Cheatsheets R

Hello there!

Here are some beginner friendly cheatsheets to use for R. Since R is open source and everyone can contribute to it, there are some absolutely amazing cheatsheets out there. I included some of the originals one in this document (credits included, of course). The cheatsheets are structured in a clear format that gives you access to quick commands and relevant pieces of information. If you want to know more, always just Google.

Last page consists of a bunch of helpful resources that you can abuse to expand your knowledge.

Have lots of fun and do not be scared! Be brave enough to expand your skills and train your logic while programming ♡

Kind regards,
Wouter & Ioana

Basics: : CHEAT SHEET

Data Import

Tidyverse is your to go package for all basics

R's tidyverse is built around tidy data stored in tibbles, which are enhanced data frames

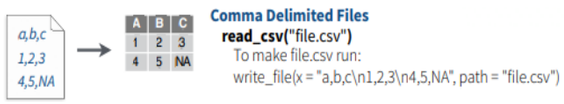
Try one of the following packages to import other types of files

- **haven -** SPSS, Stata, and SAS files
- **readxl -** excel files (.xls and .xlsx)
- **DBI –** databases
- **jsonlite -** json
- **xml2 -** XML
- **httr -** Web APIs
- **rvest -** HTML (Web Scraping)

import.packages(namepackage)
library("namepackage")

You need both in order to install and access the package. For library use " ".

Read Tabular Data



New Versions. In the new R version read() function can be used with . Eg: read.csv()

Delimitation. If your file is delimated by "," or ";" use read.csv("file.csv", sep = ".")

Assign it to a variable In order to modify and access the data easier use :

data <- read.csv("file.csv", sep = ".") #name the variable how you want

Set Working Directory

You need to make sure R can access your file.
Go into the folder where you have the file and set your working directory accordingly

getwd()
Run this command to see which folder does R access for getting your files

setwd("C:/Users/YourName/Folder")
Have all your R files in here (including the datasets).

Tick. You can use this in order to read the file directly from the folder:

data <- read.csv("C:/Users/YourName/Folder/file.csv", sep = ".")

Get help

The help() function and ? help operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.

To access documentation for the standard lm (linear model) function, for example, enter the command help(lm) or help("lm"), or ?lm or ?"lm" (i.e., the quotes are optional).

Stuck?

When Googling, try to specify R and the step you want to search for. Eg: R create a dataframe. Always try multiple websites because information might be explained easier in some.

For Googling errors your best bet is to copy the error from the Terminal and paste it in the Google search bar. Guarantee the first three result will come up with the solution for your problem.

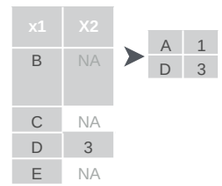
Remember that when you get errors, check your code multiple times. There might be spelling mistakes, or you forgot a sign by mistake. Do not be afraid to Google examples for your code. However, try to understand what you copy paste. If you encounter a function you are not familiar with in a code on the internet, try to research it simple as R + function name. The R documentation will give a bit more insight into it .

Remember, understating the logic behind what you code is the most important aspect of programming, Even professional developers will still google the documentation of some functions after decades of experience.

Extra Elements Missing from Cheatsheets

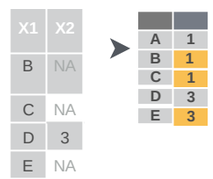
Handle Missing Values

drop_na(data, ...)
Drop rows containing NA's in columns.



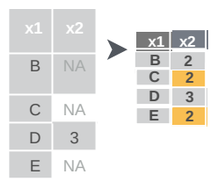
drop_na(x, x2)

fill(data, ..., .direction = c("down", "up"))
Fill in NA's in ... columns with most recent non NA values.



fill(x, x2)

replace_na(data, replace = list(), ...)
Replace NA's by column.



replace_na(x, list(x2 = 2))

Construct A Tibble In Two Ways

tibble(...)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(...) Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tribble(x, ...) **Convert data frame to tibble.**

enframe(x, name = "name", value = "value") **Convert named vector to a tibble**

is_tibble(x) **Test whether x is a tibble.**

Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.

help.search('weighted mean')
Search the help files for a word or phrase.

help(package = 'dplyr')
Find help for a package.

More aboutan object

str(iris)
Get a summary of an object's structure.

class(iris)
Find the class an object belongs to.

Using Libraries

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)
Return x sorted.

rev(x)
Return x reversed.

table(x)
See counts of values.

unique(x)
See unique values.

Selecting Vector Elements

By Position

x[4]	The fourth element.
x[-4]	All but the fourth.
x[2:4]	Elements two to four.
x[-(2:4)]	All elements except two to four.
x[c(1, 5)]	Elements one and five.

By Value

x[x == 10]	Elements which are equal to 10.
x[x < 0]	All elements less than zero.
x[x %in% c(1, 2, 5)]	Elements in the set 1, 2, 5.

Named Vectors

x['apple']	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
    Do something
}
```

Example

```
for (i in 1:4){
    j <- i + 10
    print(j)
}
```

While Loop

```
while (condition){
    Do something
}
```

Example

```
while (i < 5){
    print(i)
    i <- i + 1
}
```

If Statements

```
if (condition){
    Do
    something
} else {
    Do something different
}
```

Example

```
if (i > 3){
    print('Yes')
} else {
    print('No')
}
```

Functions

```
function_name <- function(var){
    Do something
    return(new_variable)
}
```

Example

```
square <- function(x){
    squared <- x*x
    return(squared
    )
```

Reading and Writing Data

Input	Ouput	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/ write.table.
load('file.RData')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions

a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
as.numeric	1, 0, 1	Integers or floating point numbers.
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Maths Functions

log(x)	Natural log.	sum(x)	Sum.
exp(x)	Exponential.	mean(x)	Mean.
max(x)	Largest element.	median(x)	Median.
min(x)	Smallest element.	quantile(x)	Percentage quantiles.
round(x, n)	Round to n decimal places.	rank(x)	Rank of elements.
signif(x, n)	Round to n significant figures.	var(x)	The variance.
cor(x, y)	Correlation.	sd(x)	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

ls()	List all variables in the environment.
rm(x)	Remove x from the environment.
rm(list = ls())	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrixes

m <- matrix(x, nrow = 3, ncol = 3)

Create a matrix from x.

m[2,] - Select a row

m[, 1] - Select a column

m[2, 3] - Select an element

t(m)

Transpose

m %*% n

Matrix Multiplication

solve(m, n)

Find x in: m * x = n

Lists

l <- list(x = 1:5, y = c('a', 'b'))

A list is collection of elements which can be of different types.

l[[2]]	l[1]	l$x	l['y']
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Data Frames

Also see the **dplyr** library.

df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))

A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

df$x

df[[2]]

Understanding a dataframe

View(df)	See the full data frame.
head(df)	See the first 6 rows.

Matrix subsetting

df[, 2]

df[2,]

df[2, 2]

nrow(df)
Number of rows.

ncol(df)
Number of columns.

dim(df)
Number of columns and rows.

cbind - Bind columns.

rbind - Bind rows.

Strings

Also see the **stringr** library.

paste(x, y, sep = ' ')	Join multiple vectors together.
paste(x, collapse = ' ')	Join elements of a vector together.
grep(pattern, x)	Find regular expression matches in x.
gsub(pattern, replace, x)	Replace matches in x with a string.
toupper(x)	Convert to uppercase.
tolower(x)	Convert to lowercase.
nchar(x)	Number of characters in a string.

Factors

factor(x)
Turn a vector into a factor. Can set the levels of the factor and the order.

cut(x, breaks = 4)
Turn a numeric vector into a factor but 'cutting' into sections.

Statistics

lm(x ~ y, data=df)
Linear model.

glm(x ~ y, data=df)
Generalised linear model.

summary
Get more detailed information out a model.

t.test(x, y)
Preform a t-test for difference between means.

pairwise.t.test
Preform a t-test for paired data.

prop.test
Test for a difference between proportions.

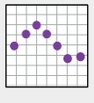
aov
Analysis of variance.

Distributions

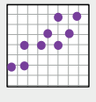
	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Poison	rpois	dpois	ppois	qpois
Binomial	rbinom	dbinom	pbinom	qbinom
Uniform	runif	dunif	punif	qunif

Plotting

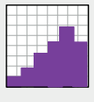
Also see the **ggplot2** library.



plot(x)
Values of x in order.



plot(x, y)
Values of x against y.



hist(x)
Histogram of x.

Dates

See the **lubridate** library.

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, ...)
Compute table of summaries.
summarise(mtcars, avg =mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in ... Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE**)**
Returns copy of table grouped by ...
g_iris <- group_by(iris, Species)

ungroup(x, ... **)**
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, ... **)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length >7)*

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace =TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, ... **)** Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, ... **)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL**)**
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1**)** Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, ... **)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)** **num_range(**prefix, range**)** **:**, e.g. mpg:cyl
ends_with(match**)** **one_of(**...**)** **-**, e.g, -Species
matches(match**)** **starts_with(**match**)**

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, ... **)**
Compute new column(s).
mutate(mtcars, gpm =1/mpg)

transmute(.data, ... **)**
Compute new column(s), drop others.
transmute(mtcars, gpm =1/mpg)

mutate_all(.tbl, .funs, ... **)** Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, ... **)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL**)** Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new =1:32)*

rename(.data, ... **)** Rename columns.
rename(iris, Length =Sepal.Length)

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()**- Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()

```
iris %>% mutate(Species = case_when(
            Species == "versicolor" ~"versi",
            Species == "virginica" ~ "virgi",
                        TRUE ~Species))
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** -# of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()**- last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile()- nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad()- median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

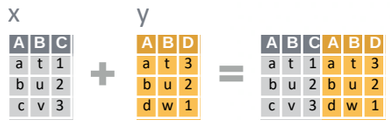
rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES



Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...**)**
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...**)**
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...**)**
Join data. Retain only rows with matches.

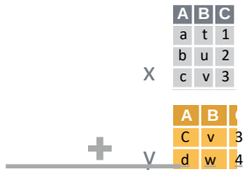
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...**)**
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C"= "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES



Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

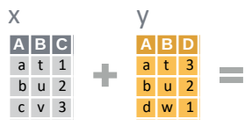
intersect(x, y, …)
Rows that appear in both x and y.

setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS



Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...**)**
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...**)**
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

R Base Graphics Cheatsheet

SET GRAPHICAL PARAMETERS

the following can only be set with **par()**

par(...)

multiple plots	`mfcol = c(nrow,ncol)` `mfrow = c(nrow,ncol)`	*plot margins (outer)*	`oma = c(bottom, left, top, right)` default: `c(0, 0, 0, 0)` lines
plot margins	`mar = c(bottom, left, top, right)` default: `c(5.1, 4.1, 4.1, 2.1)` lines	*query x & y limits*	`par("usr")`

CREATE A NEW PLOT

Bar charts	**barplot**(*height,...*)	**Histograms**	**hist**(*x,...*)
bar labels	`names.arg =`	*breakpts*	`breaks =`
border	`border =`		
fill color	`col =`	**Line charts**	**plot**(*x*, `type = "l"`)
horizontal	`horiz = TRUE`	*line type*	`lty =` `"blank"|0` `"solid"|1` `"dashed"|2` `"dotted"|3`
Box plots	**boxplot**(*x,...*)	*line width*	`lwd =`
horizontal	`horizontal = TRUE`		
box labels	`names =`		
Dot plots	**dotchart**(*x,...*)	**Scatterplots**	**plot**(*x,...*)
dot labels	`labels =`	*symbol*	`pch =`

REMOVE

axis labels	`ann = FALSE`
axis, tickmarks, and labels	`xaxt = "n"` `yaxt = "n"`
plot box	`bty = "n"`

NOTE: Many of the parameters here can be also be set in **par()**. *See R help for more options.*

ADJUST

allow plotting out of plot region	`xpd = TRUE`
aspect ratio	`asp =`
axis limits	`xlim =, ylim =`
axis lines to match axis limits	`xaxs = "i"`, `yaxs = "i"` *(internal axis calculation)*

ADD TEXT

location

axis labels	`xlab =, ylab =`
subtitle	`sub =`
title	`main =`

style

font face	`font = 1` *(plain)* 2 *(bold)* 3 *(italic)* 4 *(bold italic)*
font family	`family = "serif"` `"sans" "mono"`

size

(magnification factor)

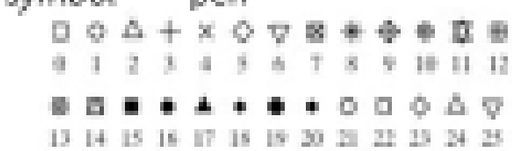
all elements	`cex =`
axis labels	`cex.lab =`
subtitle	`cex.sub =`
tick mark labels	`cex.axis =`
title	`cex.main =`

position

text direction	`las = 1` (horizontal)
justification	`adj = 0 .5 1` (left, center, right)

ADD TO AN EXISTING PLOT

Add new plot	[*any plot function*] (*...*, `add = TRUE`) ex. `barplot(x, add = TRUE)`	**Lines**	**lines** (*x,...*)
		line style	`lty =`
		line width	`lwd =`
		color	`col =`
Axes	**axis** (*side, ...*)	**Points**	**points** (*x,...*)
location	`side = 1 2 3 4` (bottom, left, top, right)	*symbol*	`pch =`
tick mark:			
labels	`labels =`	*color*	`col =`
location	`at =`	*fill color*	`bg =` (pch: 21-25 only)
remove	`tick = FALSE`		
rotate text	`las = 1` (horizontal)		
Axis labels	**mtext** (*text,...*)	**Text**	**text** (*x, y, text,...*)
location	`side = 1 2 3 4` (bottom, left, top, right)	*position (rel. to x,y)*	`pos = 1 2 3 4` (below, left, above, right) (default=center)
lines to skip	`line =` *(from plot region, default = 0)*		
position	`at =` *x or y-coord* (depending on side)	**Title**	**title** (*main,...*)
justification	`adj = 0 .5 1` (left, center, right)	*axis labels*	`xlab =, ylab =`
		subtitle	`sub =`
		title	`main =`



Joyce Robbins, joycerobbins1@gmail.com

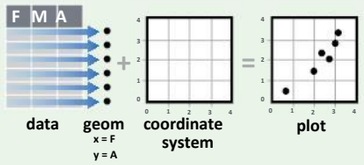
Data Visualization

with ggplot2
Cheat Sheet

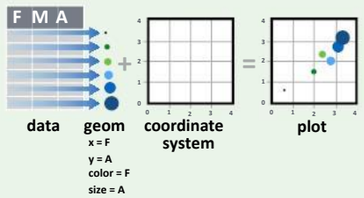
RStudio

Basics

ggplot2 is based on the grammar of graphics, the idea that you can build every graph from the same few components: a data set, a set of geoms—visual marks that represent data points, and a coordinate system.



To display data values, map variables in the data set to aesthetic properties of the geom like size, color, and x and y locations.



Build a graph with qplot() or ggplot()

aesthetic mappings | data | geom

```
qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")
```

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

```
ggplot(data = mpg, aes(x = cty, y = hwy))
```

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(color = cyl)) +
  geom_smooth(method ="lm") +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

Add a new layer to a plot with a geom_*() or stat_*() function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

```
last_plot()
```

Returns the last plot

```
ggsave("plot.png", width = 5, height = 5)
```

Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + geom_dotplot()
x, y, alpha, color, fill

a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group))
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + geom_path(lineend="butt", linejoin="round', linemitre=1)
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900))
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat))
x, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + geom_blank()

f + geom_jitter()
x, y, alpha, color, fill, shape, size

f + geom_point()
x, y, alpha, color, fill, shape, size

f + geom_quantile()
x, y, alpha, color, linetype, size, weight

f + geom_rug(sides = "bl")
alpha, color, linetype, size

f + geom_smooth(model = lm)
x, y, alpha, color, fill, linetype, size, weight

f + geom_text(aes(label = cty))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill

g + geom_violin(scale = "area")
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + geom_bin2d(binwidth = c(5, 0.5))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + geom_area()
x, y, alpha, color, fill, linetype, size

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv")
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also geom_errorbarh())

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + geom_map(aes(map_id = state), map = map) + expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

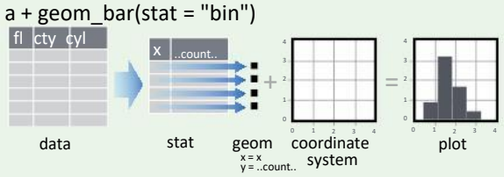
m + geom_contour(aes(z = z))
x, y, z, alpha, colour, linetype, size, weight

m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

m + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size

Stats - An alternative way to build a layer

Some plots visualize a transformation of the original data set. Use a stat to choose a common transformation to visualize, e.g. a + geom_bar(stat = "bin")



Each stat creates additional variables to map aesthetics to. These variables use a common ..name.. syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. stat_bin(geom="bar") does the same as geom_bar(stat="bin")

stat function | layer specific mappings | variable created by transformation

i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100)

geom for layer | parameters for stat

1D distributions

a + stat_bin(binwidth = 1, origin = 10)
  x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + stat_bindot(binwidth = 1, binaxis = "x")
  x, y, | ..count.., ..ncount..
a + stat_density(adjust = 1, kernel = "gaussian")
  x, y, | ..count.., ..density.., ..scaled..

2D distributions

f + stat_bin2d(bins = 30, drop = TRUE)
  x, y, fill | ..count.., ..density..
f + stat_binhex(bins = 30)
  x, y, fill | ..count.., ..density..
f + stat_density2d(contour = TRUE, n = 100)
  x, y, color, size | ..level..

3 Variables

m + stat_contour(aes(z = z))
  x, y, z, order | ..level..
m+ stat_spoke(aes(radius= z, angle = z))
  angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + stat_summary_hex(aes(z = z), bins = 30, fun = mean)
  x, y, z, fill | ..value..
m + stat_summary2d(aes(z = z), bins = 30, fun = mean)
  x, y, z, fill | ..value..

Comparisons

g + stat_boxplot(coef = 1.5)
  x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area")
  x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

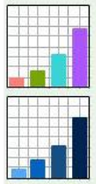
f + stat_ecdf(n = 40)
  x, y | ..x.., ..y..
f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")
  x, y | ..quantile.., ..x.., ..y..
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)
  x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))
  x | ..y..
f + stat_identity()
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5))
  sample, x, y | ..x.., ..y..
f + stat_sum()
  x, y, size | ..size..
f + stat_summary(fun.data = "mean_cl_boot")
f + stat_unique()

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



n <- b + geom_bar(aes(fill = fl))
n

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + scale_fill_manual(
  values = c("skyblue", "royalblue", "blue", "navy"),
  limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
  name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values as visual values
scale_*_manual(values = c()) - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Discrete

n <- b + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
library(RcolorBrewer)
display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

Continuous

o <- a + geom_dotplot(aes(fill = ..x..))

o + scale_fill_gradient(low = "red", high = "yellow")

o + scale_fill_gradient2(low = "red", hight = "blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours = terrain.colors(6))
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

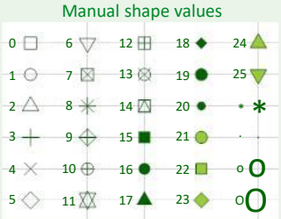
Shape scales

p <- f + geom_point(aes(shape = fl))

p + scale_shape(solid = FALSE)

p + scale_shape_manual(values = c(3:7))
Shape values shown in chart on right

Manual shape values



Size scales

q <- f + geom_point(aes(size = cyl))

q + scale_size_area(max = 6)
Value mapped to area of circle (not radius)

Coordinate Systems

r <- b + geom_bar()

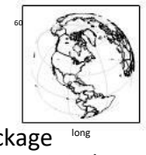
r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.



z + coord_map(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set scales to let axis limits vary across facets

t + facet_grid(y ~ x, scales = "free")
x and y axis limits adjust to individual facets
- "free_x" - x axis limits adjust
- "free_y" - y axis limits adjust

Set labeller to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(. ~ fl, labeller = label_bquote(alpha ^ .(x)))

α^{c} α^{d} α^{e} α^{p} α^{r}

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

s + geom_bar(position = "stack")
Stack elements on top of one another

f + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual width and height arguments

s + geom_bar(position = position_dodge(width = 1))

Labels

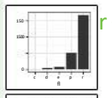
t + ggtitle("New Plot Title")
Add a main title above the plot
t + xlab("New X label")
Change the label on the X axis
t + ylab("New Y label")
Change the label on the Y axis
t + labs(title =" New title", x = "New x", y = "New y")
All of the above

Use scale functions to update legend labels

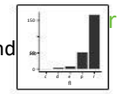
Legends

t + theme(legend.position = "bottom")
Place legend at "bottom", "top", "lef", or "right"
t + guides(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.

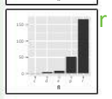
Themes



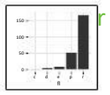
r + theme_bw()
White background with grid lines



r + theme_classic()
White background no gridlines



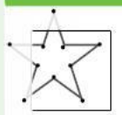
r + theme_grey()
Grey background (default theme)



r + theme_minimal()
Minimal theme

ggthemes - Package with additional ggplot2 themes

Zooming



Without clipping (preferred)

t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)

t + xlim(0, 100) + ylim(10, 20)
t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))