Barcelona R

Introduction to Data Manipulation

Nicolas Attalides

Workshop Setup:

Wi-Fi

- Network Name: N/A
- Password: N/A

Packages

- tidyverse

Resources

- R (version 4.0.1)
- RStudio (version 1.3.959)

What is tidyverse?

Tidyverse is a collection of R packages that are designed for data science tasks, more specifically for data manipulation, transformation, exploration and visualisation.

These packages share a common design philosophy and contain functions that are consistent and uniform in coding style.

You can read more at https://www.tidyverse.org/

Topics

- Workshop aim:

Learn how to do data manipulations using tidyverse packages.

- Topics:
  - Learn the "verbs" with dplyr
  - Improve your workflow with %>%
  - Simple string manipulation using stringr

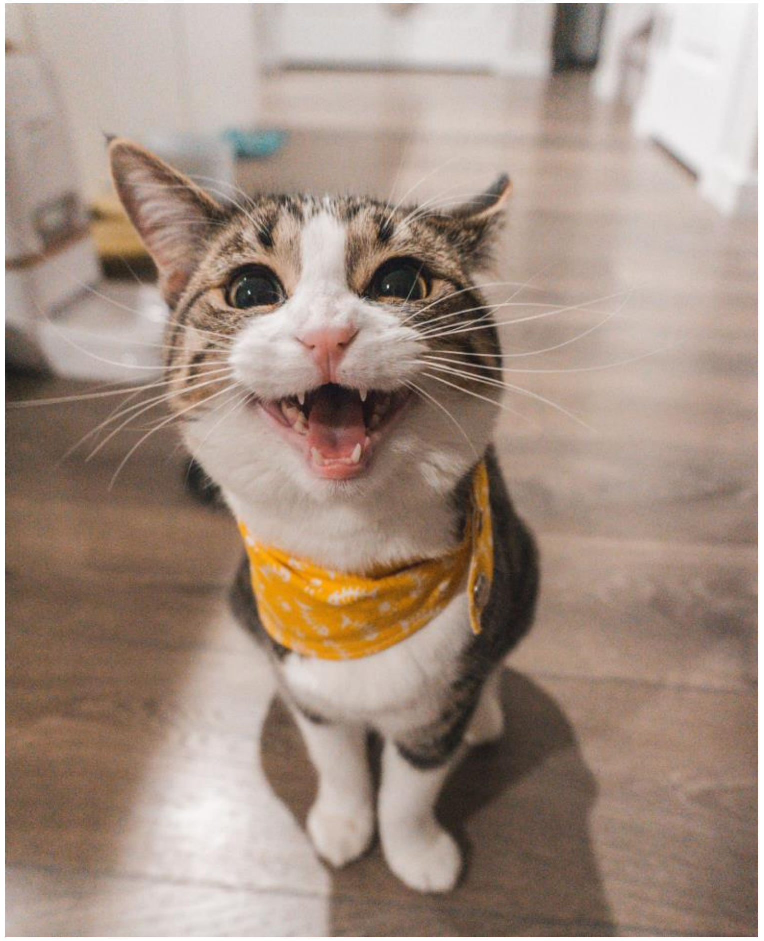
Learn the "verbs" with dplyr

One of the most commonly used R packages when dealing with data manipulations is **{dplyr}**. It is very powerful in handling tabular data such as data frames and is easy to use through "verb" functions. You can use **{dplyr}** to:

- **Select** columns from your data
- **Filter** your data to keep the rows that meet some conditions
- **Arrange** your data in some order
- **Mutate** your data and create new columns
- **Group** and **summarise** your data

```
library(tidyverse)

View(starwars)
```

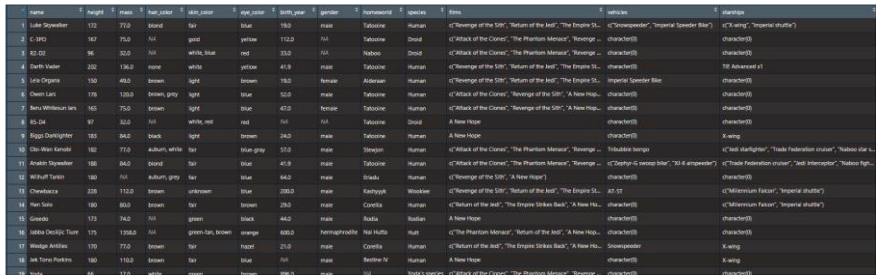
Live Coding Example 1



Use the starwars dataset from the **{dplyr}** package to:

1. Select the columns: “name”, “height”, “mass”, “species”.
2. Filter the rows to keep only those characters that are greater than or equal to 175cm.
3. Filter the rows to keep only the “Human” characters.
4. Arrange the rows according to descending “mass” values.
5. Who is the character on the first row?

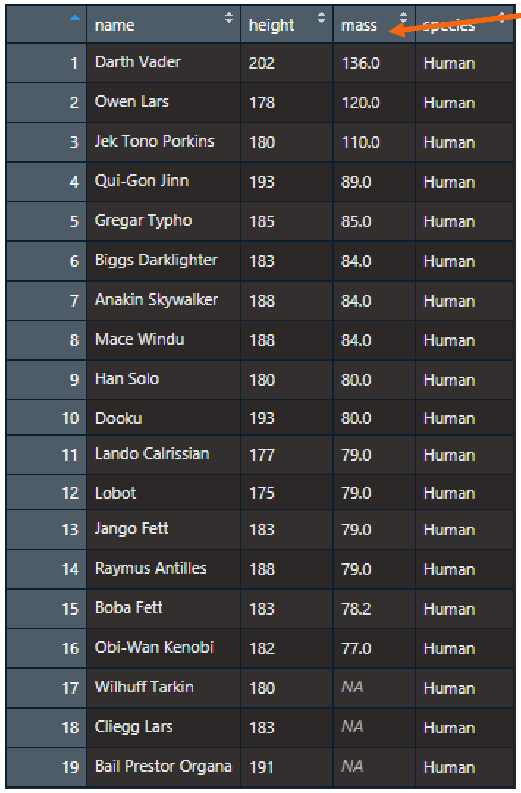
Live Coding Example 1



	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Empire St...	c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")
2	C-3PO	167	75.0	NA	gold	yellow	112.0	NA	Tatooine	Droid	c("Attack of the Clones", "The Phantom Menace", "Revenge ...	character(0)	character(0)
3	R2-D2	96	32.0	NA	white, blue	red	33.0	NA	Naboo	Droid	c("Attack of the Clones", "The Phantom Menace", "Revenge ...	character(0)	character(0)
4	Darth Vader	202	136.0	none	white	yellow	41.9	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Empire St...	character(0)	TIE Advanced x1
5	Leia Organa	150	49.0	brown	light	brown	19.0	female	Alderaan	Human	c("Revenge of the Sith", "Return of the Jedi", "The Empire St...	Imperial Speeder Bike	character(0)
6	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New Hop...	character(0)	character(0)
7	Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New Hop...	character(0)	character(0)
8	R5-D4	97	32.0	NA	white, red	red	NA	NA	Tatooine	Droid	A New Hope	character(0)	character(0)
9	Biggs Darklighter	183	84.0	black	light	brown	24.0	male	Tatooine	Human	A New Hope	character(0)	X-wing
10	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male	Stewjon	Human	c("Attack of the Clones", "The Phantom Menace", "Revenge ...	Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo star s...
11	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male	Tatooine	Human	c("Attack of the Clones", "The Phantom Menace", "Revenge ...	c("Zephyr-G swoop bike", "XJ-6 airspeeder")	c("Trade Federation cruiser", "Jedi Interceptor", "Naboo figh...
12	Wilhuff Tarkin	180	NA	auburn, grey	fair	blue	64.0	male	Eriadu	Human	c("Revenge of the Sith", "A New Hope")	character(0)	character(0)
13	Chewbacca	228	112.0	brown	unknown	blue	200.0	male	Kashyyyk	Wookiee	c("Revenge of the Sith", "Return of the Jedi", "The Empire St...	AT-ST	c("Millennium Falcon", "Imperial shuttle")
14	Han Solo	180	80.0	brown	fair	brown	29.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A New Ho...	character(0)	c("Millennium Falcon", "Imperial shuttle")
15	Greedo	173	74.0	NA	green	black	44.0	male	Rodia	Rodian	A New Hope	character(0)	character(0)
16	Jabba Desilijic Tiure	175	1358.0	NA	green-tan, brown	orange	600.0	hermaphrodite	Nal Hutta	Hutt	c("The Phantom Menace", "Return of the Jedi", "A New Hop...	character(0)	character(0)
17	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A New Ho...	Snowspeeder	X-wing
18	Jek Tono Porkins	180	110.0	brown	fair	blue	NA	male	Bestine IV	Human	A New Hope	character(0)	X-wing
19	Yoda	66	17.0	white	green	brown	896.0	male	NA	Yoda's species	c("Attack of the Clones", "The Phantom Menace", "Revenge ...	character(0)	character(0)

selected columns

arranged rows



	name	height	mass	species
1	Darth Vader	202	136.0	Human
2	Owen Lars	178	120.0	Human
3	Jek Tono Porkins	180	110.0	Human
4	Qui-Gon Jinn	193	89.0	Human
5	Gregar Typho	185	85.0	Human
6	Biggs Darklighter	183	84.0	Human
7	Anakin Skywalker	188	84.0	Human
8	Mace Windu	188	84.0	Human
9	Han Solo	180	80.0	Human
10	Dooku	193	80.0	Human
11	Lando Calrissian	177	79.0	Human
12	Lobot	175	79.0	Human
13	Jango Fett	183	79.0	Human
14	Raymus Antilles	188	79.0	Human
15	Boba Fett	183	78.2	Human
16	Obi-Wan Kenobi	182	77.0	Human
17	Wilhuff Tarkin	180	NA	Human
18	Cliegg Lars	183	NA	Human
19	Bail Prestor Organa	191	NA	Human

filtered rows

Who is the character on the first row?

Live Coding Example 1

```r
# Select columns
df <- select(starwars, name, height, mass, species)

# Filter rows by height condition
df <- filter(df, height >= 175)

# Filter rows by species condition
df <- filter(df, species == "Human")

# Arrange rows by descending mass
df <- arrange(df, desc(mass))
```

Mutate your data

Very often you will want to create new columns from your existing data. The function **`mutate()`** in the **{dplyr}** package can be used to do exactly this task.

You can actually create multiple columns in a single function call.

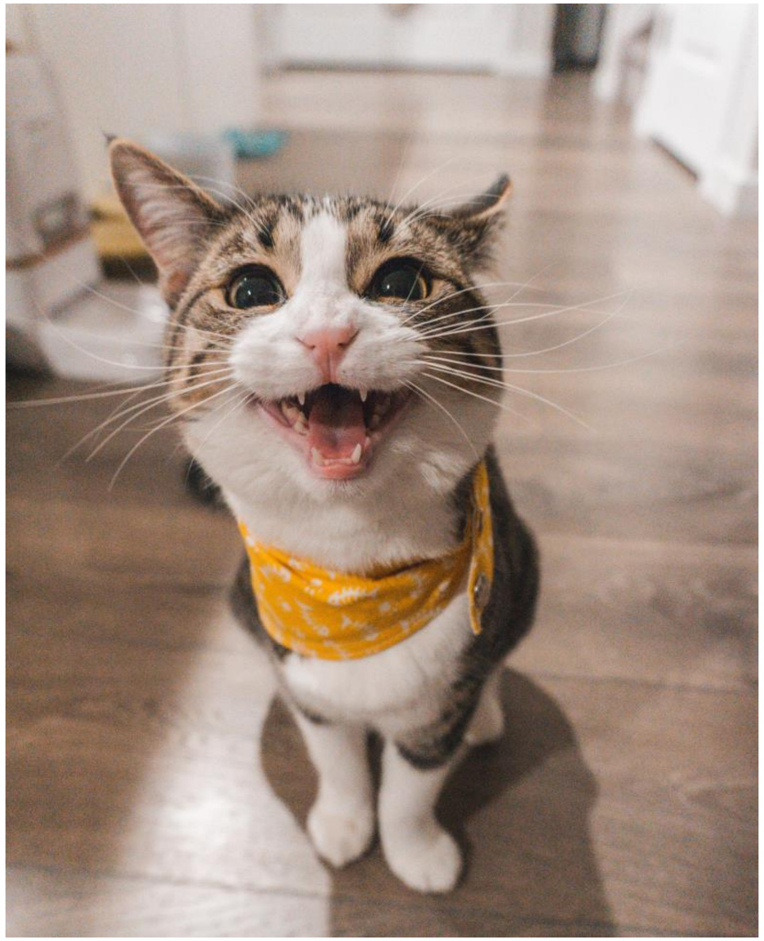
```
# Create a column for height in metres
df <- mutate(starwars, height_m = height/100)
```

Group and summarise your data

Another very common task is to group your data by a column (or more than one column) and then create summarised values for the grouped data. The functions **`group_by()`** and **`summarise()`** in the **{dplyr}** package make it very easy to do these transformations.

```
# Find the min/mean/max mass value for each species category
df <- summarise(group_by(starwars, species),
                min_mass = min(mass, na.rm = TRUE),
                mean_mass = mean(mass, na.rm = TRUE),
                max_mass = max(mass, na.rm = TRUE))
```

Live Coding Example 2



Use the starwars dataset to:

1. Remove the columns “films”, “vehicles”, “starships” from the data.
2. Remove rows that have missing mass values.
3. Calculate the Body Mass Index (BMI) for each character*.
4. Arrange the rows by descending BMI … who do you think is at the top?
5. Find the median BMI value for each gender category.

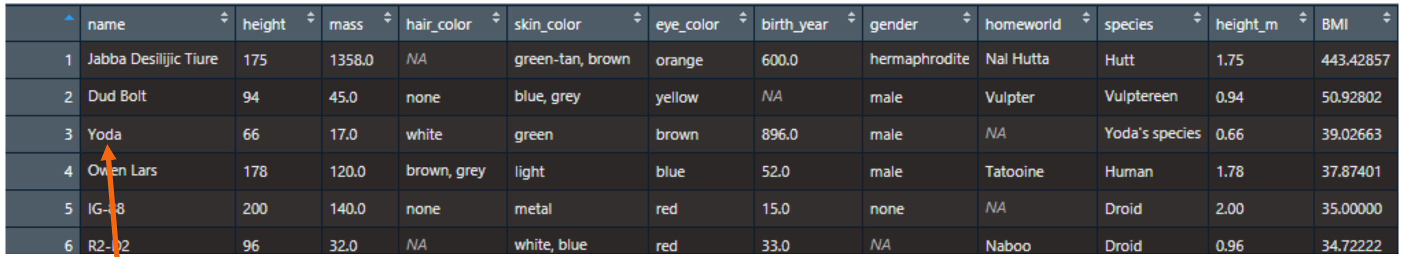
*BMI = weight (kg) / height^2 (m)

Use minus sign “-” to remove columns

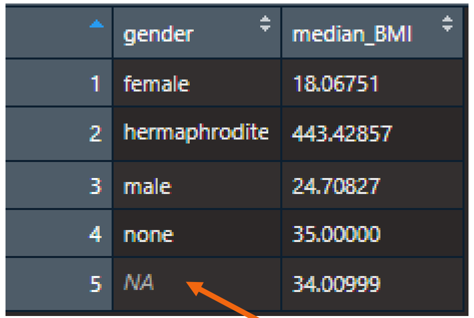
Live Coding Example 2

Who is the character

with the highest BMI?



	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	height_m	BMI
1	Jabba Desilijic Tiure	175	1358.0	NA	green-tan, brown	orange	600.0	hermaphrodite	Nal Hutta	Hutt	1.75	443.42857
2	Dud Bolt	94	45.0	none	blue, grey	yellow	NA	male	Vulpter	Vulptereen	0.94	50.92802
3	Yoda	66	17.0	white	green	brown	896.0	male	NA	Yoda's species	0.66	39.02663
4	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	Tatooine	Human	1.78	37.87401
5	IG-88	200	140.0	none	metal	red	15.0	none	NA	Droid	2.00	35.00000
6	R2-D2	96	32.0	NA	white, blue	red	33.0	NA	Naboo	Droid	0.96	34.72222



	gender	median_BMI
1	female	18.06751
2	hermaphrodite	443.42857
3	male	24.70827
4	none	35.00000
5	NA	34.00999

Investigate data quality

Live Coding Example 2

```r
# Select columns
df <- select(starwars, -films, -vehicles, -starships)

# Filter rows that have missing mass
df <- filter(df, !is.na(mass))

# Create columns: height in metres and the Body Mass Index (BMI)
df <- mutate(df, height_m = height/100, BMI = mass / (height_m)^2)

# Arrange rows according to descending “BMI” values
df <- arrange(df, desc(BMI))

# Calculate the median BMI value for each gender
df <- summarise(group_by(df, gender), median_BMI = median(BMI))
```

Summary of {dplyr} "verb" functions

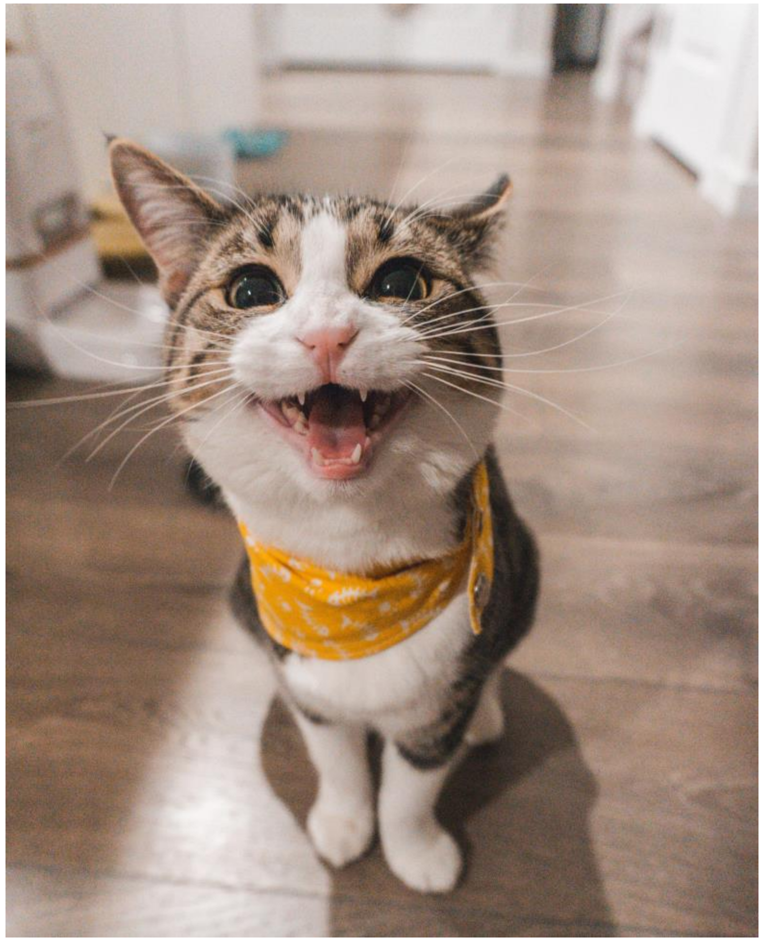
Function	Description
select	Select columns by name
filter	Filter rows that meet a condition
arrange	Arrange rows to some order
mutate	Mutate data to create new columns
group_by	Group data by columns
summarise	Summarise data to values

Improve your workflow with

A package that has changed the way we write R code is called **{magrittr}**. It has significantly improved the readability and workflow of code by introducing the "pipe" operator. It acts as a "then" operation where we can pass data from one function to another function very easily.

Fun fact: The package name is inspired by the famous artist René Magritte. One of his work, a pipe, has the text "this is not a pipe" as a caption … this is where the **{magrittr}** package gets its image.

Live Coding Example 3



Repeat Example 1 using the pipe operator from the **{magrittr}** package.

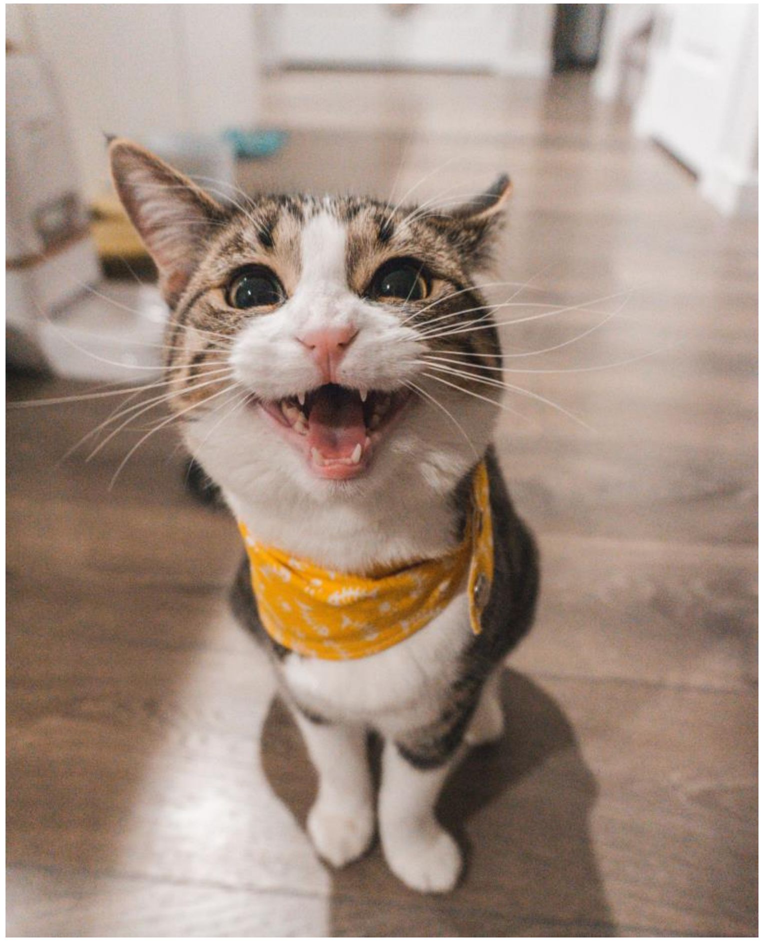
1. Select the columns: “name”, “height”, “mass”, “species” THEN filter the rows to keep only those characters that are greater than or equal to 175cm THEN filter the rows to keep only the human characters THEN arrange the rows according to descending “mass” values.

Live Coding Example 3

```
library(magrittr)


# Pipe each data manipulation operation to the next one
df <- starwars %>%
  select(name, height, mass, species) %>%
  filter(height >= 175) %>%
  filter(species == "Human") %>%
  arrange(desc(mass))
```

Try **CTRL+SHIFT+M** (Windows) **CMD+SHIFT+M** (Mac) and see what happens

Live Coding Example 4



Repeat Example 2 using the pipe operator from the **{magrittr}** package.

1. Remove the columns "films", "vehicles", "starships" from the data THEN remove rows that have missing mass values THEN calculate the Body Mass Index (BMI) for each character THEN arrange the rows by descending BMI THEN find the median BMI value for each gender category.

Live Coding Example 4

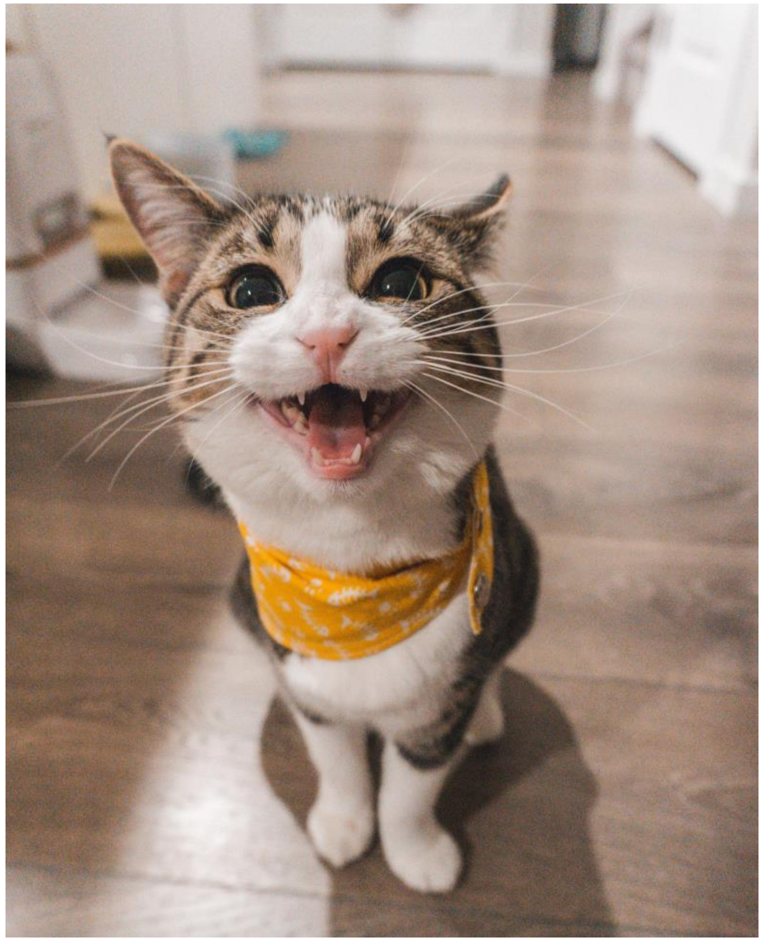
```
# Pipe each data manipulation operation to the next one
df <- starwars %>%
  select(-films, -vehicles, -starships) %>%
  filter(!is.na(mass)) %>%
  mutate(height_m = height/100,
         BMI = mass / (height_m)^2) %>%
  arrange(desc(BMI)) %>%
  group_by(gender) %>%
  summarise(median_BMI = median(BMI))
```

Simple string manipulation using stringr

The package in the tidyverse collection that helps us do data manipulations involving strings is called **{stringr}**. String manipulation is another common task, especially in data cleaning and pre-processing. Here are some examples:

```
# Make all character names as lower case
string <- str_to_lower(starwars$name)

# Combine the name, hair colour & eye colour of characters in a sentence
string <- str_c(starwars$name, " has ",
                starwars$hair_color, " hair and ",
                starwars$eye_color, " eyes.")

# Create an indicator where the specific pattern matches
ind <- str_detect(string = starwars$name, pattern = "Skywalker")
```

Live Coding Example 5



Use the starwars dataset to:

1. Transform the character names to upper case.
2. Combine the "name" and the "homeworld" to create a sentence, for example: "Luke Skywalker is from Tatooine".
3. Create an indicator for the rows where characters have green skin.

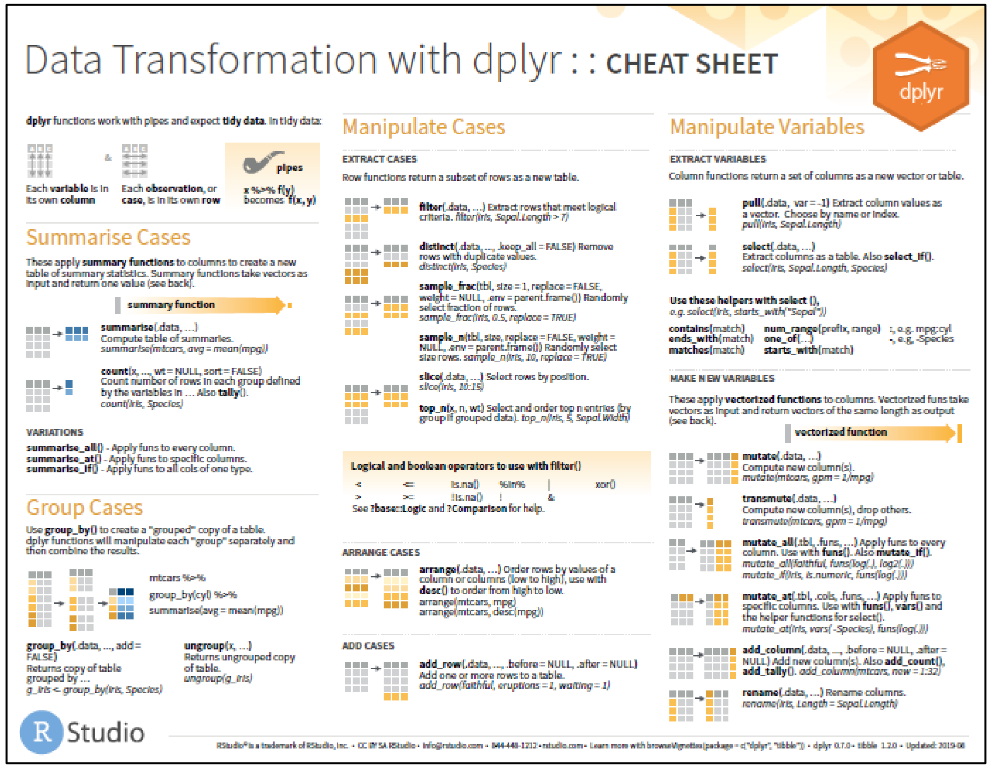
Live Coding Example 5

```
# Make all character names as upper case
string <- str_to_upper(starwars$name)

# Combine the name, hair colour & eye colour of characters in a sentence
string <- str_c(starwars$name, " is from ",
                starwars$homeworld, ".")

# Create an indicator where the specific pattern matches
ind <- str_detect(string = starwars$skin_color, pattern = "green")
```

Barcelona R

tidyverse Introduction to Data Manipulation

Other resources – {dplyr} cheat sheet



Data Transformation with dplyr : : CHEAT SHEET

dplyr

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(.)))

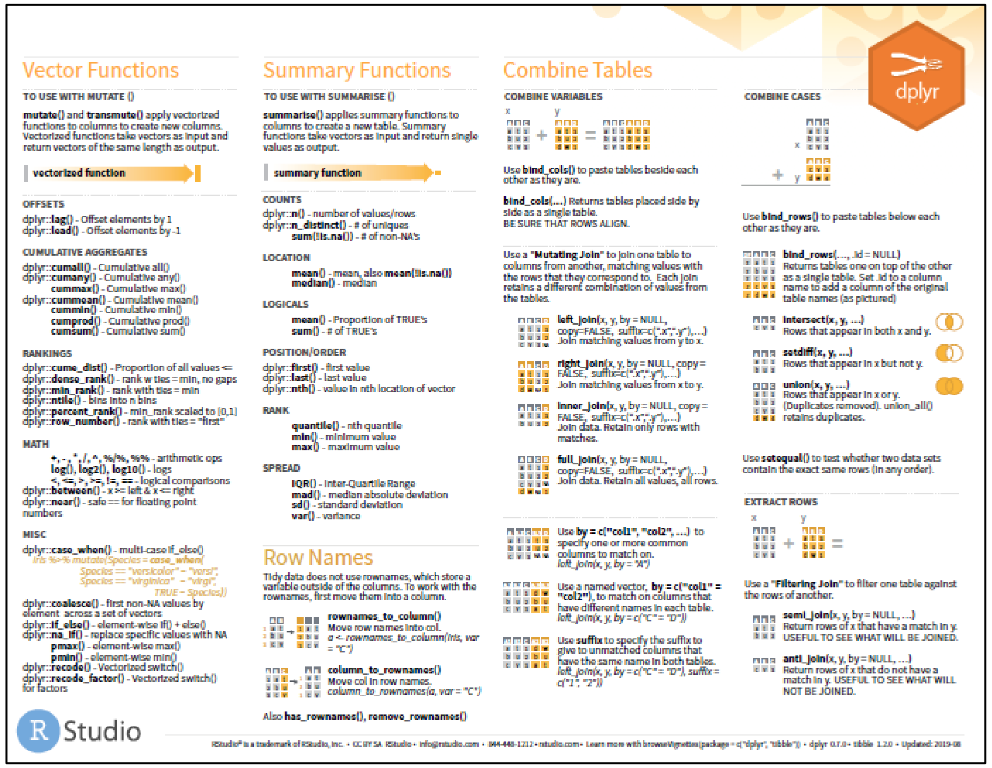
mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs**(), **vars**() and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

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)

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08



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.

RStudio

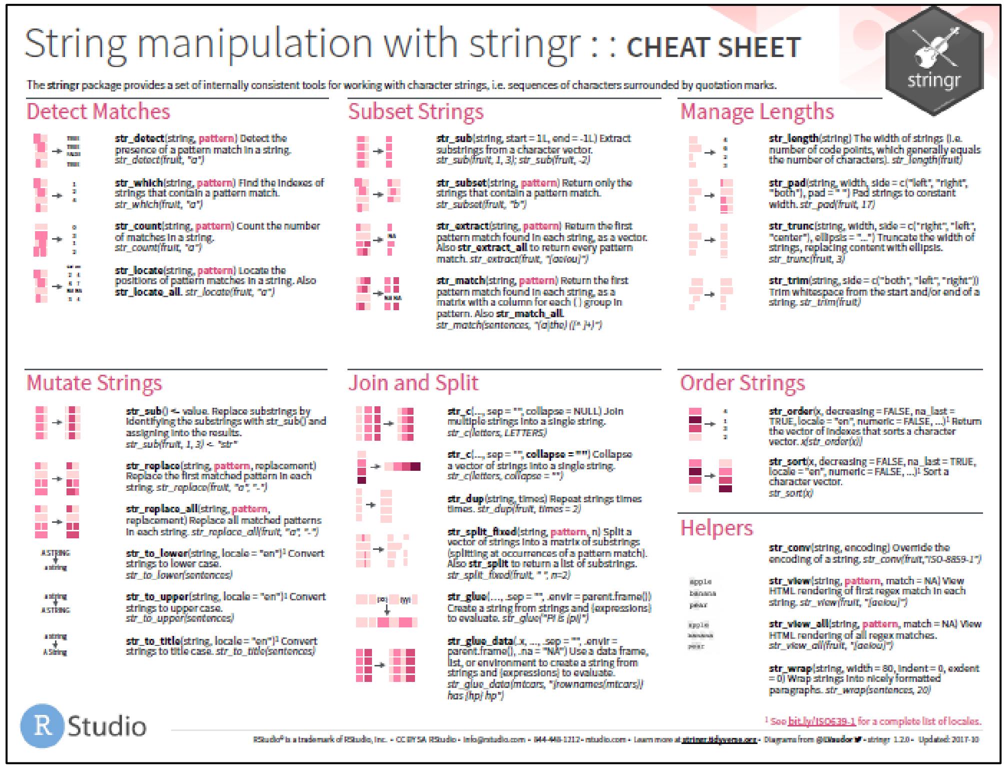
RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08

Get the cheat sheet at: https://rstudio.com/resources/cheatsheets/

Barcelona R

Introduction to Data Manipulation

Other resources – {stringr} cheat sheet



String manipulation with stringr : : CHEAT SHEET

stringr

The stringr package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

str_detect(string, pattern) Detect the presence of a pattern match in a string. *str_detect(fruit, "a")*

str_which(string, pattern) Find the indexes of strings that contain a pattern match. *str_which(fruit, "a")*

str_count(string, pattern) Count the number of matches in a string. *str_count(fruit, "a")*

str_locate(string, pattern) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

Subset Strings

str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*

str_subset(string, pattern) Return only the strings that contain a pattern match. *str_subset(fruit, "b")*

str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**. *str_match(sentences, "(a|the) ([^]+)")*

Manage Lengths

str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

Mutate Strings

str_sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*

str_replace(string, pattern, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*

str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

str_to_lower(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*

str_to_upper(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*

str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

Join and Split

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

str_c(..., sep = "", **collapse = ""**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*

str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

str_view(string, pattern, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

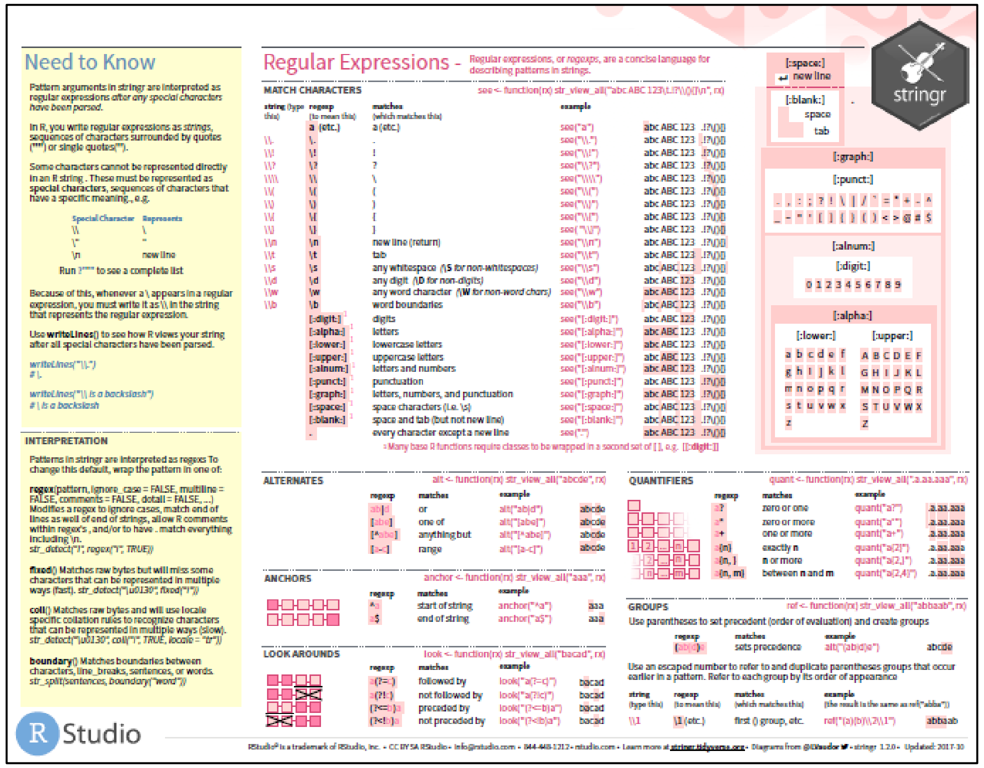
str_view_all(string, pattern, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at stringr.tidyverse.org • Diagrams from @LVaudor • stringr 1.2.0 • Updated: 2017-10



Need to Know

Pattern arguments in stringr are interpreted as regular expressions *after any special characters have been parsed.*

In R, you write regular expressions as strings, sequences of characters surrounded by quotes ("") or single quotes('').

Some characters cannot be represented directly in an R string . These must be represented as special characters, sequences of characters that have a specific meaning., e.g.

Special Character	Represents
\\	\
\"	"
\n	new line

Run ?"'" to see a complete list

Because of this, whenever a \ appears in a regular expression, you must write it as \\ in the string that represents the regular expression.

Use **writeLines**() to see how R views your string after all special characters have been parsed.

writeLines("\\.")
# \.

writeLines("\\ is a backslash")
# \ is a backslash

INTERPRETATION

Patterns in stringr are interpreted as regexs To change this default, wrap the pattern in one of:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...) Modifies a regex to ignore cases, match end of lines as well of end of strings, allow R comments within regex's , and/or to have . match everything including \n. *str_detect("I", regex("i", TRUE))*

fixed() Matches raw bytes but will miss some characters that can be represented in multiple ways (fast). *str_detect("\u0130", fixed("i"))*

coll() Matches raw bytes and will use locale specific collation rules to recognize characters that can be represented in multiple ways (slow). *str_detect("\u0130", coll("i", TRUE, locale = "tr"))*

boundary() Matches boundaries between characters, line_breaks, sentences, or words. *str_split(sentences, boundary("word"))*

Regular Expressions - Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

MATCH CHARACTERS — see <- function(rx) str_view_all("abc ABC 123\t.!?\\(){}\n", rx)

string (type this)	regexp (to mean this)	matches (which matches this)	example	
	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
\\.	\.	.	see("\\.")	abc ABC 123 .!?\(){}
\\!	\!	!	see("\\!")	abc ABC 123 .!?\(){}
\\?	\?	?	see("\\?")	abc ABC 123 .!?\(){}
\\\\	\\	\	see("\\\\")	abc ABC 123 .!?\(){}
\\(	\(	(	see("\\(")	abc ABC 123 .!?\(){}
\\)	\)	)	see("\\)")	abc ABC 123 .!?\(){}
\\{	\{	{	see("\\{")	abc ABC 123 .!?\(){}
\\}	\}	}	see("\\}")	abc ABC 123 .!?\(){}
\\n	\n	new line (return)	see("\\n")	abc ABC 123 .!?\(){}
\\t	\t	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	\s	any whitespace (\S for non-whitespaces)	see("\\s")	abc ABC 123 .!?\(){}
\\d	\d	any digit (\D for non-digits)	see("\\d")	abc ABC 123 .!?\(){}
\\w	\w	any word character (\W for non-word chars)	see("\\w")	abc ABC 123 .!?\(){}
\\b	\b	word boundaries	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:][1]	digits	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:][1]	letters	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:][1]	lowercase letters	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:][1]	uppercase letters	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:][1]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:][1]	punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:][1]	letters, numbers, and punctuation	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:][1]	space characters (i.e. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:][1]	space and tab (but not new line)	see("[:blank:]")	abc ABC 123 .!?\(){}
	.	every character except a new line	see(".")	abc ABC 123 .!?\(){}

[1] Many base R functions require classes to be wrapped in a second set of [ ], e.g. [[:digit:]]

[:space:] new line
[:blank:] space tab
[:graph:]
[:punct:] . , : ; ? ! \ | / ` = * + - ^ _ ~ " ' [ ] { } ( ) < > @ # $
[:alnum:]
[:digit:] 0 1 2 3 4 5 6 7 8 9
[:alpha:]
[:lower:] a b c d e f g h i j k l m n o p q r s t u v w x z
[:upper:] A B C D E F G H I J K L M N O P Q R S T U V W X Z

ALTERNATES — alt <- function(rx) str_view_all("abcde", rx)

regexp	matches	example	
ab|d	or	alt("ab|d")	abcde
[abe]	one of	alt("[abe]")	abcde
[^abe]	anything but	alt("[^abe]")	abcde
[a-c]	range	alt("[a-c]")	abcde

ANCHORS — anchor <- function(rx) str_view_all("aaa", rx)

regexp	matches	example	
^a	start of string	anchor("^a")	aaa
a$	end of string	anchor("a$")	aaa

LOOK AROUNDS — look <- function(rx) str_view_all("bacad", rx)

regexp	matches	example	
a(?=c)	followed by	look("a(?=c)")	bacad
a(?!c)	not followed by	look("a(?!c)")	bacad
(?<=b)a	preceded by	look("(?<=b)a")	bacad
(?<!b)a	not preceded by	look("(?<!b)a")	bacad

QUANTIFIERS — quant <- function(rx) str_view_all(".a.aa.aaa", rx)

regexp	matches	example	
a?	zero or one	quant("a?")	.a.aa.aaa
a*	zero or more	quant("a*")	.a.aa.aaa
a+	one or more	quant("a+")	.a.aa.aaa
a{n}	exactly n	quant("a{2}")	.a.aa.aaa
a{n, }	n or more	quant("a{2,}")	.a.aa.aaa
a{n, m}	between n and m	quant("a{2,4}")	.a.aa.aaa

GROUPS — ref <- function(rx) str_view_all("abbaab", rx)

Use parentheses to set precedent (order of evaluation) and create groups

regexp	matches	example	
(ab|d)e	sets precedence	alt("(ab|d)e")	abcde

Use an escaped number to refer to and duplicate parentheses groups that occur earlier in a pattern. Refer to each group by its order of appearance

string (type this)	regexp (to mean this)	matches (which matches this)	example (the result is the same as ref("abba"))	
\\1	\1 (etc.)	first () group, etc.	ref("(a)(b)\\2\\1")	abbaab

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at stringr.tidyverse.org • Diagrams from @LVaudor • stringr 1.2.0 • Updated: 2017-10

Get the cheat sheet at: https://rstudio.com/resources/cheatsheets/

Thank you to our sponsors and partners!

MANGO SOLUTIONS

RStudio®