**Cheatsheet**

**The response variable** is also commonly known as the ‘dependent variable’ (Manipulated) and the explanatory variables are sometimes referred to as ‘independent variables’ (or less frequently as ‘predictor variables’).

**Discrete variable example** < - number of children

**Continuous variable example** <- Your height (decimal number)

**Bar charts** have a categorical variable on the horizontal axis

**Histograms** have a quantitative variable on the horizontal axis.

Histograms provide a view of the data density. Higher bars represent where the data are relatively more common. For instance, there are many more loans with rates between 5% and 10% than loans with rates between 20% and 25% in the data set. The bars make it easy to see how the density of the data changes relative to the interest rate.

Histograms that have one, two, or three prominent peaks are called unimodal, bimodal, and multimodal, respectively.

**apropos("mean")** list all functions containing a specified character string.

RSiteSearch("regression") This function allows you to access the https://www.r-project.org/search.html search engine directly from the Console with the results displayed in your web browser

**args()**

**round(x2, digits = 3)** -> Round to 3 decimals

dir.create(‘data’)-> create a directory called ‘data’

dir.create(file.path("testdir2","testdir3"), recursive = TRUE)

getwd()

setwd()

file.exists()

file.info()

file.rename(from,to)

file.copy(from, to, overwrite = recursive, recursive = FALSE, copy.mode = TRUE, copy.date = FALSE)

file.create()

file.path()

dir

list.files()

ls() -> list all objects in your local workspace

pi:10

1:20 -> all numbers from 1 to 20

seq(1,20)

seq(0,10, by=0.5)

seq(5,10, length=30)

seq\_along(my\_seq) -> note that my\_seq is a variable that we created

length()

seq(along.with = my\_seq)

rep(0, times = 40) -> creating a vector that contains 40 zeros

rep(c(0, 1, 2), times = 10)-> vector to contain 10 repetitions of the vector (0, 1, 2)

rep(c(0, 1, 2), each = 10) -> vector to contain 10 zeros, then 10 ones, then 10 twos

xfun::session\_info()

mydata <- c(2, 4, 1, 6, 8, 5, NA, 4, 7)

mean(mydata, na.rm = TRUE) use of the mean() function to calculate the mean without this missing value

**Vectors come in two different flavors: atomic vectors and lists**. An **atomic vector** contains exactly one data type, | whereas a **list** may contain multiple data types. We'll explore atomic vectors further before we get to lists.

num\_vect <- c(0.5, 55, -10, 6)

num\_vect[3] # extract the 3rd value, in this case -10

num\_vect[c(1, 3)] # extract the 1st and 3rd value, in this case 0.5 and -10

vector(mode=”logical”, length= 10)

tf <- num\_vect < 1

y <- rnorm(1000) create a vector containing 1000 draws from a standard normal distribution

z <- rep(NA, 1000)

my\_data <- sample(c(y, z), 100) let's select 100 elements at random from these 2000 values (combining y and z)

y <- x[!is.na(x)]

vect <- c(foo = 11, bar = 2, norf = NA)

vect[c("foo", "bar")]

vect["bar"]

names(vect2) <- c("foo", "bar", "norf")

**dim(my\_vector)<- c(4,5)** gave my\_vector 4 rows and 5 columns essentially turning it into a matrice

(TRUE == TRUE) | (TRUE == FALSE) -> true The `|` in the middle states that AT LEAST ONE of the pieces is TRUE. Your job | is to figure out if that is an accurate statement

my\_char <- c("My", "name", "is")

paste(my\_char, collapse = " ")

my\_name <- c(my\_char, "Daniel Lachaud") - >we used the paste() function to collapse the elements of a single character vector.

paste("Hello", "world!", sep = " ") join two character vectors that are each of length 1 (i.e. join two words)

**paste(1:3, c("X", "Y", "Z"), sep = "")** -> returns "1X" "2Y" "3Z”

**paste(LETTERS, 1:4, sep = "-")** -> Vector recycling! Returns: "A-1" "B-2" "C-3" "D-4" "E-1" "F-2" "G-3" "H-4" "I-1" "J-2" "K-3" "L-4" "M-1" "N-2" "O-3" "P-4" "Q-1" "R-2" "S-3" "T-4" "U-1" "V-2" "W-3" "X-4" "Y-1" "Z-2"

**Matrices and data Frames**

The main difference, as you'll see, is that **matrices can only contain a single class of data**, while **data frames can consist of many different classes of data**.

**my\_matrix2 <- matrix(1:20, 4, 5)** create a matrix | containing the same numbers (1-20) and dimensions (4 rows, 5 columns)

**cbind(patients, my\_matrix) ->** cbind() function to combine columns -

**my\_data <- data.frame(patients, my\_matrix) ->** data.frame() function allowed us to store our character vector of names right alongside our matrix of numbers.

**cnames <- c("patient","age", "weight", "bp", "rating", "test") colnames(my\_data)<-cnames ->** set the `colnames` attribute for a data frame.

**which(ints>7) ->** find the indices of ints that are greater than 7

**any(ints<0) ->** to see if any of the elements of ints are less than zero (returns true or false)

**all(ints>0) ->** to see if all of the elements of ints are greater than zero (returns true or false)

**List**

**a.list <- list("animal"=c('dog', 'cat', 'mouse'), "pet"=c(TRUE, TRUE, FALSE)); a.list**

**a.list$animal**

**Note that 'plot' is short for scatterplot.**

**data(cars)** -> load the data frame cars

plot(x=cars$speed, y=cars$dist)

xlab=”Speed”,ylab=”Distance”,main=”My Title”, sub=”My Subtitle”, col=2

Use xlim = c(10, 15) -> limiting the x-axis to 10 through 15

pch = 2 -> change the shape of the symbols in the plot see help page for points (?points)

**Boxplot**

boxplot(formula=mpg ~ cyl, data=mtcars)

flowers$nitrogen <- factor(flowers$nitrogen,

levels = c("low", "medium", "high"))

boxplot(weight ~ nitrogen, data = flowers,

ylab = "weight (g)", xlab = "nitrogen level")

library(vioplot)

vioplot(weight ~ nitrogen, data = flowers,

ylab = "weight (g)", xlab = "nitrogen level",

col = "lightblue")

**Codplot**

coplot(flowers ~ weight|nitrogen \* treat, data = flowers,

panel = **function**(x, y, ...) {

points(x, y, ...)

abline(lm(y ~ x), col = "blue")})

**Histogram**

When looking at a single variable, histograms are a useful tool. hist() is the associated R function. Like plot(), hist() is best used by just passing in a single vector.

dens <- density(flowers$height)

hist(flowers$height, breaks = brk, main = "petunia height",

freq = FALSE)

lines(dens)

ScatterPlot

par(mar = c(4.1, 4.4, 4.1, 1.9), xaxs = "i", yaxs = "i")

plot(flowers$weight, flowers$shootarea,

xlab = "weight (g)",

ylab = expression(paste("shoot area (cm"^"2",")")),

xlim = c(0, 30), ylim = c(0, 200), bty = "l",

las = 1, cex.axis = 0.8, tcl = -0.2,

pch = 16, col = "dodgerblue1", cex = 0.9)

text(x = 28, y = 190, label = "A", cex = 2)

pch value

Calendar

Description automatically generated

Install Packages-

install.packages('remotes', dependencies = TRUE)

update.packages(ask = FALSE)

Table

Description automatically generatedlosqrt(exp

**Table

Description automatically generated**

| **Operator** | **Technical Description** | **What it means** | **Example** |
| --- | --- | --- | --- |
| && | Logical AND | Both conditions must be met | if(cond1 == test && cond2 == test) |
| || | Logical OR | Either condition must be met | if(cond1 == test || cond2 == test) |
| < | Less than | X is less than Y | if(X < Y) |
| > | Greater than | X is greater than Y | if(X > Y) |
| <= | Less than or equal to | X is less/equal to Y | if(X <= Y) |
| >= | Greater than or equal to | X is greater/equal to Y | if(X >= Y) |
| == | Equal to | X is equal to Y | if(X == Y) |
| != | Not equal to | X is not equal to Y | if(X != Y) |

**GGPLOT**

We know from the “final figure” that we want the **variable shootarea on the y axis (response/dependent variable**) and **weight on the x axis (explanatory/independent variable).**

**Data, mapping, geom**

classic <- ggplot(aes(x = weight, y = shootarea, colour = nitrogen), data = flower) +

geom\_point(aes(shape = nitrogen), size = 2, alpha = 0.6) +

geom\_smooth(method = "lm", se = FALSE) +

facet\_grid(block ~ treat) +

xlab("Weight of flower (g)") +

ylab(bquote("Area of shoot"~(cm^2))) +

labs(shape = "Nitrogen\nConcentration", colour = "Nitrogen\nConcentration",

caption = "Regression assumptions are unvalidated") +

*# Classic theme*

theme\_classic()+

coord\_cartesian(xlim = c(10, 15), ylim = c(50, 150))+

geom\_text(x = 22, y = 5, label = "A. Artist", colour = "grey", family = "serif") +

*# Changing the angle of the axis text*

theme(axis.text.x=element\_text(angle = 45, vjust = 0.5))

**Plotting multiple ggplots**

While we’ve made multiples of the same figure, what if we wanted to take two completely different figures and plot them together in the same frame?

install.packages("patchwork")

library(patchwork)

rbook\_figure / final\_figure

**Functions**

Everything that exists is an object. 2. Everything that happens is a function call.

boring\_function <- function(x) {

x

}

remainder <- function(num, divisor=2) {

num %% divisor

# Remember: the last expression evaluated will be returned!

}

evaluate(function(x){x[length(x)]}, c(8, 4, 0))

Function (syntax)

nameOfFunction <- function(argument1, argument2, ...) {expression}

Function Example

multiply\_columns <- **function**(x, y) {

temp\_var <- x \* y

**if** (any(is.na(temp\_var))) {

warning("The function has produced NAs")

return(temp\_var)

} **else** {

return(temp\_var)

}

}

Function(example)

eggs <- TRUE

n.milk <- ifelse(eggs == TRUE, yes = 6, no = 1)

Function(example)

good.day <- **function**(code.working, day) {

**if** (code.working == TRUE && day == "Friday") {

"BEST. DAY. EVER. Stop while you are ahead and go to the pub!"

} **else** **if** (code.working == FALSE && day == "Friday") {

"Oh well, but at least it's Friday! Pub time!"

} **else** **if** (code.working == TRUE && day != "Friday") {

"So close to a good day... shame it's not a Friday"

} **else** **if** (code.working == FALSE && day != "Friday") {

"Hello darkness."

}

}

Loops

**for** (i **in** 1:5) {

print(i)

}

**for** (i **in** 1:5) {

print(i + 1)

}

temp <- list()

**for** (i **in** 1:(ncol(city) - 1)) {

temp[[i]] <- multiply\_columns(x = city[, i], y = city[, i + 1])

}

i <- 0

**while** (i <= 4) {

i <- i + 1

print(i)

}

lapply / sapply

**lapply()** returns a vector/list of length one. **sapply()** instead returns a matrix and tries to simplify the result.

**lapply(flag\_colors, sum)**

**sapply(flag\_colors, sum)**

**flag\_shapes <- flags[, 19:23 -> extract columns 19 through 23 from the flags dataset and store the result in a new data**

**lapply(unique\_vals, function(elem) elem[2]) -> return a list containing the second item from each element of the unique\_vals list**

**vapply and tapply**

**tapply(flags$population, flags$landmass, summary)**

**Dates and Times**

**dates are stored as the number of days since 1970-01-01 and times are stored as either the number of seconds since 1970-01-0**

**d2 <- as.Date("1969-01-01")**

**d1 <- Sys.Date()**

**t1 <- Sys.time()**

**t2 <- as.POSIXlt(Sys.time())**

**str(unclass(t2))**

**weekdays(d1)**

**months(t1)**

**t3 <- "October 17, 1986 08:24"**

**t4 <- strptime(t3, "%B %d, %Y %H:%M")**

**there are a number of operations that you can perform on dates and times, including arithmetic operations (+ and -) and comparisons (<, ==, etc.)**

**difftime(Sys.time(), t3, units = 'days') -> to find the amount of time in DAYS that has passed since you created t3**

**Type of object in R:** Vectors, Matrices, Factors, Arrays, List, DataFrame

Simple linear modelling

**Graphical user interface

Description automatically generated**

**smoke\_lm <- lm(mortality ~ smoking, data = smoke)**

**smoke\_risk\_lm <- lm(mortality ~ risk.group, data = smoke)**

**anova(smoke\_risk\_lm)**

ggplot(mapping = aes(x = smoking, y = mortality), data = smoke) +

geom\_point() +

geom\_smooth(method = "lm", se = TRUE)

**Convert a dataframe into a tible**

**as\_tibble(nameofdataframe)**

**or**

**read\_csv(your\_file\_path)**

**Convert a tible into a dataframe**

**as.data.frame(nameoftible)**

**Tidy data**

1. **Each variable must have its own column**
2. **Each observation has its own row**
3. **Each value has its own cell**

**pivot\_longer** a function of tydr to flit data frame.

**Dplyr**

**Text

Description automatically generated**

**Syntax df <- verb(df, argument)**

**Examples:** nov\_dec <- filter(flights, month %in% c(11, 12)) filter(flights, month == 1, day == 1)

filter(flights, month == 11 | month == 12)

[filter](https://rdrr.io/r/stats/filter.html)(flights, !(arr\_delay > 120 | dep\_delay > 120))

[filter](https://rdrr.io/r/stats/filter.html)(flights, arr\_delay <= 120, dep\_delay <= 120)

**group\_by(df, color)**

**arrange(flights, desc(dep\_delay))**

**summarize(df, total =sum(value))**

**starts\_with() / ends\_with() / contains() / everything() / - columnName**

**Qr code

Description automatically generated with medium confidence**

**Date:**

**mutate**(chicago, month = **as.POSIXlt**(date)$mon + 1)

**difftime(maxDate, mindate, units =”weeks”)**

**mean(tlc$amount, na.rm =TRUE)**

**Markdown**

library(rmarkdown)

render('my\_first\_rmarkdown.Rmd', output\_format = 'html\_document')

*# alternatively if you don't want to load the rmarkdown package*

rmarkdown::render('my\_first\_rmarkdown.Rmd', output\_format = 'html\_document')

| **Goal** | **R markdown** | **output** |
| --- | --- | --- |
| bold text | \*\*mytext\*\* | **mytext** |
| italic text | \*mytext\* | *mytext* |
| strikethrough | ~~mytext~~ |  |
| superscript | mytext^2^ | mytext2 |
| subscript | mytext~2~ | mytext2 |

**To create blank space:**

These &nbsp; &nbsp; &nbsp; data were

**Heading**

You can add headings and subheadings to your R markdown document by using the # symbol at the beginning of the line. You can decrease the size of the headings by simply adding more # symbols. For example:

# Benthic Biodiversity experiment

## Benthic Biodiversity experiment

### Benthic Biodiversity experiment

**Comments**

If you want to include a comment in your R markdown document outside a code chunk which won’t be included in the final rendered document then enclose your comment between <!-- and -->.

*<!--*

*this is an example of how to format a comment using R markdown.*

*-->*

Lists

If you want to create a bullet point list of text you can format an unordered list with sub items. Notice that the sub-items need to be indented.

- item 1

- item 2

+ sub-item 2

+ sub-item 3

- item 3

- item 4

1. item 1

2. item 2

+ sub-item 2

+ sub-item 3

3. item 3

4. item 4

Include images

***```{r, echo=FALSE, fig.align='center', out.width='50%'}***

***library(knitr)***

***include\_graphics("images/Cute\_grey\_kitten.jpg")***

***```***

**Links**

You can include a text for your clickable *[*link*](https://www.worldwildlife.org)*

**Code Trunks**

There are a large number of chunk options documented [here](https://yihui.name/knitr/options) with a more condensed version [here](https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf). Perhaps the most commonly used are summarised below with the default values shown.

| **Chunk option** | **default value** | **Function** |
| --- | --- | --- |
| echo | echo=TRUE | If FALSE, will not display the code in the final document |
| results | results='markup' | If ‘hide’, will not display the code’s results in the final document. If ‘hold’, will delay displaying all output pieces until the end of the chunk. If ‘asis’, will pass through results without reformatting them. |
| include | include=TRUE | If FALSE, will run the chunk but not include the chunk in the final document. |
| eval | eval=TRUE | If FALSE, will not run the code in the code chunk. |
| message | message=TRUE | If FALSE, will not display any messages generated by the code. |
| warning | warning=TRUE | If FALSE, will not display any warning messages generated by the code. |

If we wanted to only display the output of our R code (just the summary statistics for example) and not the code itself in our final document we can use the chunk option echo=FALSE

***```{r, summary-stats, echo=FALSE}***

***x <- 1:10 # create an x variable***

***y <- 10:1 # create a y variable***

***dataf <- data.frame(x = x, y = y)***

***summary(dataf)***

***```***

**Adding Tables**

***```{r, kable-table2}***

***kable(iris[1:10,], digits = 0, align = 'c',***

***col.names = c('sepal length', 'sepal width',***

***'petal length', 'petal width', 'species'))***

***```***

**Inline R Code**

Morphological characteristics (variable names: `r names(iris)[1:4]`) were measured from

`r nrow(iris)` \*Iris sp.\* plants from `r length(levels(iris$Species))` different

species. The mean Sepal length was `r round(mean(iris$Sepal.Length), digits = 2)` mm.

**Problem-**

**When I load a package in my R markdown document my rendered output contains all of the startup messages and/or warnings.**

**Solution:**

***``{r, global\_options, include=FALSE}***

***knitr::opts\_chunk$set(message=FALSE, warning=FALSE, tidy.opts=list(width.cutoff=60))***

***suppressPackageStartupMessages(library(ggplot2))***

***```***

**Conversion**

fctr stands for factors, which R uses to represent categorical variables with fixed possible values.

**df$ID <- as.factor(df$ID)**

**The `unclass` method in R** allows you to view how a particular R object is stored.

**The function n()** returns the number of observations in a current group. A closed function to n() is **n\_distinct(),** which count the number of unique values.

**Mode:** The most frequently observed value in the data set

Contingency table <- table(var1, var2)

Read\_csv()

logs <- **read\_csv**("data/2016-07-19.csv.bz2", n\_max = 10)

Parsed with column specification:

**cols**(

date = **col\_date**(format = ""),

time = **col\_time**(format = ""),

size = **col\_double**(),

r\_version = **col\_character**(),

r\_arch = **col\_character**(),

r\_os = **col\_character**(),

package = **col\_character**(),

version = **col\_character**(),

country = **col\_character**(),

ip\_id = **col\_double**()

)

Pie chart basics R-

Syntax: pie(x, labels, main, col, clockwise, init.angle)

**Example:**

g <- c(2,4,5,7,12,14,16,18,21,24,30,35)

pie(dataF$Cost, labels = dataF$Month, main = "Electricity Usage", col = rainbow(length(g)), clockwise = TRUE, init.angle = 0)

**Statistics**

**t.test(trees$Height, mu = 70, alternative = "greater")**

You can also change the confidence level used for estimating the confidence intervals using the argument conf.level = 0.99. If specified in this way, 99% confidence intervals would be estimated.

**wilcox.test(trees$Height, mu = 70)**

Perhaps the simplest way to assess normality is the ‘quantile-quantile plot’. This graph plots the ranked sample quantiles from your distribution against a similar number of ranked quantiles taken from a normal distribution. If your data are normally distributed then the plot of your data points will be in a straight line.

qqnorm(trees$Height)

qqline(trees$Height, lty = 2)

To construct a Q-Q plot you need to use both the qqnorm() and qqline() functions. The lty = 2 argument changes the line to a dashed line.

If you insist on performing a specific test for normality you can use the function shapiro.test() which performs a Shapiro – Wilks test of normality.

shapiro.test(trees$Height)

*##*

*## Shapiro-Wilk normality test*

*##*

*## data: trees$Height*

*## W = 0.96545, p-value = 0.4034*

In the example above, the p value = 0.4 which suggests that there is no evidence to reject the null hypothesis and we can therefore assume these data are normally distributed.

To perform a two-sample t test

**t.test(atmos$moisture ~ atmos$treatment)**

To test whether the assumption of equal variances is valid you can perform an F-test on the ratio of the group variances using the var.test() function.

**var.test(atmos$moisture ~ atmos$treatment)**

Proportion Test()

| **before** | **after** |
| --- | --- |
| would buy | 45 | 71 |
| would not buy | 35 | 32 |

<https://intro2r.com/one-and-two-sample-tests.html>

buy <- c(45,71) *# creates a vector of positive outcomes*

total <-c((45 + 35), (71 + 32)) *# creates a vector of total numbers*

prop.test(buy, total) *# perform the test*

*##*

*## 2-sample test for equality of proportions with continuity correction*

*##*

*## data: buy out of total*

*## X-squared = 2.598, df = 1, p-value = 0.107*

*## alternative hypothesis: two.sided*

*## 95 percent confidence interval:*

*## -0.27865200 0.02501122*

*## sample estimates:*

*## prop 1 prop 2*

*## 0.5625000 0.6893204*

**Proportion**

**prop.test(Qtybrooklyn, Qtyall, p=0.40, alternative=”two.sided”, conf.level=0.95)**

**Mean**

**t.test(Airbnb$price, mu =150, alternative = “greater”, conf.level =0.95)**

**reject the null Hypothesis is p-value < 0.05**

**Confidence Interval**

**t.test(Airbnb$price, conf.level =0.95) used to calculate the CI only**

You can then perform a Chi-square test to test whether the number of cat owners buying the cat food is independent of the advertising campaign using the chisq.test() function. In this example the only argument is our matrix of counts.

**chisq.test(buyers)**

Correlation

**cor(trees, use = "complete.obs")**

*## Girth Height Volume*

*## Girth 1.0000000 0.5192801 0.9671194*

*## Height 0.5192801 1.0000000 0.5982497*

*## Volume 0.9671194 0.5982497 1.000000*

The function cor() will return the correlation coefficient of two variables, but gives no indication whether the coefficient is significantly different from zero. To do this you need to use the function cor.test().

**cor.test(trees$Height, trees$Volume)**

Two non-parametric equivalents to Pearson correlation are available within the cor.test() function; Spearman’s rank and Kendall’s tau coefficient. To use either of these simply include the argument method = "spearman" or method = "kendall" depending on the test you wish to use. For example

**cor.test(trees$Height, trees$Volume, method = "spearman")**

POINT ESTIMATES AND SAMPLING VARIABILITY

# 1. Create a set of 250 million entries, where 88% of them are "support"

# and 12% are "not".

pop size <- 250000000

possible entries <- c(rep("support", 0.88 \* pop size), rep("not", 0.12 \* pop size))

# 2. Sample 1000 entries without replacement.

sampled entries <- sample(possible entries, size = 1000)

# 3. Compute p-hat: count the number that are "support", then divide by

# the sample size.

sum(sampled entries == "support") / 1000

Git

git config --global user.name "Daniel Lachaud"

git config --global user.email [dlachaud@gmail.com](mailto:dlachaud@gmail.com)

git config --global -e

mkdir foldername to create a new directory

navigate to new directory and then

git init

-add file to directory. With linux commands or simply use windows explorer

-git status

-git add filename to add file to stage area – necessary for all new files

-git commit -m “message of choice”

-git rm filename to remove file from both local and staging area and then commit the changes with git commit