The Data (error) Generating Process

Emily Riederer

When monitoring systems, we prioritize what might go wrong

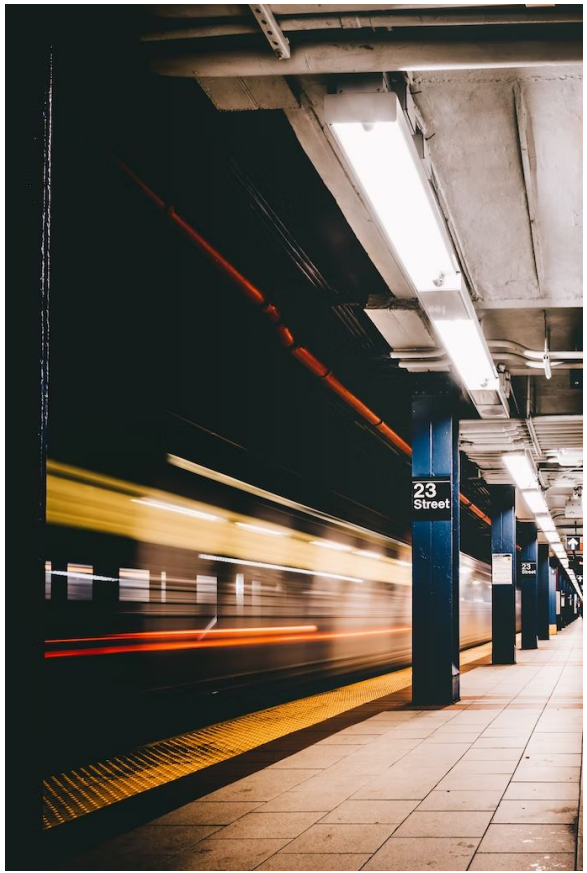


Photo Credit: Lerone Pieters on Unsplash

You should worry about...

✅ What other trains are on the same track (collisions)

✅ Damaged sections of the track (derailment)

✅ Insufficient crew

When monitoring systems, we prioritize what might go wrong

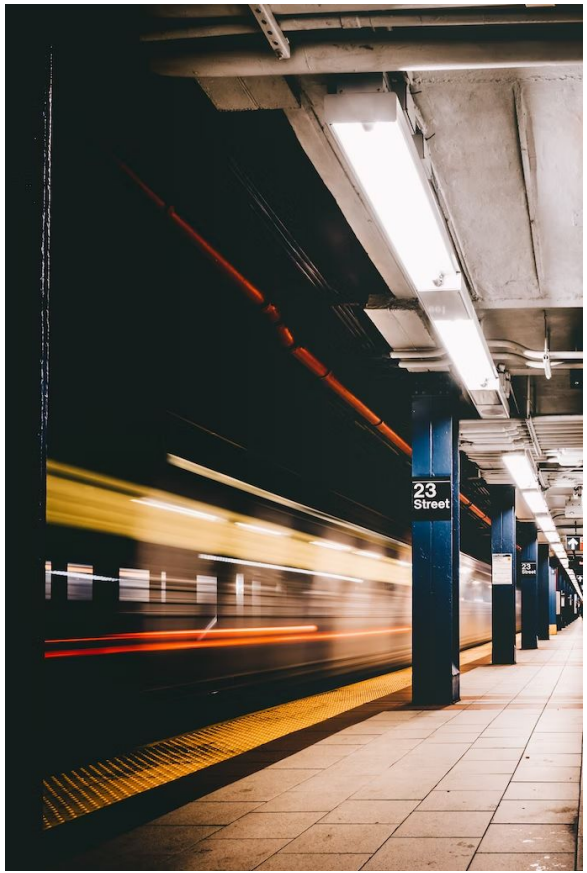


Photo Credit: Lerone Pieters on Unsplash

You should worry about...

✅ What other trains are on the same track (collisions)

✅ Damaged sections of the track (derailment)

✅ Insufficient crew

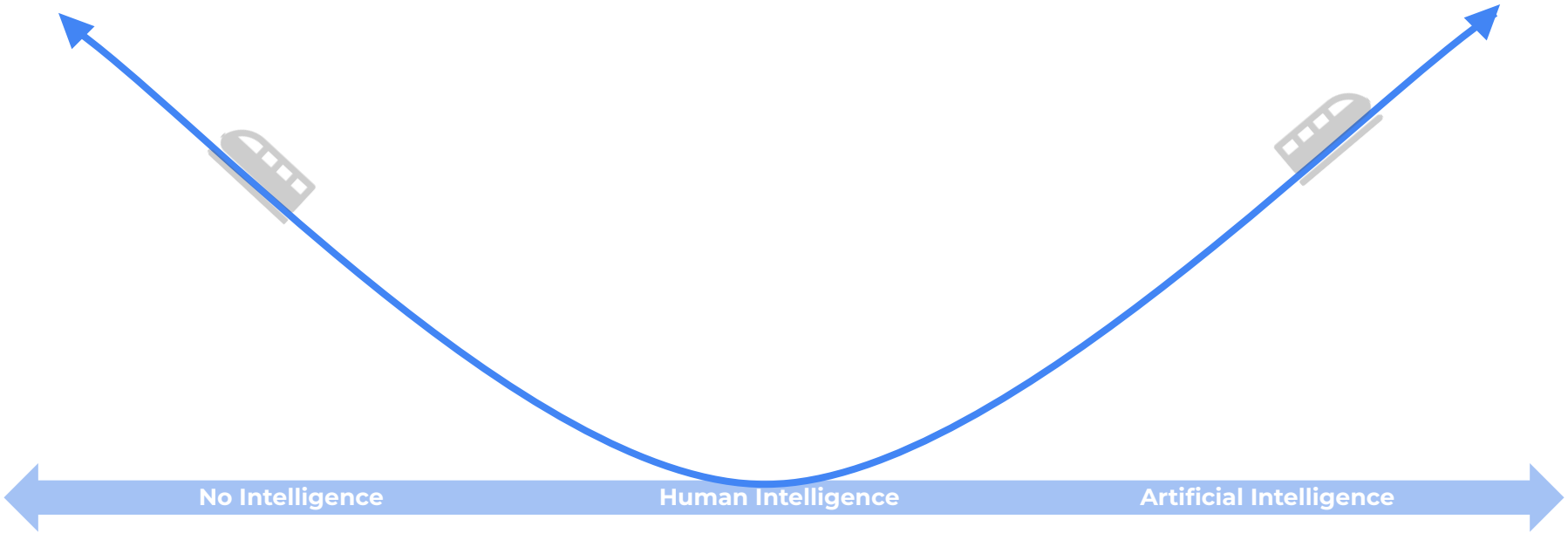
You wouldn't worry about...

❌ All passengers sitting on the same side and tipping it over

❌ Accidental teleportation

❌ Landing gears stuck

When monitoring data quality, we often don't



Data movers are best equipped to devise good checks since they can see from origin to destination

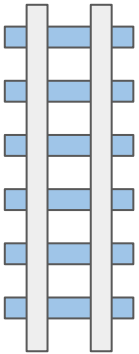
Data movers are best equipped to devise good checks since they can see from origin to destination

What do I need to be true?

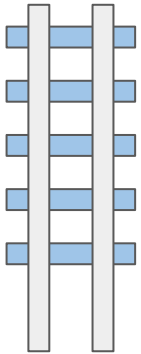
What are likely failure modes?

There are many ways for data loads to fail

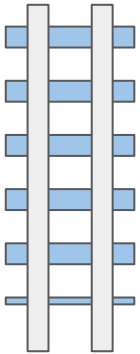
Complete



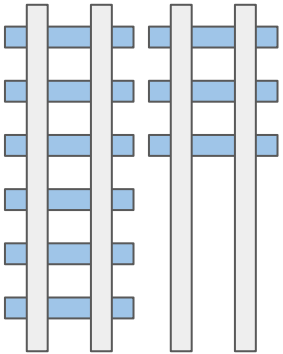
Stale



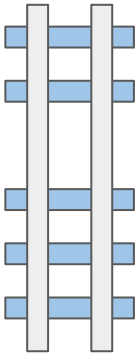
Partial



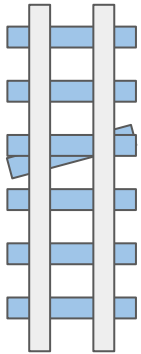
Multisource



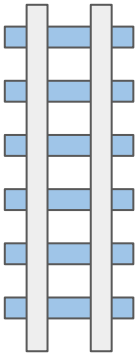
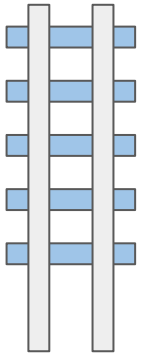
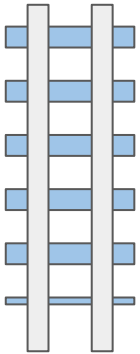
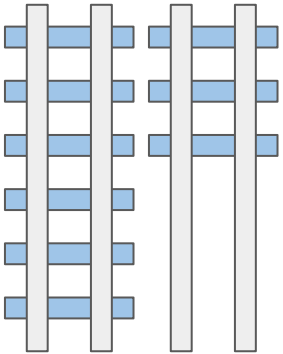
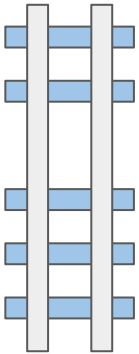
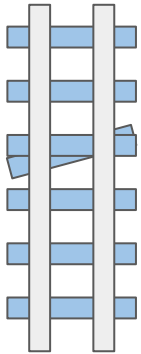
Missing



Duplicate



There are many ways for data loads to fail that go undetected by a standard recency check

Complete	Stale	Partial	Multisource	Missing	Duplicate
					
`max(date) == today`	✅	❌	❌	❌	❌

The NYC subway ridership data generating process

Photo Credit: Tim Hufner on Unsplash

Data Collection

✅ Individual turnstile records cumulative counts

✅ Information uploaded 4x daily

✅ Panel data structure (turnstile x time period)

The idealized transformation process for NYC subway data

Station	Turnstile	TS	Entries (Accum)
A	1	2022-10-08 00:00	1000
A	1	2022-10-08 06:00	1500
A	1	2022-10-08 12:00	1750
A	2	2022-10-08 00:00	700
A	2	2022-10-08 06:00	1000
A	2	2022-10-08 12:00	1200

The idealized transformation process for NYC subway data

Station	Turnstile	TS	Entries (Accum)	New Entries
A	1	2022-10-08 00:00	1000	- -
A	1	2022-10-08 06:00	1500	1500 - 1000 = 500
A	1	2022-10-08 12:00	1750	1750 - 1500 = 250
A	2	2022-10-08 00:00	700	- -
A	2	2022-10-08 06:00	1000	1000 - 700 = 300
A	2	2022-10-08 12:00	1200	1200 - 1000 = 200

The idealized transformation process for NYC subway data

Station	Turnstile	TS	Entries (Accum)
A	1	2022-10-08 00:00	1000
A	1	2022-10-08 06:00	1500
A	1	2022-10-08 12:00	1750
A	2	2022-10-08 00:00	700
A	2	2022-10-08 06:00	1000
A	2	2022-10-08 12:00	1200

New Entries
-
-
1500 - 1000 = 500
1750 - 1500 = 250
-
-
1000 - 700 = 300
1200 - 1000 = 200

Station	TS	Entries
A	2022-10-08 00:00	
A	2022-10-08 06:00	500 + 300 = 800
A	2022-10-08 12:00	250 + 200 = 450

The idealized transformation process for NYC subway data

Station	Turnstile	TS	Entries (Accum)
A	1	2022-10-08 00:00	1000
A	1	2022-10-08 06:00	1500
A	1	2022-10-08 12:00	1750
A	2	2022-10-08 00:00	700
A	2	2022-10-08 06:00	1000
A	2	2022-10-08 12:00	1200

New Entries
- -
1500 - 1000 = 500
1750 - 1500 = 250
- -
1000 - 700 = 300
1200 - 1000 = 200

Station	TS	Entries
A	2022-10-08 00:00	
A	2022-10-08 06:00	500 + 300 = 800
A	2022-10-08 12:00	250 + 200 = 450

Assumes Entries field is:

- Non-Missing
- Monotonic

The NYC subway ridership data generating process

Photo Credit: Tim Hufner on Unsplash

Data Collection

✅ Individual turnstile records cumulative counts

✅ Information uploaded 4x daily

✅ Panel data structure (turnstile x time period)

Failure modes?

❌ Missing data

Turnstile stops recording counts (broken sensor)
Turnstile stops transmitting data upstream (disconnected)

❌ Non-cumulative data

Turnstile stops counting correctly (broken sensor)
Turnstile history is reset (maintenance, replacement)

Grouped checks can add value in multiple ways

Only Expressible

Turnstile ID not unique
versus
Turnstile ID *by Station*

More Rigorous

Max Date
versus
Max Date *by Sensor*

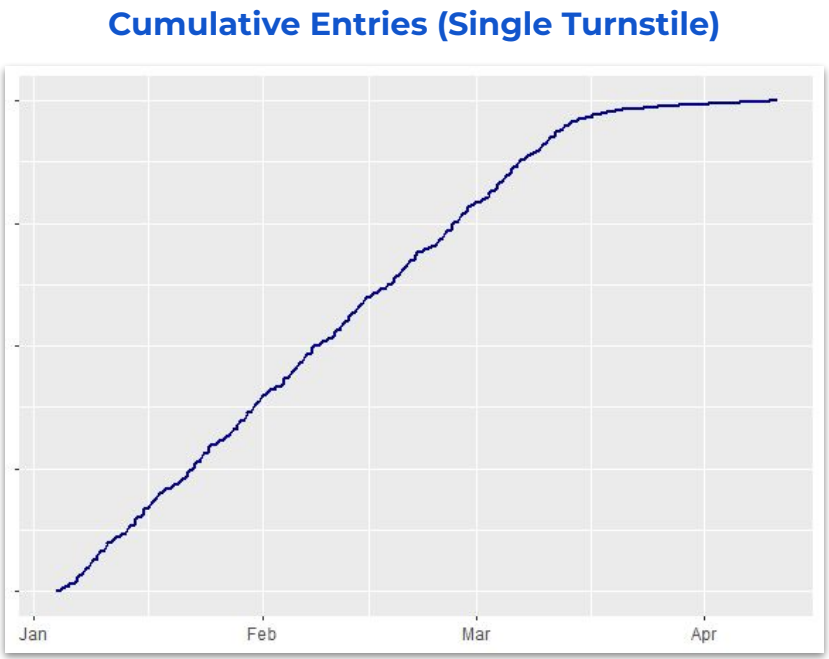
Semantically Intuitive

Think about *by group* “facts”
then
Implement with composites

Only expressible: monotonicity (keeps going up)

Station	Turnstile	TS	Entries (Accum)
A	1	2022-10-08 00:00	1000
A	2	2022-10-08 00:00	600
A	1	2022-10-08 06:00	1500
A	2	2022-10-08 06:00	800
A	1	2022-10-08 08:00	1750
A	2	2022-10-08 08:00	1000

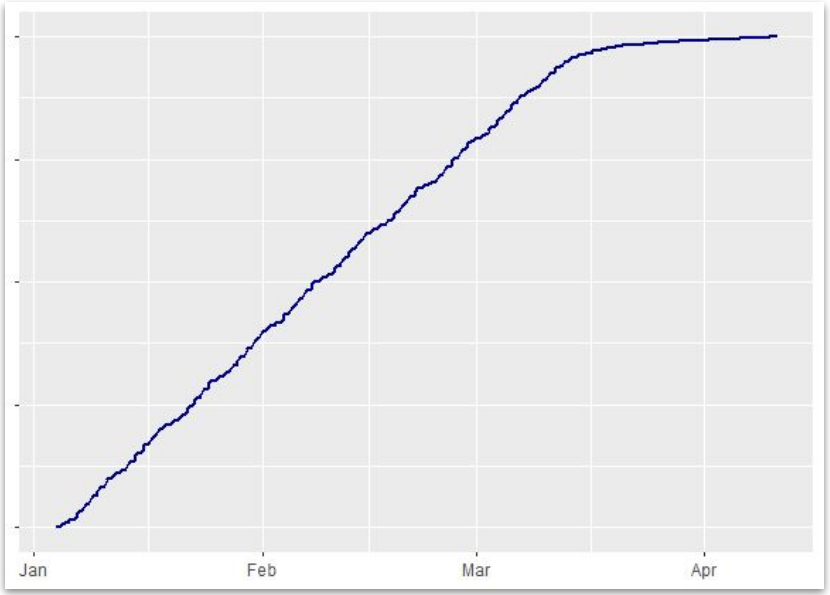
Only expressible: monotonicity (keeps going up)



CA = A033, Unit = R170, SCP = 02-00-05

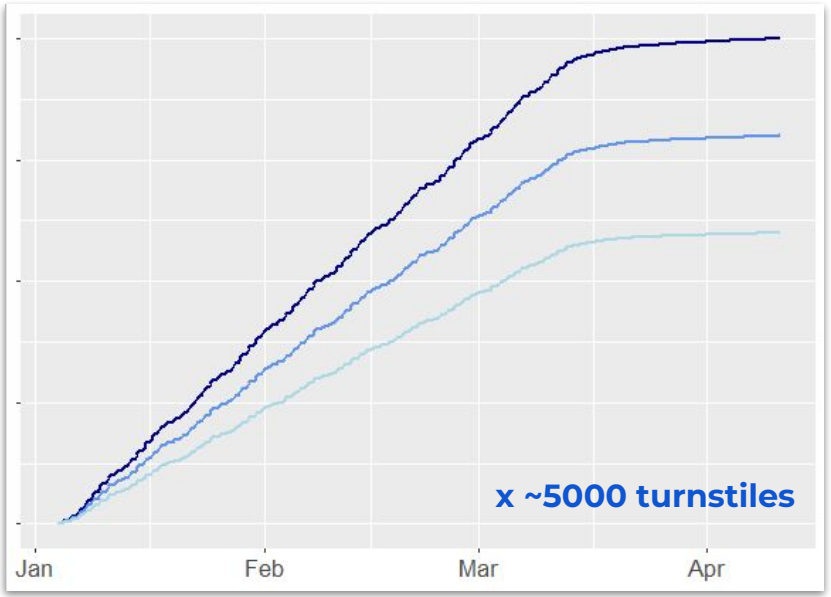
Only expressible: monotonicity (keeps going up)

Cumulative Entries (Single Turnstile)



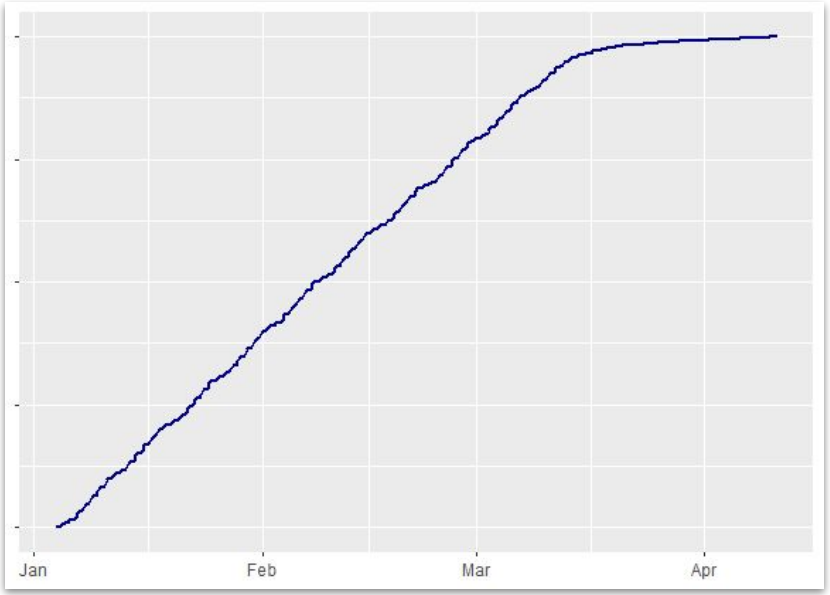
CA = A033, Unit = R170, SCP = 02-00-05

Cumulative Entries (Idealized)



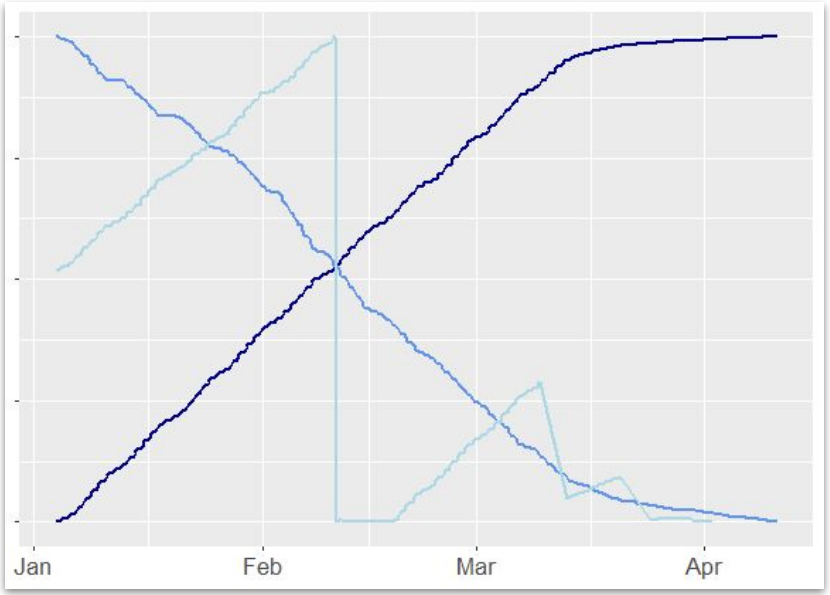
Only expressible: monotonicity (keeps going up)

Cumulative Entries (Single Turnstile)



CA = A033, Unit = R170, SCP = 02-00-05

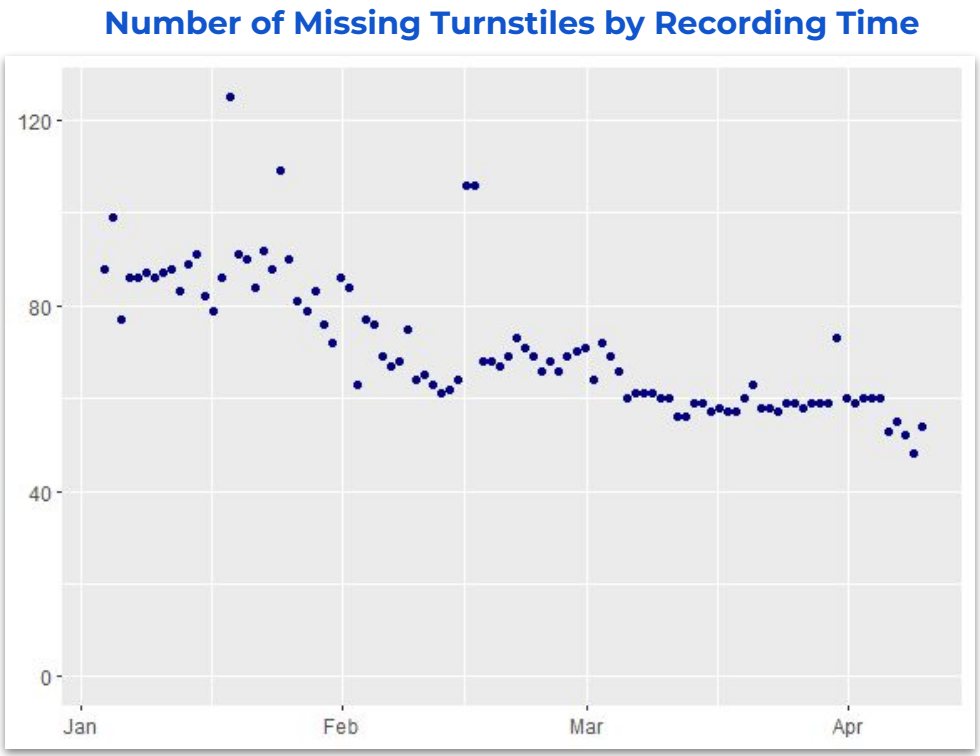
Cumulative Entries (Three Turnstiles)



CA = A033, Unit = R170, SCP = 02-00-05
CA = N559, Unit = R425, SCP = 00-06-01
CA = PTH03, Unit = R552, SCP = 00-00-07

Scales standardized for visualization

More rigorous: maximum date



Data movers are best equipped to devise good checks since they can see from origin to destination

What do I need to be true?

What are likely failure modes?

Now in dbt-utils

V1.0 implements *by group* testing for:

- equal_rowcount()
- fewer_rows_than()
- recency()
- at_least_one()
- not_constant()
- sequential_values()
- non_null_proportion()

Now in dbt-utils

V1.0 implements *by group* testing for:

- equal_rowcount()
- fewer_rows_than()
- recency()
- at_least_one()
- not_constant()
- sequential_values()
- non_null_proportion()

```
models:
  - name: model_name
    tests:
      - dbt_utils.recency:
          datepart: day
          field: recorded_at
          interval: 1
          group_by_columns:
            - station_id
            - turnstile_id
```

Example excerpt models/schema.yml file

Questions?

↓ Get the data ↓
(NYC MTA Data | GitHub Gist)

↓ Read more on ↓
(D(e)GP for Validation | Grouped checks)