 3 month    4 1971-04-01 1993-04-01
```

Helper Functions

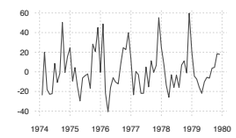
Transform time series of **all classes** and **frequencies**

TRANSFORM



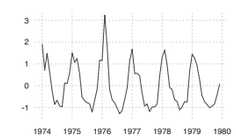
ts_trend(): Trend estimation based on loess

```
ts_trend(fdeaths)
```



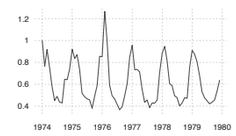
ts_pc(), ts_pcy(), ts_pca(), ts_diff(), ts_diffy(): (annualized) Percentage change rates or differences to previous period, year

```
ts_pc(fdeaths)
```



ts_scale(): normalize mean and variance

```
ts_scale(fdeaths)
```



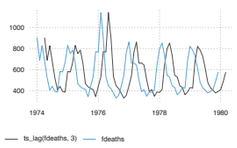
ts_index():Index, based on levels
ts_compound(): Index, based on growth rates

```
ts_index(fdeaths, base = 1976)
```

ts_seas(): seasonal adjustment using X-13

```
ts_seas(fdeaths)
```

SPAN AND FREQUENCY



ts_lag(): Lag or lead of time series

```
ts_lag(fdeaths, 4)
```

ts_frequency(): convert to frequency

```
ts_frequency(fdeaths, "year")
```

ts_span(): filter time series for a time span.

```
ts_span(fdeaths, "1976-01-01")
ts_span(fdeaths, "-5 year")
```

Class Conversion

tsbox is built around a set of converters, which convert time series of the following **supported classes** to each other:

converter function	ts-boxable class
ts_ts()	ts, mts
ts_data.frame(), ts_df()	data.frame
ts_data.table(), ts_dt()	data.table
ts_tbl()	df_tbl, "tibble"
ts_xts()	xts
ts_zoo()	zoo
ts_tibbletime()	tibbletime
ts_timeSeries()	timeSeries
ts_tsibble()	tsibble
ts_tslist()	a list with ts objects

Time Series in data frames

LONG STRUCTURE

Default structure to store multiple time series in long data frames (or data tables, or tibbles)

```
ts_df(ts_c(fdeaths, mdeaths))
```

id	time	value
fdeaths	1974-01-01	901
fdeaths	1974-02-01	689
fdeaths	1974-03-01	827
...	...	...

AUTO-DETECT COLUMN NAMES

tsbox auto-detects a *value*-, a *time*- and zero, one or several *id*-columns. Alternatively, the *time*- and the *value*-column can be explicitly named **time** and **value**.

ts_default(): standardize column names in data frames

RESHAPE

ts_wide(): convert default long structure to wide
ts_long(): convert wide structure to default long

USE WITH PIPE

tsbox plays well with tibbles and with `%>%`, so it can be easily integrated into a dplyr/pipe workflow

```
library(dplyr)
ts_c(fdeaths, mdeaths) %>%
  ts_tbl() %>%
  ts_trend() %>%
  ts_pc()
```

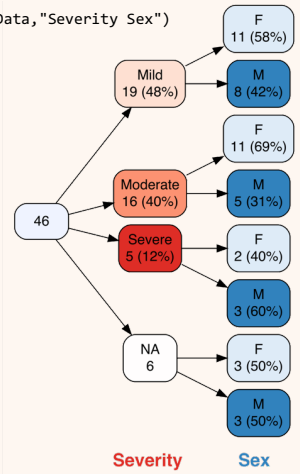
pass return value as first argument to the next function

vtree cheatsheet

Basics

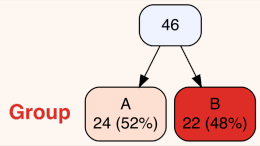
Draw a basic variable tree

```
vtree(FakeData,"Severity Sex")
```



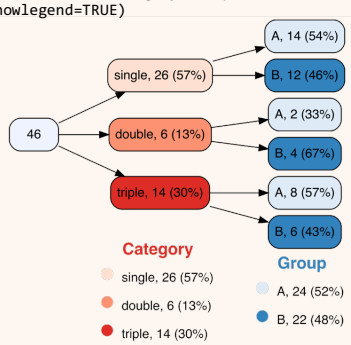
Draw the tree vertically

```
vtree(FakeData,"Group",horiz=FALSE)
```



Display a legend

```
vtree(FakeData,"Category Group",sameline=TRUE,
  showlegend=TRUE)
```



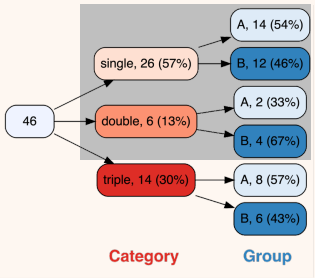
Variable specification

Modifier	Effect
prefix `is.na:`	is.na(variable)
prefix `stem:`	all REDCap variables with stem
prefix `tri:`	trichotomize in each node
`variable=`*x*	dichotomize at *x*
`variable<`*x*	dichotomize below x
`variable>`*x*	dichotomize above *x*

Pruning

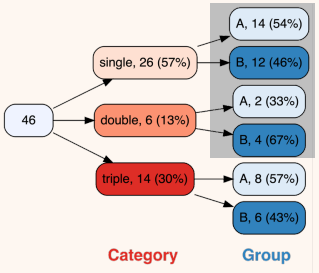
Prune single and double and their descendants

```
vtree(FakeData,"Category Group",sameline=TRUE,
  prune=list(Category=c("single","double")))
```



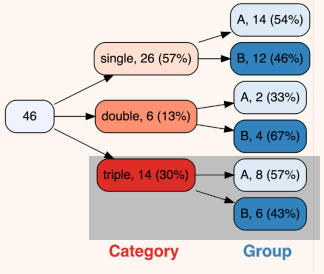
Prune nodes below single and double

```
vtree(FakeData,"Category Group",sameline=TRUE,
  prunebelow=list(Category=c("single","double")))
```



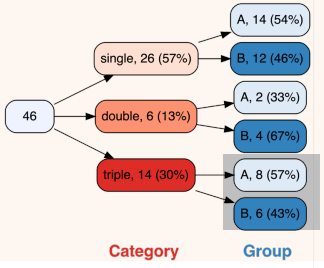
Only keep single and double and their descendants

```
vtree(FakeData,"Category Group",sameline=TRUE,
  keep=list(Category=c("single","double")))
```



Only include descendants of single and double

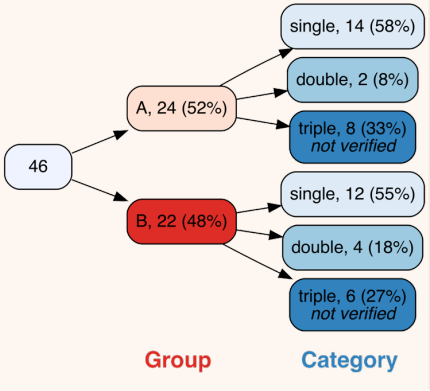
```
vtree(FakeData,"Category Group",sameline=TRUE,
  follow=list(Category=c("single","double")))
```



Text

Add text to nodes

```
vtree(FakeData,"Group Category",sameline=TRUE,
  text=list(Category=c(triple="\n*not verified*")))
```



Code	Meaning
`\n`	Line break
`*...*`	Italics
`**...**`	Bold
`^...^`	Superscript
`~...~`	Subscript
`%%red ...%%`	Make text red (or another color)

Text in a specific node ("targeted text")

```
ttext=list(Group="A",Severity="Moderate",text="hi")
```

Labeling

Parameter

Change variable labels
`labelvar=c(`
`Severity="New label for Severity")`

Change node labels for a variable
`labelnode=list(MyVar=`
`c(New="Old",New2="Old2"))`

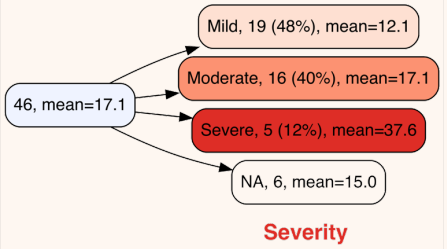
Change a specific node label
`tlabelnode=list(`
`c(Group="A",Sex="F",label="girl"))`

Font size (points) for variable names `varnamepointsize=15`
Specify an optional label for the root node `title="All patients"`
Show node labels? `shownodelabels=TRUE`
Show variable names? `showvarnames=TRUE`

Summary information

Add summary statistics to nodes

```
vtree(FakeData,"Severity",sameline=TRUE,
  summary="Score , mean=%mean%")
```



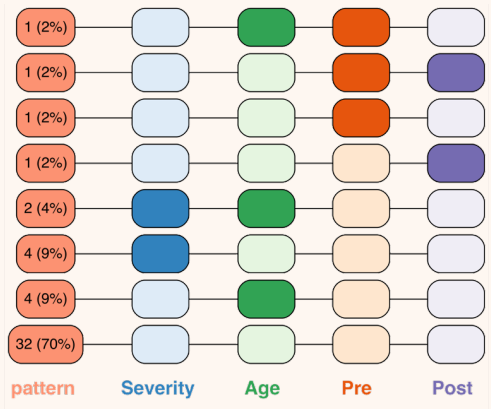
Code	Result
`%mean%`	mean
`%SD%`	standard deviation
`%min%`	minimum
`%max%`	maximum
`%pX%`	Xth percentile (e.g. p50 means the 50th percentile)
`%median%`	median, i.e. p50
`%IQR%`	IQR, i.e. p25, p75
`%npct%`	frequency and percentage
`%pct%`	just percentage
`%list%`	list of individual values, separated by commas
`%listlines%`	list of individual values, each on a separate line
`%mv%`	the number of missing values

Code	Summary information restricted to:
`%noroot%`	all nodes *except* the root
`%leafonly%`	leaf nodes
`%var=v%`	nodes of variable *v*
`%node=n%`	nodes named *n*

Checking for missing values

Show missing value patterns; dark color = missing, light = not missing

```
vtree(FakeData,"Severity Age Pre Post",check.is.na=TRUE)
```



xplain Cheat Sheet

Important Links

xplain package on CRAN	https://cran.r-project.org/web/packages/xplain/index.html
xplain web tutorial	http://www.zuckarelli.de/xplain/index.html
xplain cheat sheet	http://www.zuckarelli.de/xplain/xplain_cheatsheet.pdf
xplain on GitHub	https://www.github.com/jsugarelli/xplain

Purpose & Application

- xplain allows to **write interpretation/explanation texts** for statistical functions in the form of XML files.
- The user of the functions can read these explanations **while working on his/her specific problems**.
- xplain explanations **can react to the user's results** and provide meaningful insights related to the user's problem.
- For this, the xplain **XML files can contain R code** and can **work with the return object** of the user's function call.

```
> xplain("lm(education ~ young + income + urban)")
> Your R^2 is 0.11 which is quite low. There is a serious
risk your model is misspecified. You should reconsider the
selection of variables included in your model.
```

xplain XML files

<xplain>

1 Any valid xplain XML must be enclosed in an `<xplain>` block. Multiple `<xplain>` blocks per XML file are possible.

<package>

2 A `<package>` block combines all functions from the same package.

<function>

3 Within a `<function>` block, explanations/interpretations for the function as such or for specific elements of the return object can be provided.

<result>

4 Packages explanations/ interpretations related to one element of the function's return object.

```
<xml>
 1 <xplain>
     2 <package name = "stats">
         3 <function name = "lm">
             4 <title>This is about lm</title>
             5 <text>…</text>
             6 <result name = "coefficients">
                 4 <title>…<title>
                 5 <text>…</text>
               </result>
           </function>
       </package>
   </xplain>
</xml>
```

Not case-sensitive

<title>

5 Structures explanations with headers.

<text>

6 The actual explanations/interpretations. Can include R code with references to the function's return object.

Including R code

R code can be easily integrated into `<text></text>` elements:

`<text>` `!%%` `R code` `%%!` `</text>`

R code delimiter tags

Access the explained function's (`<function name="…">`) return object:

- Access the full return object with `@`. Example: `summary(@)`.
- Access the current `<result name="…">` item of the return object with `##`. Example: `mean(##)`.

Using placeholders

`<define name=` `"placeholder"` `> !%%` `R code` `%%!` `</define>`

`</text>` `Text…` `!**` `"placeholder"` `**!` `Text…` `</text>`

Placeholder name delimiter tags

Example:

```
<define name="s">!%% summary(@) %%!</define>
<text>And here is the summary !**s**! for your model</text>
```

Iterating through (items of) the return object

- To apply a `<text>` element to a whole matrix, data frame, vector or list, use the **`foreach`** attribute.
- Value of `foreach` defines what is iterated over and (for 2D structures) in which sequence; `items` is for lists.
- `$` is a placeholder for the index of the current element.
- **Example** (shows all 1st column elements of the coefficient matrix):

```
<text foreach="rows">!%% @$coefficients[$,1] %%!</text>
```

`foreach =`
`"rows"`
`"columns"`
`"rows, columns"`
`"columns, rows"`
`"items"`

Main attributes: Overview

name	Name of the element (package, function, result).
lang	Language (ISO code) of the explanation (e.g. `"EN"`).
level	Complexity level; integer number; cumulative, i.e. `level=1` explanations will also be presented when `level=2` or `level=3` are called.

Attributes: Inheritance and necessity

- Elements **inherit attributes from higher-level** elements; e.g., if only one language, definition on `<xplain>` level suffices. Lower-level attributes overrule higher-level.
- **`name`** attribute required for `<package>`, `<function>` and `<result>` elements.
- All levels shown, if no **`level`** is given to `xplain()`.

Calling xplain()

1 Direct call of `xplain()`

call	Call of the explained function as string
xml	Path of the XML file providing the explanations
lang	Language of the explanations to be shown (default means English)
level	Complexity level of the explanations (cumulative! Default means "all")

2 Wrapper function with `xplain.getcall()`

Example: lm

```
lm.xplain <- function(formula, data, subset, weights, na.action,
method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
singular.ok = TRUE, contrasts = NULL, offset, ...) {
  call<-xplain.getcall("lm")
  xplain(call, xml="http://www.zuckarelli.de/example_lm.xml")
}
```

Joachim Zuckarelli 2018 jsugarelli joachim@zuckarelli.de