Data Validation in R

From Principles to Tools and Packages

Dr Caterina Constantinescu

Tesco Bank

October 21, 2020

About me

caterina.constantinescu@gmail.com
https://datapowered.io/
@c__constantine
CaterinaC

- Data scientist @ Tesco Bank
- Background in psychology
- R user since 2012
- Ex-EdinbR organiser: @edinb_r EdinburghRusers
- Why data validation?

Outline

Outline

What is data validation?

- Checking the quality of source data before importing or otherwise processing it
- More about the formal aspects of the data, rather than knowing it is necessarily 'correct'

What is data validation?

- Checking the quality of source data before importing or otherwise processing it
- More about the formal aspects of the data, rather than knowing it is necessarily 'correct'

...and can you tell if your data is valid?

- Data validation $\neq$ Data validity

What is data validation?

- Checking the quality of source data before importing or otherwise processing it
- More about the formal aspects of the data, rather than knowing it is necessarily 'correct'

...and can you tell if your data is valid?

- Data validation $\neq$ Data validity
  - Data validity often can't be assessed formally
  - Classical Test Theory: `Observed score = True score + Error`

What is data validation?

- Checking the quality of source data before importing or otherwise processing it
- More about the formal aspects of the data, rather than knowing it is necessarily 'correct'

...and can you tell if your data is valid?

- Data validation $\neq$ Data validity
  - Data validity often can't be assessed formally
  - Classical Test Theory: `Observed score = True score + Error`
- Same as data cleaning? Not quite...
  - One-off, quick & dirty, fixes in the here and now, vs. robust, 'future-proofed' code that can anticipate the ways in which future data might be problematic
  - Flagging issues vs actually fixing them

Outline


WHEN YOU HAVE A CODING STANDARD TO IMPOSE
AND YOU HAVEN'T TOLD ANYONE FOR 5MIN.
memegenerator.net

Principles: Data validation

Steve McConnell's *Code complete* (`Defensive programming`)

Principles: Data validation

Steve McConnell's *Code complete* (`Defensive programming`)

- Garbage in, garbage out (aka user beware)? Maybe not.

Principles: Data validation

Steve McConnell's *Code complete* (`Defensive programming`)

- Garbage in, garbage out (aka user beware)? Maybe not.
- A good program never puts out garbage, regardless what it takes in.

Principles: Data validation

Steve McConnell's *Code complete* (`Defensive programming`)

- Garbage in, garbage out (aka user beware)? Maybe not.
- A good program never puts out garbage, regardless what it takes in.
- Alternatives:
  - Garbage in, nothing out
  - Garbage in, error message out
  - No garbage allowed in

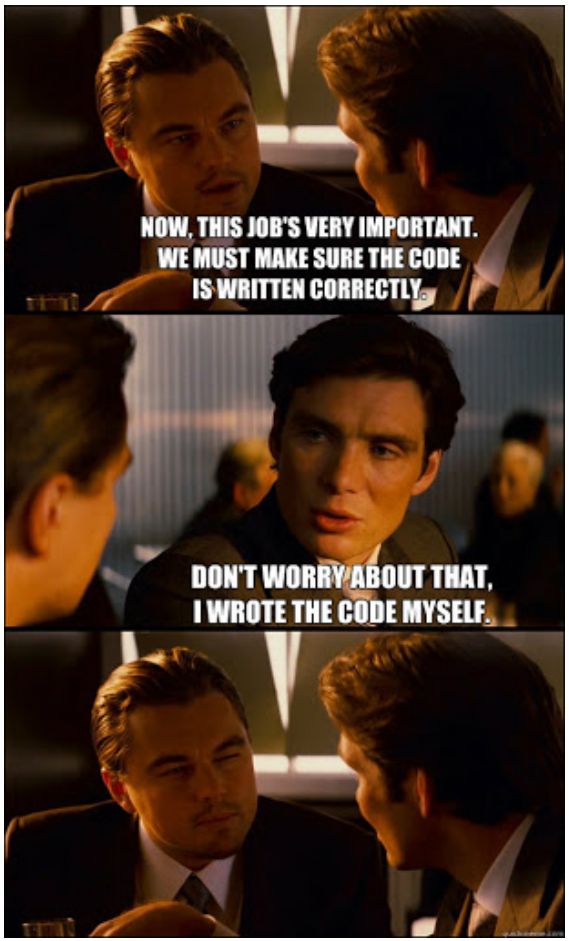
You can't make me...

Story time

Healthcare records

Story time

Date formats



Principles: Data validation & defensive programming

- Decide how to handle bad inputs

Principles: Data validation & defensive programming

- Decide how to handle bad inputs
  - Should the offending situation trigger an abrupt halt?
  - Should it be handled elegantly and the processing be allowed to continue?

Principles: Data validation & defensive programming

- Decide how to handle bad inputs
  - Should the offending situation trigger an abrupt halt?
  - Should it be handled elegantly and the processing be allowed to continue?
- Error Handling (`tryCatch()`, `purrr::safely()`)

Principles: Data validation & defensive programming

- Decide how to handle bad inputs
  - Should the offending situation trigger an abrupt halt?
  - Should it be handled elegantly and the processing be allowed to continue?
- Error Handling (`tryCatch()`, `purrr::safely()`)
  - Replace illegal value with a 'neutral' value (zero/missing)
  - Substitute with closest legal value (e.g., winsorize)
  - Any of the above & append issue to a log that you can then make available in some way (if appropriate)
  - Display an error message to the user
  - Return previous answer
  - System halt or shutdown

Principles: Data validation & defensive programming

- Decide how to handle bad inputs
  - Should the offending situation trigger an abrupt halt?
  - Should it be handled elegantly and the processing be allowed to continue?
- Error Handling (`tryCatch()`, `purrr::safely()`)
  - Replace illegal value with a 'neutral' value (zero/missing)
  - Substitute with closest legal value (e.g., winsorize)
  - Any of the above & append issue to a log that you can then make available in some way (if appropriate)
  - Display an error message to the user
  - Return previous answer
  - System halt or shutdown
- Assertions (to follow)

Principles: Data validation & defensive programming

- Decide how to handle bad inputs
  - Should the offending situation trigger an abrupt halt?
  - Should it be handled elegantly and the processing be allowed to continue?
- Error Handling (`tryCatch()`, `purrr::safely()`)
  - Replace illegal value with a 'neutral' value (zero/missing)
  - Substitute with closest legal value (e.g., winsorize)
  - Any of the above & append issue to a log that you can then make available in some way (if appropriate)
  - Display an error message to the user
  - Return previous answer
  - System halt or shutdown
- Assertions (to follow)
- When to use each?

Principles: Data validation & defensive programming

The Code Complete way:

Principles: Data validation & defensive programming

The Code Complete way:

- Error-handling
  - Checks for bad data
  - For conditions you *expect to occur* and *can foresee* (usually a result of bad input data)
  - Allows the program responds gracefully
  - Often suitable for data coming in from external (less trusted) sources.

Principles: Data validation & defensive programming

The Code Complete way:

- Error-handling
  - Checks for bad data
  - For conditions you *expect to occur* and *can foresee* (usually a result of bad input data)
  - Allows the program responds gracefully
  - Often suitable for data coming in from external (less trusted) sources.
- Assertions
  - Checks for bugs / flushing out errors
  - For conditions that should *never occur*
  - Document any assumptions about what is normal for a function's pre- and post-conditions
  - Corrective action is to alter source code and release a new version
  - Suitable for data coming in from internal, trusted sources, which have informed software design.

Principles: Data validation & defensive programming

The Code Complete way:

- Error-handling
  - Checks for bad data
  - For conditions you *expect to occur* and *can foresee* (usually a result of bad input data)
  - Allows the program responds gracefully
  - Often suitable for data coming in from external (less trusted) sources.
- Assertions
  - Checks for bugs / flushing out errors
  - For conditions that should *never occur*
  - Document any assumptions about what is normal for a function's pre- and post-conditions
  - Corrective action is to alter source code and release a new version
  - Suitable for data coming in from internal, trusted sources, which have informed software design.
- Choice also depends on a correctness vs robustness trade-off.

Principles: Data validation & defensive programming

The Code Complete way:

- Error-handling
  - Checks for bad data
  - For conditions you *expect to occur* and *can foresee* (usually a result of bad input data)
  - Allows the program responds gracefully
  - Often suitable for data coming in from external (less trusted) sources.
- Assertions
  - Checks for bugs / flushing out errors
  - For conditions that should *never occur*
  - Document any assumptions about what is normal for a function's pre- and post-conditions
  - Corrective action is to alter source code and release a new version
  - Suitable for data coming in from internal, trusted sources, which have informed software design.
- Choice also depends on a correctness vs robustness trade-off.
- **For highly robust code, assert and then handle the error anyway.**

Principles: Data validation & defensive programming

Principles: Data validation & defensive programming

The data science way:

- Error-handling
  - For conditions you *can't control* (but can potentially foresee, e.g., packages on which you depend may change; resources may be temporarily unavailable, wrong file encoding etc).
- Assertions
  - For conditions that should *not occur* - but can use this for untrusted data sources especially (not just flushing out errors).
  - Still act as documentation to make explicit any assumptions about what is normal
  - Not necessarily indicators that a new release is needed.
- Choice also depends on a correctness vs robustness trade-off.
- **For highly robust code, assert and then handle the error anyway.**

When/where to validate?

- Usually, as early as possible, although that could be problematic in some cases (e.g., when to `janitor::clean_names()`?)

When/where to validate?

- Usually, as early as possible, although that could be problematic in some cases (e.g., when to `janitor::clean_names()`?)
- Key decision is where to set up 'barricades' between a safe vs non-safe zone
  - May not be so obvious if sanitizing inputs needs to happen at multiple levels... e.g., formal checks of names, dims, vs anything more subtle like overlap in keys between multiple datasets

When/where to validate?

- Usually, as early as possible, although that could be problematic in some cases (e.g., when to `janitor::clean_names()`?)
- Key decision is where to set up 'barricades' between a safe vs non-safe zone
  - May not be so obvious if sanitizing inputs needs to happen at multiple levels... e.g., formal checks of names, dims, vs anything more subtle like overlap in keys between multiple datasets
- Relationship between barricades and assertions (the Code Complete way:)
  - Error handling outside the barricades, as less safe to make assumptions about the data
  - Assertions inside the barricade, as the data there should already be sanitised when it reaches them. Hence, any issues are more likely to mean the code itself is buggy, rather than that there are errors in the data (beware of domain-specific/data science differences)

Outcomes when failing validation tests

Fix or flag an issue?

- *Offensive programming* helpful here, especially in development: makes problems very noticeable. i.e., fail hard when an issue is encountered.

Outcomes when failing validation tests

Fix or flag an issue?

- *Offensive programming* helpful here, especially in development: makes problems very noticeable. i.e., fail hard when an issue is encountered.

- Can fix (silently even), e.g., duplicated columns or cases, unnecessary columns, strip special characters.

Outline

Validation criteria

1. Cell
   - Class (e.g., integer, numeric, character)
   - Range (e.g., A number between 35-40)
   - Fixed set of legal values
   - Date format

Validation criteria

1. Cell
   - Class (e.g., integer, numeric, character)
   - Range (e.g., A number between 35-40)
   - Fixed set of legal values
   - Date format
2. Column
   - Means, SDs, Outlier detection
   - No missing values/ max number of NAs
   - Uniqueness for cross-sectional data (e.g., patient ID)
   - Consistent expressions (e.g., using one of St., Str, Street)

Validation criteria

1. Cell
   - Class (e.g., integer, numeric, character)
   - Range (e.g., A number between 35-40)
   - Fixed set of legal values
   - Date format
2. Column
   - Means, SDs, Outlier detection
   - No missing values/ max number of NAs
   - Uniqueness for cross-sectional data (e.g., patient ID)
   - Consistent expressions (e.g., using one of St., Str, Street)
3. Row
   - Equality or summing up to constant across rows, or multivariate distance

Validation criteria

1. Cell
   - Class (e.g., integer, numeric, character)
   - Range (e.g., A number between 35-40)
   - Fixed set of legal values
   - Date format
2. Column
   - Means, SDs, Outlier detection
   - No missing values/ max number of NAs
   - Uniqueness for cross-sectional data (e.g., patient ID)
   - Consistent expressions (e.g., using one of St., Str, Street)
3. Row
   - Equality or summing up to constant across rows, or multivariate distance
4. Bivariate/Multivariate
   - Correlations
   - BMI, or when age $<$ 16, isjob_status $==$ "no job"

Validation criteria

1. Cell
   - Class (e.g., integer, numeric, character)
   - Range (e.g., A number between 35-40)
   - Fixed set of legal values
   - Date format
2. Column
   - Means, SDs, Outlier detection
   - No missing values/ max number of NAs
   - Uniqueness for cross-sectional data (e.g., patient ID)
   - Consistent expressions (e.g., using one of St., Str, Street)
3. Row
   - Equality or summing up to constant across rows, or multivariate distance
4. Bivariate/Multivariate
   - Correlations
   - BMI, or when age $<$ 16, isjob_status $==$ "no job"
5. Full dataset
   - Consistent column names & types
   - Expected dimensions (nrows/ncols)

Validation criteria

1. Cell
   - Class (e.g., integer, numeric, character)
   - Range (e.g., A number between 35-40)
   - Fixed set of legal values
   - Date format
2. Column
   - Means, SDs, Outlier detection
   - No missing values/ max number of NAs
   - Uniqueness for cross-sectional data (e.g., patient ID)
   - Consistent expressions (e.g., using one of St., Str, Street)
3. Row
   - Equality or summing up to constant across rows, or multivariate distance
4. Bivariate/Multivariate
   - Correlations
   - BMI, or when age $<$ 16, isjob_status $==$ "no job"
5. Full dataset
   - Consistent column names & types
   - Expected dimensions (nrows/ncols)
6. Cross datasets
   - If merges are required, correct overlap between keys

Outline

Example: Anscombe's quartet

`datasets::anscombe`

	x1	x2	x3	x4	y1	y2	y3	y4
1	10	10	10	8	8.04	9.14	7.46	6.58
2	8	8	8	8	6.95	8.14	6.77	5.76
3	13	13	13	8	7.58	8.74	12.74	7.71
4	9	9	9	8	8.81	8.77	7.11	8.84
5	11	11	11	8	8.33	9.26	7.81	8.47
6	14	14	14	8	9.96	8.10	8.84	7.04
7	6	6	6	8	7.24	6.13	6.08	5.25
8	4	4	4	19	4.26	3.10	5.39	12.50
9	12	12	12	8	10.84	9.13	8.15	5.56
10	7	7	7	8	4.82	7.26	6.42	7.91
11	5	5	5	8	5.68	4.74	5.73	6.89

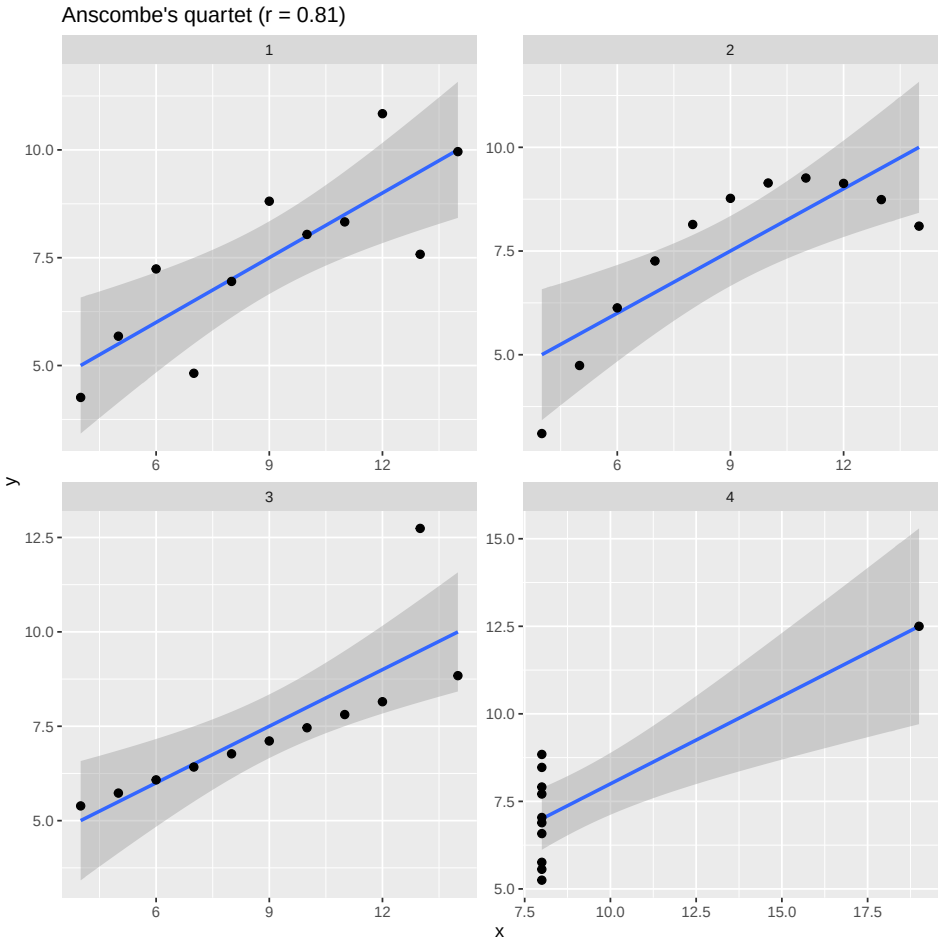
Example: Anscombe's quartet

```
Description
Four x-y datasets which have the same traditional statistical
properties (mean, variance, correlation, regression line, etc.),
yet are quite different.

Format
A data frame with 11 observations on 8 variables.

x1 == x2 == x3 the integers 4:14, specially arranged
x4 values 8 and 19
y1, y2, y3, y4 numbers in (3, 12.5) with mean 7.5 and sdev 2.03
```

Example: Anscombe's quartet



validate

validate

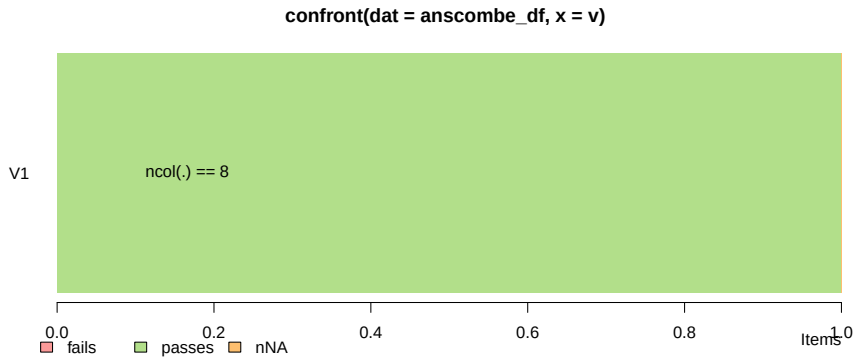
```
1  library(validate)
2  library(magrittr)
3
4  anscombe_df <- anscombe
5
6  anscombe_df %>%
7    check_that(
8        nrow(.) > 100 # Ooops!
9      , ncol(.) == 8
10     , is.numeric(x1)
11     , x1 == x2
12     , x1 == x3
13     , x2 == x4 # Ooops again!
14     , x4 %vin% c(8, 19)
15     , is_complete(x1, x2)
16     , cor1 := cor(x1, y1) %>% round(2)
17     , cor2 := cor(x2, y2) %>% round(2)
18     , cor1 == cor2
19     , m_y1 := mean(y1) %>% round(2)
20     , m_y1 == 7.5
21     , y1_sd := sd(y1) %>% round(2)
22     , y1_sd == 2.03) %>%
23   summary()
```

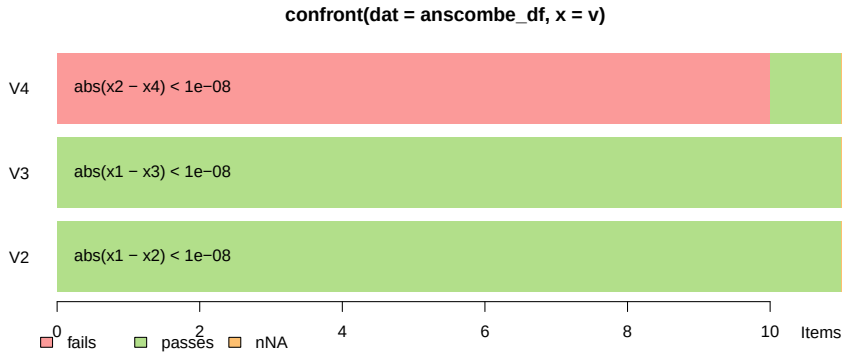
validate

	name	items	passes	fails	nNA	error	warning	expression
1	V01	1	0	1	0	FALSE	FALSE	nrow(.) > 100
2	V02	1	1	0	0	FALSE	FALSE	ncol(.) == 8
3	V03	1	1	0	0	FALSE	FALSE	is.numeric(x1)
4	V04	11	11	0	0	FALSE	FALSE	abs(x1 - x2) < 1e-08
5	V05	11	11	0	0	FALSE	FALSE	abs(x1 - x3) < 1e-08
6	V06	11	1	10	0	FALSE	FALSE	abs(x2 - x4) < 1e-08
7	V07	11	11	0	0	FALSE	FALSE	x4 %vin% c(8, 19)
8	V08	11	11	0	0	FALSE	FALSE	is_complete(x1, x2)
9	V11	1	1	0	0	FALSE	FALSE	cor(x1, y1) %>% round(2) == cor(x2, y2)...
10	V13	1	1	0	0	FALSE	FALSE	mean(y1) %>% round(2) == 7.5
11	V15	1	1	0	0	FALSE	FALSE	sd(y1) %>% round(2) == 2.03

validate

```
1  v <- validator( ncol(.) == 8
2                , x1 == x2
3                , x1 == x3
4                , x2 == x4 )
5  cf <- confront(anscombe_df, v)
6  plot(cf)
```





assertr

assertr + assertive

```
1  library(assertr)
2  library(assertive)
3
4  anscombe_df %>%
5    # whole df
6    verify(has_all_names("x1", "x2", "x3", "x4", "y1", "y2", "y3", "y4")) %>%
7    verify(nrow(.) == 11) %>%
8    verify(ncol(.) == 8) %>%
9    # specific values:
10   verify(x1 == x2) %>%
11   verify(x2 == x3) %>%
12   verify(is.numeric(x1)) %>%
13   verify(mean(y1) %>% round(2) == 7.5) %>%
14   verify(sd(y1) %>% round(2) == 2.03) %>%
15   verify(cor(x1, y1) %>% round(2) == cor(x2, y2) %>% round(2) ) %>%
16   # by column:
17   assert(within_bounds(4, 14), x1:x2) %>%
18   assert(in_set(8, 19), x4) %>%
19   insist(within_n_sds(4), x1) %>%
20   # by row:
21   assert_rows(num_row_NAs, in_set(0), everything()) %>%
22   assert_rows(col_concat, is_uniq, x1:x4) %>%
23   insist_rows(maha_dist, within_n_mads(10), everything()) %>%
24   # extending via assertive:
25   verify(assertive::is_linux())
```

assertr + assertive

	x1	x2	x3	x4	y1	y2	y3	y4
1	10	10	10	8	8.04	9.14	7.46	6.58
2	8	8	8	8	6.95	8.14	6.77	5.76
3	13	13	13	8	7.58	8.74	12.74	7.71
4	9	9	9	8	8.81	8.77	7.11	8.84
5	11	11	11	8	8.33	9.26	7.81	8.47
6	14	14	14	8	9.96	8.10	8.84	7.04
7	6	6	6	8	7.24	6.13	6.08	5.25
8	4	4	4	19	4.26	3.10	5.39	12.50
9	12	12	12	8	10.84	9.13	8.15	5.56
10	7	7	7	8	4.82	7.26	6.42	7.91
11	5	5	5	8	5.68	4.74	5.73	6.89

assertr + assertive

```
1  anscombe_df %>%
2    chain_start() %>%
3    # whole df
4    verify(has_all_names("x1", "x2", "x3", "x4", "y1", "y2", "y3", "y4")) %>%
5    verify(nrow(.) == 12) %>% # Ooops!
6    verify(ncol(.) == 8) %>%
7    # specific values:
8    verify(x1 == x2) %>%
9    verify(x2 == x3) %>%
10   verify(is.numeric(x1)) %>%
11   verify(mean(y1) %>% round(2) == 7.5) %>%
12   verify(sd(y1) %>% round(2) == 2.03) %>%
13   verify(cor(x1, y1) %>% round(2) == cor(x2, y2) %>% round(2) ) %>%
14   # by column:
15   assert(within_bounds(4, 14), x1:x2) %>%
16   assert(in_set(8, 19), x4) %>%
17   insist(within_n_sds(4), x1) %>%
18   # by row:
19   assert_rows(num_row_NAs, in_set(0), everything()) %>%
20   assert_rows(col_concat, is_uniq, x1:x4) %>%
21   insist_rows(maha_dist, within_n_mads(10), everything()) %>%
22   # extras:
23   verify(assertive::is_windows()) %>% # Nope !
24   chain_end()
```

assertr + assertive

```
There are 2 errors across 2 verbs:
-
    verb redux_fn                predicate column index value
1 verify       NA           nrow(.) == 12     NA     1    NA
2 verify       NA assertive::is_windows()     NA     1    NA

Error: assertr stopped execution
```

ensurer

ensurer

```
1  library(ensurer)
2  library(data.table)
3
4  # Set rules & data template
5  ensure_df_dt <- ensures_that(any(is.data.frame(.), is.data.table(.)),
6                               ! any(is.na(.)))
7
8  anscombe_tmpl <- anscombe_df[FALSE, ]
9  ensure_as_template <- function(x, tpl){
10   ensure_that(x,
11               identical(names(.), names(tpl)),
12               identical(sapply(., class), sapply(tpl, class)),
13               err_desc = "Please check column names and types, and try again.")}
14
15 # Apply to data
16 anscombe_df %>%
17   ensure_df_dt %>%
18   ensure_as_template(anscombe_tmpl) %>%
19   ensure_that(nrow(.) == 11,
20               ncol(.) == 8) %>%
21   # continuing with assertr chain:
22   verify(x1 == x2) %>%
23   verify(x2 == x3) %>%
24   verify(mean(y1) %>% round(2) == 7.5)
```

ensurer

	x1	x2	x3	x4	y1	y2	y3	y4
1	10	10	10	8	8.04	9.14	7.46	6.58
2	8	8	8	8	6.95	8.14	6.77	5.76
3	13	13	13	8	7.58	8.74	12.74	7.71
4	9	9	9	8	8.81	8.77	7.11	8.84
5	11	11	11	8	8.33	9.26	7.81	8.47
6	14	14	14	8	9.96	8.10	8.84	7.04
7	6	6	6	8	7.24	6.13	6.08	5.25
8	4	4	4	19	4.26	3.10	5.39	12.50
9	12	12	12	8	10.84	9.13	8.15	5.56
10	7	7	7	8	4.82	7.26	6.42	7.91
11	5	5	5	8	5.68	4.74	5.73	6.89

ensurer

```
anscombe_df %>%
  ensure_df_dt %>%
  ensure_as_template(anscombe_tmpl) %>%
  ensure_that(all(. > 200), err_desc = "Case(s) under 200!", # Oops !
              nrow(.) == 11,
              ncol(.) == 9) %>% # Oops again !
  # continuing with assertr chain:
  verify(x1 == x2) %>%
  verify(x2 == x3) %>%
  verify(mean(y1) %>% round(2) == 7.5)
```

ensurer

```
Error: conditions failed for call 'anscombe_df %>%
ensure_df_dt %>% ensure_as_template(anscombe_tmpl) %>% ':
 * all(. > 200)
 * ncol(.) == 9
 Description: Case(s) under 200!
```

checkr

checkr

```
1  library(checkr)
2
3  new_copy <- anscombe_df %>%
4    check_colnames(names(anscombe_tmpl), exclusive = TRUE, error = TRUE) %>%
5    check_ncol(ncol = 8, error = TRUE) %>%
6    check_nrow(nrow = 11, error = TRUE) %>%
7    check_classes("data.frame") # can't easily check individual col classes
8
9  another_new_copy <- check_data(anscombe_df,
10                                values = list(x1 = 10000, x2 = 10000, x3 = c(4, 14), x4 = 10000,
11                                              y1 = 10000, y2 = 10000, y3 = 10000, y4 = 10000),
12                                exclusive = TRUE,
13                                order = TRUE,
14                                nrow = 11L, # no ncol
15                                error = TRUE)
```

checkr

	x1	x2	x3	x4	y1	y2	y3	y4
1	10	10	10	8	8.04	9.14	7.46	6.58
2	8	8	8	8	6.95	8.14	6.77	5.76
3	13	13	13	8	7.58	8.74	12.74	7.71
4	9	9	9	8	8.81	8.77	7.11	8.84
5	11	11	11	8	8.33	9.26	7.81	8.47
6	14	14	14	8	9.96	8.10	8.84	7.04
7	6	6	6	8	7.24	6.13	6.08	5.25
8	4	4	4	19	4.26	3.10	5.39	12.50
9	12	12	12	8	10.84	9.13	8.15	5.56
10	7	7	7	8	4.82	7.26	6.42	7.91
11	5	5	5	8	5.68	4.74	5.73	6.89

checkr

```
1  library(checkr)
2
3  another_new_copy <- check_data(anscombe_df,
4                                 values = list(x1 = 10000, x2 = 10000, x3 = c(4, 10), # Ooops!
5                                               x4 = "",  # Ooops again!
6                                               y1 = 10000, y2 = 10000, y3 = 10000, y4 = 10000),
7                                 exclusive = TRUE,
8                                 order = TRUE,
9                                 nrow = 11L, # no ncol
10                                error = TRUE)
11
12 # Error: the values in column x3 of anscombe_df must lie between 4 and 10
13
14 # check_rbind(datasets::mtcars, datasets::mtcars)
15 # check_join(data1, data2, by = c(x = "y"), error = FALSE)
16 # check_key()
```

pointblank

pointblank

```
1  library(pointblank)
2
3  create_agent(anscombe_df) %>% # optional !
4    col_exists(names(anscombe)) %>%
5    col_schema_match(col_schema(.tbl = anscombe_tmpl)) %>%
6
7    col_vals_not_null(names(anscombe_tmpl)) %>%
8    col_vals_in_set(x4, c(8, 19)) %>%
9    col_vals_between(x1, 4, 14) %>%
10
11   col_vals_expr(expr(nrow(.) == 11)) %>%
12   col_vals_expr(expr(ncol(.) == 8)) %>%
13
14   col_vals_expr(expr(x1 == x2)) %>%
15   col_vals_expr(expr(x1 == x3)) %>%
16   col_vals_expr(expr(mean(y1) %>% round(2) == 7.5)) %>%
17   col_vals_expr(expr(cor(x1, y1) %>% round(2) == cor(x2, y2) %>% round(2))) %>%
18   interrogate() # optional !
```

pointblank

Pointblank Validation

agent_2020-10-20_20:15:28 (2020-10-20 20:15:29)

	STEP	COLUMNS	VALUES	TBL	EVAL	UNITS	PASS	FAIL	W	S	N	EXT
1	col_exists()	x1	—		✓	1	1 1.00	0 0.00	—	—	—	—
2	col_exists()	x2	—		✓	1	1 1.00	0 0.00	—	—	—	—
3	col_exists()	x3	—		✓	1	1 1.00	0 0.00	—	—	—	—
4	col_exists()	x4	—		✓	1	1 1.00	0 0.00	—	—	—	—
5	col_exists()	y1	—		✓	1	1 1.00	0 0.00	—	—	—	—
6	col_exists()	y2	—		✓	1	1 1.00	0 0.00	—	—	—	—
7	col_exists()	y3	—		✓	1	1 1.00	0 0.00	—	—	—	—
8	col_exists()	y4	—		✓	1	1 1.00	0 0.00	—	—	—	—
9	col_schema_match()	—	SCHEMA R TYPES		✓	1	1 1.00	0 0.00	—	—	—	—
10	col_vals_not_null()	x1	—		✓	11	11 1.00	0 0.00	—	—	—	—
11	col_vals_not_null()	x2	—		✓	11	11 1.00	0 0.00	—	—	—	—

pointblank

16	col_vals_not_null()	y3	—	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
17	col_vals_not_null()	y4	—	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
18	col_vals_in_set()	x4	8, 19	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
19	col_vals_between()	x1	[4, 14]	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
20	col_vals_expr()	—	nrow(.) == 11	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
21	col_vals_expr()	—	ncol(.) == 8	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
22	col_vals_expr()	—	x1 == x2	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
23	col_vals_expr()	—	x1 == x3	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
24	col_vals_expr()	—	mean(y1) %>% r…	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
25	col_vals_expr()	—	cor(x1, y1) %>…	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—

pointblank

```
1  anscombe_df %>%
2  col_exists(names(anscombe_tmpl)) %>%
3  col_schema_match(col_schema(.tbl = anscombe_tmpl)) %>%
4
5  col_vals_not_null(names(anscombe_tmpl)) %>%
6  col_vals_in_set(x4, c(8, 19)) %>%
7  col_vals_between(x1, 4, 10) %>% # Ooops !
8
9  col_vals_expr(expr(nrow(.) == 60)) %>% # Oops again !
10 col_vals_expr(expr(ncol(.) == 8)) %>%
11
12 col_vals_expr(expr(x1 == x2)) %>%
13 col_vals_expr(expr(x1 == x3)) %>%
14 col_vals_expr(expr(mean(y1) %>% round(2) == 7.5)) %>%
15 col_vals_expr(expr(cor(x1, y1) %>% round(2) == cor(x2, y2) %>% round(2)))
```

pointblank

```
 +   col_vals_expr(expr(cor(x1, y1) %>% round(2) == cor(x2, y2) %>% round(2))) # %>%
Error: Exceedance of failed test units where values in ‘x1‘ should have been
between ‘4‘ and ‘10‘.
The ‘col_vals_between()‘ validation failed beyond the absolute threshold level (1).
* failure level (4) >= failure threshold (1)
```

pointblank

If re-adding `create_agent()` + `interrogate()`

16	col_vals_not_null()	y3	—	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
17	col_vals_not_null()	y4	—	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
18	col_vals_in_set()	x4	8, 19	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
19	col_vals_between()	x1	[4, 10]	ℐ	✓	11	7 0.64	4 0.36	—	—	—	CSV
20	col_vals_expr()	—	nrow(.) == 60	ℐ	✓	11	0 0.00	11 1.00	—	—	—	CSV
21	col_vals_expr()	—	ncol(.) == 8	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
22	col_vals_expr()	—	x1 == x2	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
23	col_vals_expr()	—	x1 == x3	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
24	col_vals_expr()	—	mean(y1) %>% r…	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—
25	col_vals_expr()	—	cor(x1, y1) %>…	ℐ	✓	11	11 1.00	0 0.00	—	—	—	—

ruler

ruler

```
1  library(ruler)
2  anscombe_dims_rules <- . %>%
3    dplyr::summarise(nrow = nrow(.) == 11,
4                     ncol = ncol(.) == 8)
5
6  anscombe_na_rules <- . %>% dplyr::summarise(all_not_na = Negate(anyNA)(.))
7
8  anscombe_class_rules <- . %>% dplyr::summarise_at(vars(names(anscombe_df)), rules(is.numeric(.)))
9
10 anscombe_data_pack <- data_packs(data_nrow = anscombe_dims_rules,
11                                  data_na = anscombe_na_rules,
12                                  data_classes = anscombe_class_rules)
13
14 x1_value_rules <- . %>% dplyr::transmute_at(c("x1"), rules(. %in% 4:14))
15 x4_value_rules <- . %>% dplyr::transmute_at(c("x4"), rules(. %in% c(8, 19)))
16 y1_value_rules <- . %>% dplyr::transmute_at(c("y1"), rules(y1_mean = mean(.) %>% round(2) == 7.5))
17
18 anscombe_cell_packs <- cell_packs(x1_test = x1_value_rules,
19                                   x4_test = x4_value_rules,
20                                   y1_test = y1_value_rules)
21
22 anscombe_df %>%
23   expose(anscombe_data_pack, .remove_obeyers = TRUE) %>%
24   expose(anscombe_cell_packs, .remove_obeyers = TRUE) %>% # get_report() %>% # or:
25   assert_any_breaker()
```

ruler

	x1	x2	x3	x4	y1	y2	y3	y4
1	10	10	10	8	8.04	9.14	7.46	6.58
2	8	8	8	8	6.95	8.14	6.77	5.76
3	13	13	13	8	7.58	8.74	12.74	7.71
4	9	9	9	8	8.81	8.77	7.11	8.84
5	11	11	11	8	8.33	9.26	7.81	8.47
6	14	14	14	8	9.96	8.10	8.84	7.04
7	6	6	6	8	7.24	6.13	6.08	5.25
8	4	4	4	19	4.26	3.10	5.39	12.50
9	12	12	12	8	10.84	9.13	8.15	5.56
10	7	7	7	8	4.82	7.26	6.42	7.91
11	5	5	5	8	5.68	4.74	5.73	6.89

ruler

```
1  y1_value_rules <- . %>% dplyr::transmute_at(c("y1"), rules(y1_mean = mean(.) %>% round(2) == 7.8)) # Ooops!
2
3
4  anscombe_cell_packs <- cell_packs(x1_test = x1_value_rules,
5                                    x4_test = x4_value_rules,
6                                    y1_test = y1_value_rules)
7
8
9  anscombe_df %>%
10   expose(anscombe_data_pack, .remove_obeyers = TRUE) %>%
11   expose(anscombe_cell_packs, .remove_obeyers = TRUE) %>%
12   get_report() # or assert_any_breaker()
```

ruler

```
# get_report() or assert_any_breaker()
  Breakers report
Tidy data validation report:
# A tibble: 11 x 5
   pack    rule    var      id value
   <chr>   <chr>   <chr> <int> <lgl>
 1 y1_test y1_mean ""        1 FALSE
 2 y1_test y1_mean ""        2 FALSE
 3 y1_test y1_mean ""        3 FALSE
 4 y1_test y1_mean ""        4 FALSE
 5 y1_test y1_mean ""        5 FALSE
 6 y1_test y1_mean ""        6 FALSE
 7 y1_test y1_mean ""        7 FALSE
 8 y1_test y1_mean ""        8 FALSE
 9 y1_test y1_mean ""        9 FALSE
10 y1_test y1_mean ""       10 FALSE
11 y1_test y1_mean ""       11 FALSE

Error: assert_any_breaker: Some breakers found in exposure.
```

checkmate

checkmate

```
1  library(checkmate)
2
3  anscombe_df %>%
4    assert_data_frame(types = lapply(anscombe_tmpl, class) %>% unlist(),
5                      any.missing = FALSE,
6                      nrows = 11,
7                      ncols = 8) # returns output invisibly if successful
8
9  anscombe_df %>%
10   names() %>%
11   assertNames(permutation.of = names(anscombe_tmpl))
12
13 assertTRUE(with(anscombe_df, identical(x1, x2)))
14 assertSetEqual(anscombe_df$x4, c(8, 19))
15 assertTRUE(with(anscombe_df,
16                 identical(cor(x1, y1) %>% round(2),
17                           cor(x2, y2) %>% round(2))))
```

checkmate

```
1  library(checkmate)
2
3  anscombe_df %>%
4    assert_data_frame(types = lapply(anscombe_tmpl, class) %>% unlist(),
5                      any.missing = FALSE,
6                      nrows = 12,
7                      ncols = 8)
8  # Error in function_list[[k]](value) :
9  #  Assertion on '.' failed: Must have exactly 12 rows, but has 11 rows.
10
11 anscombe_df %>%
12   names() %>%
13   assertNames(permutation.of = names(mtcars))
14 # Error in function_list[[k]](value) :
15 #  Assertion on '.' failed: Must be a permutation of set {mpg,cyl,disp,hp,drat,wt,qsec,vs,am,gear,carb}.
16
17 assertTRUE(with(anscombe_df, identical(x1, x4)))
18 # Error: Assertion on 'with(anscombe_df, identical(x1, x4))' failed: Must be TRUE.
```

Outline

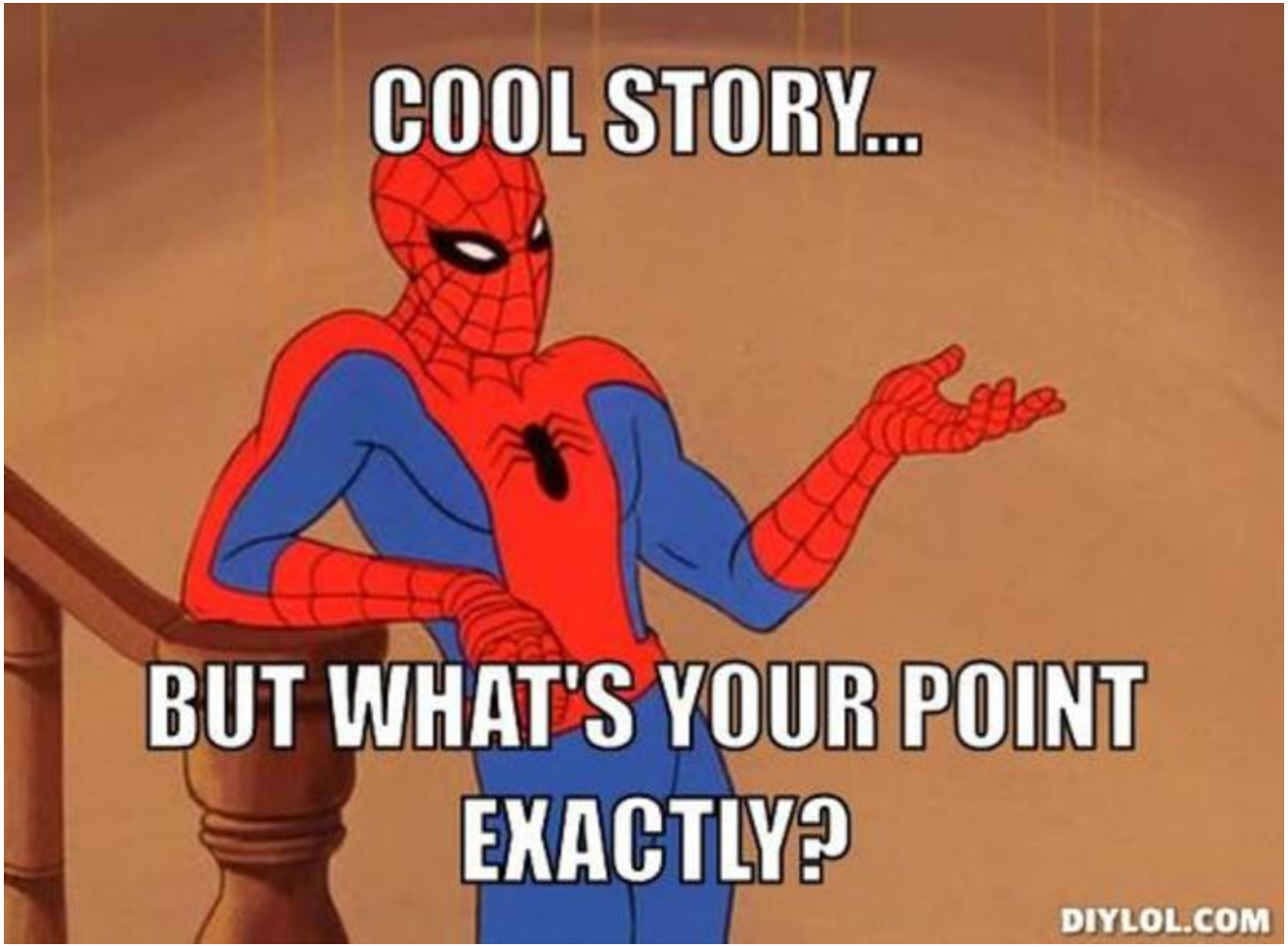
readr::read_csv + readr:::n_problems

```
1  library(readr)
2  inputPath <- "/home/caterina/Desktop/anscombe.csv"
3
4  anscombe_upload <- read_csv(file = inputPath,
5                              col_types = list(
6                                x1 = col_double(), x2 = col_double(), x3 = col_double(), x4 = col_double(),
7                                y1 = col_double(), y2 = col_date(), # !!!
8                                y3 = col_double(), y4 = col_double()))
9
10 if (readr:::n_problems(anscombe_upload) > 0) {
11
12   pbs <- problems(anscombe_upload)
13   surface_parsing_errs <- paste0("Found ",
14                                  nrow(pbs),
15                                  " parsing error(s) in column(s): '",
16                                  paste(unique(pbs$col), collapse = "' & '"),
17                                  "'. Please check the upload, correct any problematic values and try again.")
18   anscombe_upload <- surface_parsing_errs
19
20 } else {
21    anscombe_upload # continue with extra processing, checking, etc
22 }
```

readr::read_csv + readr:::n_problems

```
>  problems(anscombe_upload)
# A tibble: 11 x 5
     row col   expected     actual file
   <int> <chr> <chr>        <chr>  <chr>
 1     1 y2    "date like " 9.14   '/home/caterina/Desktop/anscombe.csv'
 2     2 y2    "date like " 8.14   '/home/caterina/Desktop/anscombe.csv'
 3     3 y2    "date like " 8.74   '/home/caterina/Desktop/anscombe.csv'
 4     4 y2    "date like " 8.77   '/home/caterina/Desktop/anscombe.csv'
 5     5 y2    "date like " 9.26   '/home/caterina/Desktop/anscombe.csv'
 6     6 y2    "date like " 8.1    '/home/caterina/Desktop/anscombe.csv'
 7     7 y2    "date like " 6.13   '/home/caterina/Desktop/anscombe.csv'
 8     8 y2    "date like " 3.1    '/home/caterina/Desktop/anscombe.csv'
 9     9 y2    "date like " 9.13   '/home/caterina/Desktop/anscombe.csv'
10    10 y2    "date like " 7.26   '/home/caterina/Desktop/anscombe.csv'
11    11 y2    "date like " 4.74   '/home/caterina/Desktop/anscombe.csv'
```

Outline


COOL STORY...
BUT WHAT'S YOUR POINT EXACTLY?
DIYLOL.COM

Conclusions

- Discussed packages/tools:
  - `validate`
  - `assertr` (`assertive`)
  - `ensurer`
  - `checkr`
  - `pointblank`
  - `ruler`
  - `checkmate`
  - `read_csv` / `n_problems`

  But... no "one-stop shop" - it's OK/necessary to customise

Conclusions

- Discussed packages/tools:
  - `validate`
  - `assertr` (`assertive`)
  - `ensurer`
  - `checkr`
  - `pointblank`
  - `ruler`
  - `checkmate`
  - `read_csv` / `n_problems`

  But... no "one-stop shop" - it's OK/necessary to customise

- Data validation teaches you to think defensively: code *into* a language, not *in* it

Conclusions

- Discussed packages/tools:
  - `validate`
  - `assertr` (`assertive`)
  - `ensurer`
  - `checkr`
  - `pointblank`
  - `ruler`
  - `checkmate`
  - `read_csv` / `n_problems`

  But... no "one-stop shop" - it's OK/necessary to customise

- Data validation teaches you to think defensively: code *into* a language, not *in* it
- Allows you to build more robust code

Conclusions

- Discussed packages/tools:
  - `validate`
  - `assertr` (`assertive`)
  - `ensurer`
  - `checkr`
  - `pointblank`
  - `ruler`
  - `checkmate`
  - `read_csv` / `n_problems`

  But... no "one-stop shop" - it's OK/necessary to customise

- Data validation teaches you to think defensively: code *into* a language, not *in* it
- Allows you to build more robust code
- Forces you to think more deeply about your assumptions/ check your own understanding of the data

Conclusions

- Discussed packages/tools:
  - `validate`
  - `assertr` (`assertive`)
  - `ensurer`
  - `checkr`
  - `pointblank`
  - `ruler`
  - `checkmate`
  - `read_csv` / `n_problems`

  But... no "one-stop shop" - it's OK/necessary to customise

- Data validation teaches you to think defensively: code *into* a language, not *in* it
- Allows you to build more robust code
- Forces you to think more deeply about your assumptions/ check your own understanding of the data
- Supports collaboration

Thanks!