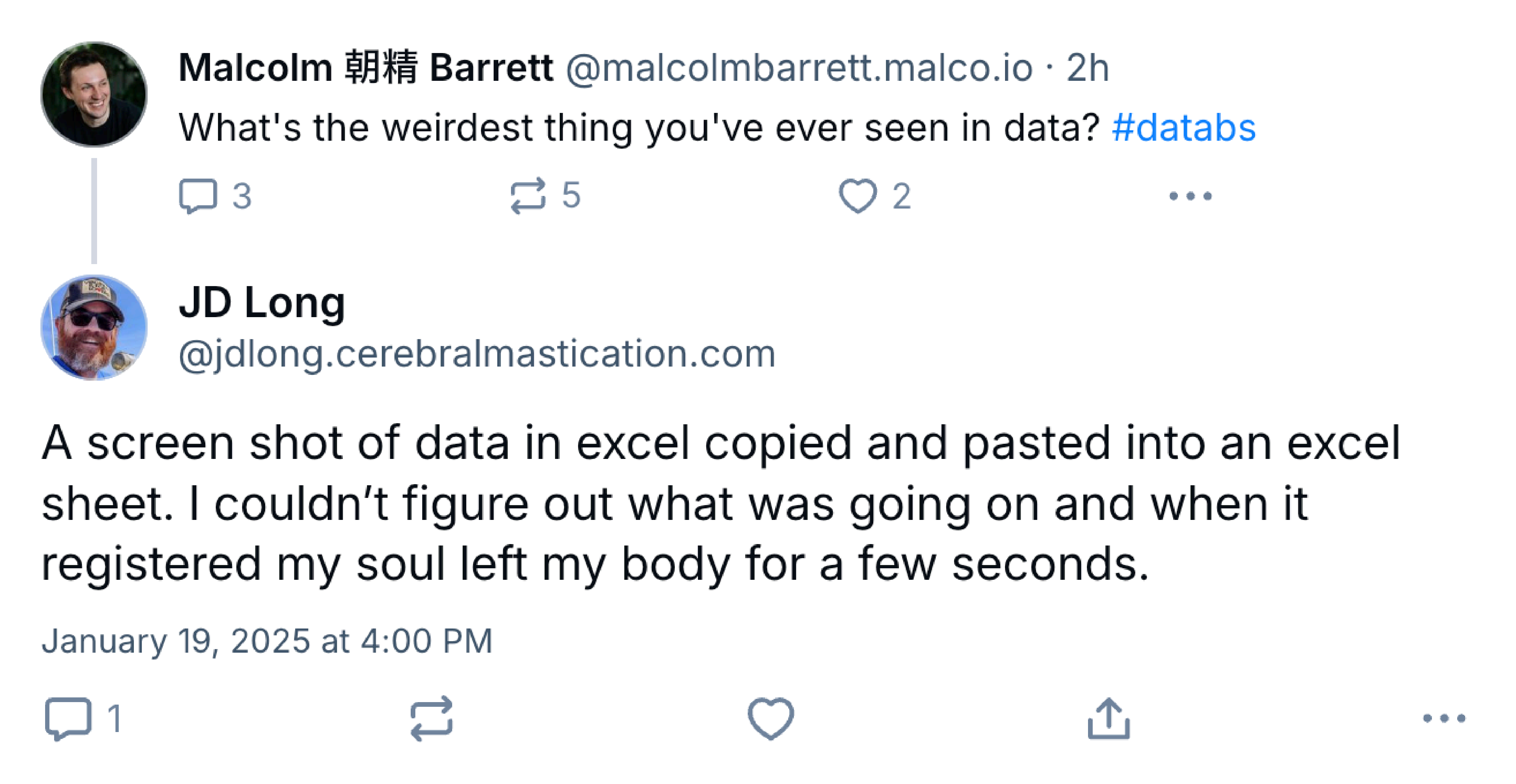
Data validation in R and Python with pointblank

2025-01-21

Data is weird



Malcolm 朝精 Barrett @malcolmbarrett.malco.io · 2h

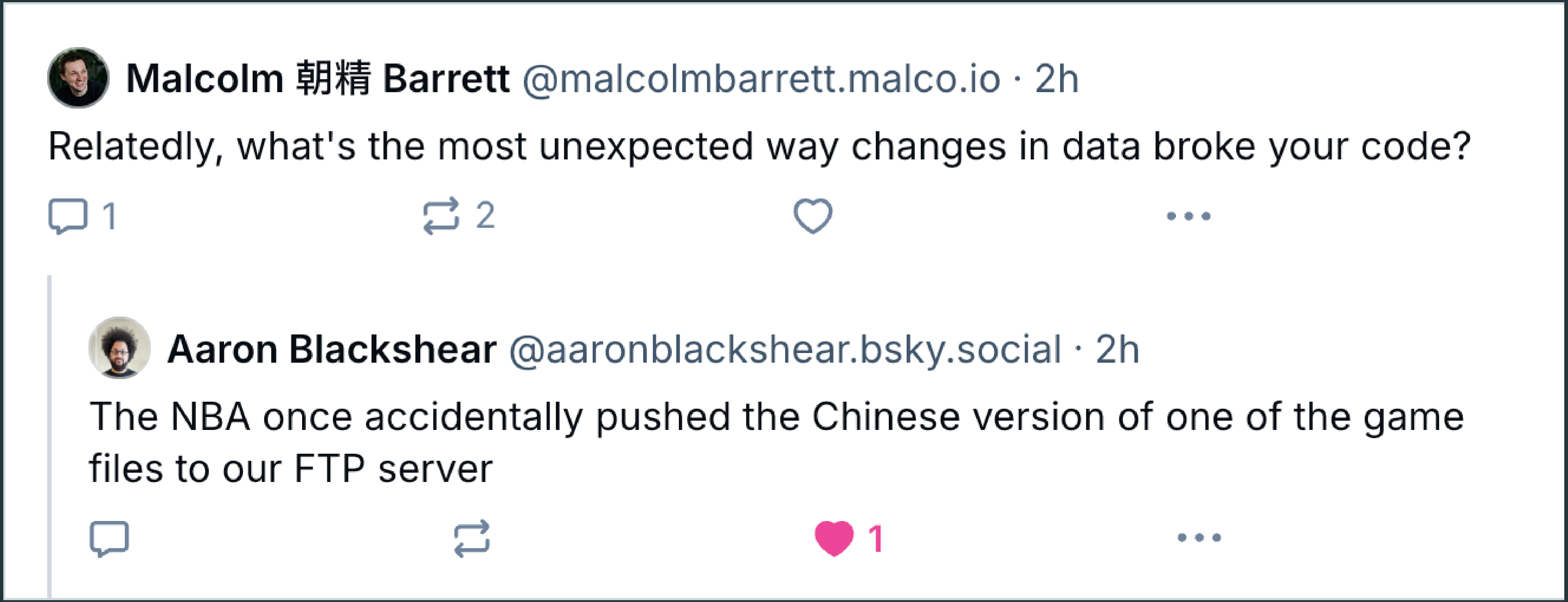
What's the weirdest thing you've ever seen in data? #databs

JD Long
@jdlong.cerebralmastication.com

A screen shot of data in excel copied and pasted into an excel sheet. I couldn't figure out what was going on and when it registered my soul left my body for a few seconds.

January 19, 2025 at 4:00 PM

Data gets weirder



Malcolm 朝精 Barrett @malcolmbarrett.malco.io · 2h

Relatedly, what's the most unexpected way changes in data broke your code?

Aaron Blackshear @aaronblackshear.bsky.social · 2h

The NBA once accidentally pushed the Chinese version of one of the game files to our FTP server

Your Turn 1 (`exercises_r.qmd`, `exercises_py.qmd`)

Discussion: What’s the strangest thing you’ve ever seen in your data? What’s are some times changes in data broke your code?

Data validation

1. Values
2. Rows and columns
3. Dataset properties
4. Logical consistency
5. Scientific consistency

> It’s not that we don’t test our code, it’s that we don’t store our tests so they can be re-run automatically. —Hadley Wickham

Writing down and testing expectations about data

pointblank

- methodically validate your data by writing down expectations and testing them

- Works in R and Python, although the Python version is less mature

- Works with local data frames and remote databases

create_agent()/pb.Validate()

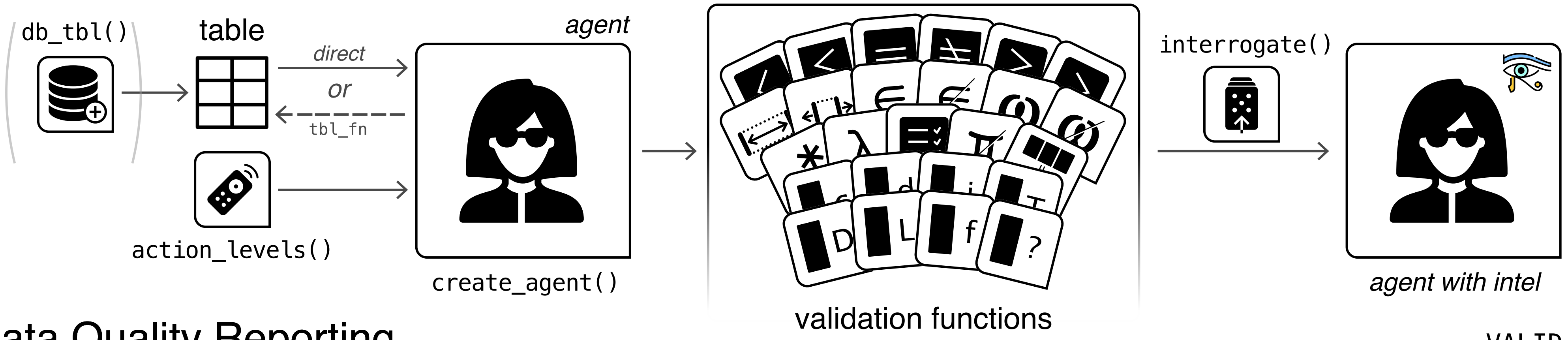
R

```r
library(pointblank)
df |>
  create_agent() |>
  # ... validation steps
  interrogate()
```

Python

```python
import pointblank as pb
validation = (
  pb.Validate(data=df)
  # ... validation steps
  .interrogate()
)
```

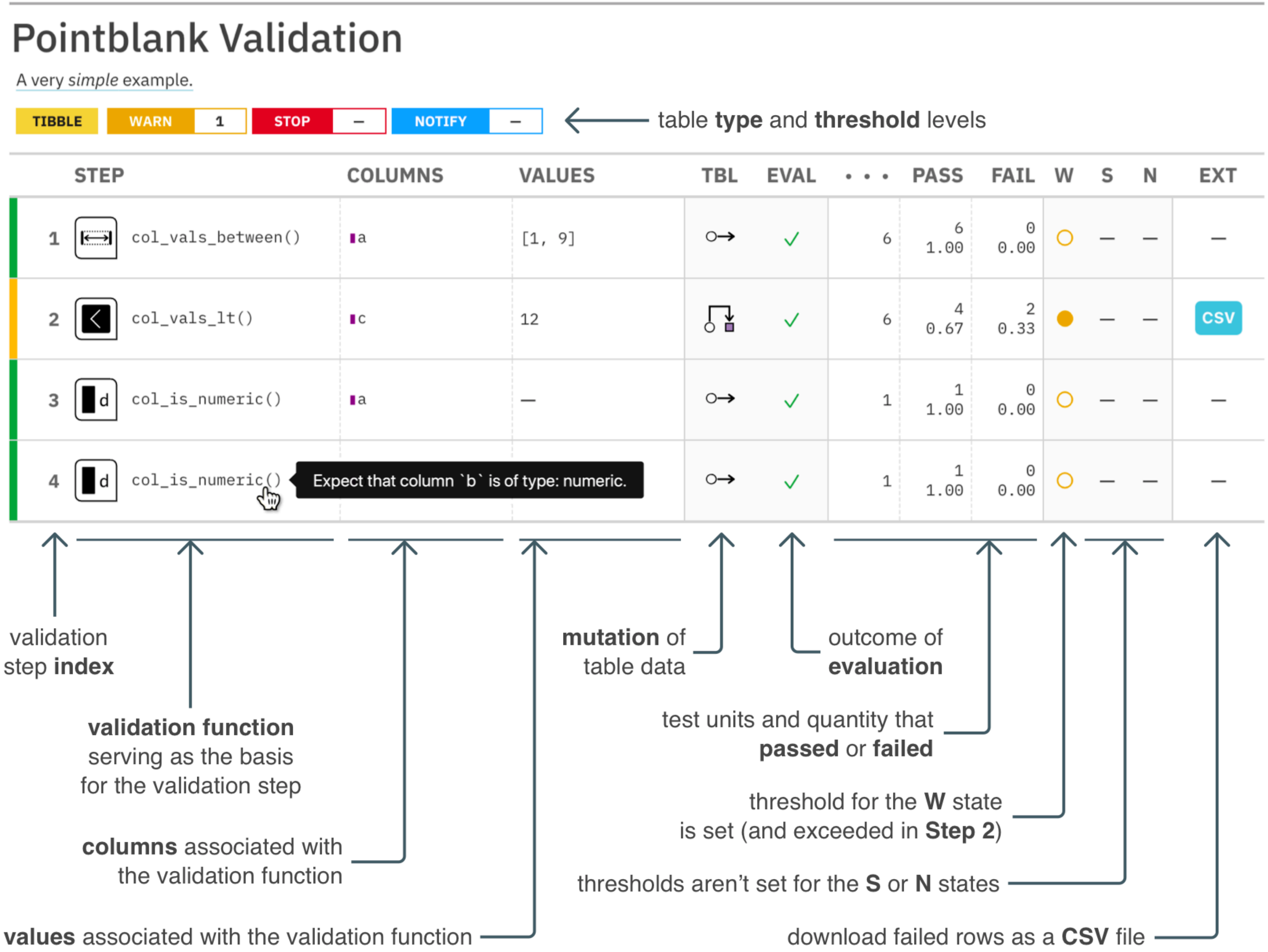
create_agent()/pb.Validate()



Data Quality Reporting

VALID-I

Interoggation reports



Pointblank Validation

A very *simple* example.

TIBBLE | WARN 1 | STOP — | NOTIFY —

table **type** and **threshold** levels

	STEP	COLUMNS	VALUES	TBL	EVAL	• • •	PASS	FAIL	W	S	N	EXT
1	col_vals_between()	a	[1, 9]	○→	✓	6	6 1.00	0 0.00	○	—	—	—
2	col_vals_lt()	c	12		✓	6	4 0.67	2 0.33	●	—	—	CSV
3	col_is_numeric()	a	—	○→	✓	1	1 1.00	0 0.00	○	—	—	—
4	col_is_numeric()			○→	✓	1	1 1.00	0 0.00	○	—	—	—

Expect that column `b` is of type: numeric.

validation step **index**

validation function serving as the basis for the validation step

columns associated with the validation function

values associated with the validation function

mutation of table data

outcome of **evaluation**

test units and quantity that **passed** or **failed**

threshold for the **W** state is set (and exceeded in **Step 2**)

thresholds aren't set for the **S** or **N** states

download failed rows as a **CSV** file

small_table / pb.load_dataset("small_table")

```r
library(pointblank)
small_table
```

```
# A tibble: 13 × 8
   date_time           date           a b             c       d
   <dttm>              <date>     <int> <chr>     <dbl>   <dbl>
 1 2016-01-04 11:00:00 2016-01-04     2 1-bcd-…       3   3423.
 2 2016-01-04 00:32:00 2016-01-04     3 5-egh-…       8  10000.
 3 2016-01-05 13:32:00 2016-01-05     6 8-kdg-…       3   2343.
 4 2016-01-06 17:23:00 2016-01-06     2 5-jdo-…      NA   3892.
 5 2016-01-09 12:36:00 2016-01-09     8 3-ldm-…       7    284.
 6 2016-01-11 06:15:00 2016-01-11     4 2-dhe-…       4   3291.
 7 2016-01-15 18:46:00 2016-01-15     7 1-knw-…       3    843.
 8 2016-01-17 11:27:00 2016-01-17     4 5-boe-…       2   1036.
 9 2016-01-20 04:30:00 2016-01-20     3 5-bce-…       9    838.
10 2016-01-20 04:30:00 2016-01-20     3 5-bce-…       9    838.
11 2016-01-26 20:07:00 2016-01-26     4 2-dmx-…       7    834.
12 2016-01-28 02:51:00 2016-01-28     2 7-dmx-…       8    108.
13 2016-01-30 11:23:00 2016-01-30     1 3-dka-…      NA   2230.
# ℹ 2 more variables: e <lgl>, f <chr>
```

Testing cell values: `col_vals_*()`

R

```r
library(pointblank)
small_table |>
  create_agent() |>
  col_vals_gte(a, 0) |>
  interrogate()
```

Python

```python
import pointblank as pb
validation = (
  pb.Validate(
    data=pb.load_dataset("small_table")
  )
  .col_vals_gte("a", 0)
  .interrogate()
)

validation
```

Your Turn 2

```r
worlds_fairs <- read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/main/

worlds_fairs
```

```
# A tibble: 70 × 14
   start_month start_year end_month end_year
         <dbl>      <dbl>     <dbl>    <dbl>
 1           4       1851        10     1851
 2           5       1855        11     1855
 3           5       1862        11     1862
 4           4       1867        11     1867
 5           5       1873        10     1873
 6           5       1876        11     1876
 7           5       1878        11     1878
 8          10       1880         4     1881
 9           4       1888        12     1888
10           5       1889        10     1889
# ℹ 60 more rows
# ℹ 10 more variables: name_of_exposition <chr>,
#   country <chr>, city <chr>, category <chr>, theme <chr>,
#   notables <chr>, …
```

Testing cell values: arguments

Your Turn 3

Validate the steps in the exercise file.

Testing columns: `col_is_*()`/`col_schema_match()`

```r
create_agent(tbl = small_table) |>
  col_is_date(columns = date) |>
  interrogate() |>
  all_passed()
```

```
[1] TRUE
```

Your Turn 4

Validate the steps in the exercise file.

Testing rows: `row_*()`

- `rows_distinct()`
- `rows_complete()` (R only)
- `rows_distinct(c(var1, var2, ...))`
- `rows_complete(c(var1, var2, ...))` (R only)

```r
create_agent(tbl = small_table) |>
  rows_distinct() |>
  interrogate() |>
  all_passed()
```

```
[1] FALSE
```

```r
create_agent(tbl = small_table) |>
  rows_complete() |>
  interrogate() |>
  all_passed()
```

```
[1] FALSE
```

Your Turn 5

Validate the steps in the exercise file.

Testing table properties: `*_match()`

- `col_schema_match(schema)`
- `row_count_match(n)`, `row_count_match(tbl)`
- `col_count_match(n)`, `col_count_match(tbl)`

Testing table properties: `*_match()`

```r
create_agent(small_table) |>
  row_count_match(13) |>
  col_count_match(8) |>
  interrogate() |>
  all_passed()
```

```
[1] TRUE
```

Exploring test results

- Extract failures from a given step: `get_data_extracts()`

- Get passing or failing rows: `get_sundered_data()`

Exploring test results

```r
agent <- create_agent(tbl = small_table) |>
  col_vals_gte(a, 0) |>
  col_vals_lt(b, 1110) |>
  rows_complete() |>
  interrogate()

get_agent_x_list(agent)$n_failed
```

```
[1]  0 11  2
```

Exploring test results

```r
get_data_extracts(agent, i = 3)
```

```
# A tibble: 2 × 8
  date_time           date           a b             c     d
  <dttm>              <date>     <int> <chr>     <dbl> <dbl>
1 2016-01-06 17:23:00 2016-01-06     2 5-jdo-903    NA 3892.
2 2016-01-30 11:23:00 2016-01-30     1 3-dka-303    NA 2230.
# ℹ 2 more variables: e <lgl>, f <chr>
```

Exploring test results

```r
get_sundered_data(agent)
```

```
# A tibble: 2 × 8
  date_time           date               a b                 c     d
  <dttm>              <date>         <int> <chr>         <dbl> <dbl>
1 2016-01-04 11:00:00 2016-01-04         2 1-bcd-345         3 3423.
2 2016-01-15 18:46:00 2016-01-15         7 1-knw-093         3  843.
# ℹ 2 more variables: e <lgl>, f <chr>
```

Exploring test results

```r
get_sundered_data(agent, type = "fail")
```

```
# A tibble: 11 × 8
   date_time           date           a b             c       d
   <dttm>              <date>     <int> <chr>     <dbl>   <dbl>
 1 2016-01-04 00:32:00 2016-01-04     3 5-egh-…       8 10000.
 2 2016-01-05 13:32:00 2016-01-05     6 8-kdg-…       3  2343.
 3 2016-01-06 17:23:00 2016-01-06     2 5-jdo-…      NA  3892.
 4 2016-01-09 12:36:00 2016-01-09     8 3-ldm-…       7   284.
 5 2016-01-11 06:15:00 2016-01-11     4 2-dhe-…       4  3291.
 6 2016-01-17 11:27:00 2016-01-17     4 5-boe-…       2  1036.
 7 2016-01-20 04:30:00 2016-01-20     3 5-bce-…       9   838.
 8 2016-01-20 04:30:00 2016-01-20     3 5-bce-…       9   838.
 9 2016-01-26 20:07:00 2016-01-26     4 2-dmx-…       7   834.
10 2016-01-28 02:51:00 2016-01-28     2 7-dmx-…       8   108.
```

Exploring test results

```r
get_sundered_data(agent, pass_fail = "fail")
```

```
# A tibble: 2 × 8
  date_time           date           a b             c     d
  <dttm>              <date>     <int> <chr>     <dbl> <dbl>
1 2016-01-04 11:00:00 2016-01-04     2 1-bcd-345     3 3423.
2 2016-01-15 18:46:00 2016-01-15     7 1-knw-093     3  843.
# ℹ 2 more variables: e <lgl>, f <chr>
```

Severity and action (R)

```r
al <- action_levels(warn_at = .001, stop_at = .2)

agent <- create_agent(
  tbl = small_table,
  actions = al
) |>
  col_vals_gte(a, 0) |>
  col_vals_lt(d, 1110) |>
  interrogate()

get_agent_x_list(agent)$warn
```

```
[1] FALSE  TRUE
```

```r
get_agent_x_list(agent)$stop
```

```
[1] FALSE  TRUE
```

Severity and action (Python)

```python
import pointblank as pb
tld = pb.Thresholds(warn_at=.001, stop_at=.2)
validation = (
    pb.Validate(
      data=pb.load_dataset("small_table"),
      thresholds=tld
    )
    .col_vals_ge("a", 0)
    .col_vals_lt("d", 1110)
    .interrogate()
)

validation.warn()
```

```
{1: False, 2: True}
```

```python
validation.stop()
```

```
{1: False, 2: True}
```

Your Turn 6

Validate the steps in the exercise file.

Your Turn 7: Challenge!

```r
english_monarchs_marriages <- read_csv('https://raw.githubuserco

english_monarchs_marriages
```

```
# A tibble: 83 × 5
   king_name        king_age consort_name       consort_age
   <chr>            <chr>    <chr>              <chr>
 1 Æthelwulf        ?        Osburh             ?
 2 Æthelwulf        50(?)    Judith of Flanders 12
 3 Æthelbald        24       Judith of Flanders 14
 4 Æthelberht       —        —                  —
 5 Æthelred         ?        Wulfthryth?        ?
 6 Alfred the Great 19       Ealhswith          16
 7 Edward the Elder 19       Ecgwynn            ?
 8 Edward the Elder 28       Aelffaed           ?
 9 Edward the Elder 31       Eadgifu of Kent    ?
10 Æthelstan        —        —                  —
# ℹ 73
```

Bonus functions! (R only… for now!)

- `data_scan()`
- `draft_validation()`
- Other workflows (testing, YAML, etc)

Data validation

1. Values
2. Rows and columns
3. Dataset properties
4. Logical consistency
5. Scientific consistency