Summarizing Data Part 1

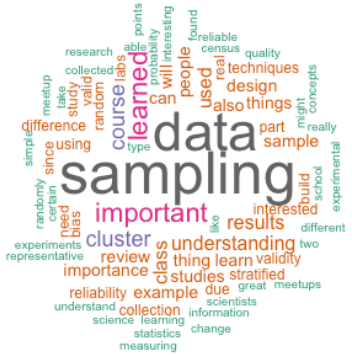
DATA 606 - Statistics & Probability for Data Analytics

Jason Bryer, Ph.D. and Angela Lui, Ph.D.

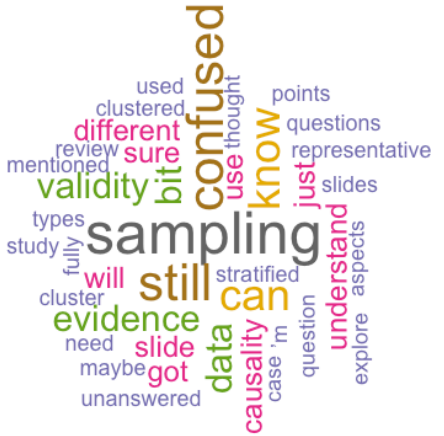
September 8, 2025

One Minute Paper Results

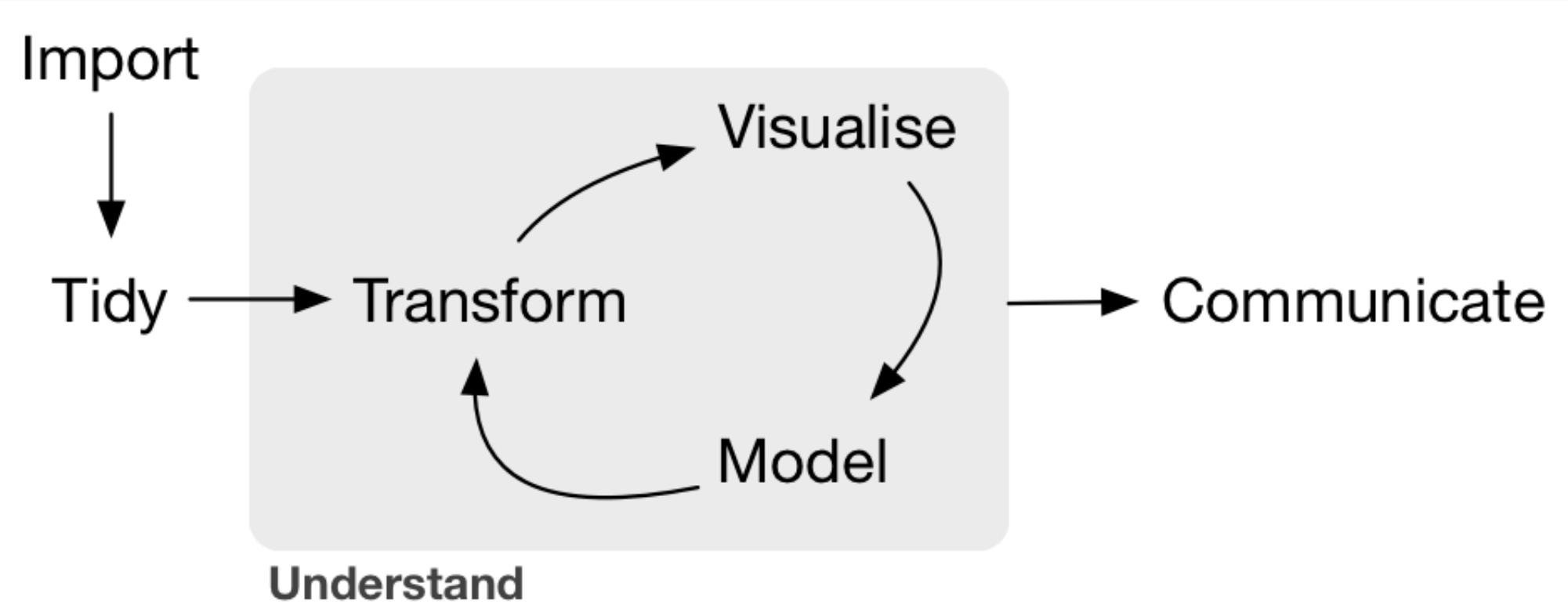
What was the most important thing you learned during this class?



What important question remains unanswered for you?



Workflow



Source: Wickham & Grolemund, 2017

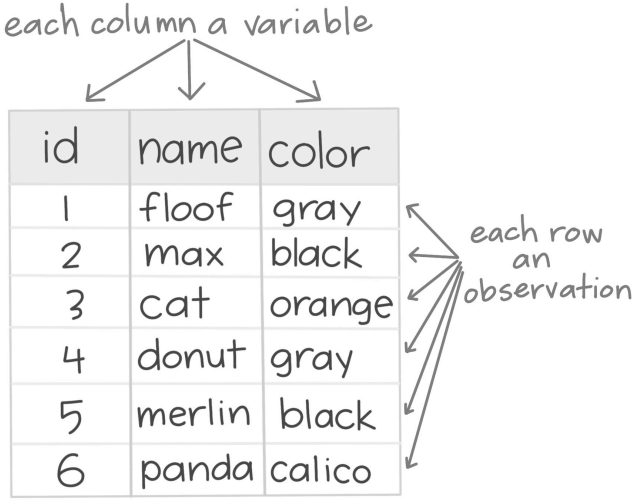
Tidy Data

"TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."

—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



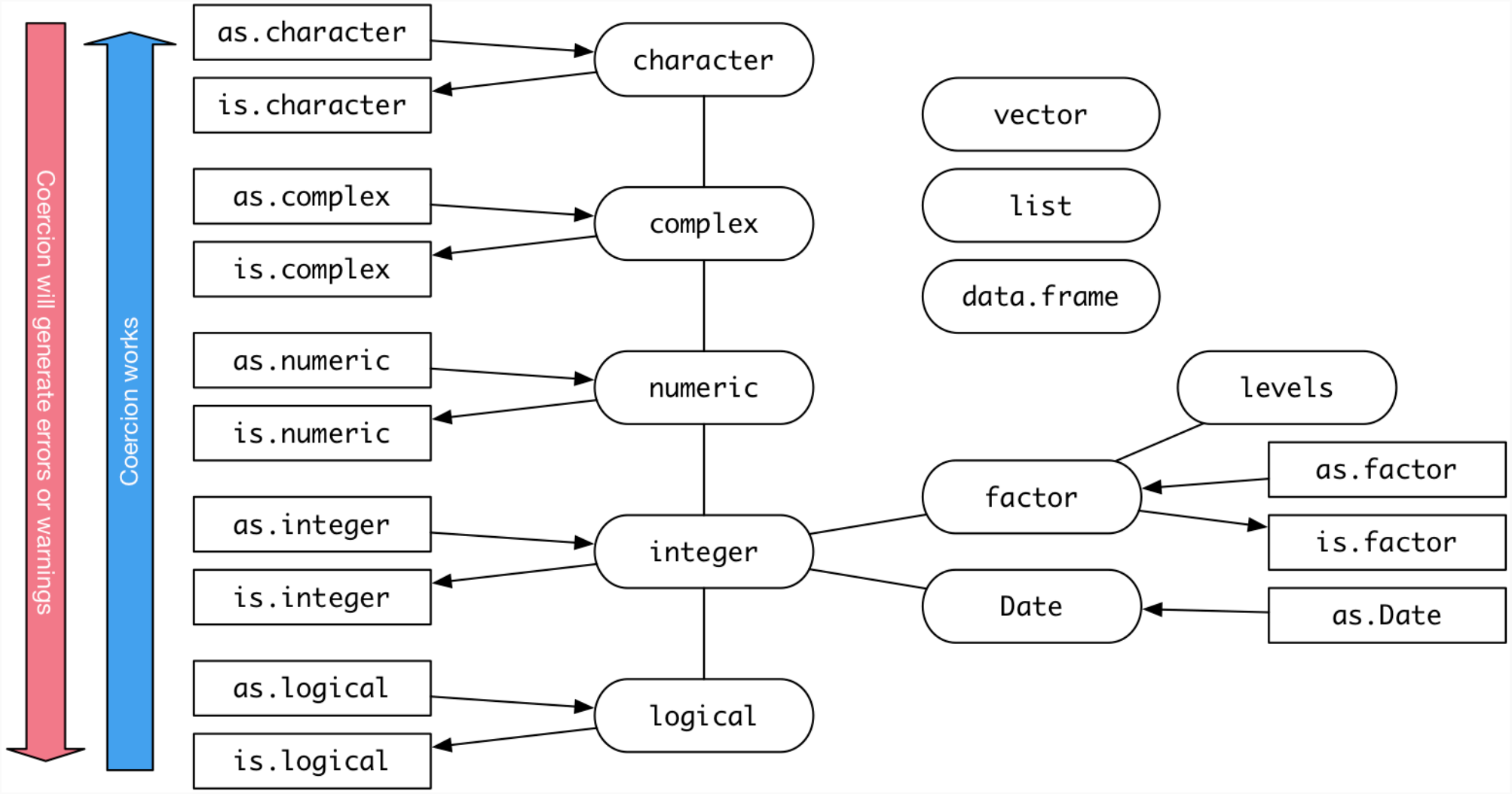
Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

See Wickham (2014) Tidy data.

Types of Data

- Numerical (quantitative)
  - Continuous
  - Discrete
- Categorical (qualitative)
  - Regular categorical
  - Ordinal

Data Types in R



Data Types / Descriptives / Visualizations

Data Type	Descriptive Stats	Visualization
Continuous	mean, median, mode, standard deviation, IQR	histogram, density, box plot
Discrete	contingency table, proportional table, median	bar plot
Categorical	contingency table, proportional table	bar plot
Ordinal	contingency table, proportional table, median	bar plot
Two quantitative	correlation	scatter plot
Two qualitative	contingency table, chi-squared	mosaic plot, bar plot
Quantitative & Qualitative	grouped summaries, ANOVA, t-test	box plot

Statistics

When describing a quantitative variable we are often interested in two things:

1. A measure of center
2. A measure of spread

The most common measures we will use in this class is the mean and median.

$$\bar{x} = \frac{\Sigma(x_i)}{n}$$

$$S^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

Note that in the numerator for the variance calculation we square the differences (also known as deviations). Squaring terms is common practice in statistics that serves two purposes:

1. It makes all the values positive.
2. It weighs observations that are further from the center more.

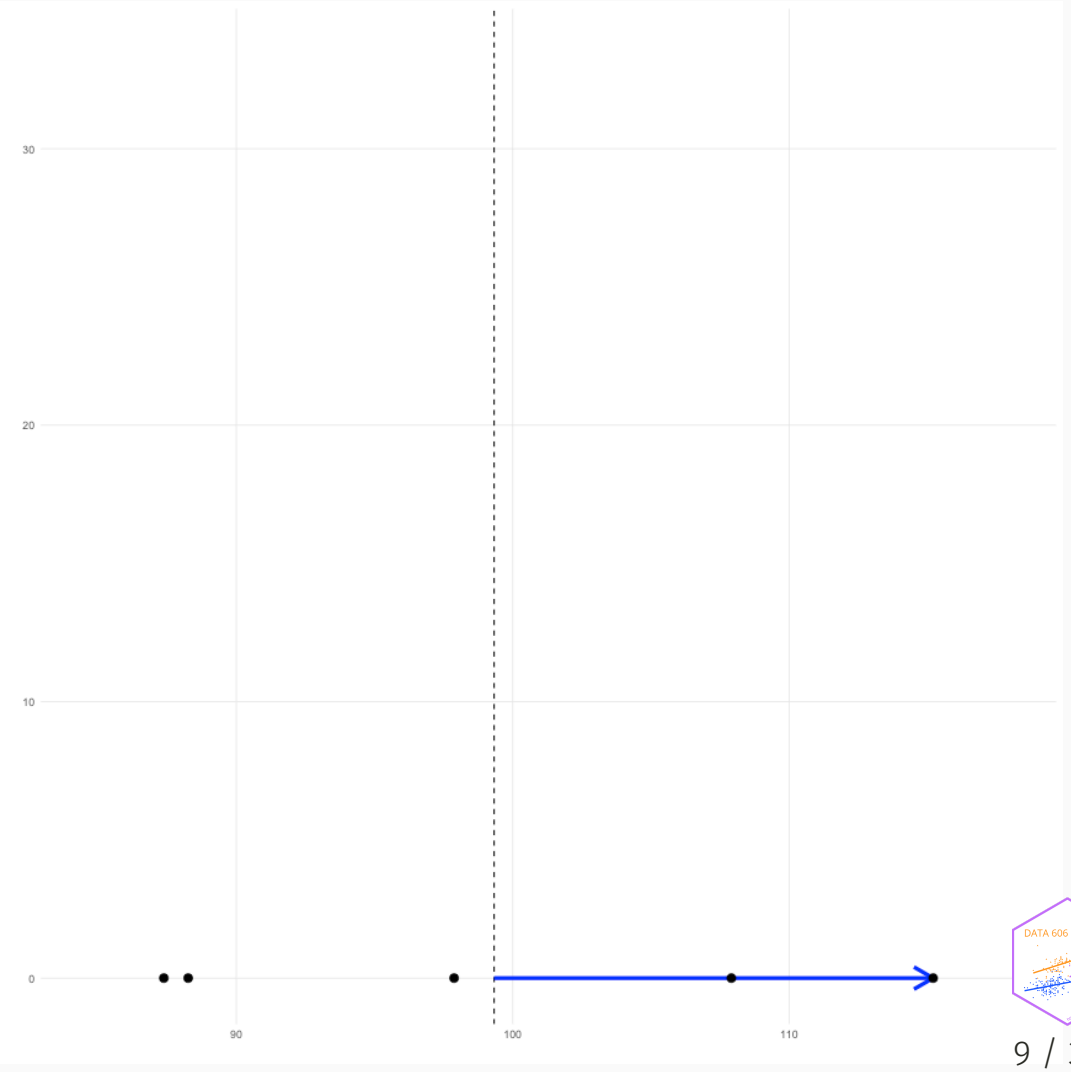
Variance

Population Variance:

$$S^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

Consider a dataset with five values (black points in the figure). For the largest value, the deviance is represented by the blue line ($x_i - \bar{x}$).

See also:
https://shiny.rit.albany.edu/stat/visualizess/
https://github.com/jbryer/VisualStats/

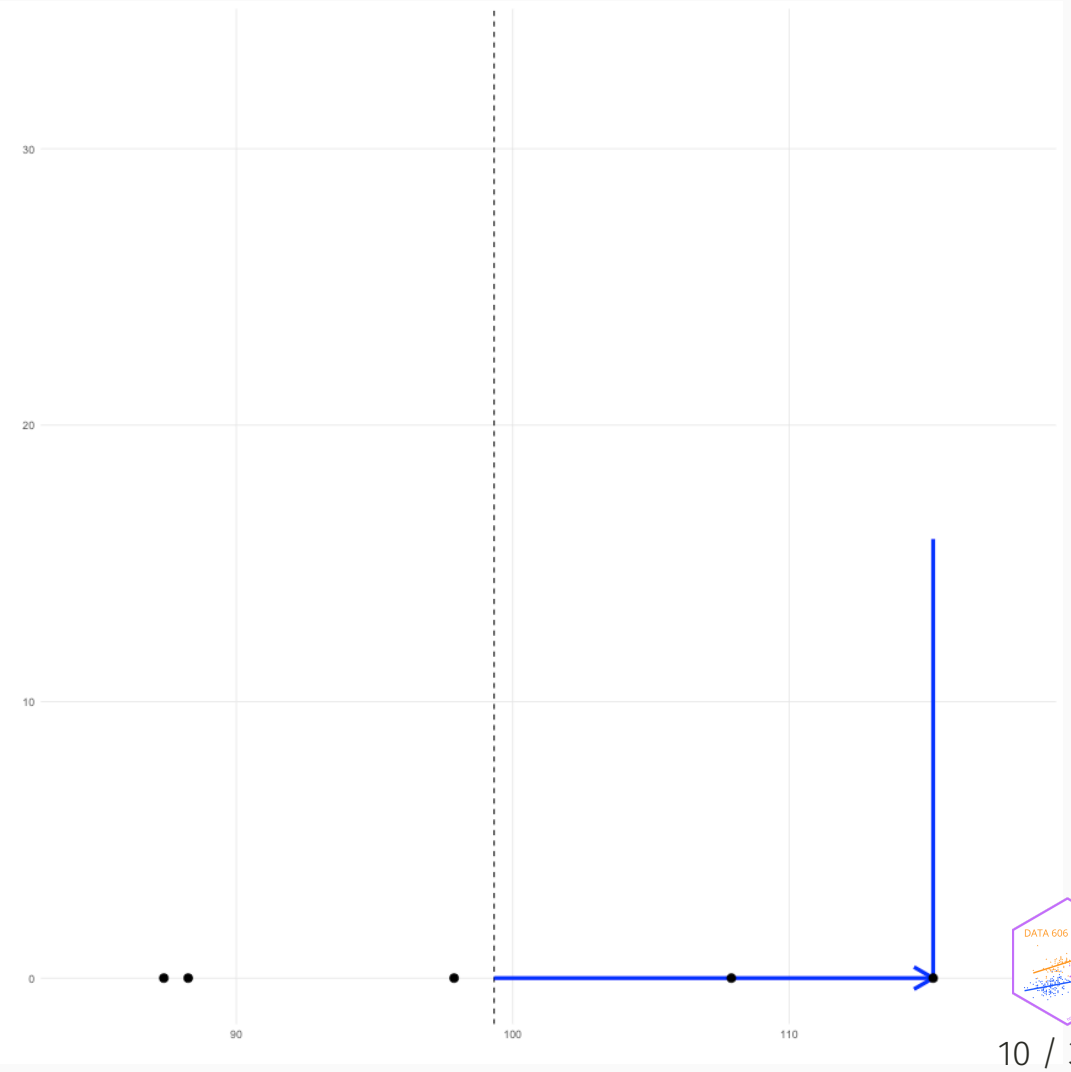


Variance (cont.)

Population Variance:

$$S^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

In the numerator, we square each of these deviances. We can conceptualize this as a square. Here, we add the deviance in the *y* direction.

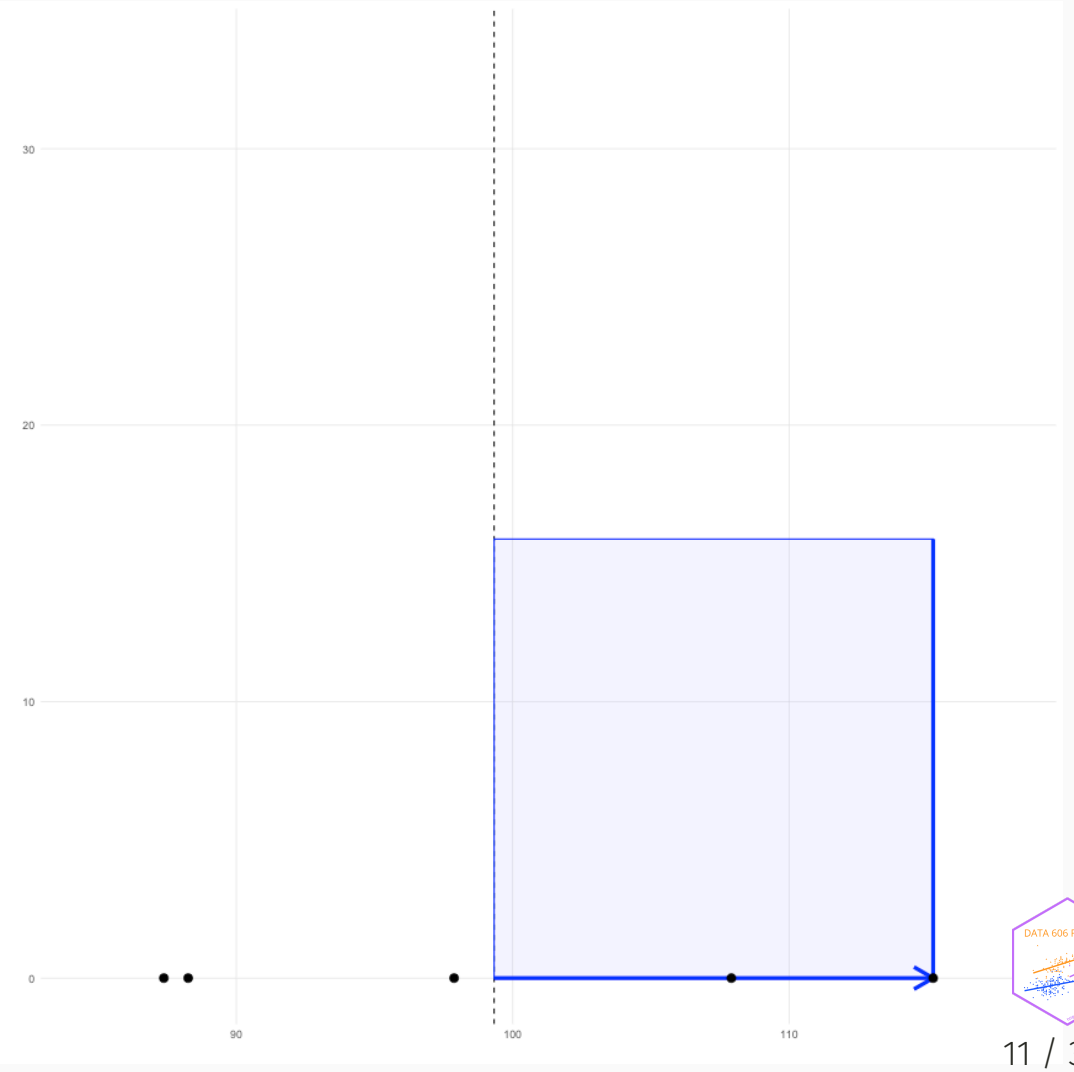


Variance (cont.)

Population Variance:

$$S^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

We end up with a square.

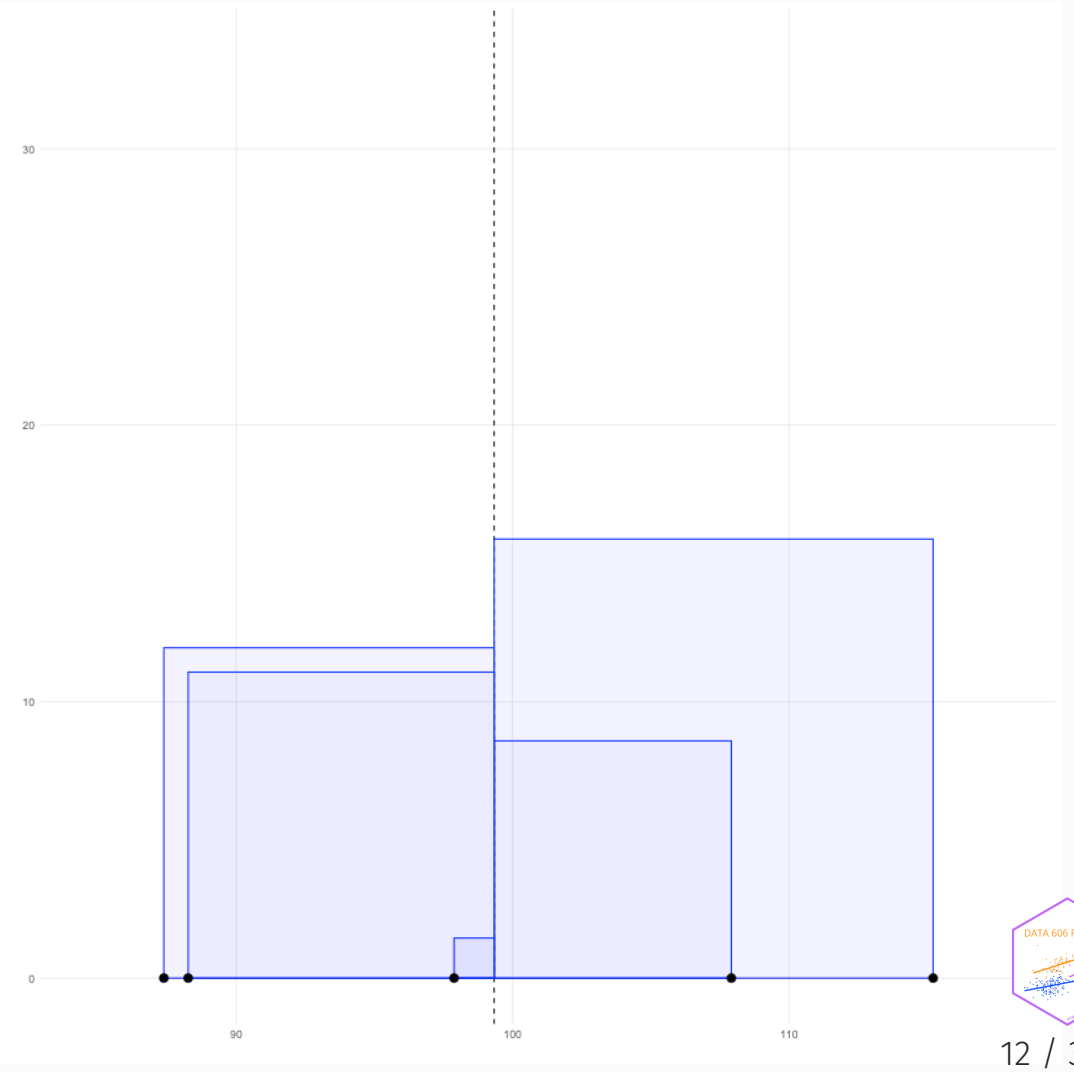


Variance (cont.)

Population Variance:

$$S^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

We can plot the squared deviance for all the data points. That is, each component in the numerator is the area of each of these squares.

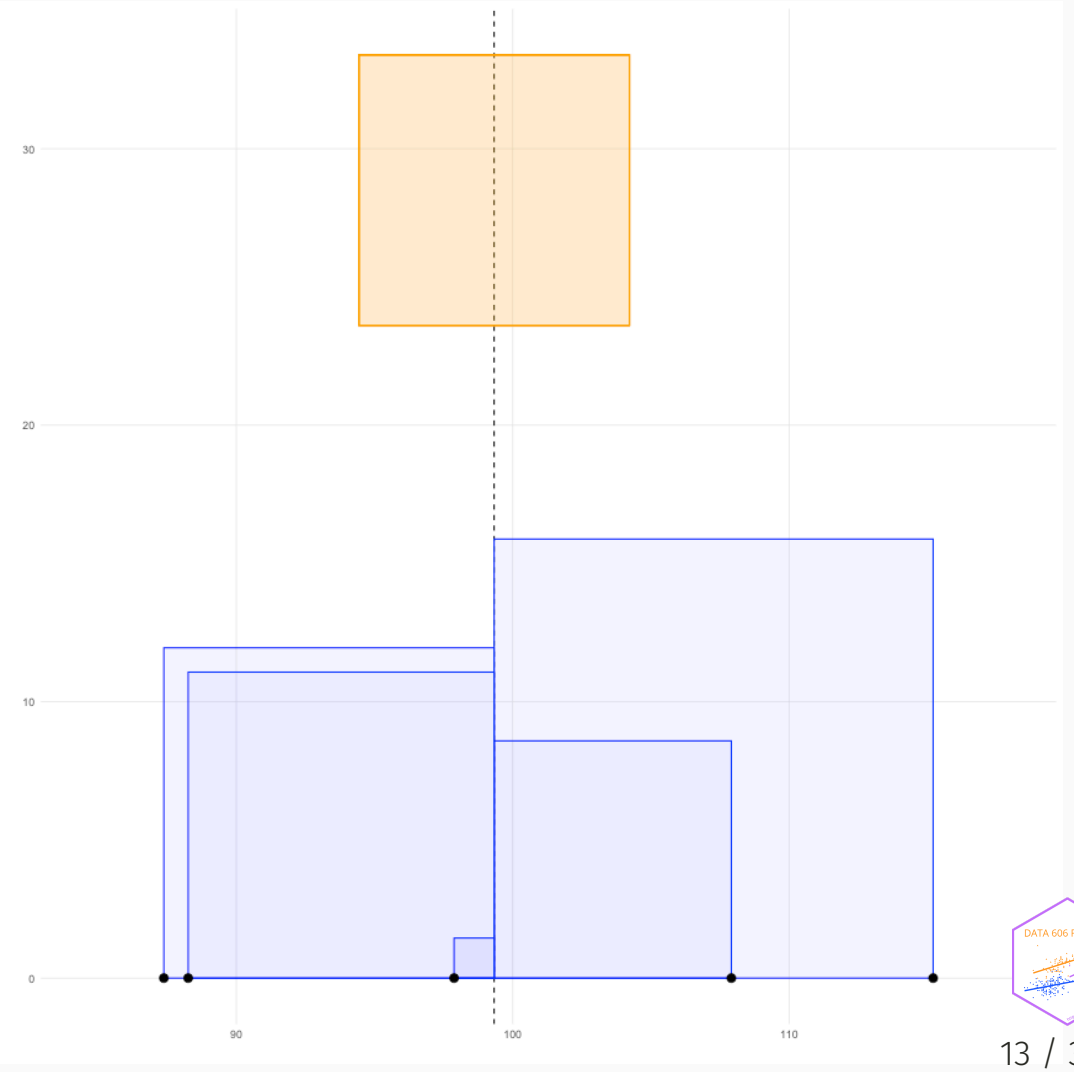


Variance (cont.)

Population Variance:

$$S^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

The variance is therefore the average of the area of all these squares, here represented by the orange square.



Population versus Sample Variance

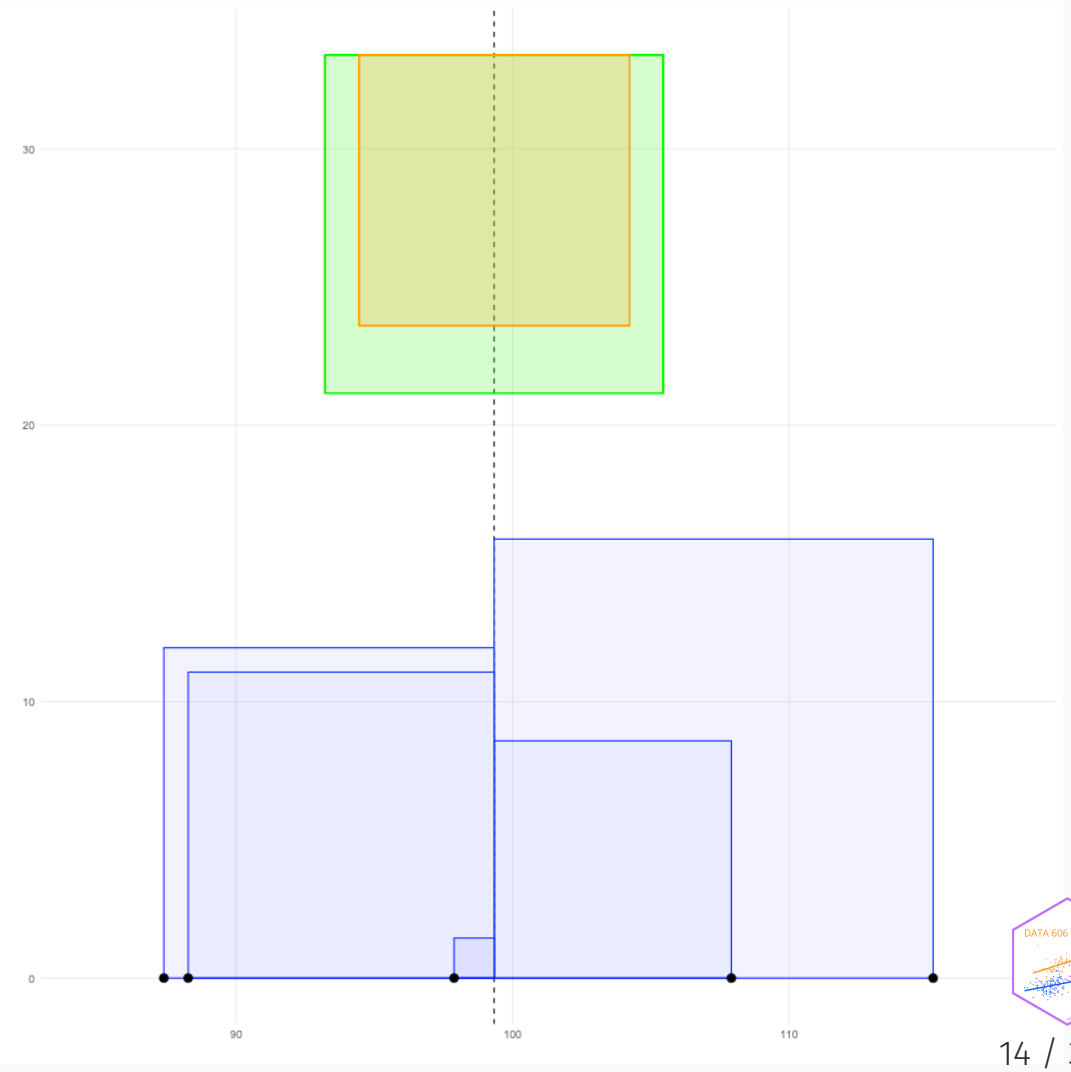
Typically we want the sample variance. The difference is we divide by $n-1$ to calculate the sample variance. This results in a slightly larger area (variance) then if we divide by n.

Population Variance (yellow):

$$S^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

Sample Variance (green):

$$s^2 = \frac{\Sigma(x_i - \bar{x})^2}{n-1}$$



Robust Statistics

Consider the following data randomly selected from the normal distribution:

```
set.seed(41)
x <- rnorm(30, mean = 100, sd = 15)
mean(x); sd(x)
```

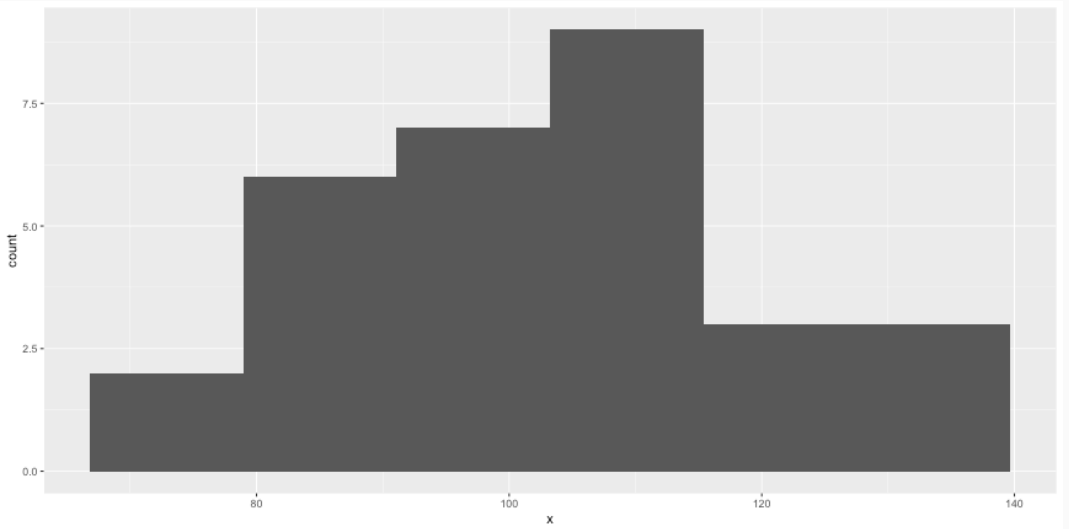
```
## [1] 103.1934
```

```
## [1] 16.8945
```

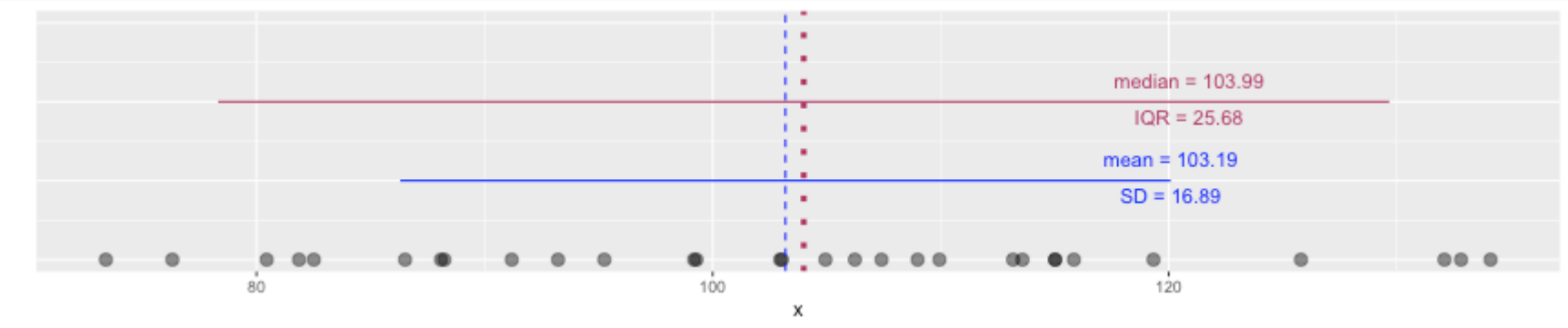
```
median(x); IQR(x)
```

```
## [1] 103.9947
```

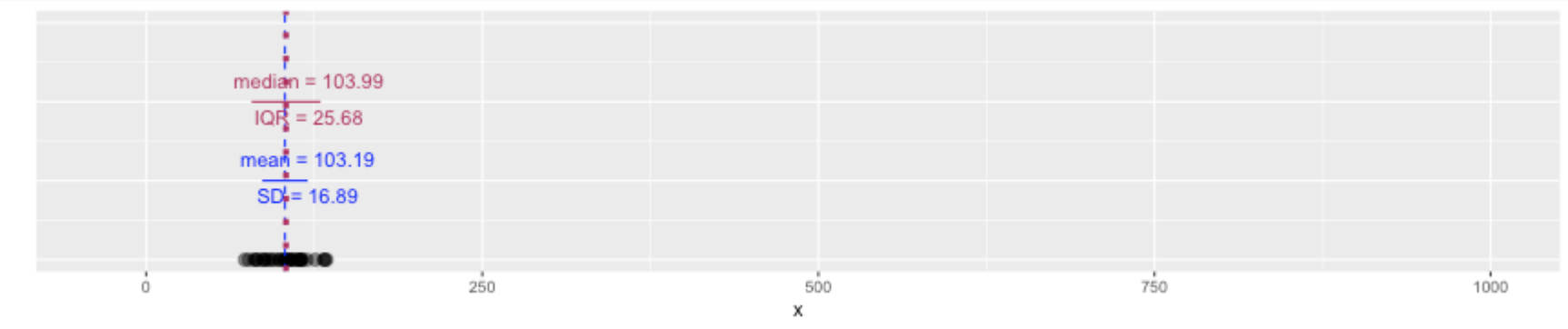
```
## [1] 25.68004
```



Robust Statistics

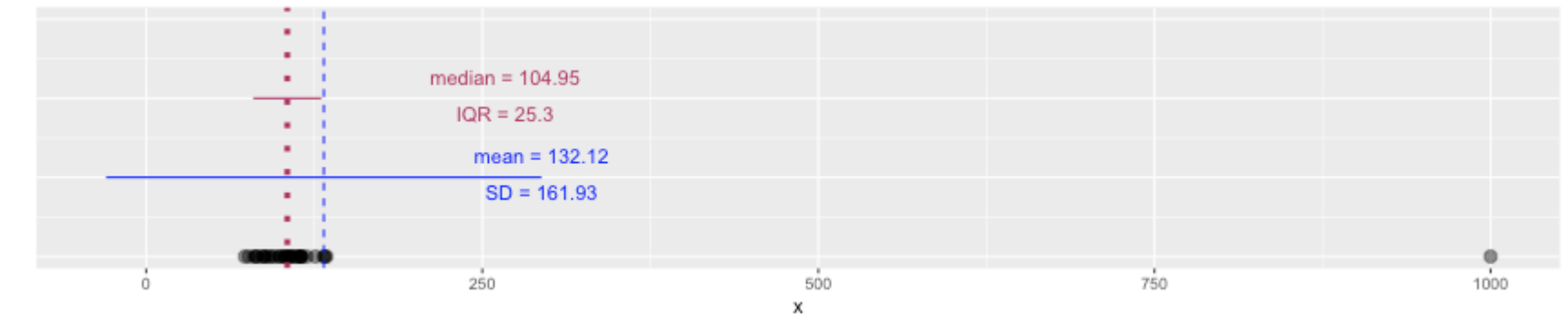


Robust Statistics



Let's add an extreme value:

```
x <- c(x, 1000)
```



Robust Statistics

Median and IQR are more robust to skewness and outliers than mean and SD. Therefore,

- for skewed distributions it is often more helpful to use median and IQR to describe the center and spread
- for symmetric distributions it is often more helpful to use the mean and SD to describe the center and spread

About `legosets`

To install the `brickset` package:

```
remotes::install_github('jbryer/brickset')
```

To load the load the `legosets` dataset.

```
data('legosets', package = 'brickset')
```

The `legosets` data has 20420 observations of 36 variables.

```
names(legosets)
```

```
##  [1] "setID"                "number"               "numberVariant"
##  [4] "name"                 "year"                 "theme"
##  [7] "themeGroup"           "subtheme"             "category"
## [10] "released"             "pieces"               "minifigs"
## [13] "bricksetURL"          "rating"               "reviewCount"
## [16] "packagingType"        "availability"         "agerange_min"
## [19] "thumbnailURL"         "imageURL"             "US_retailPrice"
## [22] "US_dateFirstAvailable" "US_dateLastAvailable" "UK_retailPrice"
## [25] "UK_dateFirstAvailable" "UK_dateLastAvailable" "CA_retailPrice"
## [28] "CA_dateFirstAvailable" "CA_dateLastAvailable" "DE_retailPrice"
## [31] "DE_dateFirstAvailable" "DE_dateLastAvailable" "height"
## [34] "width"                "depth"                "weight"
```

Structure (`str`)

```
str(legosets)
```

```
## 'data.frame':    20420 obs. of  36 variables:
##  $ setID                : int  7693 7695 7697 7698 25534 7418 7419 6020 22704 7421 ...
##  $ number               : chr  "1" "2" "3" "4" ...
##  $ numberVariant        : int  8 8 6 4 6 1 1 1 3 4 ...
##  $ name                 : chr  "Small house set" "Medium house set" "Medium house set" "Large house set" ...
##  $ year                 : int  1970 1970 1970 1970 1970 1970 1970 1970 1970 1970 ...
##  $ theme                : chr  "Minitalia" "Minitalia" "Minitalia" "Minitalia" ...
##  $ themeGroup           : chr  "Vintage" "Vintage" "Vintage" "Vintage" ...
##  $ subtheme             : chr  NA NA NA NA ...
##  $ category             : chr  "Normal" "Normal" "Normal" "Normal" ...
##  $ released             : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
##  $ pieces               : int  67 109 158 233 NA 1 1 60 65 NA ...
##  $ minifigs             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ bricksetURL          : chr  "https://brickset.com/sets/1-8" "https://brickset.com/sets/2-8" "https://brickset.com/sets/3-6" "https://brickset.com/sets/4-4" ...
##  $ rating               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ reviewCount          : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ packagingType        : chr  "{Not specified}" "{Not specified}" "{Not specified}" "{Not specified}" ...
##  $ availability         : chr  "{Not specified}" "{Not specified}" "{Not specified}" "{Not specified}" ...
##  $ agerange_min         : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ thumbnailURL         : chr  "https://images.brickset.com/sets/small/1-8.jpg" "https://images.brickset.com/sets/small/2-8.jpg" "https://images.brickset.com/sets/small/3-6.jpg" "https:
##  $ imageURL             : chr  "https://images.brickset.com/sets/images/1-8.jpg" "https://images.brickset.com/sets/images/2-8.jpg" "https://images.brickset.com/sets/images/3-6.jpg" "ht
##  $ US_retailPrice       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ US_dateFirstAvailable: Date, format: NA NA ...
##  $ US_dateLastAvailable : Date, format: NA NA ...
##  $ UK_retailPrice       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ UK_dateFirstAvailable: Date, format: NA NA ...
##  $ UK_dateLastAvailable : Date, format: NA NA ...
##  $ CA_retailPrice       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ CA_dateFirstAvailable: Date, format: NA NA ...
##  $ CA_dateLastAvailable : Date, format: NA NA ...
##  $ DE_retailPrice       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ DE_dateFirstAvailable: Date, format: NA NA ...
##  $ DE_dateLastAvailable : Date, format: NA NA ...
##  $ height               : num  NA NA NA NA NA ...
##  $ width                : num  NA NA NA NA NA ...
```

RStudio Eenvironment tab can help

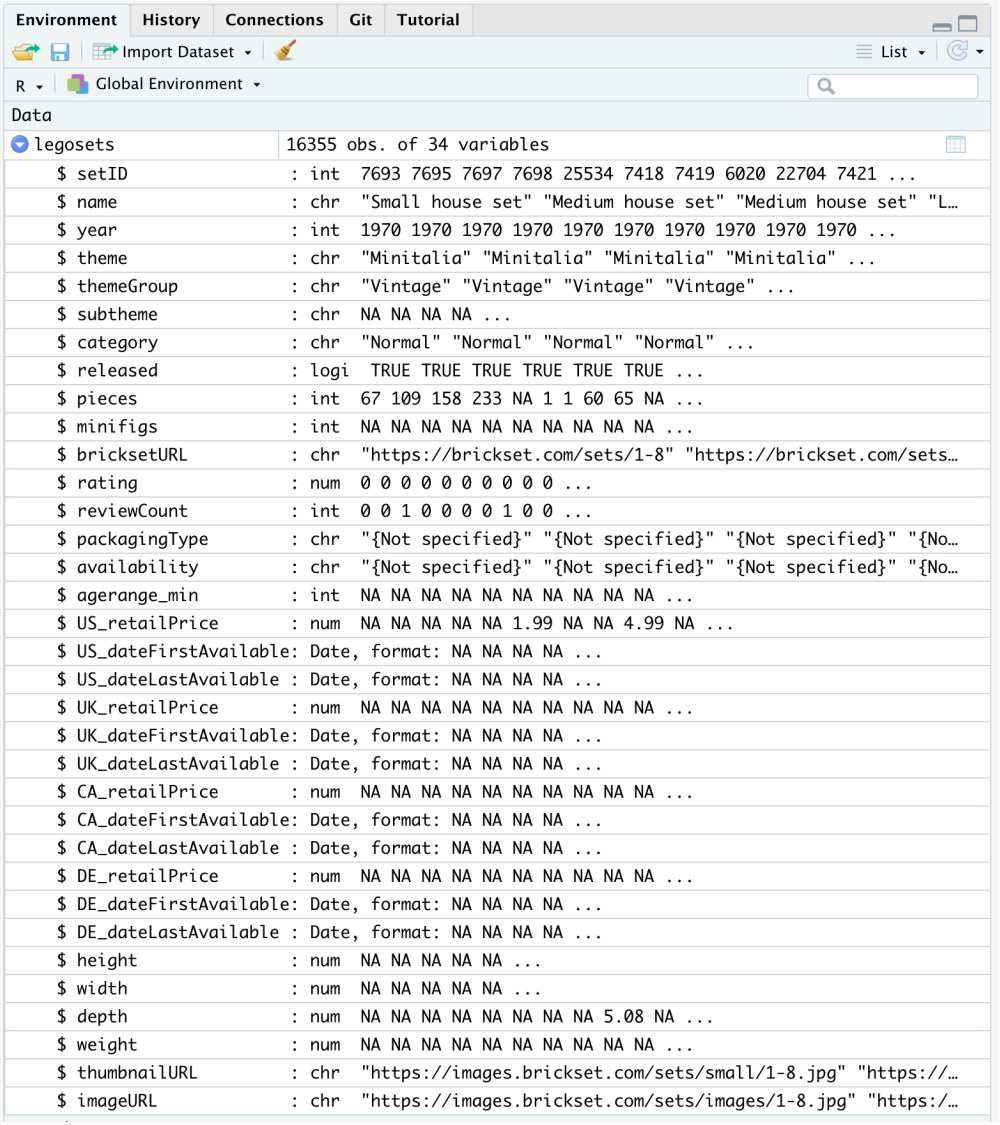


Table View

Show 10 entries

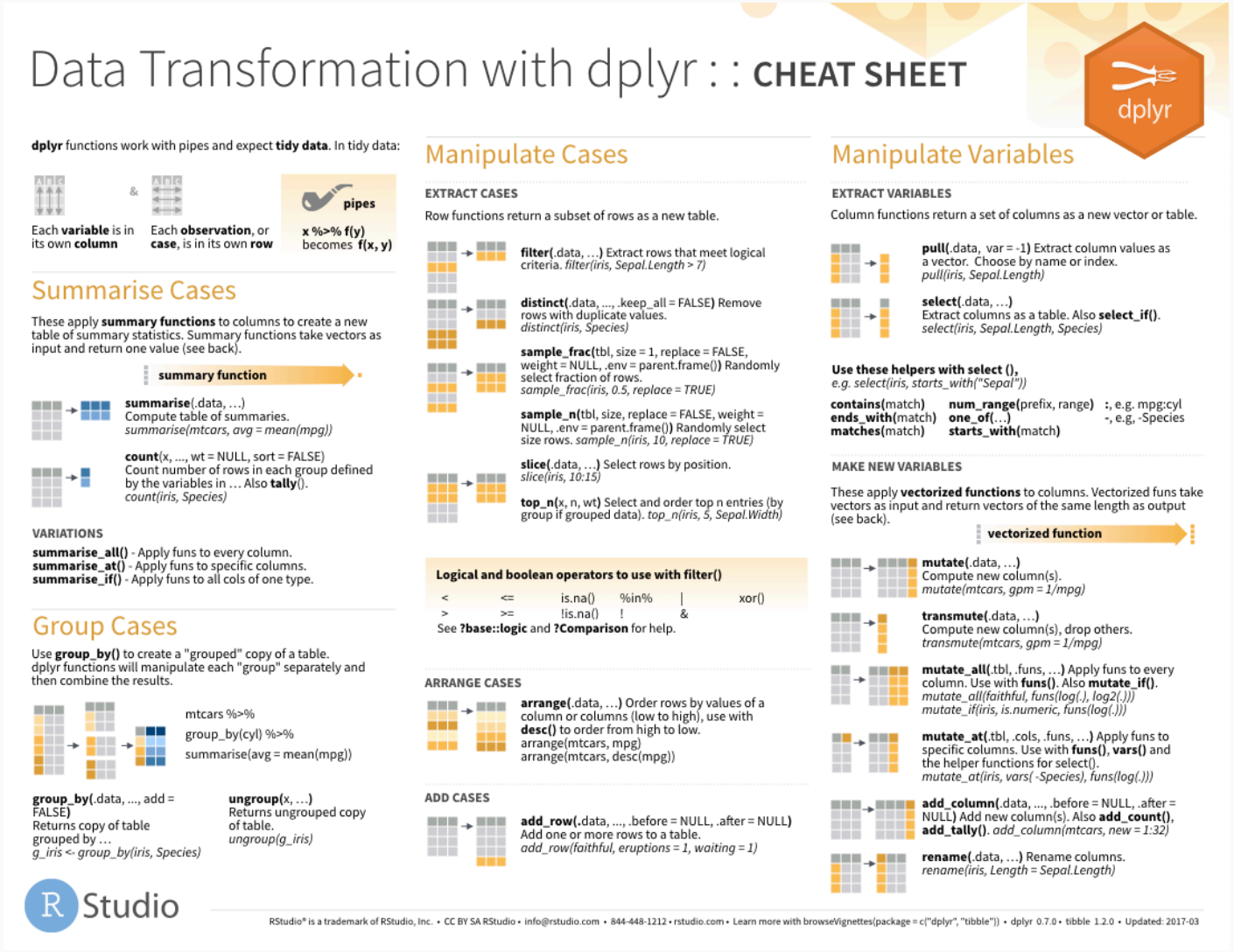
Search:

	setID	name	year	theme	themeGroup	category	US_retailPrice	pieces	minifigs	rating
1	10007	Super Pack 3-in-1	2012	Cars	Licensed	Collection		381		0
2	29171	Vintage Minnie	2019	Collectable Minifigures	Miscellaneous	Normal		7	1	3.7
3	23968	Onua - Master of Earth	2015	Bionicle	Constraction	Normal	19.99	108		4.3
4	1391	Party Dress and Accessories	1998	Scala	Girls	Normal		22		0
5	23267	Technology Resource Set	1995	Dacta	Educational	Normal		1733		0
6	33983	Birthday Cake	2018	Promotional	Miscellaneous	Extended		28		0
7	47192	Eldorado Fortress	2023	Icons	Model making	Normal	214.99	2509	9	4.4
8	4516	Designer's Manual	1991	Books	Miscellaneous	Book		1		0
9	31823	Stuttgart brand store associate figure	2021	Promotional	Miscellaneous	Extended		129		0
10	29482	Newbury Abandoned Prison	2020	Hidden Side	Action/Adventure	Normal	39.99	400	4	3.8

Showing 1 to 10 of 100 entries

Previous 1 2 3 4 5 ... 10 Next

Data Wrangling 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)
ends_with(match)
matches(match)
num_range(prefix, range)
one_of(…)
starts_with(match)
:, e.g. mpg:cyl
-, e.g, -Species

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

Tidyverse vs Base R

R Syntax Comparison : : CHEAT SHEET

Dollar sign syntax	Formula syntax	Tidyverse syntax
`goal(data$x, data$y)`	`goal(y~x|z, data=data, group=w)`	`data %>% goal(x)`
SUMMARY STATISTICS:	**SUMMARY STATISTICS:**	**SUMMARY STATISTICS:**
one continuous variable: `mean(mtcars$mpg)`	one continuous variable: `mosaic::mean(~mpg, data=mtcars)`	one continuous variable: `mtcars %>% dplyr::summarize(mean(mpg))`

one categorical variable:
`table(mtcars$cyl)`
	one categorical variable:
	`mosaic::tally(~cyl, data=mtcars)`
		one categorical variable:
		`mtcars %>% dplyr::group_by(cyl) %>%`
		`dplyr::summarize(n())`

two categorical variables:
`table(mtcars$cyl, mtcars$am)`
	two categorical variables:
	`mosaic::tally(cyl~am, data=mtcars)`
		two categorical variables:
		`mtcars %>% dplyr::group_by(cyl, am) %>%`
		`dplyr::summarize(n())`

one continuous, one categorical:
`mean(mtcars$mpg[mtcars$cyl==4])`
`mean(mtcars$mpg[mtcars$cyl==6])`
`mean(mtcars$mpg[mtcars$cyl==8])`
	one continuous, one categorical:
	`mosaic::mean(mpg~cyl, data=mtcars)`
		one continuous, one categorical:
		`mtcars %>% dplyr::group_by(cyl) %>%`
		`dplyr::summarize(mean(mpg))`

PLOTTING:	**PLOTTING:**	**PLOTTING:**

one continuous variable:
`hist(mtcars$disp)`
`boxplot(mtcars$disp)`
	one continuous variable:
	`lattice::histogram(~disp, data=mtcars)`
	`lattice::bwplot(~disp, data=mtcars)`
		one continuous variable:
		`ggplot2::qplot(x=mpg, data=mtcars, geom = "histogram")`
		`ggplot2::qplot(y=disp, x=1, data=mtcars, geom="boxplot")`

one categorical variable: `barplot(table(mtcars$cyl))`	one categorical variable: `mosaic::bargraph(~cyl, data=mtcars)`	one categorical variable: `ggplot2::qplot(x=cyl, data=mtcars, geom="bar")`
two continuous variables: `plot(mtcars$disp, mtcars$mpg)`	two continuous variables: `lattice::xyplot(mpg~disp, data=mtcars)`	two continuous variables: `ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")`

two categorical variables:
`mosaicplot(table(mtcars$am, mtcars$cyl))`
	two categorical variables:
	`mosaic::bargraph(~am, data=mtcars, group=cyl)`
		two categorical variables:
		`ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") +`
		`facet_grid(.~am)`

one continuous, one categorical:
`histogram(mtcars$disp[mtcars$cyl==4])`
`histogram(mtcars$disp[mtcars$cyl==6])`
`histogram(mtcars$disp[mtcars$cyl==8])`
`boxplot(mtcars$disp[mtcars$cyl==4])`
`boxplot(mtcars$disp[mtcars$cyl==6])`
`boxplot(mtcars$disp[mtcars$cyl==8])`
	one continuous, one categorical:
	`lattice::histogram(~disp|cyl, data=mtcars)`
	`lattice::bwplot(cyl~disp, data=mtcars)`
		one continuous, one categorical:
		`ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") +`
		`facet_grid(.~cyl)`
		`ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars,`
		`geom="boxplot")`

WRANGLING:		**WRANGLING:**
subsetting: `mtcars[mtcars$mpg>30, ]`		subsetting: `mtcars %>% dplyr::filter(mpg>30)`

making a new variable:
`mtcars$efficient[mtcars$mpg>30] <- TRUE`
`mtcars$efficient[mtcars$mpg<30] <- FALSE`
		making a new variable:
		`mtcars <- mtcars %>%`
		`dplyr::mutate(efficient = if_else(mpg>30, TRUE, FALSE))`

tilde

the pipe

The variety of R syntaxes give you many ways to "say" the same thing

read **across** the cheatsheet to see how different syntaxes approach the same problem

SMITH COLLEGE

RStudio® is a trademark of RStudio, Inc. CC BY Amelia McNamara • amcnamara@smith.edu • @AmeliaMN • science.smith.edu/~amcnamara/ • Updated: 2018-01

Pipes %>% and |>

The pipe operator (%>%) introduced with the `magrittr` R package allows for the chaining of R operations. As of version 4.1, R now has a native pipe operator (|>). They take the output from the left-hand side and passes it as the first parameter to the function on the right-hand side.

You can do this in two steps:

```
tab_out <- table(legosets$category)
prop.table(tab_out)
```

Or as nested function calls.

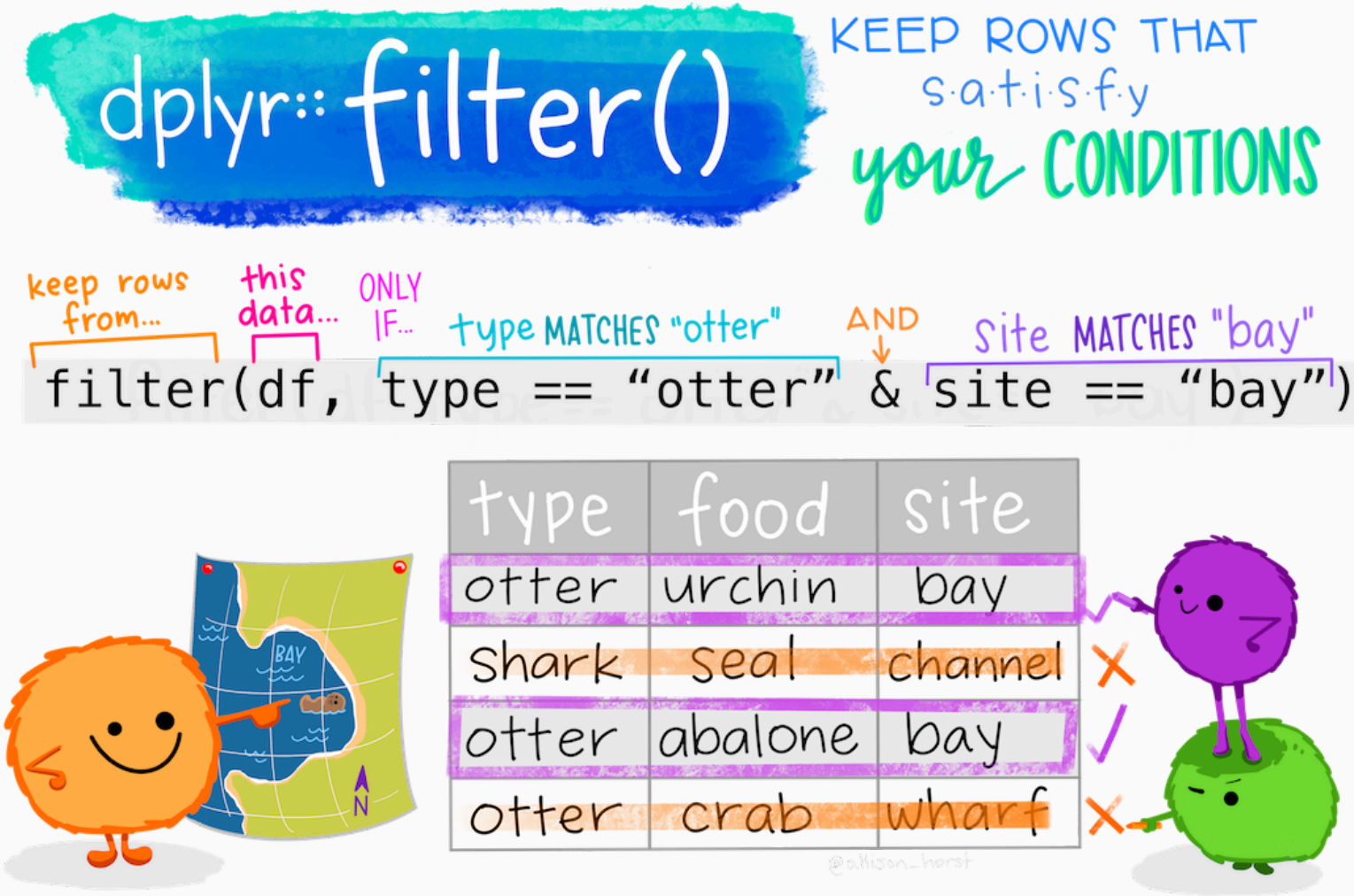
```
prop.table(table(legosets$category))
```

Using the pipe (|>) operator we can chain these calls in a what is arguably a more readable format:

```
table(legosets$category) |> prop.table()
```

```
##
##        Book  Collection    Extended        Gear      Normal       Other
## 0.034818805 0.030509305 0.031586680 0.156513222 0.677864838 0.064985309
##      Random
## 0.003721841
```

Filter


dplyr::filter()
KEEP ROWS THAT satisfy your CONDITIONS
keep rows from...
this data...
ONLY IF...
type MATCHES "otter"
AND
site MATCHES "bay"
filter(df, type == "otter" & site == "bay")
BAY
N
type food site
otter urchin bay
shark seal channel
otter abalone bay
otter crab wharf
@allison_horst

Logical Operators

- `!a` - TRUE if a is FALSE
- `a == b` - TRUE if a and be are equal
- `a != b` - TRUE if a and b are not equal
- `a > b` - TRUE if a is larger than b, but not equal
- `a >= b` - TRUE if a is larger or equal to b
- `a < b` - TRUE if a is smaller than be, but not equal
- `a <= b` - TRUE if a is smaller or equal to b
- `a %in% b` - TRUE if a is in b where b is a vector

```
which( letters %in% c('a','e','i','o','u') )
```

```
## [1]  1  5  9 15 21
```

- `a | b` - TRUE if a *or* b are TRUE
- `a & b` - TRUE if a *and* b are TRUE
- `isTRUE(a)` - TRUE if a is TRUE

Filter

dplyr

```
mylego <- legosets %>% filter(themeGroup == 'Educational' & year > 2015)
```

Base R

```
mylego <- legosets[legosets$themeGroup == 'Educational' & legosets$year > 2015,]
```

```
nrow(mylego)
```

```
## [1] 103
```

Select

dplyr

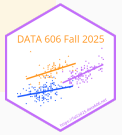
```
mylego <- mylego %>% select(setID, pieces, theme, availability, US_retailPrice, minifigs)
```

Base R

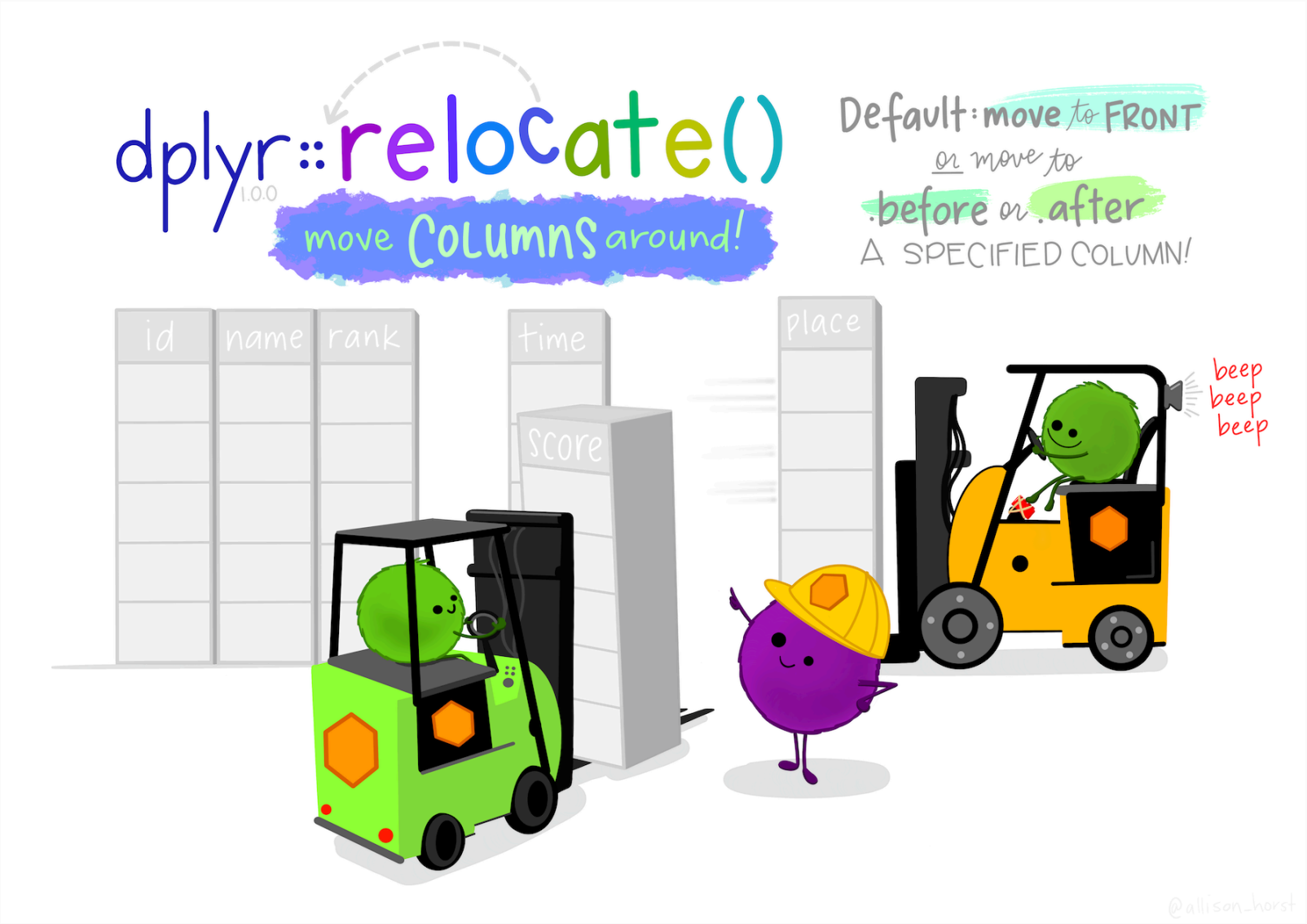
```
mylego <- mylego[,c('setID', 'pieces', 'theme', 'availability', 'US_retailPrice', 'minifigs')]
```

```
head(mylego, n = 4)
```

```
##   setID pieces     theme    availability US_retailPrice minifigs
## 1 26803    109 Education {Not specified}             NA        6
## 2 26277    188 Education     Educational          94.95       NA
## 3 27742    160 Education {Not specified}             NA       NA
## 4 26805   1000 Education {Not specified}             NA       NA
```



Relocate


dplyr::relocate()
1.0.0
move columns around!
Default: move to FRONT
or move to
.before or .after
A SPECIFIED COLUMN!
id
name
rank
time
score
place
beep
beep
beep
@allison_horst

Relocate

dplyr

```
mylego %>% relocate(where(is.numeric), .after = where(is.character)) %>% head(n = 3)
```

```
##       theme    availability setID pieces US_retailPrice minifigs
## 1 Education {Not specified} 26803    109             NA        6
## 2 Education     Educational 26277    188          94.95       NA
## 3 Education {Not specified} 27742    160             NA       NA
```

Base R

```
mylego2 <- mylego[,c('theme', 'availability', 'setID', 'pieces', 'US_retailPrice', 'minifigs')]
head(mylego2, n = 3)
```

```
##       theme    availability setID pieces US_retailPrice minifigs
## 1 Education {Not specified} 26803    109             NA        6
## 2 Education     Educational 26277    188          94.95       NA
## 3 Education {Not specified} 27742    160             NA       NA
```

Rename

dplyr::rename()

RENAME COLUMNS*

df %>% rename(lair = site)

~~species~~ nemesis	status	~~site~~ lair
narwhal	unknown	ocean
chicken	active	coop
pika	active	mountain

*See **rename_with()** to rename using a function.

Rename

dplyr

```
mylego %>% dplyr::rename(USD = US_retailPrice) %>% head(n = 3)
```

```
##   setID pieces     theme    availability   USD minifigs
## 1 26803    109 Education {Not specified}    NA        6
## 2 26277    188 Education     Educational 94.95       NA
## 3 27742    160 Education {Not specified}    NA       NA
```

Base R

```
names(mylego2)[5] <- 'USD'
head(mylego2, n = 3)
```

```
##       theme    availability setID pieces   USD minifigs
## 1 Education {Not specified} 26803    109    NA        6
## 2 Education     Educational 26277    188 94.95       NA
## 3 Education {Not specified} 27742    160    NA       NA
```

Mutate


dplyr::mutate
add Column(s),
keep existing.
A
B
C
1
5
10
0
2
-3
4
9
6
3
2
14
9
8
THE PLAN:
A
B
C
+
=
HORST '18

Mutate

dplyr

```
mylego %>% filter(!is.na(pieces) & !is.na(US_retailPrice)) %>%
    mutate(Price_per_piece = US_retailPrice / pieces) %>% head(n = 3)
```

```
##   setID pieces     theme availability US_retailPrice minifigs Price_per_piece
## 1 26277    188 Education  Educational          94.95       NA       0.5050532
## 2 25949    280 Education  Educational         224.95       NA       0.8033929
## 3 25954      1 Education  Educational          14.95       NA      14.9500000
```

Base R

```
mylego2 <- mylego[!is.na(mylego$US_retailPrice) & !is.na(mylego$Price_per_piece),]
mylego2$Price_per_piece <- mylego2$Price_per_piece / mylego2$US_retailPrice
head(mylego2, n = 3)
```

```
## [1] setID           pieces          theme           availability
## [5] US_retailPrice  minifigs        Price_per_piece
## <0 rows> (or 0-length row.names)
```

Group By and Summarize

```
legosets %>% group_by(themeGroup) %>% summarize(mean_price = mean(US_retailPrice, na.rm = TRUE),
                                                sd_price = sd(US_retailPrice, na.rm = TRUE),
                                                median_price = median(US_retailPrice, na.rm = TRUE),
                                                n = n(),
                                                missing = sum(is.na(US_retailPrice)))
```

```
## # A tibble: 17 × 6
##    themeGroup       mean_price sd_price median_price     n missing
##    <chr>                 <dbl>    <dbl>        <dbl> <int>   <int>
##  1 Action/Adventure       41.9     40.8         30.0  1551     813
##  2 Art and crafts         37.7     50.5         20.0   100       9
##  3 Basic                  22.5     19.6         15.0   879     734
##  4 Constraction           16.4     12.4         13.0   502     284
##  5 Educational           184.     188.          138.    520     482
##  6 Girls                  35.8     24.0         23.0   240     227
##  7 Historical             34.2     32.4         20.0   473     400
##  8 Junior                 22.0     10.1         20.0   228     165
##  9 Licensed               54.0     70.9         30.0  3054    1160
## 10 Miscellaneous          21.6     30.4         13.0  6657    4221
## 11 Model making           78.2     94.1         40.0   832     401
## 12 Modern day             39.0     36.1         30.0  2577    1567
## 13 Pre-school             31.0     22.6         25.0  1587    1108
## 14 Racing                 26.8     26.5         15.0   270     176
```

Describe and Describe By

```
library(psych)
describe(legosets$US_retailPrice)
```

```
##    vars    n  mean    sd median trimmed   mad  min    max range skew kurtosis
## X1    1 8027 40.33 57.36  19.99   28.86 17.79 1.49 849.99 848.5 5.05    40.51
##      se
## X1 0.64
```

```
describeBy(legosets$US_retailPrice, group = legosets$availability, mat = TRUE, skew = FALSE)
```

```
##      item                          group1 vars    n      mean        sd median   min    max range         se
## X11     1                {Not specified}    1 1904  27.15558  39.42703  19.99  1.49 789.99 788.5  0.9035676
## X12     2                     Educational    1   12 217.03333 108.17617 232.45 14.95 399.95 385.0 31.2277699
## X13     3 Gift with Purchase at LEGO.com    1    0       NaN        NA     NA   Inf   -Inf  -Inf         NA
## X14     4                  LEGO exclusive    1 1129  60.45079 108.56718  14.99  1.99 849.99 848.0  3.2311088
## X15     5              LEGOLAND exclusive    1    2   4.99000   0.00000   4.99  4.99   4.99   0.0  0.0000000
## X16     6                        Not sold    1    1  12.99000        NA  12.99 12.99  12.99   0.0         NA
## X17     7                     Promotional    1    5   4.79000   0.83666   4.99  3.99   5.99   2.0  0.3741657
## X18     8           Promotional (Airline)    1    0       NaN        NA     NA   Inf   -Inf  -Inf         NA
## X19     9                          Retail    1 4658  38.88535  39.14642  26.99  1.99 699.99 698.0  0.5735778
## X110   10              Retail - limited    1  315  63.48270  69.97310  39.99  2.49 449.99 447.5  3.9425374
## X111   11                       Unknown    1    1   3.99000        NA   3.99  3.99   3.99   0.0         NA
```

Additional Resources

For data wrangling:

- `dplyr` website: https://dplyr.tidyverse.org
- R for Data Science book: https://r4ds.had.co.nz/wrangle-intro.html
- Wrangling penguins tutorial: https://allisonhorst.shinyapps.io/dplyr-learnr/#section-welcome
- Data transformation cheat sheet: https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

One Minute Paper

1. What was the most important thing you learned during this class?
2. What important question remains unanswered for you?

https://forms.gle/JN16YgSegWVR2csc7