https://stochasticsolutions.com/pdf/tdda-london-2026.pdf

TEST-DRIVEN DATA ANALYSIS

Data analysis as if the answers actually matter

PyData London

2026-06-14T14:10:00+01:00

Nicholas J. Radcliffe
Stochastic Solutions Limited
& Department of Mathematics,
University of Edinburgh

Stochastic Solutions

https://stochasticsolutions.com/pdf/tdda-london-2026.pdf

TEST-DRIVEN DATA ANALYSIS

Data analysis as if the answers actually matter

SLIDES

PyData London

2026-06-14T14:10:00+01:00

INSTALLATION

Nicholas J. Radcliffe
Stochastic Solutions Limited
& Department of Mathematics,
University of Edinburgh

Stochastic Solutions

OUTLINE

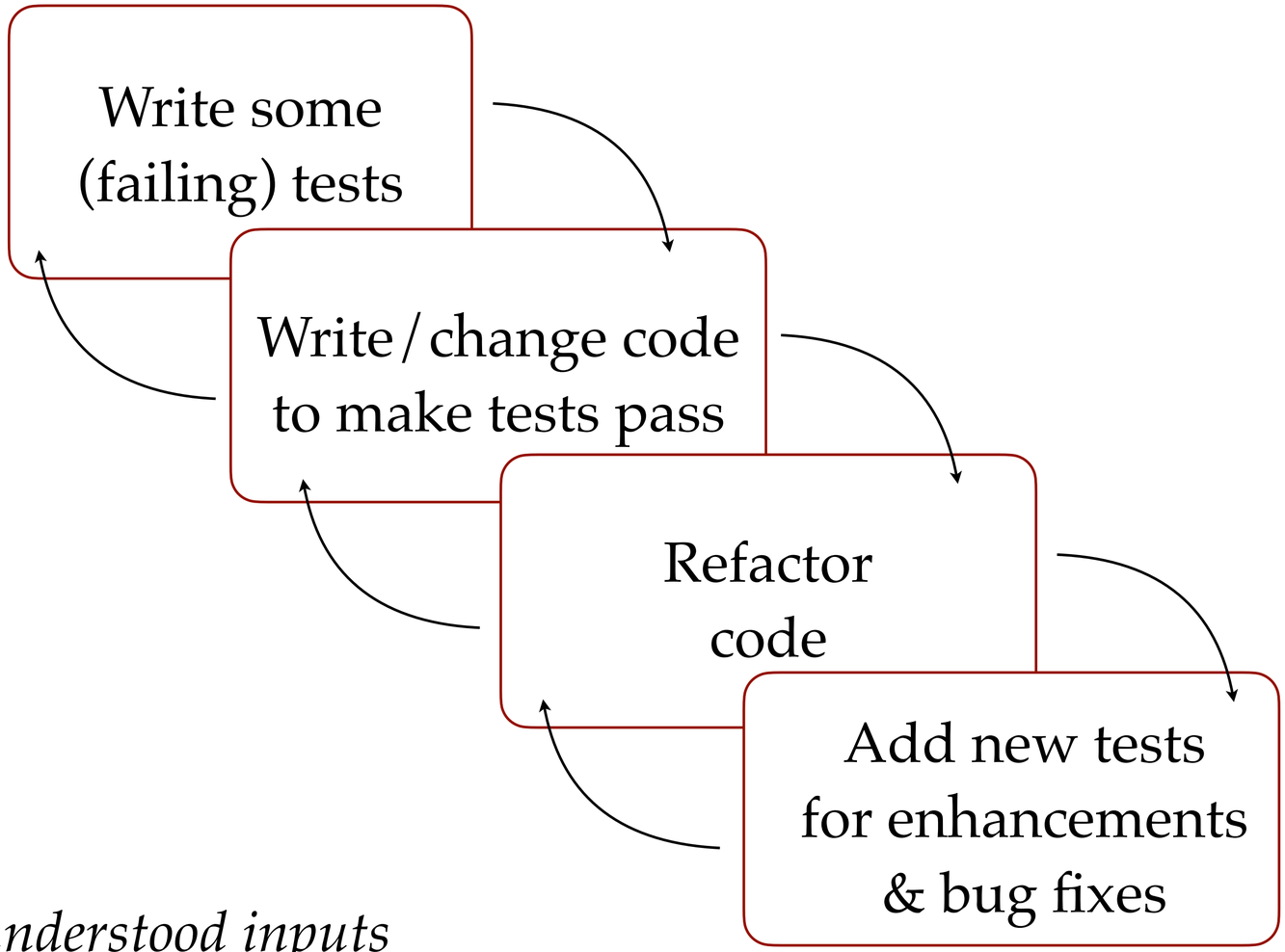
- TDDA Motivation & Methodology
- Data Validation with Constraints

  `tdda discover` • `tdda verify` • `tdda detect`

  `tdda.serial` • tdda utilities
- Reference Tests (*Semantic Regression Tests*) for analytical code
  - The Parables of Anne and Beth
  - `tdda.referencetest` for `unittest` & `pytest`-based tests
  - Automatic Test Generation with `tdda gentest`
- If time dilates / a miracle happens:
  - `tdda diff`
  - Test-Driven Document Development (TDDD)

Why TDDA?

MODERN SOFTWARE DEVELOPMENT (WITH TDD*)

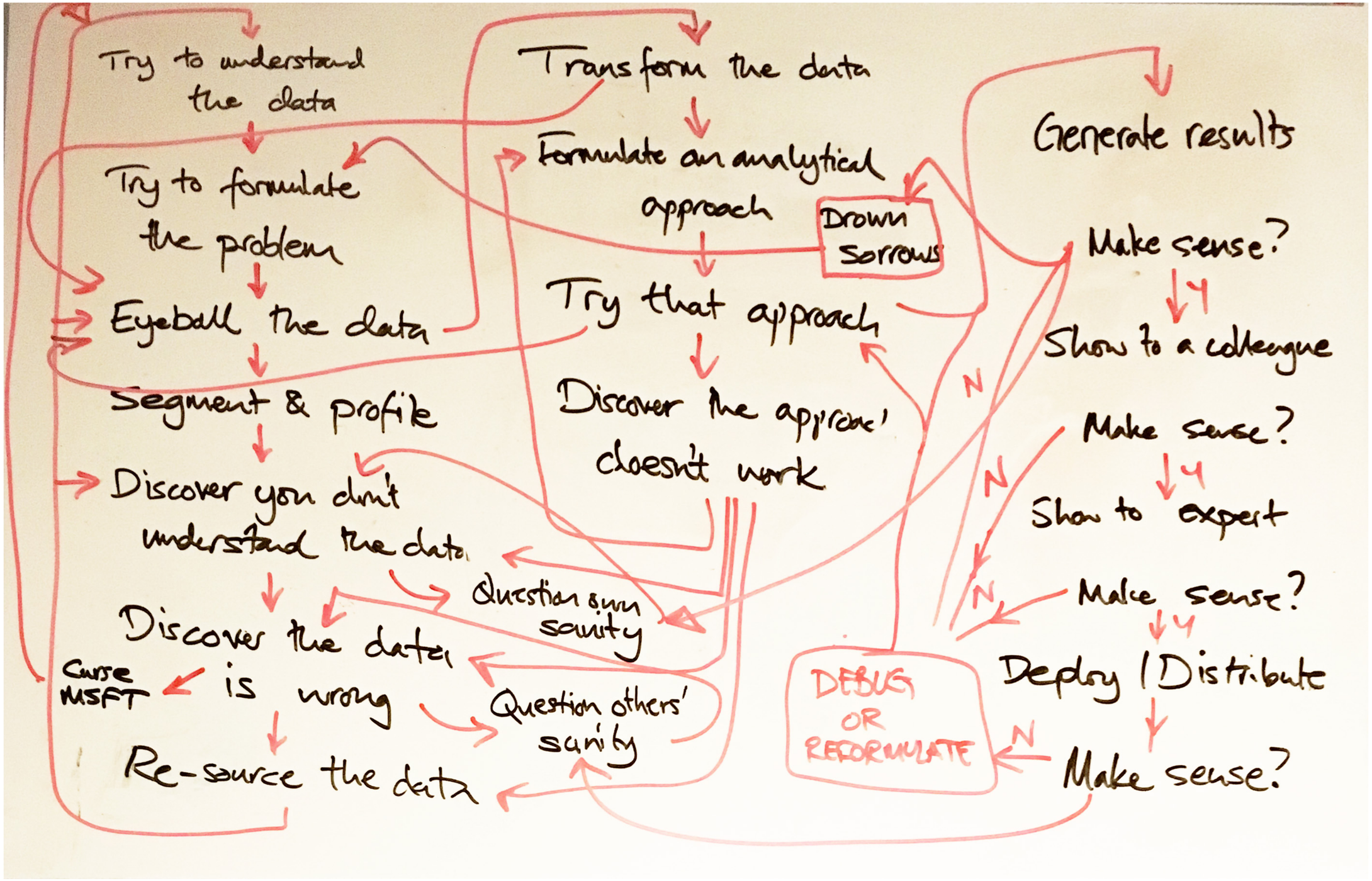


**test-driven development*

Constantly run tests with CI?

Often:

- *Well-understood inputs*
- *Well-understood goal*
- *Many kinds of errors/failures are unmistakable*


Try to understand the data
Try to formulate the problem
Eyeball the data
Segment & profile
Discover you don't understand the data
Discover the data is wrong
Curse MSFT
Re-source the data
Transform the data
Formulate an analytical approach
Drown Sorrows
Try that approach
Discover the approach doesn't work
Question own sanity
Question others' sanity
DEBUG OR REFORMULATE
Generate results
Make sense?
Y
Show to a colleague
Make sense?
Y
Show to expert
Make sense?
Y
Deploy / Distribute
Make sense?
N
N
N
N

TDD ↦ TDDA

TDDA extends TDD's idea of testing for

software correctness

with the idea of testing for

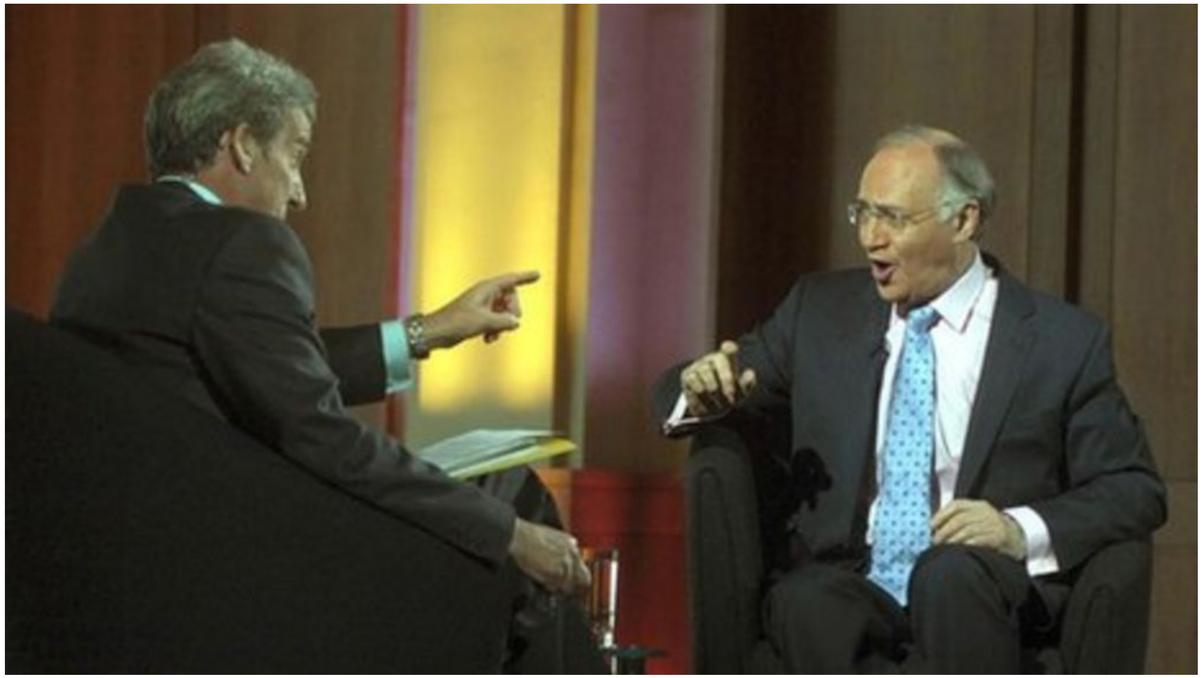
meaningfulness of analysis,

correctness and validity of input and output data,

& *correctness of interpretation*.

"test-driven data analysis"

Jeremy Paxman



PRESENTER, BBC NEWSNIGHT

Interviewing Michael Howard

1997-05-13

HOST, UNIVERSITY CHALLENGE

Winning Team, Final, 2023

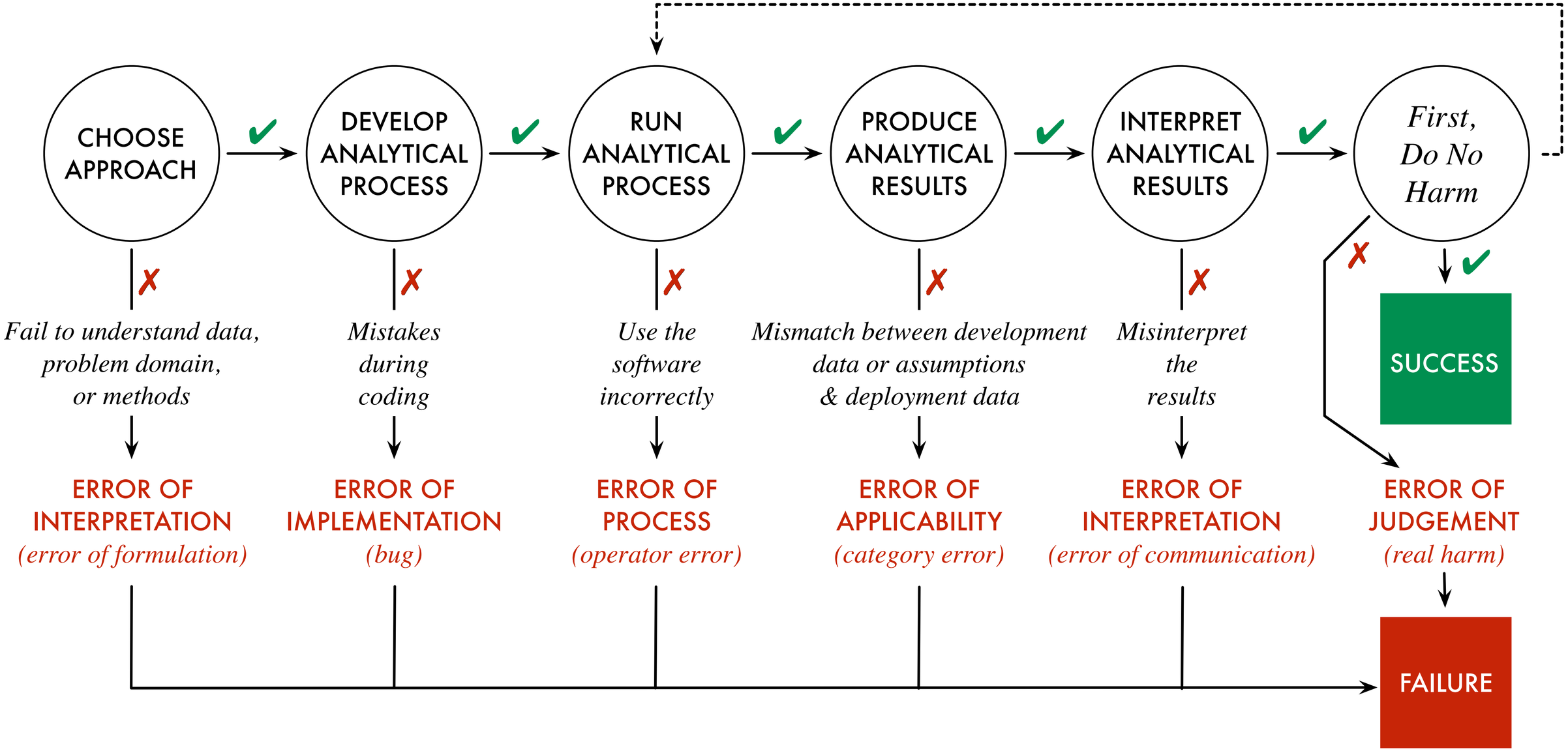
Why is this
lying bastard
lying to me?

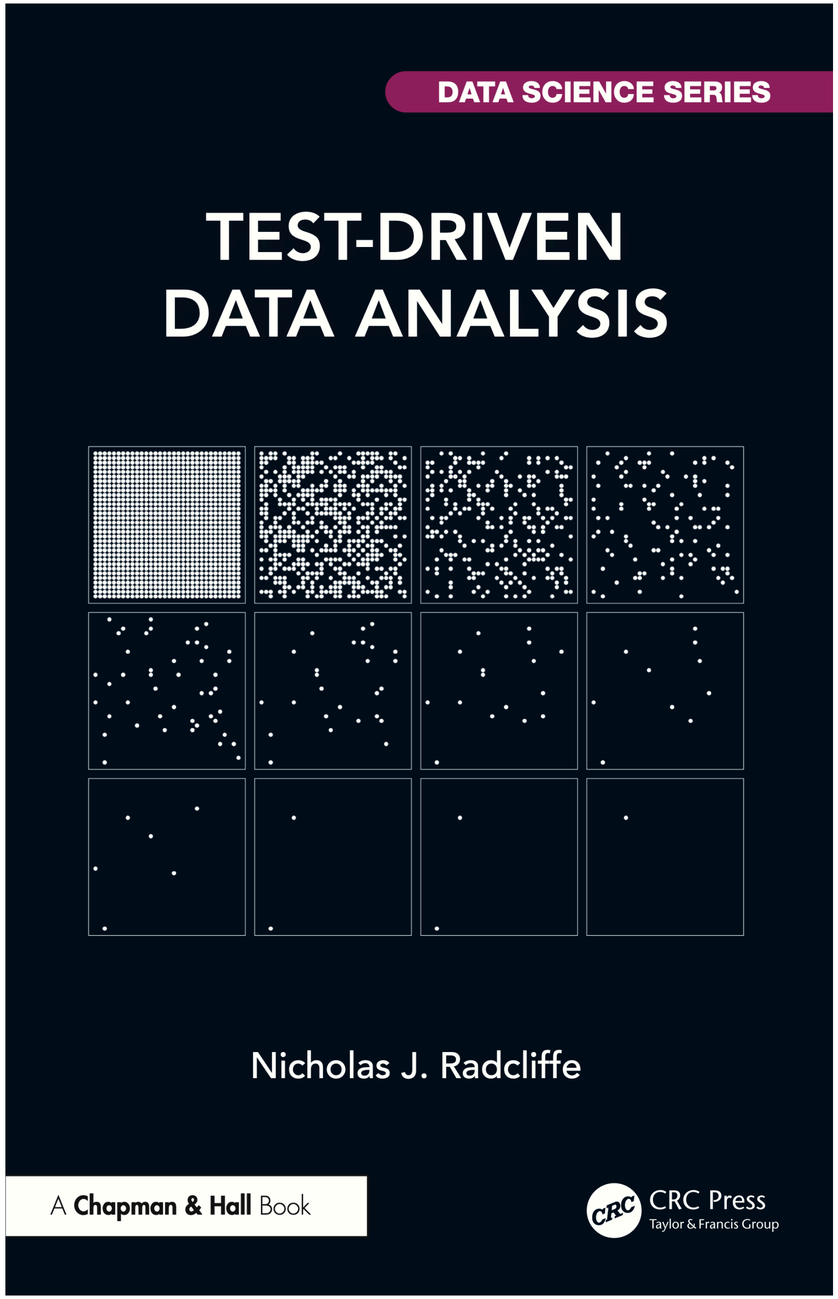
— Jeremy Paxman

(paraphrasing Louis Heron,
paraphrasing unnamed mentor
from Daily Worker)

How is this misleading data misleading me?

How is this misleading data misleading me?





*Available **now** from all good booksellers*
and all sellers of good books

20% discount with **26SMA1** till **2026-06-30**

https://www.routledge.com

Or read for free online at:

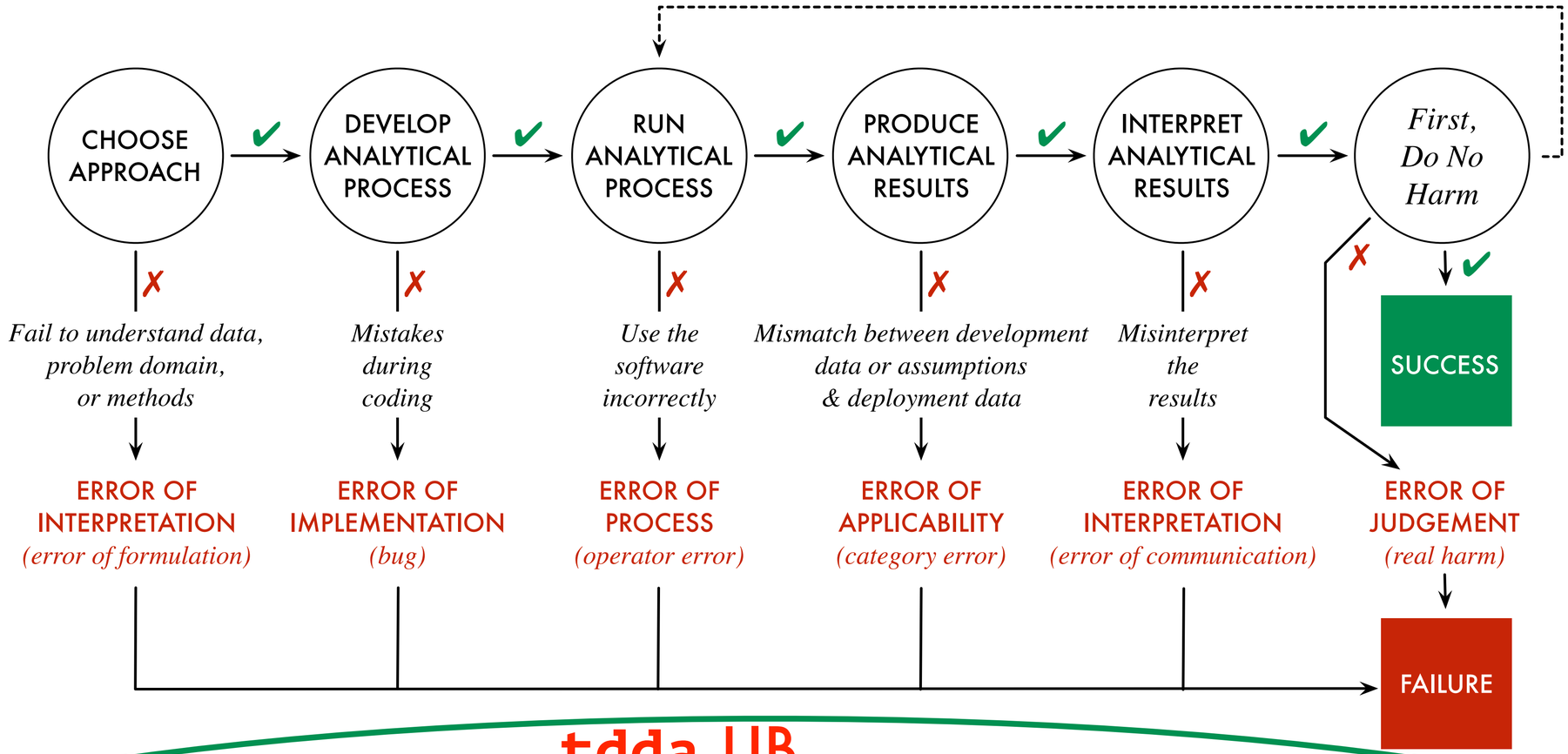
https://book.tdda.info

Chapters released weekly May–Sept 2026

All auxiliary material available now

figures • *exercises* • *TDDD Tests* • *checklists*
glossary • *data dictionaries* • *profiles* • *errata*

How is this misleading data misleading me?



tdda LIB

Read documentation
Understand data domain
Check with domain experts
Create/read data dictionaries
Use checklists

Create reference tests
Use coding standards
Perform code reviews
Use linters
Validate data
Use checklists

Automate & monitor
Use reference tests
Create documentation
Use version control
Use checklists
Adopt coding & data standards
Maintain and utilize metadata

Validate data
Make Fermi estimates
Pay attention to errors
Be careful with the time structure of data
Use checklists

Follow graphing & output best practices
Use checklists

Act responsibly

CHECKLISTS

BOOK

CHAPTER 13
ERRORS OF FORMULATION

CHAPTERS 9–12
REFERENCE TESTING

CHAPTER 16
ERRORS OF PROCESS

PART I & CHAPTER 17
DATA VALIDATION
ERRORS OF APPLICABILITY
& JUDGEMENT

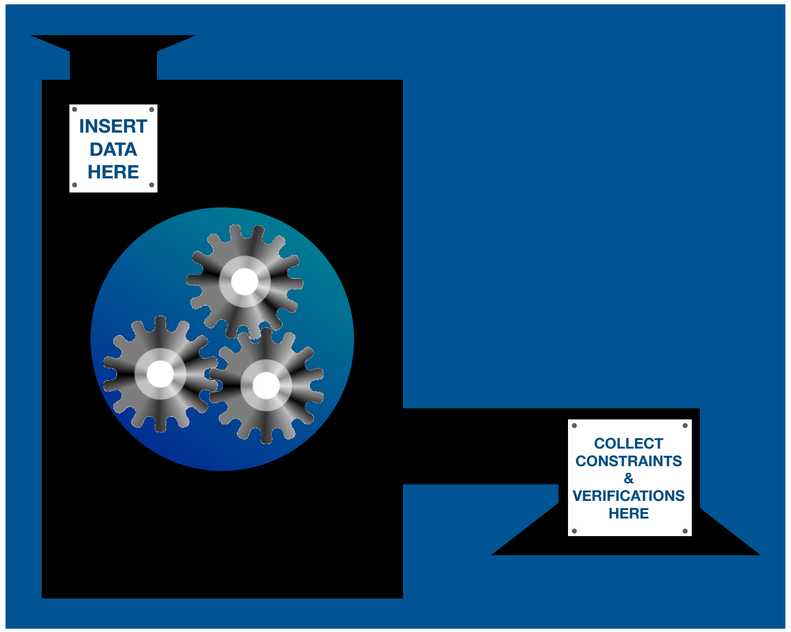
CHAPTERS 14 & 15
ERRORS OF COMMUNICATION
GRAPHING SINS

CHAPTER 17
ERRORS OF APPLICABILITY
& JUDGEMENT

PYTHON TDDA LIBRARY (tdda)

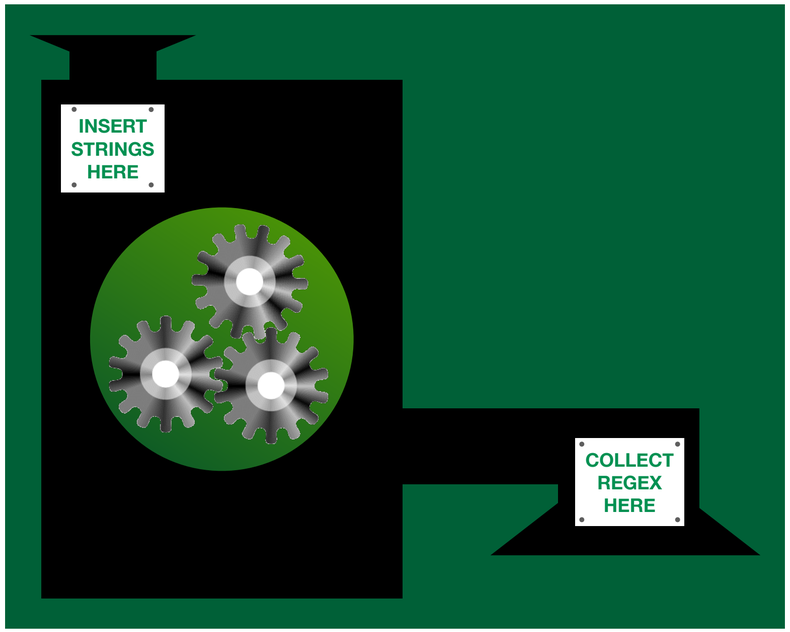
Discover • Verify • Detect

VERIFYING DATA
WITH
CONSTRAINTS



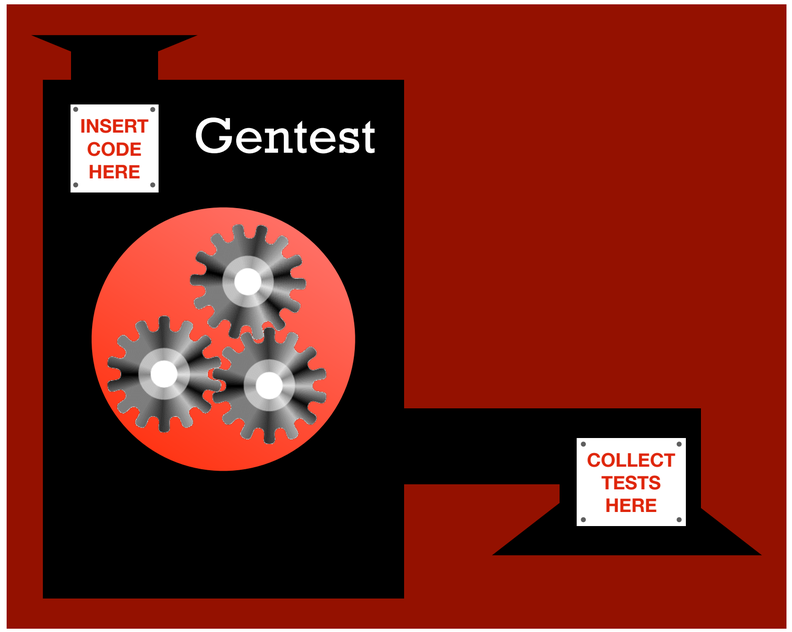
Rexpy

GENERATING
REGULAR EXPRESSIONS
FROM EXAMPLES



Reference Tests • Gentest

TESTING
DATA
PROCESSES



Gentest writes tests,
so you don't have to™

PYTHON TDDA LIBRARY (tdda)

ReferenceTest API

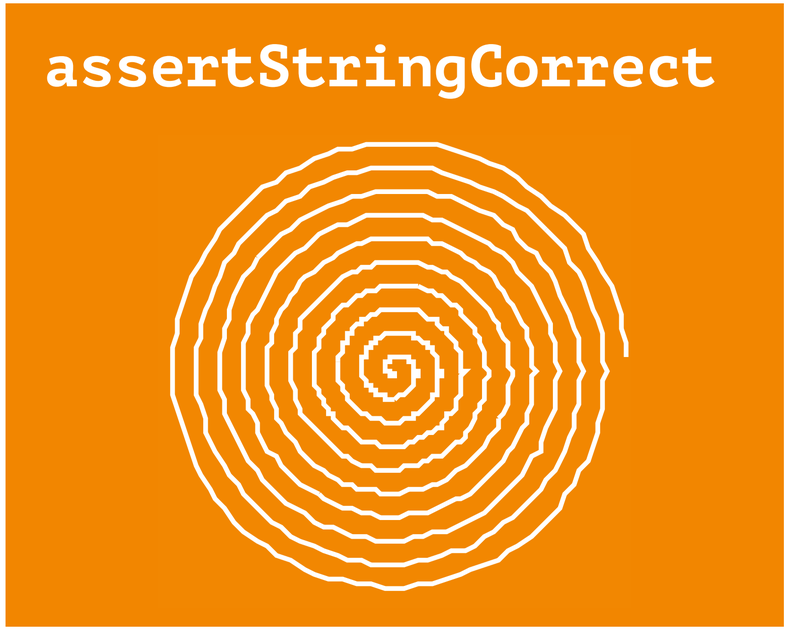
SEMANTIC
REGRESSION TESTS

tdda diff

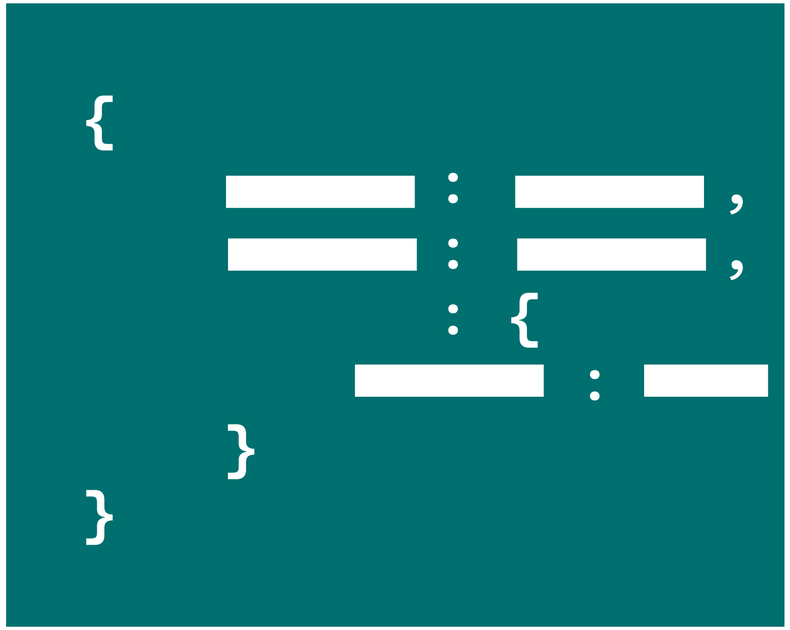
SEMANTIC
DATAFRAME DIFF

tdda.serial

METADATA FOR
FLAT (CSV) FILES



N	N	S	S
111	111	AAA	BBB
1	1	aaa	Aaa
7	7	yy	zz
98	99	q	P



TDDA: MAIN IDEAS

1. Checking data: Constraint Discovery & Verification
    - *"Unit tests for data"*
    - Can cover inputs, outputs and intermediate results
    - Automatically discovered (and refined by humans)
    - Use as part of analysis to verify inputs, outputs and intermediates (as appropriate)
2. Checking analytical processes & pipelines: "Reference" Tests
    - *cf.* system/integration tests in TDD
    - With support for new assertions, exclusions, regeneration, helpful reporting etc.
    - Re-run these tests *all the time, everywhere*

    2a. Automatic Test Generation
        - Give **`tdda gentest`** a command/script to run.
        - It generates tests for you (Python tests for any language).
3. Utilities: serial metadata (`tdda.serial`); dataframe comparison (`tdda diff`); unicode utilities (`tdda.utils` and CLI utilities, `cat`, `ls`, `head`, `tail`, `sample`)

TDDA LIBRARY: INSTALLATION

Install from PyPI

```
python -m pip install -U tdda
```

or from Github (source)

```
git clone https://github.com/tdda/tdda.git
cd tdda
pip install .
```

Or works fine in virtual environment or with **uv**

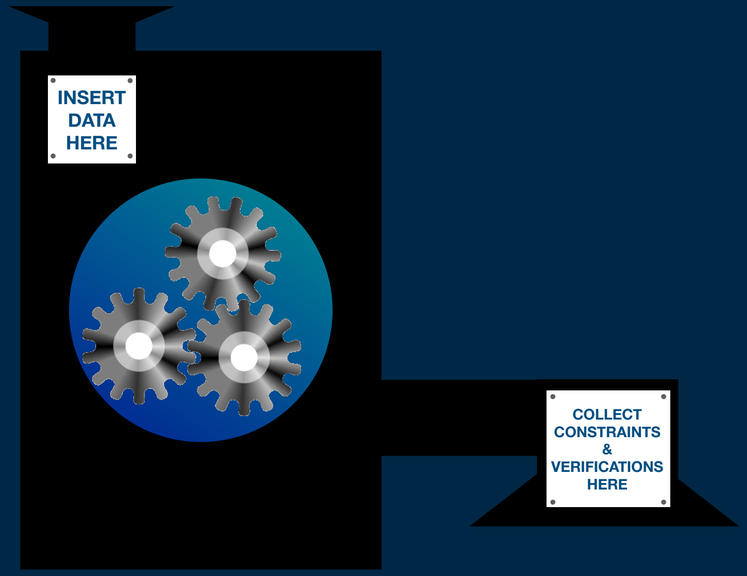
- But must be ***installed*** for the command line and API to work

TDDA LIBRARY

- Python 3 • Open source (MIT Licensed)
- Documentation:
  - Built copy at `https://tdda.readthedocs.io`
    - Sphinx source in `doc` subdirectory
  - Book: `https://book.tdda.info`
  - Man pages: `tdda installman` *then* `man tdda-discover` etc.
  - Built-in help: `tdda help COMMAND` *or* `tdda COMMAND --help`
  - Checklists: `https://book.tdda.info/checklists`
- Example data: `tdda examples` *or* `tdda examples all`
- Quick reference: `https://www.tdda.info/pdf/tdda-quickref.pdf`

TDDA LIBRARY

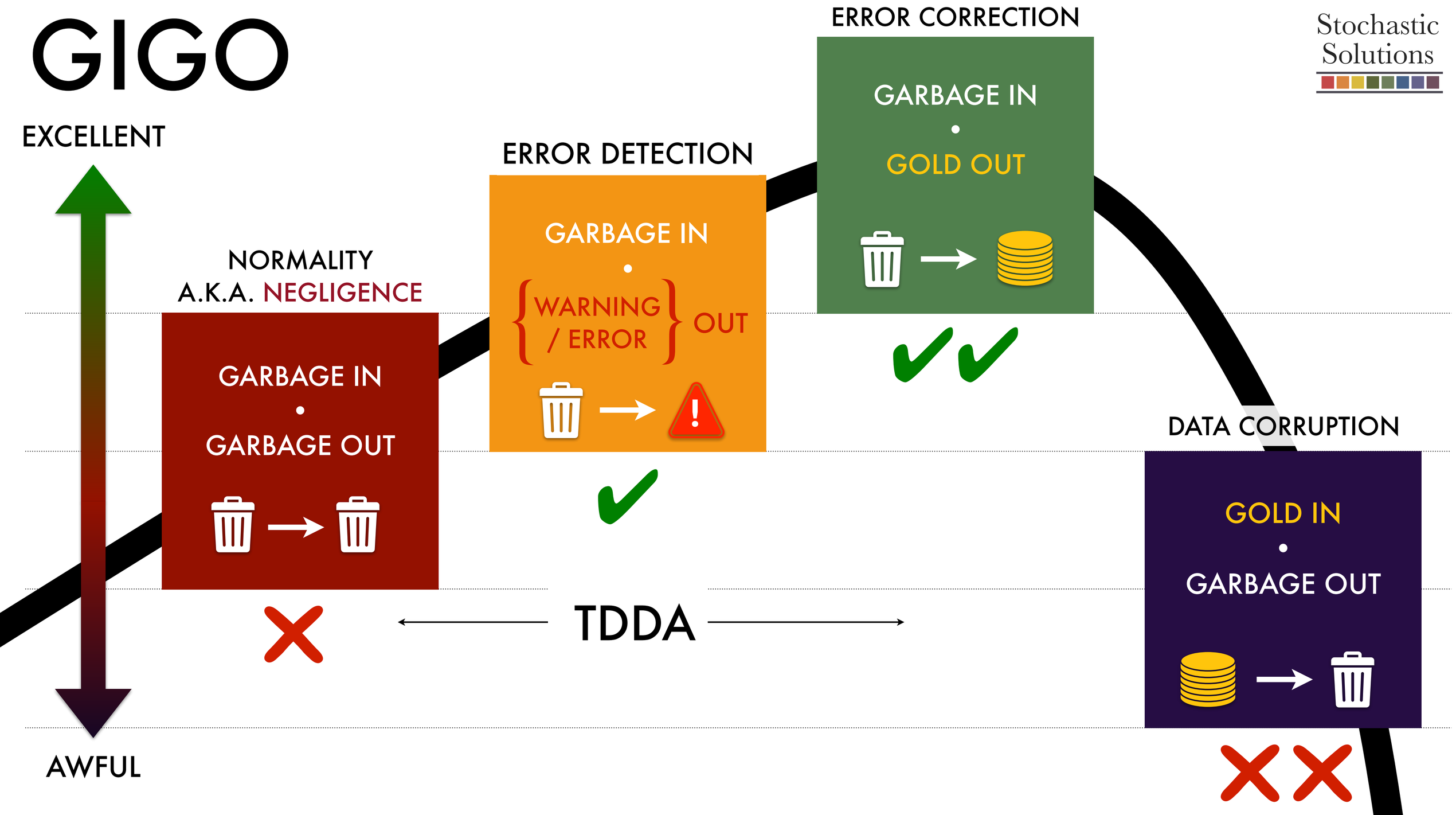
- Works on Linux + Unix, Mac, Windows (and—better!—under WSL)
- Supports:
  - **DataFrames:**
    - Pandas 2 & 3 (*original*, *numpy_nullable*, & *py_arrow backends*)
    - Polars
  - **Serial Data**
    - Parquet
    - Flat files (tdda.serial, CSVW, Frictionless metadata)
  - **Databases** (constraint discovery & data validation)
    - PostgreSQL, MySQL/MariaDB, SQLite3



DATA VALIDATION: CONSTRAINT GENERATION, VERIFICATION & ANOMALY DETECTION

GIGO

Stochastic Solutions



CONSTRAINTS

- Very commonly, data analysis uses data tables (e.g. DataFrames) as inputs, outputs and intermediate results
- There are many things we know (or at least expect) to be true about these data tables
- *Could* write down all these expectations as constraints and check that they are actually satisfied during analysis . . . *but life's too short!* (Also: humans are rather error-prone)

THE BIG IDEA

- Get the computer to find ("discover") constraints satisfied by example datasets automatically.
- Verify against these constraints, modifying as required
- (Humans much happier to make tweaks than start from scratch)

EXAMPLE CONSTRAINTS

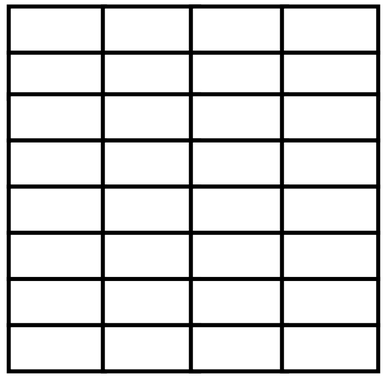
SINGLE FIELD CONSTRAINTS	DATASET CONSTRAINTS
$0 \leq \text{Age} \leq 150$	The data frame must contain field **CID**
type(Age) = int	Number of records must be 118
CID ≠ NULL	One field should be tagged O
CID unique *(in a customer table)*	**Date** should be sorted ascending
len(CardNumber) = 16	**MULTI-FIELD CONSTRAINTS**
Base in {"C", "G", "A", "T"}	StartDate ≤ EndDate
0 ≤ Pass Rate ≤ 100%	AlmostEqual(F, m × a, 6)
StartDate < tomorrow()	sum(Favourite*) = 1
v < 2.97e8 (m/s)	minVal ≤ medianVal ≤ maxVal
Height ~ N(1.6, 0.2)	$V \leq H \times W \times D$

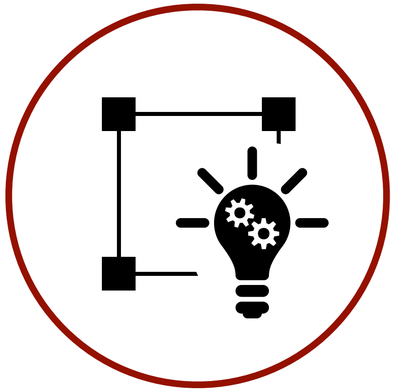
CONSTRAINTS SUPPORTED BY TDDA LIBRARY

KIND	DESCRIPTION	BOOLEAN	INTEGER	REAL	DATE	STRING
`min`	*Minimum allowed value; on verification interpreted with proportionate tolerance* `epsilon`.	✔	✔	✔	✔	✗
`max`	*Maximum allowed value; on verification interpreted with proportionate tolerance* `epsilon`.	✔	✔	✔	✔	✗
`sign`	`"positive"`, `"non-negative"`, `"zero"`, `"non-positive"` *or* `"negative"`.	✔	✔	✔	✗	✗
`max_nulls`	`0` *if nulls not allowed. In principle, can be higher values (in particular,* `1`*), but* `discover` *function does not use these at present.*	✔	✔	✔	✔	✔
`no_duplicates`	`true` *if duplicates are not allowed.*	✔	✔	✔	✔	✔
`min_length`	*smallest allowed string length*	✗	✗	✗	✗	✔
`max_length`	*largest allowed string length*	✗	✗	✗	✗	✔
`allowed_values`	*list of allowed; strings must be one of those values.*	✗	✗	✗	✗	✔
`rex`	*list of regular expressions; strings must match at least one.*	✗	✗	✗	✗	✔

+ allowed and required fields in the dataset

AUTOMATIC CONSTRAINT GENERATION





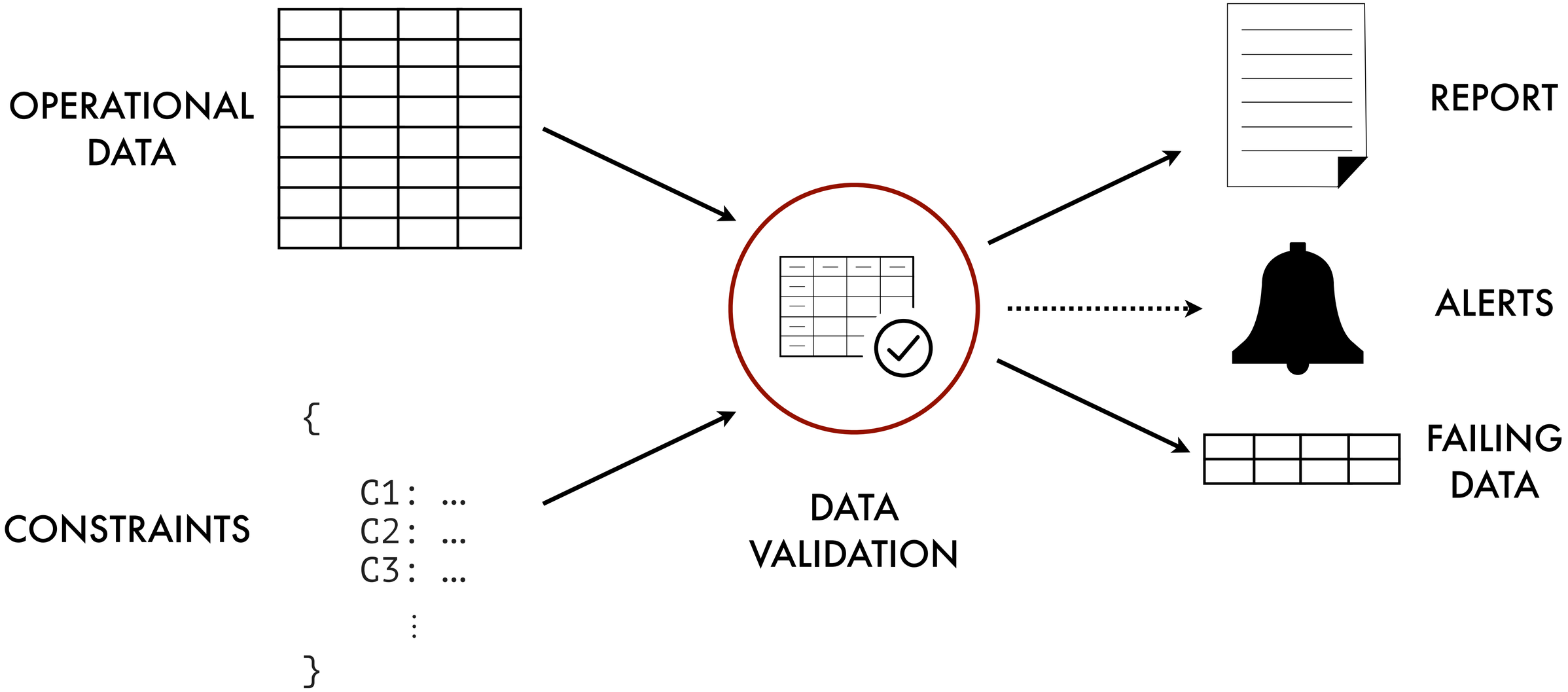
```
{
   C1: Age ⩾ 0
   C2: ID is not null
   C3: CardNumber ~
      DDDD DDDD DDDD DDDD
                ⋮
}
```

TRAINING DATA
(believed to be "good")

AUTOMATIC GENERATION OF CONSTRAINTS
("Discovery")

DISCOVERED CONSTRAINTS
(ideally, now refine by hand)

DATA VERIFICATION/DETECTION



DATABANK PLC

Databank is a synthetic UK-based retail bank operating in the UK and accepting only *UK-resident customers*.

It was established in 2000 and opened for business on 2nd October 2001.

It offers a range of current accounts, with (internal) names `basic`, `current`, `current+`, `offset`, and `premium`.

The datasets used here have been nominally generated on 1st January 2026.

FIRST 20 RECORDS (tdda cat)

```
cd ~/tmp
tdda examples
cd constraints_examples
```

```
$
$ tdda cat accounts1k.parquet 20
accounts1k.parquet: 20 records; 16 fields. [Pandas]
```

tdda head accounts1k.parquet

account_number	open_date	close_date	title	forename	middle_name	surname	address1	city	postcode	home_tel	mobile_tel	email	account_type	overdraft_limit	cash_card_number
12094757	2010-03-07 00:00:00	∅	Mrs	Clara	Ava	Walker	5 Ditton Reach	Thames Ditton	KT7 0ZT	0843 295 2714	0705 784 4779	ye4jxzzywz4i472j@runbox.co.uk	offset	0	1202 8605 9840 2291
12824955	2005-03-09 00:00:00	∅	Mrs	Bonnie	∅	Kay	28 Little Woodbury Drive	Derby	DE23 8TH	0505 300 4446	0705 426 8196	bonnie753@yahoo.co.uk	current	800	1202 8620 1506 4579
12736845	2011-11-28 00:00:00	∅	Mr	Charles	Ronnie	Chapman	61 Kingsley Close	Thornton	FY5 3RT	0140 939 8078	0775 815 3161	x2yh@hotmail.co.uk	current	9600	1202 8663 1022 4459
12978556	2006-12-15 00:00:00	∅	Mr	Mohammed	Charles	Young	26 Larkfield Park	Chepstow / Cas-Gwent	NP16 2DX	0247 120 7888	0779 914 4439	mohammed.young59@gmail.com	current	2400	1202 8662 2487 3776
11783426	2008-11-02 00:00:00	∅	Mr	Ryan	∅	Reid	1 Matheson Place	Wick	BN17 6YC	0850 610 9741	0743 555 1523	ryan1@googlemail.co.uk	current+	2000	1202 8612 4044 5859
10306278	2016-03-15 00:00:00	∅	Ms	Lyla	∅	Jackson	29 Ferris Green	West End	RG42 0EU	0601 572 4938	0753 164 3648	lyla30@hotmail.com	current	10400	1202 8677 5335 4488
10099638	2014-11-09 00:00:00	∅	Miss	Nancy	Sienna	Clarke	42 Crieff Walk	Hartlepool	TS25 5EY	0884 134 8768	0745 190 5358	nancy.clarke6884@runbox.co.uk	current	0	1202 8659 5033 8345
12067823	2002-11-02 00:00:00	2018-10-11 00:00:00	Miss	Jessica	∅	Palmer	24 Melbury Road	Cheadle Hulme	SK8 9EU	0656 559 5460	0794 211 1635	jessica.palmer@gmail.com	current	6900	1202 8636 9333 2534
11401145	2009-12-20 00:00:00	∅	Mr	Arlo	∅	Fraser	7 Seymour Close	Basildon	SS15 3GU	0569 168 3516	0781 950 3390	arlo.fraser3@runbox.co.uk	current+	0	1202 8670 0078 4203
11948703	2007-06-01 00:00:00	∅	Mr	Victor	∅	James	15 Oak Tree Close	Long Bennington	NG23 0TN	0864 390 9897	0775 119 1215	jxklxlpcw@hotmail.co.uk	current	15600	1202 8624 7860 9489
10156515	2011-08-28 00:00:00	2016-03-29 00:00:00	Mrs	Elsie	∅	Watt	11 Southwell Road	Wisbech	PE13 9AP	0688 669 8749	0709 616 4494	elsie54@gmail.com	current	0	1202 8666 3520 9593
10848668	2003-09-09 00:00:00	2017-03-05 00:00:00	Mr	Paul	Liam	Davis	29 Knight's Place	Twickenham	TW2 2VB	0353 500 9086	0752 602 6491	paul.davis36@yahoo.co.uk	basic	0	1202 8606 3898 1124
11687931	2015-12-19 00:00:00	∅	Mrs	Robyn	Hannah	Douglas	41 Maes Gwynedd	Caernarfon	LL55 0UH	0928 802 0169	0714 301 0006	n1921pyn@runbox.com	current	0	1202 8659 0077 6660
10446129	2014-01-26 00:00:00	∅	Miss	Lottie	∅	Gray	88 Sidney Terrace	Bishop's Stortford	CM23 0ZV	0161 718 2281	0723 923 3884	lottie.gray5125@hotmail.com	current	12800	1202 8634 3696 7944
11521624	2006-01-21 00:00:00	∅	Mrs	Bonnie	∅	Clarke	82 Warden's Close	Edinburgh	EH1 5VW	0596 251 2050	0774 824 4470	v6v62zo@googlemail.co.uk	offset	18200	1202 8614 0752 4497
11740257	2005-10-17 00:00:00	2010-02-26 00:00:00	Mr	Nathan	Charlie	Reid	21 Buttercup Square	Oxford	OX4 6SV	0756 820 4314	0768 854 8716	nathan7539@googlemail.co.uk	current+	0	1202 8616 5320 7863
10166275	2017-09-24 00:00:00	∅	Miss	Isla	∅	Duncan	17 Primrose Bank	Oldham	OL8 1HC	0391 858 6851	0776 381 2554	isla.duncan0@googlemail.com	current	9500	1202 8693 3300 2705
10853973	2004-05-24 00:00:00	∅	Ms	Clara	∅	Corbyn	19 Rose Street	Annan	DG12 9YW	0269 532 5580	0716 612 4106	clara.corbyn56@runbox.com	current	0	1202 8686 6350 9678
10872786	2014-06-07 00:00:00	∅	Mr	Ralph	Jackson	Bennett	20 Bedford Avenue	Wallsend	NE28 0WZ	0832 300 0158	0788 612 5980	ralph.bennett@runbox.com	current	0	1202 8637 2049 6960
10358958	2011-05-04 00:00:00	∅	Mrs	Arabella	Ruby	Carter	37 Bedford Road	Casnewydd / Newport	NP19 2GH	0867 741 1060	0716 363 4650	arabella7563@runbox.co.uk	current	0	1202 8635 4511 3346

```
$
```

"Known-good data"

(or "Believed to be reasonably good data")

FIRST 10 RECORDS IN TWO BITS

tdda head accounts1k.parquet '[a-e]*' 10

```
$ tdda cat accounts1k.parquet '[a-e]*' 10
accounts1k.parquet: 10 records; 7 fields. [Pandas]
```

account_number	close_date	address1	city	email	account_type	cash_card_number
12094757	ø	5 Ditton Reach	Thames Ditton	ye4jxzzywz4i472j@runbox.co.uk	offset	1202 8605 9840 2291
12824955	ø	28 Little Woodbury Drive	Derby	bonnie753@yahoo.co.uk	current	1202 8620 1506 4579
12736845	ø	61 Kingsley Close	Thornton	x2yh@hotmail.co.uk	current	1202 8663 1022 4459
12978556	ø	26 Larkfield Park	Chepstow / Cas-Gwent	mohammed.young59@gmail.com	current	1202 8662 2487 3776
11783426	ø	1 Matheson Place	Wick	ryan1@googlemail.co.uk	current+	1202 8612 4044 5859
10306278	ø	29 Ferris Green	West End	lyla30@hotmail.com	current	1202 8677 5335 4488
10099638	ø	42 Crieff Walk	Hartlepool	nancy.clarke6884@runbox.co.uk	current	1202 8659 5033 8345
12067823	2018-10-11 00:00:00	24 Melbury Road	Cheadle Hulme	jessica.palmer@gmail.com	current	1202 8636 9333 2534
11401145	ø	7 Seymour Close	Basildon	arlo.fraser3@runbox.co.uk	current+	1202 8670 0078 4203
11948703	ø	15 Oak Tree Close	Long Bennington	jxklxlpcw@hotmail.co.uk	current	1202 8624 7860 9489

tdda head accounts1k.parquet '[f-z]*' 10

```
$
$ tdda cat accounts1k.parquet '[f-z]*' 10
accounts1k.parquet: 10 records; 9 fields. [Pandas]
```

open_date	title	forename	middle_name	surname	postcode	home_tel	mobile_tel	overdraft_limit
2010-03-07 00:00:00	Mrs	Clara	Ava	Walker	KT7 0ZT	0843 295 2714	0705 784 4779	0
2005-03-09 00:00:00	Mrs	Bonnie	ø	Kay	DE23 8TH	0505 300 4446	0705 426 8196	800
2011-11-28 00:00:00	Mr	Charles	Ronnie	Chapman	FY5 3RT	0140 939 8078	0775 815 3161	9600
2006-12-15 00:00:00	Mr	Mohammed	Charles	Young	NP16 2DX	0247 120 7888	0779 914 4439	2400
2008-11-02 00:00:00	Mr	Ryan	ø	Reid	BN17 6YC	0850 610 9741	0743 555 1523	2000
2016-03-15 00:00:00	Ms	Lyla	ø	Jackson	RG42 0EU	0601 572 4938	0753 164 3648	10400
2014-11-09 00:00:00	Miss	Nancy	Sienna	Clarke	TS25 5EY	0884 134 8768	0745 190 5358	0
2002-11-02 00:00:00	Miss	Jessica	ø	Palmer	SK8 9EU	0656 559 5460	0794 211 1635	6900
2009-12-20 00:00:00	Mr	Arlo	ø	Fraser	SS15 3GU	0569 168 3516	0781 950 3390	0
2007-06-01 00:00:00	Mr	Victor	ø	James	NG23 0TN	0864 390 9897	0775 119 1215	15600

```
$
```

LOOK AT THE METADATA ETC. (tdda ls -l)

tdda ls -l accounts1k.parquet

```
$ tdda ls -l accounts1k.parquet
accounts1k.parquet: 1,000 records; 16 fields. [Pandas]
```

Field	Pandas dtype	Min	Max	Nulls
account_number	int64	10000801	12994290	0
open_date	datetime64	2002-10-02 00:00:00	2018-11-17 00:00:00	0
close_date	datetime64	2003-08-12 00:00:00	2018-11-14 00:00:00	850
title	category	Dr	Prof	0
forename	category	Abigail	Zara	0
middle_name	category	Adam	Zara	510
surname	category	Adams	Young	0
address1	object	1 Alwold Crescent	9 Youngs Rise	0
city	category	Aberdeen	Yoxford	0
postcode	object	AB12 5PN	ZE1 3CN	0
home_tel	object	0103 774 2993	0997 919 1100	0
mobile_tel	object	0700 058 8424	0799 901 9613	0
email	object	abi.c@runbox.com	zyz@gmail.co.uk	0
account_type	category	basic	premium	0
overdraft_limit	int64	0	20000	0
cash_card_number	object	1202 8600 1578 0240	1202 8699 6870 9338	0

```
$
```

CONSTRAINT GENERATION (tdda discover)

1. ~~Copy examples somewhere:~~

   ~~`cd ~/tmp`~~
   ~~`tdda examples`~~
   ~~`cd constraints_examples`~~

Data also provided as .csv for inspection and alternative source.

2. Generate constraints from 1,000 bank customer records (**`accounts1k.parquet`**)

```
tdda discover -x accounts1k.parquet accounts1k.tdda
```

3. Examine output (**`accounts1k.tdda`**)

   (*Alternatively, also generate report,* **`accounts1k.html`**, *and examine that; see later)*

The **`-x`** *flag tells discover to include regular expression constraints for string fields.*

This is off by default, because it can be slow on large datasets.

less accounts1k.tdda


```
{
    "creation_metadata": {
        "local_time": "2026-06-02T21:53:58",
        "utc_time": "2026-06-02T20:53:58+00:00",
        "creator": "TDDA 3.0.03",
        "source": "accounts1k.parquet",
        "host": "gardot.local",
        "user": "njr",
        "dataset": "accounts1k.parquet",
        "n_records": 1000,
        "n_selected": 1000,
        "tddafile": "accounts1k.tdda"
    },
    "fields": {
        "account_number": {
            "type": "int",
            "min": 10000801,
            "max": 12994290,
            "sign": "positive",
            "max_nulls": 0,
            "no_duplicates": true
        },
        "open_date": {
            "type": "date",
            "min": "2002-10-02 00:00:00",
            "max": "2018-11-17 00:00:00",
            "max_nulls": 0
        },
        "close_date": {
            "type": "date",
            "min": "2003-08-12 00:00:00",
            "max": "2018-11-14 00:00:00"
        },
        "title": {
            "type": "string",
            "min_length": 1,
            "max_length": 4,
            "max_nulls": 0,
            "allowed_values": [
                "Dr",
                "M",
                "Miss",
                "Mr",
                "Mrs",
                "Ms",
                "Prof"
            ],
            "rex": [
                "^([A-Za-z]+)$"
            ]
        },
        "forename": {
            "type": "string",
            "min_length": 3,
            "max_length": 9,
            "max_nulls": 0,
            "rex": [
                "^([A-Z])([a-z]+)$"
            ]
        },
        "middle_name": {
            "type": "string",
            "min_length": 3,
            "max_length": 9,
            "rex": [
                "^([A-Z])([a-z]+)$"
            ]
        },
        "surname": {
            "type": "string",
            "min_length": 3,
            "max_length": 10,
            "max_nulls": 0,
            "rex": [
                "^([A-Za-z]+)$"
            ]
        },
        "address1": {
            "type": "string",
            "min_length": 9,
            "max_length": 26,
            "max_nulls": 0,
            "no_duplicates": true,
            "rex": [
                "^([0-9]{1,2}) ([A-Z])([a-z]+)$",
                "^4 Foresters' Close$",
                ⋮
            ]
        },
        "city": {
            "type": "string",
            "min_length": 3,
            "max_length": 29,
            "max_nulls": 0,
            "rex": [
                "^([^\\W0-9_]+)$",
                "^([A-Za-z]+) ([^\\W0-9_]+)$",
                ⋮
            ]
        },

        "postcode": {
            "type": "string",
            "min_length": 6,
            "max_length": 8,
            "max_nulls": 0,
            "no_duplicates": true,
            "rex": [
                "^([A-Z0-9]{2,4}) ([0-9])([A-Z]{2})$"
            ]
        },
        "home_tel": {
            "type": "string",
            "min_length": 13,
            "max_length": 13,
            "max_nulls": 0,
            "no_duplicates": true,
            "rex": [
                "^([0-9]{4}) ([0-9]{3}) ([0-9]{4})$"
            ]
        },
        "mobile_tel": {
            "type": "string",
            "min_length": 13,
            "max_length": 13,
            "max_nulls": 0,
            "no_duplicates": true,
            "rex": [
                "^([0-9]{4}) ([0-9]{3}) ([0-9]{4})$"
            ]
        },
        "email": {
            "type": "string",
            "min_length": 14,
            "max_length": 37,
            "max_nulls": 0,
            "no_duplicates": true,
            "rex": [
                "^([a-z0-9]+)@([a-z]+)\\.com$",
                "^([a-z0-9]+)[.@]([a-z0-9]+)[.@]([a-z]+)\\.([a-z]{2,3})$",
                "^([a-z]+)\\.([a-z0-9]+)@([a-z]+)\\.co\\.uk$"
            ]
        },
        "account_type": {
            "type": "string",
            "min_length": 5,
            "max_length": 8,
            "max_nulls": 0,
            "allowed_values": [
                “basic", "current",
                “current+", “offset", "premium"
            ],
            "rex": [
                "^([a-z]{5,7})$",
                "^current\\+$"
            ]
        },
        "overdraft_limit": {
            "type": "int",
            "min": 0,
            "max": 20000,
            "sign": "non-negative",
            "max_nulls": 0
        },
        "cash_card_number": {
            "type": "string",
            "min_length": 19,
            "max_length": 19,
            "max_nulls": 0,
            "no_duplicates": true,
            "rex": [
                "^1202 ([0-9]{4}) ([0-9]{4}) ([0-9]{4})$"
            ]
        }
    },
    "dataset": {
        "allowed_fields": [],
        "required_fields": [
            "*"
        ]
    }
}
```

CONSTRAINT VERIFICATION (DATA VALIDATION; `tdda verify`)

4. Perform verification of same data (as DataFrame). Should **pass**.

```
tdda verify accounts1k.parquet accounts1k.tdda
```

SUCCESSFUL VERIFICATION ON TRAINING DATA

tdda verify accounts1k.parquet accounts1k.tdda

```
$
$ tdda verify accounts1k.parquet accounts1k.tdda --dense
FIELDS:

account_number: 0 failures  6 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓  no_duplicates ✓
open_date: 0 failures  4 passes  type ✓  min ✓  max ✓  max_nulls ✓
close_date: 0 failures  3 passes  type ✓  min ✓  max ✓
title: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  allowed_values ✓  rex ✓
forename: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
middle_name: 0 failures  4 passes  type ✓  min_length ✓  max_length ✓  rex ✓
surname: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
address1: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓  rex ✓
city: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
postcode: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓  rex ✓
home_tel: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓  rex ✓
mobile_tel: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓  rex ✓
email: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓  rex ✓
account_type: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  allowed_values ✓  rex ✓
overdraft_limit: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓
cash_card_number: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓  rex ✓

DATASET:

Extra (disallowed) fields: None  Missing (required) fields: None

SUMMARY:

Constrained Fields: 16  Failing Fields: 0 (0.00%)
Constraints: 87  Failing Constraints: 0 (0.00%)
Extra (disallowed) fields: 0  Missing (required) fields: 0
$
```

SANITY CHECKING AND REFINEMENT

5. The next stage should be either to look at the constraints generated, or to try them on some "holdout" data.

   In either case we should then adapt the constraints using our knowledge and judgement, possibly with a domain expert, either by

   - Directly editing the `.tdda` file. (Careful of trailing commas!)
   - Or by editing the training *data*
     - removing the things that cause false positives and false negatives
     - e.g. remove out of range data, duplicates, illegal nulls, bad categories etc.
     - e.g. add for missing categories or over-tight ranges.

   Then run discover again.

accounts1k.tdda: METADATA (JSON)


```
"creation_metadata": {
    "local_time": "2026-06-02T08:24:37",
    "utc_time": "2026-06-02T07:24:37+00:00",
    "creator": "TDDA 3.0.03",
    "source": "accounts1k.parquet",
    "host": "gardot.local",
    "user": "njr",
    "dataset": "accounts1k.parquet",
    "n_records": 1000,
    "n_selected": 1000,
    "tddafile": "accounts1k.tdda"
},
```

FIELD CONSTRAINTS: ACCOUNT NUMBER

```
"fields": {
    "account_number": {
        "type": "int",
        "min": 10000801,
        "max": 12994290,
        "sign": "positive",
        "max_nulls": 0,
        "no_duplicates": true
    },
```

Sensible? 10000000 ?
99999999 ?

Technically redundant but userful?

✔✔✔

✔✔✔

FIELD CONSTRAINTS: DATE

```
"open_date": {
    "type": "date",
    "min": "2002-10-02 00:00:00",
    "max": "2018-11-17 00:00:00",
    "max_nulls": 0
},
```

Right date? Opened 2001-10-02

Really want "not in future"?

✔✔✔

```
"close_date": {
    "type": "date",
    "min": "2003-08-12 00:00:00",
    "max": "2018-11-14 00:00:00"
    (No max_nulls constraint)
},
```

As above

Really want not close before open

✔✔✔

STRING FIELD CONSTRAINTS

```
"title": {
    "type": "string",
    "min_length": 1,
    "max_length": 4,
    "max_nulls": 0,
    "allowed_values": [
        "Dr",
        "M",
        "Miss",
        "Mr",
        "Mrs",
        "Ms",
        "Prof"
    ],
    "rex": [
        "^([A-Za-z]+)$"
    ]
},
```

Don't need rex and allowed values: Pick one.

```
^[A-Z][a-z]*$
```

better

```
"forename": {
    "type": "string",
    "min_length": 3,
    "max_length": 9,
    "max_nulls": 0,
    "rex": [
        "^([A-Z])([a-z]+)$"
    ]
},
```

Not bad.
Lengths a bit tight?
No accents, hyphens etc.

```
"postcode": {
    "type": "string",
    "min_length": 6,
    "max_length": 8,
    "max_nulls": 0,
    "no_duplicates": true,
    "rex": [
        "^([A-Z0-9]{2,4}) ([0-9])([A-Z]{2})$"
    ]
},
```

Not bad.
Lengths a bit tight?
No accents, hyphens etc.

Better:

```
^[A-Z]{1,2}[0-9]{1,2}[A-Z?] [0-9][A-Z]{2}$
```

CONSTRAINT GENERATION (`tdda discover`)

4. Can add **`-r html`** to add an html report (**`accounts1k.html`**).

   Formats after **`-r`** can be any space-separated

   `html`

   `txt`

   `md`

   `json`

   `yaml`

   `toml`

```
tdda discover -x accounts1k.parquet accounts1k.tdda -r html md
```

THE DISCOVER (HTML) REPORT

TDDA Discover Report

accounts1k.html

Individual Field Constraints								
Field name	Type Allowed	Min Allowed	Max Allowed	Sign Allowed	Nulls Allowed	Duplicates Allowed	Values Allowed	Regular Expressions
account_number	int	10000801	12994290	positive	0	no		
open_date	date	2002-10-02	2018-11-17		0			
close_date	date	2003-08-12	2018-11-14					
title	string	1	4		0		"Dr" "M" "Miss" "Mr" "Mrs" "Ms" "Prof"	^([A-Za-z]+)$
forename	string	3	9		0			^([A-Z])([a-z]+)$
middle_name	string	3	9					^([A-Z])([a-z]+)$
surname	string	3	10		0			^([A-Za-z]+)$
address1	string	9	26		0	no		^([0-9]{1,2}) ([A-Z])([a-z]+)$ ^4 Foresters' Close$ ^([0-9]{1,2}) ([A-Z])([a-z]+) ([A-Z])([a-z]+)$ ^81 Golwg\-Y\-Mynydd$ ^2 Colne And Broughton Road$ ^([0-9]{1,2}) ([A-Z])([a-z]+) ([A-Za-z]+) ([A-Z])([a-z]+)$ ^([0-9]{1,2}) ([A-Z])([a-z]+)'s ([A-Z])([a-z]+)$ ^([0-9]{1,2}) St ([A-Z])([a-z]+)'s ([A-Z])([a-z]{3,5})$ ^([0-9]{2}) ([A-Z])([a-z]{2,3})\-Y\-([A-Z])([a-z]{2,3}) ([A-Z])([a-z]{3,4})$
city	string	3	29		0			^([^\W0-9_]+)$ ^([A-Za-z]+) ([^\W0-9_]+)$ ^([A-Z])([a-z]{2,3})\-([^\W0-9_]{4})$ ^([A-Z])([a-z]+) / ([A-Z])([a-z]+)$ ^([A-Z])([a-z]+) ([A-Za-z]+) ([A-Z])([a-z]+)$ ^([A-Z])([a-z]+)\-([a-z]+)\-([A-Z])([a-z]+)$ ^([A-Z])([a-z]{3,5})'s ([A-Z])([a-z]+)$ ^Lytham St Anne's$ ^Waltham on the Wolds$ ^([A-Z])([a-z]+) / ([A-Za-z]+) ([^\W0-9_]{4,5})$ ^([A-Z])([a-z]+) / ([A-Z])([a-z]+)\-([A-Za-z]{4,5})$ ^([A-Z])([a-z]+)\-([a-z]{2})\-the\-([A-Z])([a-z]{2,4})$ ^Bae Colwyn / Colwyn Bay$ ^Pontypool / Pont\-y\-pŵl$ ^Connah's Quay / Cei Connah$ ^Pen\-y\-bont ar Ogwr / Bridgend$
postcode	string	6	8		0	no		^([A-Z0-9]{2,4}) ([0-9])([A-Z]{2})$
home_tel	string	13	13		0	no		^([0-9]{4}) ([0-9]{3}) ([0-9]{4})$
mobile_tel	string	13	13		0	no		^([0-9]{4}) ([0-9]{3}) ([0-9]{4})$
email	string	14	37		0	no		^([a-z0-9]+)@([a-z]+)\.com$ ^([a-z0-9]+)([.@])([a-z0-9]+)([.@])([a-z]+)\.([a-z]{2,3})$ ^([a-z]+)\.([a-z0-9]+)@([a-z]+)\.co\.uk$
account_type	string	5	8		0		"basic" "current" "current+" "offset" "premium"	^([a-z]{5,7})$ ^current\+$
overdraft_limit	int	0	20000	non-negative	0			
cash_card_number	string	19	19		0	no		^1202 ([0-9]{4}) ([0-9]{4}) ([0-9]{4})$

DATA VALIDATION & ANOMALY DETECTION

6. Now run verification of larger dataset against the same constraints. Should **fail** (because, for example, range of account numbers is now greater).

```
tdda verify accounts25k.parquet accounts1k.tdda
```

You can use the `.csv` *files instead of the parquet files if you prefer.*

There's no difference in this case because the CSV files are "good" CSV files and conform to what the `tdda` *CSV file reader expects.*

CONSTRAINT FAILURES ON NON-TRAINING DATA

tdda verify accounts25k.parquet accounts1k.tdda

```
$
$ tdda verify accounts25k.parquet accounts1k.tdda --dense
FIELDS:

account_number: 2 failures  4 passes  type ✓  min ×  max ×  sign ✓  max_nulls ✓  no_duplicates ✓
open_date: 2 failures  2 passes  type ✓  min ×  max ×  max_nulls ✓
close_date: 2 failures  1 pass  type ✓  min ×  max ×
title: 0 failures  6 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  allowed_values ✓  rex ✓
forename: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
middle_name: 0 failures  4 passes  type ✓  min_length ✓  max_length ✓  rex ✓
surname: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
address1: 4 failures  2 passes  type ✓  min_length ×  max_length ×  max_nulls ✓  no_duplicates ×  rex ×
city: 2 failures  3 passes  type ✓  min_length ✓  max_length ×  max_nulls ✓  rex ×
postcode: 1 failure  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ×  rex ✓
home_tel: 1 failure  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ×  no_duplicates ✓  rex ✓
mobile_tel: 2 failures  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ×  no_duplicates ×  rex ✓
email: 2 failures  4 passes  type ✓  min_length ✓  max_length ×  max_nulls ✓  no_duplicates ×  rex ✓
account_type: 5 failures  1 pass  type ✓  min_length ×  max_length ×  max_nulls ×  allowed_values ×  rex ×
overdraft_limit: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓
cash_card_number: 2 failures  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ×  rex ×

DATASET:

Extra (disallowed) fields: None  Missing (required) fields: None

SUMMARY:

Constrained Fields: 16  Failing Fields: 11 (68.75%)
Constraints: 87  Failing Constraints: 25 (28.74%)
Extra (disallowed) fields: 0  Missing (required) fields: 0
$
```

REDUCED FAILURES WITH REFINED CONSTRAINTS

tdda verify accounts25k.parquet testdata/accounts1k_refined.tdda

```
$
$ tdda verify accounts25k.parquet testdata/accounts1k_refined.tdda --dense
FIELDS:

account_number: 0 failures  6 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓  no_duplicates ✓
open_date: 1 failure  2 passes  type ✓  min ×  max_nulls ✓
close_date: 1 failure  1 pass  type ✓  min ×
title: 2 failures  3 passes  type ✓  min_length ×  max_length ✓  max_nulls ✓  allowed_values ×
forename: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
middle_name: 0 failures  4 passes  type ✓  min_length ✓  max_length ✓  rex ✓
surname: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
address1: 0 failures  2 passes  type ✓  max_nulls ✓
city: 0 failures  2 passes  type ✓  max_nulls ✓
postcode: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  rex ✓
home_tel: 1 failure  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ×  rex ✓
mobile_tel: 1 failure  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ×  rex ✓
email: 0 failures  4 passes  type ✓  min_length ✓  max_nulls ✓  rex ✓
account_type: 4 failures  1 pass  type ✓  min_length ×  max_length ×  max_nulls ×  allowed_values ×
overdraft_limit: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓
cash_card_number: 2 failures  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ×  rex ×

DATASET:

Extra (disallowed) fields: None  Missing (required) fields: None

SUMMARY:

Constrained Fields: 16  Failing Fields: 7 (43.75%)
Constraints: 71  Failing Constraints: 12 (16.90%)
Extra (disallowed) fields: 0  Missing (required) fields: 0
$
```

FAILURE (ANOMALY) DETECTION (tdda detect)

Don't need the backslashes if you type the whole command on one line

6. "Detect" the failing records

```
tdda detect accounts25k.parquet \
     testdata/accounts1k_refined.tdda bads.csv \
     --per-constraint --output-fields --interleave
```

- This writes out the failing records to **bads.csv**. (Can use **.parquet** instead.)
- **--per-constraint** says write out a column for every constraint that ever fails, as well as an **nfailures** column.
- **--output-fields** says write all the original fields as well as the results fields (otherwise, it just writes a row number).
- **--interleave** says to interleave boolean columns saying which constraints failed with the original columns (otherwise they all go at after the input columns)

(ANOMALY) DETECTION ON NON-TRAINING DATA

```
tdda detect accounts25k.parquet                          \
     testdata/accounts1k_refined.tdda bads.csv           \
     --per-constraint --output-fields --interleave
```

```
Constrained Fields: 16
$ tdda detect accounts25k.parquet testdata/accounts1k_refined.tdda bads.csv --per-constraint --output-fields --in
terleave --dense
FIELDS:

open_date: 1 failure  2 passes  type ✓  min ×  max_nulls ✓
close_date: 1 failure  1 pass  type ✓  min ×
title: 2 failures  3 passes  type ✓  min_length ×  max_length ✓  max_nulls ✓  allowed_values ×
home_tel: 1 failure  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ×  rex ✓
mobile_tel: 1 failure  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ×  rex ✓
account_type: 4 failures  1 pass  type ✓  min_length ×  max_length ×  max_nulls ×  allowed_values ×
cash_card_number: 2 failures  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ×  rex ×

SUMMARY:

Records: 25,000  Failing Records: 543 (2.17%)
Constrained Fields: 16  Failing Fields: 7 (43.75%)
Constrained Values: 400,000  Failing Values: 552 (0.14%)
Constraints: 71  Failing Constraints: 12 (16.90%)
Extra (disallowed) fields: 0  Missing (required) fields: 0
$
```

tdda head -s bads.csv 10

Break headers between words

```
$ tdda head bads.csv 10 -s
bads.csv: 10 records; 29 fields. [Pandas]
Read by tdda.serial.csv_to_pandas with no external metadata
```

account number	open date	open date min ok	close date	close date min ok	title	title min length ok	title values ok	forename	middle name	surname	address1	city	postcode	home tel	home tel nonnull ok	mobile tel	mobile tel nonnull ok	email	account type	account type max length ok	account type min length ok	account type nonnull ok	account type values ok	overdraft limit	cash card number	cash card number nodups ok	cash card number rex ok	n failures
10001859	2009-08-05	True	∅	∅	M	False	False	Amelia	Mia	Graham	11 Sling Lane	Lower Broadheath	WR2 2PC	0848 351 6222	True	0712 677 0637	True	amelia.graham387@gmail.com	current	True	True	True	True	0	1202 8696 2349 4223	True	True	2
10014314	2004-04-13	True	∅	∅	M	False	False	Paul	∅	Millar	13 Chestnut Drive	Leigh	WN7 8EM	0923 483 2749	True	0742 657 3319	True	paul75@yahoo.com	current	True	True	True	True	900	1202 8688 8560 2451	True	True	2
10018147	2007-11-24	True	∅	∅	M	False	False	Harvey	Freddie	Jenkins	49 Northorpe Close	Kingston upon Hull	HU9 5KD	0666 416 5255	True	0733 365 0903	True	aiaytv6b3lvbkyt6@yahoo.co.uk	current	True	True	True	True	18100	1202 8646 5803 5150	True	True	2
10022662	2008-09-16	True	∅	∅	M	False	False	Louie	∅	Clarke	4 Estuary Boulevard	Liverpool	L24 1NA	0820 750 4223	True	0702 749 9611	True	louie.clarke578@googlemail.co.uk	basic	True	True	True	True	6800	1202 8624 9552 9598	True	True	2
10025862	2014-10-09	True	∅	∅	M	False	False	Isla	Clara	Russell	27 The Crescent	Whickham	NE16 0XV	0394 634 8715	True	0735 214 2517	True	isla9581@googlemail.co.uk	current	True	True	True	True	6000	1202 8634 9145 1456	True	True	2
10027221	2002-10-30	True	∅	∅	M	False	False	Grace	Rose	Collins	12 Cairnton Street	Tillicoultry	FK13 8HR	0909 388 4856	True	0770 287 8623	True	grace3972@yahoo.co.uk	current	True	True	True	True	0	1202 8668 9889 3518	True	True	2
10029977	2009-09-11	True	2009-09-14	True	M	False	False	Erin	∅	Fisher	24 Clough Avenue	Bamber Bridge	PR5 8VS	0319 861 1093	True	0756 955 9400	True	erin458885@googlemail.co.uk	offset	True	True	True	True	0	1202 8676 3873 7980	True	True	2
10033576	2003-11-07	True	∅	∅	M	False	False	Nancy	∅	Knight	21 Templars Way	Penkridge	ST19 6HN	0504 722 0846	True	0719 976 2859	True	nancy.knight0342@yahoo.co.uk	current	True	True	True	True	13600	1202 8666 1311 7498	True	True	2
10037105	2007-12-21	True	∅	∅	M	False	False	Yasmin	∅	Hughes	3 Plantagenet Square	Northampton	NN4 6CR	0263 079 9033	True	0755 381 5920	True	ws7arkz5d@gmail.com	current	True	True	True	True	6000	1202 8651 6887 2482	True	True	2
10043363	2004-02-11	True	∅	∅	M	False	False	Mila	Rosie	Murray	14 Gilfrid Close	Hillingdon	UB8 6LR	0696 861 4151	True	0759 696 9109	True	mila.murray1931@googlemail.com	current	True	True	True	True	9000	1202 8632 3417 6401	True	True	2


```
$ tdda head bads.csv 10 -s
bads.csv: 10 records; 29 fields. [Pandas]
Read by tdda.serial.csv_to_pandas with no external metadata
```


account number	open date	open date min ok	close date	close date min ok	title	title min length ok	title values ok	forenar	n failures
10001859	2009-08-05	True	∅	∅	M	False	False	Amel	2
10014314	2004-04-13	True	∅	∅	M	False	False	Pa	2
10018147	2007-11-24	True	∅	∅	M	False	False	Harv	2
10022662	2008-09-16	True	∅	∅	M	False	False	Lou	2
10025862	2014-10-09	True	∅	∅	M	False	False	Is	2
10027221	2002-10-30	True	∅	∅	M	False	False	Gra	2
10029977	2009-09-11	True	2009-09-14	True	M	False	False	Er	2
10033576	2003-11-07	True	∅	∅	M	False	False	Nan	2
10037105	2007-12-21	True	∅	∅	M	False	False	Yasm	2
10043363	2004-02-11	True	∅	∅	M	False	False	Mi	2

$

FAILURE (ANOMALY) DETECTION (`tdda detect`)

6. Can add extra reports to **`tdda detect`**

Don't need the backslashes if you type the whole command on one line

```
-o report.html  -r html txt  --key account_number
```

Report filename (stem for all)

Report formats

Field field(s): Field value added to each failure

Name	Failures		Type			Minimum			Maximum			Sign			Max Nulls			Duplicates			Values			Rex		
	Values	Constraints	Allowed	Actual	✓	Allowed	Actual	✓	Allowed	Actual	✓	Allowed	Actual	✓	Allowed	Actual	✓	Allowed	Actual	✓	Allowed	Actual	✓	Allowed	Actual	✓
account_number	56	2	int	int	✓	10000801	10000191 ✗		12994290	12999930 ✗		positive	positive	✓	0	0	✓	no	no	✓						
open_date	13	2	date	date	✓	"2002-10-02 00:00:00"	1970-01-01 00:00:00 ✗		"2018-11-17 00:00:00"	2018-11-19 00:00:00 ✗					0	0	✓									
close_date	20	2	date	date	✓	"2003-08-12 00:00:00"	1990-03-09 00:00:00 ✗		"2018-11-14 00:00:00"	2018-11-19 00:00:00 ✗																
title	0	0	string	string	✓	1	1	✓	4	4	✓				0	0	✓				"Dr" "M" "Miss" "Mr" "Mrs" "Ms" "Prof"		✓	^([A-Za-z]+)$		✓
forename	0	0	string	string	✓	3	3	✓	9	9	✓				0	0	✓							^([A-Z])([a-z]+)$		✓
middle_name	0	0	string	string	✓	3	3.0	✓	9	9.0	✓													^([A-Z])([a-z]+)$		✓
surname	0	0	string	string	✓	3	3	✓	10	10	✓				0	0	✓							^([A-Za-z]+)$		✓
address1	1,043	4	string	string	✓	9	6 ✗		26	37 ✗					0	0	✓	no	395 dups ✗					^([0-9]{1,2}) ([A-Z])([a-z]+)$ ^4 Foresters' Close$ ^([0-9]{1,2}) ([A-Z])([a-z]+) ([A-Z])([a-z]+)$ ^81 Golwg\-Y\-Mynydd$ ^2 Colne And Broughton Road$ ^([0-9]{1,2}) ([A-Z])([a-z]+) ([A-Za-z]+) ([A-Z])([a-z]+)$ ^([0-9]{1,2}) ([A-Z])([a-z]+)'s ([A-Z])([a-z]+)$ ^([0-9]{1,2}) St ([A-Z])([a-z]+)'s ([A-Z])([a-z]{3,5})$ ^([0-9]{2}) ([A-Z])([a-z]{2,3})\-Y\-([A-Z])([a-z]{2,3}) ([A-Z])([a-z]{3,4})$	"17 Glyn-Gaer Road"	✗
city	328	2	string	string	✓	3	3	✓	29	40 ✗					0	0	✓							^([^\W0-9_]+)$ ^([A-Za-z]+) ([^\W0-9_]+)$ ^([A-Z])([a-z]{2,3})\-([^\W0-9_]{4})$ ^([A-Z])([a-z]+) / ([A-Z])([a-z]+)$ ^([A-Z])([a-z]+) ([A-Za-z]+) ([A-Z])([a-z]+)$ ^([A-Z])([a-z]+)\-([a-z]+)\-([A-Z])([a-z]+)$ ^([A-Z])([a-z]{3,5})'s ([A-Z])([a-z]+)$ ^Lytham St Anne's$ ^Waltham on the Wolds$ ^([A-Z])([a-z]+) / ([A-Za-z]+) ([^\W0-9_]{4,5})$ ^([A-Z])([a-z]+) / ([A-Z])([a-z]+)\-([A-Za-z]{4,5})$ ^([A-Z])([a-z]+)\-([a-z]{2})\-the\-([A-Z])([a-z]{2,4})$ ^Bae Colwyn / Colwyn Bay$ ^Pontypool / Pont\-y\-pŵl$ ^Connah's Quay / Cei Connah$ ^Pen\-y\-bont ar Ogwr / Bridgend$	"Pen-pedair-heol"	✗
postcode	744	1	string	string	✓	6	6	✓	8	8	✓				0	0	✓	no	374 dups ✗					^([A-Z0-9]{2,4}) ([0-9])([A-Z]{2})$		✓
home_tel	2	1	string	string	✓	13	13	✓	13	13	✓				0	2 ✗		no	no	✓				^([0-9]{4}) ([0-9]{3}) ([0-9]{4})$		✓
mobile_tel	4	2	string	string	✓	13	13.0	✓	13	13.0	✓				0	2 ✗		no	1 dups ✗					^([0-9]{4}) ([0-9]{3}) ([0-9]{4})$		✓
email	19	2	string	string	✓	14	14	✓	37	39 ✗					0	0	✓	no	3 dups ✗					^([a-z0-9]+)@([a-z]+)\.com$ ^([a-z0-9]+)([.@])([a-z0-9]+)([.@])([a-z]+)\.([a-z]{2,3})$ ^([a-z]+)\.([a-z0-9]+)@([a-z]+)\.co\.uk$		✓
account_type	5	5	string	string	✓	5	0.0 ✗		8	15.0 ✗					0	1 ✗					"basic" "current" "current+" "offset" "premium"	e.g. "" ✗		^([a-z]{5,7})$ ^current\+$	"no longer valid"	✗
overdraft_limit	0	0	int	int	✓	0	0	✓	20000	20000	✓	non-negative	non-negative	✓	0	0	✓									
cash_card_number	3	2	string	string	✓	19	19	✓	19	19	✓				0	0	✓	no	1 dups ✗					^1202 ([0-9]{4}) ([0-9]{4}) ([0-9]{4})$	"4003 6070 5668 5816"	✗

Fields:

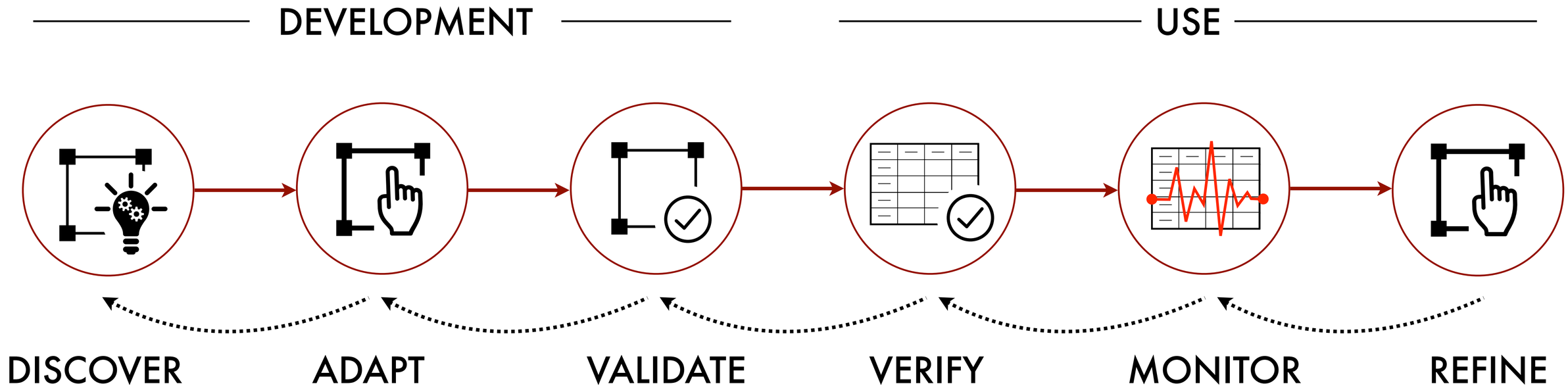
Field: account_number

- **Constraint:** `min` : `10000801`
- **Failures:** 6 / 25,000 (0.02%)
  - `10000191`: `10000191`
  - `10000307`: `10000307`
  - `10000335`: `10000335`
  - `10000422`: `10000422`
  - `10000514`: `10000514`
  - `10000600`: `10000600`
- **Constraint:** `max` : `12994290`

Failing Values (dataset level)

Failures summary (dataset level)

GENERATING CONSTRAINTS & VERIFYING DATA



training data

operational data

holdout data?

REFINING CONSTRAINTS

IDEAL PROCESS

- Discover on subset of data (“training data”)
- Read the constraints
- Adapt*
- Apply to holdout data
- Adapt*
- Operationalise
- Monitor
- Adapt*

* Tighten / Relax / Add / Delete / Choose Among

REDUCED PROCESS

- Discover on the training data
- Operationalise
- Monitor

FALSE NEGATIVES

Add or tighten constraints

TRUE POSITIVES

Act / Reject
Or fix root cause
Or improve normalisation, cleansing, upstream pipeline…

FALSE POSITIVES

Relax or remove constraints

ABSENT CONSTRAINTS

Gregory (Scotland Yard detective): *"Is there any other point to which you would wish to draw my attention?"*
Holmes: *"To the curious incident of the dog in the night-time."*
Gregory: *"The dog did nothing in the night-time."*
Holmes: *"That was the curious incident."*

— *Silver Blaze*, in *Memoirs of Sherlock Holmes*
Arthur Conan Doyle, 1892.

ABSENT CONSTRAINTS

- *Why is there not a no-duplicates on that field that clearly shouldn't have duplicates?*
- *Why is the maximum age 2026?*
- *Why does our categorical field that only takes five simple values show a regular expression from hell?*
- *Why is there no* `max_nulls=0` *constraint on our primary key field?*
- *Why is the bound on temperature in Kelvin negative?*
- *Why is min date* `1970-01-01T00:00:00.000`*?*
- *Should allowed values really include* `"claude_test_ünîcode_value_🤖💪"`*?*

ENGINE AND BACKEND

- For parquet files and flat files you can use
  - `--pandas` to force use of pandas engine (currently, the default)
  - `--polars` to force use of polars engine (might become the default)
  - `-B o`, with pandas, to force use of original dtype backend (ints promoted to reals with nulls, object for strings, booleans etc.)
  - `-B n`, with pandas, to force use of `numpy_nullable` dtype backend (`Int64` nullable ints etc)
  - `-B a`, with pandas, to force use of Apache `pyarrow` dtype backend (`int64[pyarrow]` etc)

RDBMS

- TDDA's constraints commands/APIs also work with data in relational databases (Postgres, MySQL/MariaDB, SQLite3)
- Interface is almost identical
- Use **`pg:tablename`** etc. after setting up a **`.dbCredentials.pg`** file in the home directory
- In general, data does not need to be extracted from database (SQL issued) except for regular expression inference.
- Out of scope for today. Documented in book and at tdda.readthedocs.info.

PYTHON API

- All functionality also available through Python APIs
- Out of scope for today
- Documented at tdda.readthedocs.info

tdda.serial

"safe sex for CSV files"

FLAT FILES

- Everything we did with `tdda discover`, `verify`, `detect` (and indeed, `head`, `cat`, `ls`) will also work with the flat files provided ("csv files", tab-separated, pipe-separated etc.).
- But flat files usually have no type information / other metadata
  - Easy to misread/miswrite
  - Problems with encodings, types, dates, nulls, quoting, integers (Pandas!), separators and more
  - See *Falsehoods Programmers Believe About CSVs*, Jesse Donat

    `donatstudios.com/Falsehoods-Programmers-Believe-About-CSVs`

PANDAS, POLARS, ETC.

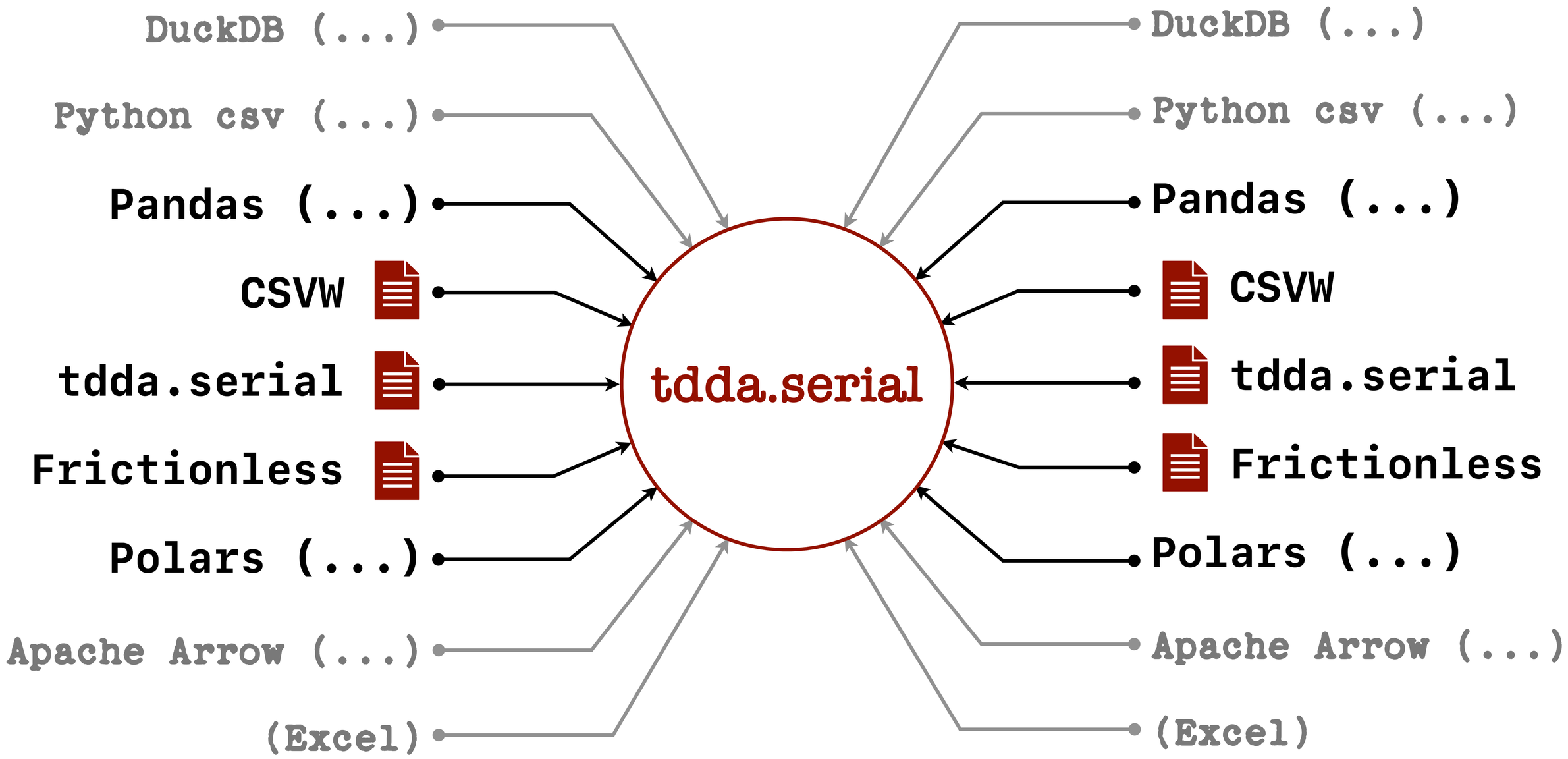
- Most flat-file readers have pretty flexible read functions (`pandas.read_csv`, `polars.read_csv`) with many parameters to control reading flat files.
- Some also have “intelligent” readers that try to infer formats of flat files, and are often quite effective
- Most flat-file writers have some flexibility in writing flat files, though typically less than in reading (which makes sense)
- Despite this, exchanging data with flat files is error prone and subtle errors are often only discovered later
- Using read-time inference can give different results on different datasets.
- Validating typed data in a data frame, database table, or from a parquet file is well-defined
- Validating data from a flat file depend on reading the file correctly/appropriately and validation results are affected by how the file is read.

tdda.serial is a JSON description
of the format used (or to be used)
in a flat file.

A tdda.serial file is usually shared between
multiple flat files and forms a contract
between the writer and reader.

The tdda library provides tools for reading and writing
flat files with serial metadata (pandas, polars, …) and
converting between tdda.serial, CSVW, and Frictionless.

VISION FOR tdda.serial



serial_examples/alk.csv.serial ("VANILLA" FORMAT)


```
{
    "format": "http://tdda.info/ns/tdda.serial",
    "writer": "tdda.serial-3.0.03",
    "tdda.serial": {
        "fields": [
            {
                "name": "account_number",
                "fieldtype": "int"
            },
            {
                "name": "open_date",
                "fieldtype": "date"
            },
            {
                "name": "close_date",
                "fieldtype": "date"
            },
            {
                "name": "title",
                "fieldtype": "string"
            },
            {
                "name": "forename",
                "fieldtype": "string"
            },
            {
                "name": "middle_name",
                "fieldtype": "string"
            },
            {
                "name": "surname",
                "fieldtype": "string"
            },
            {
                "name": "address1",
                "fieldtype": "string"
            },
            {
                "name": "city",
                "fieldtype": "string"
            },
            {
                "name": "postcode",
                "fieldtype": "string"
            },
            {
                "name": "home_tel",
                "fieldtype": "string"
            },
            {
                "name": "mobile_tel",
                "fieldtype": "string"
            },
            {
                "name": "email",
                "fieldtype": "string"
            },
            {
                "name": "account_type",
                "fieldtype": "string"
            },
            {
                "name": "overdraft_limit",
                "fieldtype": "int"
            },
            {
                "name": "cash_card_number",
                "fieldtype": "string"
            },
            {
                "name": "is_open",
                "fieldtype": "bool"
            }
        ],
        "encoding": "utf-8",
        "delimiter": ",",
        "quote_char": "\"",
        "date_format": "YYYY-MM-DD",
        "null_indicator": "",
        "header_row_count": 1,
        "header_row": 0,
        "quoting": "QUOTE_STRINGS_ONLY"
    }
}
```

serial_examples

a1k.csv.serial

```
"encoding": "utf-8",
"delimiter": ",",
"quote_char": "\"",
"date_format": "2000-10-31",
"null_indicator": "",
"header_row_count": 1,
"header_row": 0,
"quoting": "QUOTE_STRINGS_ONLY"
```

b1k-crazy.tsv.serial

```
"encoding": "latin-1",
"delimiter": "\t",
"quote_char": "'",
"stutter_quotes": true,
"date_format": "2000-10-31",
"null_indicator": "NA",
"header_row_count": 1,
"header_row": 0,
"quoting": "QUOTE_STRINGS_ONLY"
```

tdda.serial & CONSTRAINTS

In `examples_serial`:

```
tdda discover -x a1k-NULL.csv: accounts.tdda

tdda verify a1k-eurod.csv: accounts.tdda

tdda detect b1k-crazy.tsv:
     accounts.tdda bads.csv
     --per-constraint --output-fields --interleave
```

tdda.serial: Wildcards

Throughout tdda library, when working with flat files (e.g. .csv files) you can:

- Add a colon to pathnames to ask tdda to find a matching metadata file describing the flat-file format:

  e.g. **`foo.txt:`** *matches e.g.*

  `foo.serial, foo.txt.serial @.serial, f@.serial` *tdda.serial*

  `foo-metadata.json, foo.json` *CSV on the Web (CSVW)*

  `foo.schema.json, foo.resource.yaml, foo.package.json` *Frictionless*

- Add the name of a metadata file after a colon to specify a particular metadata file:

  e.g. **`foo.txt:metadata.serial`**

  `foo.txt:metadata.package.json`

  `foo.txt:metadata.json`

This can assist with more accurate flat-file reading.

tdda.serial: example formats (../serial_examples)

All the `a1k` files contain the same data as `accounts1k.csv` and `a1l.csv`

`a1k.csv`	"Vanilla": `UTF-8`, blank nulls, double quotes, stutter, `ISO 8601` dates…
`a1k-eurod.csv`	`31/10/2000`
`a1k-NULL.csv`	NULL as null marker
`a1k-sq-esc.csv`	Single quotes with backslash escapes for embedded quotes (`'I\'ll'`)
`a1k-sq-stutter.csv`	Single quotes with stuttering for embedded quotes `'I''ll'`
`a1k-usd.csv`	`10/31/2000`
`a1k-yes-no.csv`	Boolean values `yes` and `no`
`a1k.psv`	Pipe separator (also `a1k-pipe.txt`)
`a1k.tsv`	Tab separator (also `a1k-tab.txt`)

All have `.serial` files and `tdda cat`, `tdda verify` etc. should work if you add colon to CSV path.

They should validate against `accounts.tdda`.

`tdda verify a1k-usd.csv: accounts.tdda`

tdda.serial: reading flat files

- Given a metadata file (tdda.serial file, CSVW file, or Frictionless)
  - Read flat file with `csv_to_pandas` or `csv_to_polars`, e.g.

```
from tdda.serial import csv_to_pandas, csv_to_polars

df = csv_to_polars('a1k-usd.csv:')
df = csv_to_pandas('a1k.csv', find_md = True)

df = csv_to_polars('a1k-eurod.csv', 'a1k-eurod.csv.serial')
df = csv_to_pandas('a1k-yes-no-metadata.json')
df = csv_to_polars('a1k-pipe.txt',
                   md_path='a1k-pipe.txt.resource.yaml')
```

Use name matching inc. wildcards (tdda.serial, & Frictionless

tdda.serial: writing flat files

Write flat file with `pandas_to_csv` (`polars_to_csv` soon), e.g.

- Write *with format determined by metadata file:*

```
from tdda.serial import csv_to_pandas, pandas_to_csv

df = csv_to_pandas('a1k-usd.csv:')
pandas_to_csv(df, 'out.csv', md_inpath='a1k-usd.csv.serial')
```

- Write *data and metadata file specifying format together:*

```
from tdda.serial import csv_to_pandas, pandas_to_csv

df = csv_to_pandas('a1k-pipe.txt:')
pandas_to_csv(df, 'out.psv', md_outpath='out.serial', sep='|')
```

Can include format specifiers

tdda.serial: inferring format

`tdda serial a1k-NULL.csv a1k-NULL.serial`	tdda.serial
`tdda serial a1k-tab.txt a1k-tab-metadata.json`	CSVW
`tdda serial a1k-usd.csv a1k-usd.resource.json`	Frictionless resource
`tdda serial a1k-eurod.csv a1k-eurod.package.yaml`	Frictionless package

There are switches for more specificity and to help in cases of ambiguity

tdda.serial: conversions

```
tdda serial IN-METADATA OUT-METADATA
```

e.g.

```
tdda serial a1k.csv.serial a1k-metadata.json

tdda serial a1k-metadata.json a1k.schema.yaml

tdda serial a1k-schema.yaml a1k.serial
```

tdda.serial: code generation

e.g.

metadata file *script to write*

```
tdda serial b1k-crazy.tsv.serial read.py    \

            --to pd.r --for b1k-crazy.tsv
```

target *optional data file to read*

(`pandas.read_csv`) *(Without this, it writes a read function but does not call it)*

Generated Python code does *not* import `tdda` or read the metadata file

tdda serial b1k-crazy.tsv.serial read.py \
--to pd.r --for b1k-crazy.tsv

```
$
$ tdda serial b1k-crazy.tsv read.py --to pd.r --for b1k-crazy.tsv
$ cat read.py
import pandas as pd

def read_data(inpath):
    return pd.read_csv(
        inpath,
        sep='\t',
        encoding='latin-1',
        quotechar="'",
        doublequote=True,
        dtype={
            'account_number': 'Int64',
            'title': 'string',
            'forename': 'string',
            'middle_name': 'string',
            'surname': 'string',
            'address1': 'string',
            'city': 'string',
            'postcode': 'string',
            'home_tel': 'string',
            'mobile_tel': 'string',
            'email': 'string',
            'account_type': 'string',
            'city': 'string',
            'postcode': 'string',
            'home_tel': 'string',
            'mobile_tel': 'string',
            'email': 'string',
            'account_type': 'string',
            'overdraft_limit': 'Int64',
            'cash_card_number': 'string',
            'is_open': 'boolean'
        },
        date_format={
            'open_date': 'ISO8601',
            'close_date': 'ISO8601'
        },
        parse_dates=[
            'open_date',
            'close_date'
        ],
        na_values='NA',
        keep_default_na=False
    )

df = read_data('b1k-crazy.tsv')
$
```

Reference Testing

The Parables of Anne and Beth

SITREP

Anne & Beth are data scientists with parallel roles in different organizations.

Each previously performed an analysis using data to the end of Q2 2024.

The analyses were good, and were seen to be good.

2024-10-25 (Friday): "Re-run the numbers using data to the end of Q3".

THE PARABLE OF PARAMETERIZATION

BETH

1. Copies her previous Jupyter Notebook. Renames as `analysis-Q3.ipynb`.
2. Points to the new data.
3. Tries to run it: fails with errors (crashes).
    - Beth had previously run cells out of order; never fixed up.
4. Spends rest of day fixing it.
5. 16:30, calls her partner to say will be home late.

ANNE

1. Types `make test`
    - Tests (based on Q2) data pass.
2. Types `make analysis-2024Q3.pdf`.
    - Script picks up corresponding data `data-2024Q3.csv`; builds PDF.
    - "Output consistency checks: Pass".
3. Anne reads and sanity checks the report.
4. 09:30: Anne plans the rest of her Friday.

THE PARABLE OF PARAMETERIZATION

Duplication (encourages Errors of Process);
Violates DRY/single source of truth

BETH

1. Copies her previous Jupyter Notebook.
   Renames as `analysis-Q3.ipynb`.
2. Points to the new data.
3. Tries to run it: fails with errors (crashes).
   - Beth had previously run cells out of order; never fixed up.
4. Spends rest of day fixing it.
5. 16:30, calls her partner to say will be home late.

Error of implementation

Avoid Errors of Implementation
Reference tests

Avoid Errors of Process
Parameterization

ANNE

1. Types `make test`
   - Tests (based on Q2) data pass.
2. Types `make analysis-2024Q3.pdf`.
   - Script picks up corresponding data `data-2024Q3.csv`; builds PDF.
   - "Output consistency checks: Pass".
3. Anne reads sanity checks, issues the report.
4. 09:30: Anne plans the rest of her Friday.

Data validation / live testing / defensive programming

Sanity checking of results
Avoid Errors of Interpretation etc.

Detect Errors of Applicablity, Process, Implementation

THE PARABLE OF TESTING

BETH

1. Copies her previous Jupyter Notebook.
2. Renames as `analysis-Q3.ipynb`.
3. Points to new data: runs without error.
4. Issues results; plans rest of her Friday.
5. Monday: email flood: "Obviously wrong".
6. Puzzled, opens `analysis-Q2.ipynb`, clears and reruns original analysis:
    - All the graphs and numbers change.
7. Beth faces a week of toil and embarrassment.

ANNE

1. Types `make test`. Q2-based tests FAIL.
2. Realises updated helper library now does transformations previously in her script.
3. Removes transformations; reruns tests. Pass.
4. Types `make analysis-2024Q3.pdf`
    - Script builds PDF.
    - "Output consistency checks: Pass".
5. Anne commits updated code to Git.
6. Reads, sanity checks report; issues results.

THE PARABLE OF TESTING

BETH

1. Copies her previous Jupyter Notebook.
2. Renames as `analysis-Q3.ipynb`.
3. Points to new data: runs without error.
4. Issues results; plans rest of her Friday.
5. Monday: email flood: "Obviously wrong".
6. Puzzled, opens `analysis-Q2.ipynb`, clears and reruns original analysis:
   - All the graphs and numbers change.
7. Beth faces a week of toil and embarrassment.

Error of Implementation and/or Error of Process

Detects Error of Implementation
Reference test

Avoids Error of Applicabilty
Reference test

ANNE

1. Types `make test`. Q2-based tests FAIL.
2. Realises updated helper library now does transformations previously in her script.
3. Removes transformations; reruns tests. Pass.
4. Types `make analysis-2024Q3.pdf`
   - Script builds PDF.
   - "Output consistency checks: Pass".
5. Anne commits updated code to Git.
6. Reads, sanity checks report; issues results.

Single source of truth (DRY)
Avoids future Errors of Process

THE PARABLE OF UNITS

BETH

1. Copies, renames, runs, issues results.
2. 16:00: Phone call. "Obviously wrong".
3. Checks Q2 notebook is stable; it is.
4. Stares at notebooks and data. Spots `PurchasePrice` is three orders of magnitude bigger in Q3.
5. Calls data supplier: £k changed to £.
6. Beth divides by 1,000 in Q3 notebook. Renames as `analysis-Q3_FIXED.ipynb`.
7. Fires off "Garbage-in-garbage out" tirade.
8. Issues updated results.
9. Why is everyone else is lazy and incompetent?

ANNE

1. Types `make test`. Q2-based tests pass.
2. Types `make analysis-2024Q3.pdf`:

```
Input Data Check failed for PurchasePrice_kGBP
  Max expected: GBP 10.0k
  Max found:    GBP 7,843.21k
```

   - Anne had noticed Q2 purchase price was in £k so her script uses a renamed copy variable after load.
3. Calls data supplier: Confirms £k changed to £.
4. Gets supplier to change field name and resupply.
5. Anne adds conditional logic based on field name.
6. Adds new test with `PurchasePriceGBP`. Pass.
7. Commits change to Git repo.
8. Runs Q3, reads, sanity checks report; issues results.

THE PARABLE OF UNITS

BETH

Error of Applicability

1. Copies, renames, runs, issues results.
2. 16:00: Phone call. "Obviously wrong".
3. Checks Q2 notebook is stable; it is.
4. Stares at notebooks and data. Spots `Purchase Price` is three orders of magnitude bigger in Q3.
5. Calls data supplier: £k changed to £.
6. Beth divides by 1,000 in Q3 notebook. Renames as `analysis-Q3_FIXED.ipynb`.
7. Fires off "Garbage-in-garbage out" tirade.
8. Issues updated results.
9. Why is everyone else is lazy and incompetent?

The slacker's lament: "Not my fault, guv"
Error of Judgement?

Avoid future Error of Applicability

Detects Error of Applicability
Data Validation

ANNE

Avoid Error of Applicability
Defensive programming & good documentation

1. Types `make test`. Q2-based tests pass.
2. Types `make analysis-2024Q3.pdf`:

```
Input Data Check failed for PurchasePrice_kGBP
  Max expected: GBP 10.0k
  Max found:    GBP 7,843.21k
```

- Anne had noticed Q2 purchase price was in £k so her script uses a renamed copy variable after load.

3. Calls data supplier: Confirms £k changed to £.
4. Gets supplier to change field name and resupply.
5. Anne adds conditional logic based on field name.
6. Adds new test with `PurchasePriceGBP`. Pass.
7. Commits change to Git repo.
8. Runs Q3, reads, sanity checks report; issues results.

New reference test

DRY/*Parameterization*

THE PARABLE OF APPLICABILITY

BETH

1. Copies, renames, runs:
   - Crashes with `ZeroDivisionError: division by zero`
2. Can see that reference category used for indexing is absent in Q_3 data.
3. Calls data supplier; confirms is correct.
4. Adds extra code to handle case where reference category is absent.
5. Renames as `analysis-Q3_FIXED_2.ipynb`.
6. Issues updated report with note grumbling about no one ever telling Beth stuff she needs to know.
7. Makes mental note to base her next analysis on the `FIXED_2` Notebook.

ANNE

1. Types `make test`. Q2-based tests pass.
2. Types `make analysis-2024Q3.pdf`. Error:

   ```
   No data for Reference Store 001.
   If this is right, you need to
   choose a different Reference Store.
   ```
3. Checks with supplier who confirms data is correct.
4. Updates code to handle absent data for up to three candidate reference stores.
5. Adds tests for missing Reference Stores. Pass.
6. Anne commits change to Git repo.
7. Runs Q_3, reads, sanity checks report; issues results.

THE PARABLE OF APPLICABILITY

BETH

Error of Implementation / Error of Applicability

1. Copies, renames, runs:
   - Crashes with `ZeroDivisionError: division by zero`
2. Can see that reference category used for indexing is absent in Q_3 data.
3. Calls data supplier; confirms is correct.
4. Adds extra code to handle case where reference category is absent.
5. Renames as `analysis-Q3_FIXED_2.ipynb`.
6. Issues updated report with note grumbling about no one ever telling Beth stuff she needs to know.
7. Makes mental note to base her next analysis on the `FIXED_2` Notebook.

Error of Judgement?

Error of Process in the making

ANNE

Avoid Error of Applicability

Defensive programming & good documentation

1. Types `make test`. Q2-based tests pass.
2. Types `make analysis-2024Q3.pdf`. Error:

   ```
   No data for Reference Store 001.
   If this is right, you need to
   choose a different Reference Store.
   ```
3. Checks with supplier who confirms data is correct.
4. Updates code to handle absent data for up to three candidate reference stores.

Avoid future Error of Applicability

5. Adds tests for missing Reference Stores. Pass.
6. Anne commits change to Git repo.

New reference test

7. Runs Q_3, reads, sanity checks report; issues results.

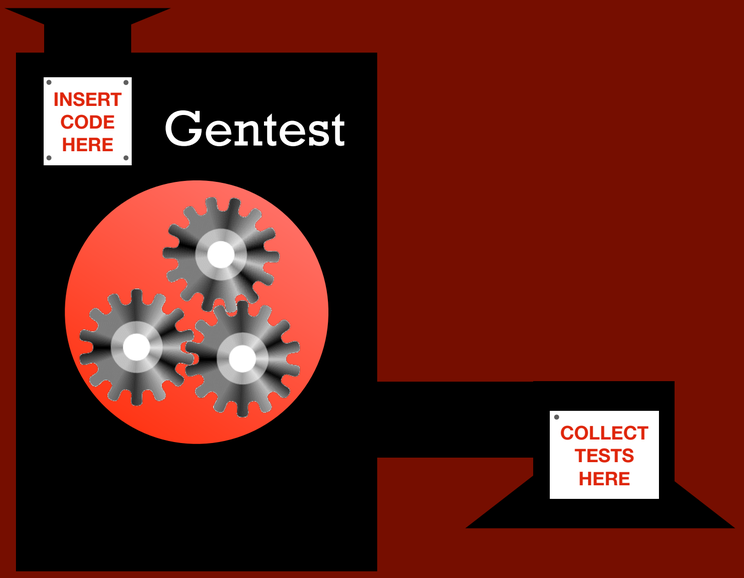
Avoid future Error of Process

Happy families are alike;
each unhappy family
is unhappy in its own way

— Leo Tolstoy
Anna Karenina

Good analyses are alike;
each problematical analysis
is problematical in its own way

Here endeth
the sermon



REFERENCE TESTS & AUTOMATIC TEST GENERATION WITH GENTEST

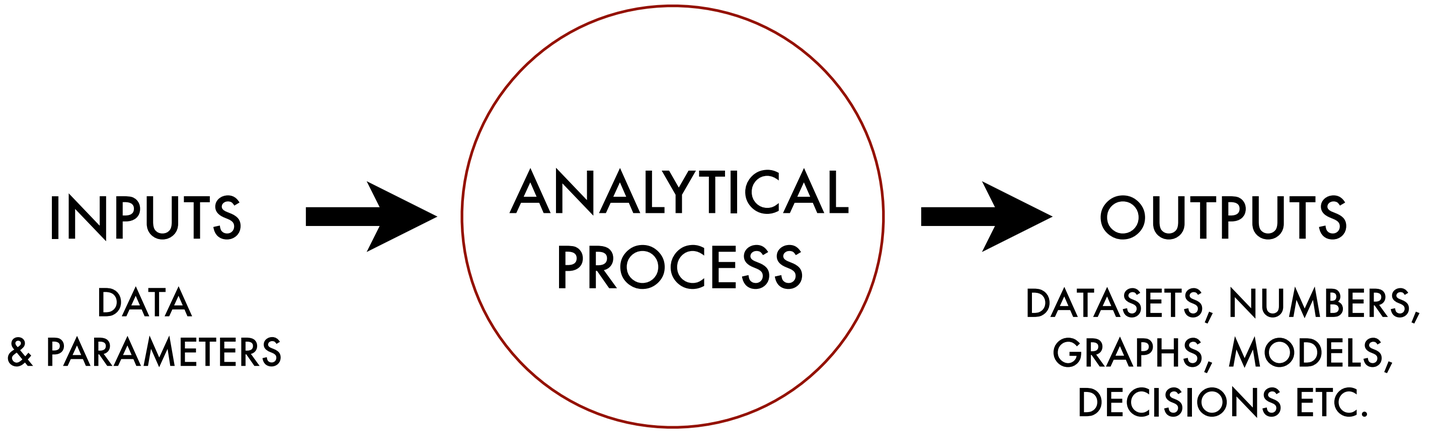
TESTING DATA PIPELINES

… is different from testing other code

- Can't write the tests first because the output is too hard to generate; ⟶ *Generate reference results from code*
- Output artefacts are not completely fixed;
  "Semantic" equivalence vs. "syntactic" equivalence:
  - Same graph, different files (random labels, different serialisation order, embedded metadata);
  - Same dictionary/set; different files / different ordering;
  - Equivalent outputs but different metadata (versions, host, datestamp etc.);
  - Important parts of output fixed; unimportant parts vary;

  } *New kinds of assertion*
  Ignore substrings
  Ignore patterns (regex)
  Pre-assert normalisation functions
  DataFrame Comparators
  DataFrame Comparison Specifiers
- Looping tests with multiple outputs: one failure hides later results; ⟶ *Multi-assertions*
- Slow to run — often want to re-run a single or a few tests; ⟶ *Tag tests; option to run only tagged tests*
- Output generated in memory but want to compare to file;
  hard to understand differences when tests fail if data is in memory; } *Automatic writing of strings in memory to file on failure;* `diff` *command generated*
- Hard to update reference results after code change/bug fix; ⟶ *Option to rewrite actual results to reference results*
- Systematic change affects many tests. ⟶ *Option to rewrite actual results to reference results; tagging allows focused rewrite*

REFERENCE TESTS



Develop a verification procedure (`diff`) and periodically rerun:
do the same inputs (still) produce the same (or equivalent) outputs?

REFERENCE TEST SUPPORT

1: UNSTRUCTURED (STRING) RESULTS

- Comparing actual string (in memory or in file) to reference (*expected*) string (in file)
- Exclude lines with substrings or parts that match regular expressions
- Preprocess output before comparison
- Write actual string produced to file when different
- Show specific `diff` command needed to examine differences
- Check multiple files in single test; report all failures
- Automatically re-write reference results after human verification.

REFERENCE TEST SUPPORT

UNSTRUCTURED (STRING) METHODS

Check a single (in-memory) string against a refererence file

```
self.assertStringCorrect(string, ref_path, . . .)
```

Check a single generated file against a reference file:

```
self.assertTextFileCorrect(actual_path, ref_path, . . .)
```

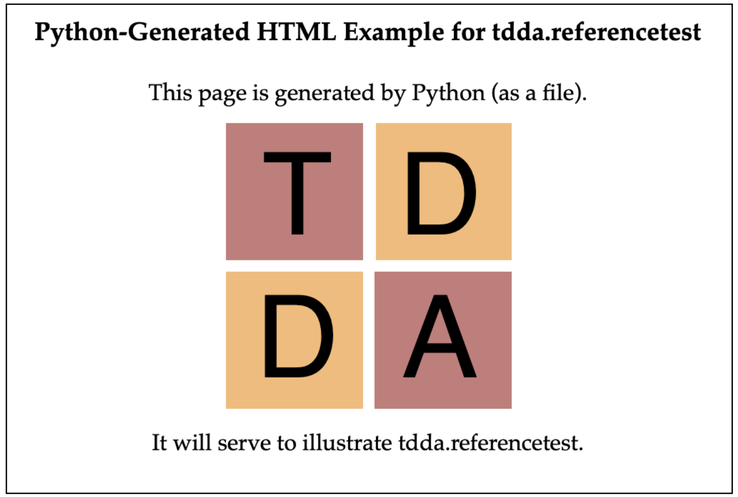
Check a multiple generated files against respective reference files:

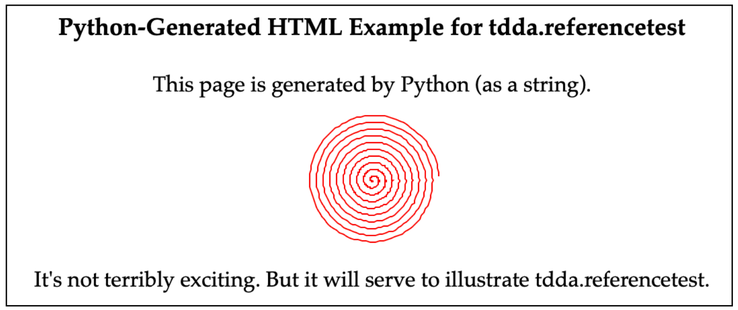
```
self.assertTextFilesCorrect(actual_paths, ref_paths, . . .)
```

EXERCISE 1: STRING DATA REFERENCE TESTS

BACKGROUND

1. `referencetest_examples/generators.py` has two functions, each of which generates HTML.
   - **`generate_string()`** returns the top web HTML page as a *file* ⟶
   - **`generate_spiral()`** returns the bottom HTML web page as a *string* ⟶





- We're going to look at two tests, and what happens if we change the output.

  — for the first page, our test will write it to file

  — for the second page, our test will keep it in memory

EXERCISE 1: STRING DATA REFERENCE TESTS

I. CHECK THE TESTS PASS

1. Copy examples somewhere:

```
cd ~/tmp
tdda examples
cd referencetest_examples
```

2. Look at reference output:

```
open reference/string_result.html
open reference/file_result.html
```

… Use whatever your platform's command for opening an HTML file is

3. Run tests (should **pass**).

```
python unittest/test_using_referencetestcase.py
```

or `(cd pytest; pytest)`

NOTE Although tests pass, output is *not* identical

— version number and copyright lines in reference files are different

```
self.assertTextFileCorrect(outpath, 'file_result.html',
                           ignore_strings=['Copyright', 'Version'])
```

(This will be clearer after next part of exercise.)

EXERCISE 1 (CTD): STRING DATA REFERENCE TESTS

II. MODIFY THE GENERATOR, CHECK RESULTS

4. Modify `generators.py`

   e.g. Change `terribly` to `very` in the `generate_string` function

   e.g. Change `C08080` to `8080C0` in the `generate_file` function

5. Repeat step 3 to run tests again. Two tests should **fail**.

   ```
   python unittest/test_using_referencetestcase.py -F
   ```

   or `(cd pytest; pytest --log-failures)`

6. Use the `diff` command suggested or something similar (`opendiff`, `fc`, …). Note that two commands are suggested, one for a raw comparison and one that prevents the allowed differences from showing up.

TWO DIFF COMMANDS

python unittest/test_using_unittest.py

```
F1 line is different, starting at line 31
Expected file /Users/njr/tmp/referencetest_examples/reference/string_result.html
Compare raw with:
    diff /var/folders/w7/lhtph66x7h33t9pns0616qk00000gn/T/actual-raw-string_resu
lt.html /Users/njr/tmp/referencetest_examples/reference/string_result.html

Compare post-processed with:
    diff /var/folders/w7/lhtph66x7h33t9pns0616qk00000gn/T/actual-string_result.h
tml /var/folders/w7/lhtph66x7h33t9pns0616qk00000gn/T/expected-string_result.html

Note exclusions:
    ignore_substrings:
        Copyright
        Version
F
```

TAGGING TESTS TO RUN A SUBSET WITH @tag

You can "tag" single individual tests or whole test classes to allow only those ones to be run with when running with `--tagged` (`unittest` also supports `-1`)

```
from tdda.referencetest import ReferenceTestCase, tag
class TestDemo(ReferenceTestCase):
    def testOne(self):
        self.assertEqual(1, 1)

    @tag
    def testTwo(self):
        self.assertEqual(2, 2)

    @tag
    def testThree(self):
        self.assertEqual(3, 3)

    def testFour(self):
        self.assertEqual(4, 4)

if __name__ == '__main__':
     ReferenceTestCase.main()
```

```
$ python3 tests.py -1
..
----------------------------
Ran 2 tests in 0.000s

OK
```

See what classes have tagged tests with `--istagged` (`unittest`: or `-0`)

```
$ python3 tests.py -0
__main__.TestDemo

----------------------------
OK
```

This is especially recommended if when you want to rewrite test results;
it's better only to re-write the results for a specific test, rather than for all tests.

AUTOTAGGING TESTS TO RUN A SUBSET

	unittest	**pytest**
Run tests logging failures	`python tests.py -F`	`pytest --log-failures`
Clear any existing tags on tests	`python tests.py -9`	`pytest --untag`
Tag last logged failing tests	`tdda tag`	`tdda tag`
Run tagged tests only	`python tests.py -1`	`pytest --tagged`
Rewrite tagged test references *if code correct*	`python tests.py -1W`	`pytest --tagged --write-all`
Retest	`python tests.py`	`pytest`
Clear tags on all tests	`python tests.py -9`	`pytest --untag`

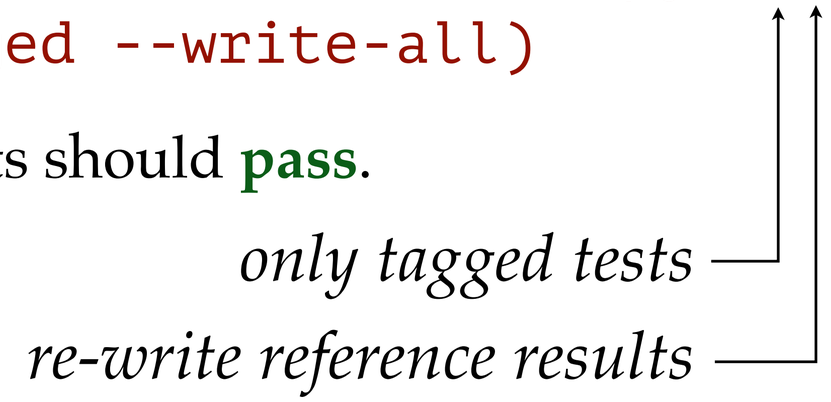
EXERCISE 1 (CTD): STRING DATA REFERENCE TESTS

III. RE-WRITE REFERENCE RESULTS

7. On the assumption that these now represent the verified,* new target results, re-write the reference output with:

```
tdda tag

python unittest/test_using_referencetestcase.py -1W
or (cd pytest; pytest --tagged --write-all)
```

8. Repeat step 5 to run tests again. All tests should **pass**.



*** WARNING**

If you habitually re-write results when tests fail without carefully verifying the new results, your tests will quickly become worthless.

With great power comes great responsibility: use TDDA referencetest's (re-)write flags wisely!

EXERCISE 1 (CTD): STRING DATA REFERENCE TESTS

IV. MODIFY THE RESULTS VERSION NUMBER; CHECK STILL OK

9. Modify `generators.py` code to change version number in output.
10. Repeat step 3 to run tests again. All tests should still **pass** since version number is excluded by

```
ignore_substrings=['Copyright', 'Version']
```

parameter to `assertStringCorrect`.

REFERENCE TEST SUPPORT

2: STRUCTURED DATA METHODS (DATAFRAMES & CSV)

- Comparing generated DataFrame or parquet or CSV file to a reference DataFrame or parquet file or CSV file. (Can include : for metadata.)
- Show specific `diff` command needed to examine differences
- Check multiple CSV/parquet files in single test; report all failures
- Choose subset of columns (with list or function) to compare
- Choose whether to check (detailed) types
- Choose whether to check column order
- Choose precision for floating-point comparisons
- Automatic re-writing of verified (changed) results.

REFERENCE TEST SUPPORT

STRUCTURED DATA METHODS (DATAFRAMES & CSV)

Check a single generated CSV/parquet file against a refererence CSV/parquet file

```
self.assertStoredDataFrameCorrect(actual_path, ref_path, . . .)
```

Check multiple generated files against respective reference CSV/parquet files:

```
self.assertStoredDataFramesCorrect(actual_paths, ref_paths, . . .)
```

Check an (in-memory) DataFrame against a reference CSV/parquet file

```
self.assertDataFrameCorrect(df, ref_path, . . .)
```

Check an (in-memory) DataFrame against another (in-memory) DataFrame

```
self.assertDataFramesEquivalent(df, ref_df . . .)
```

EXERCISE 2: DATAFRAME/CSV REFERENCE TESTS

I. CHECK THAT THE TESTS PASS

1. If you've done Exercise 1, you already have the reference examples in a (sibling) `referencetest_examples` directory

   ```
   cd ../referencetest_examples
   ```

2. Look at reference output:

   ```
   tdda cat reference/dataframe_result.csv
   tdda cat reference/dataframe_result2.csv
   ```

3. Run tests (should **pass**).

   ```
   python unittest/test_using_referencetestcase.py
   (cd pytest; pytest)
   ```

NOTE You can look at the data frame being generated with the 2-line program (`show.py`)

```
from dataframes import generate_dataframe
print(generate_dataframe())
```

tdda cat reference/dataframe_result.csv

```
$
$ tdda cat reference/dataframe_result.csv
dataframe_result.csv: 10 records; 7 fields. [Pandas]
Read by tdda.serial.csv_to_pandas with no external metadata
```

i	r	random	b	i_square	r_square	s
0	0.0	3	False	0	0.0	result 0.0
1	1.11111111111	6	False	1	1.23456790123	result 1.235
2	2.22222222222	4	False	4	4.93827160494	result 4.938
3	3.33333333333	2	False	9	11.1111111111	result 11.111
4	4.44444444444	1	False	16	19.7530864198	result 19.753
5	5.55555555556	0	True	25	30.8641975309	result 30.864
6	6.66666666667	8	True	36	44.4444444444	result 44.444
7	7.77777777778	6	True	49	60.4938271605	result 60.494
8	8.88888888889	0	True	64	79.012345679	result 79.012
9	10.0	1	True	81	100.0	result 100.0

```
$
```

tdda cat reference/dataframe_result2.csv

```
$
$ tdda cat reference/dataframe_result2.parquet
dataframe_result2.parquet: 20 records; 7 fields. [Pandas]
```

i	r	random	b	i_square	r_square	s
0	0.0	14	False	0	0.0	result 0.0
1	1.1111111111	3	False	1	1.23456790123	result 1.235
2	2.2222222222	13	False	4	4.93827160494	result 4.938
3	3.3333333333	10	False	9	11.1111111111	result 11.111
4	4.4444444444	12	False	16	19.7530864198	result 19.753
5	5.5555555556	1	False	25	30.8641975309	result 30.864
6	6.6666666667	1	False	36	44.4444444444	result 44.444
7	7.7777777778	5	False	49	60.4938271605	result 60.494
8	8.8888888889	11	False	64	79.012345679	result 79.012
9	10.0	10	False	81	100.0	result 100.0
10	11.1111111111	13	True	100	123.456790123	result 123.457
11	12.2222222222	1	True	121	149.382716049	result 149.383
12	13.3333333333	7	True	144	177.777777778	result 177.778
13	14.4444444444	15	True	169	208.641975309	result 208.642
14	15.5555555556	15	True	196	241.975308642	result 241.975
15	16.6666666667	10	True	225	277.777777778	result 277.778
16	17.7777777778	8	True	256	316.049382716	result 316.049
17	18.8888888889	6	True	289	356.790123457	result 356.79
18	20.0	7	True	324	400.0	result 400.0
19	21.1111111111	11	True	361	445.679012346	result 445.679

EXERCISE 2: DATAFRAME/CSV REFERENCE TESTS

II. MODIFY THE DATA GENERATOR, VERIFY RESULTS

4. Modify `dataframes.py`, e.g. Change the default precision from `3` to `2` in the `generate_dataframe` function. This will cause the string column `s` to be different.

5. Repeat step 3 to run tests again. Five tests should **fail**.

   ```
   python unittest/test_using_referencetestcase.py -F
   ```
   or `(cd pytest; pytest --log-failures)`

6. Look at the way differences are reported, and check that the only material change is to column `s`, as expected.

7. Clear any tags and tag the failures:

   ```
   python unittest/test_using_referencetestcase.py -9
   ```
   or `(cd pytest; pytest --untag)`

   Then `tdda tag`

EXERCISE 2: DATAFRAME/CSV REFERENCE TESTS

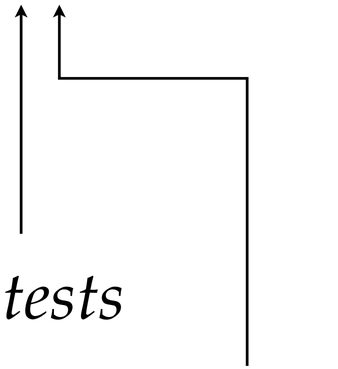
II. RE-WRITE REFERENCE RESULTS; RE-RUN

8. On the assumption that this new output now represents the new, verified target result,* re-write the reference output with

```
python unittest/test_using_referencetestcase.py -1W
```

or `(cd pytest; pytest --write-all --tagged)`

9. Repeat step 5 to run tests again. All tests should now **pass**.



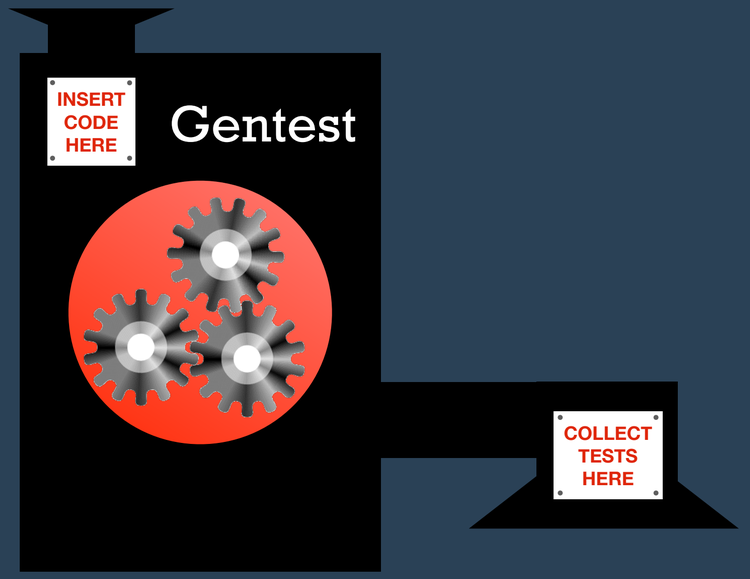
only tagged tests

re-write reference results

*** WARNING**

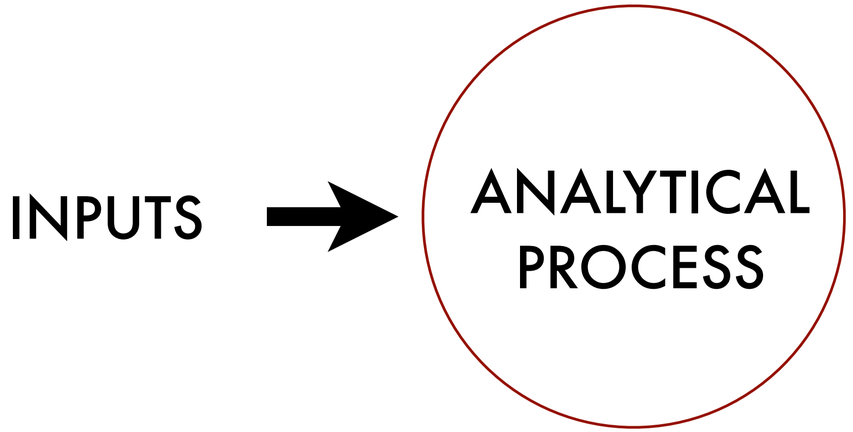
If you habitually re-write results when tests fail without carefully verifying the new results, your tests will quickly become worthless.

With great power comes great responsibility: use TDDA Reference Tests wisely!



tdda gentest

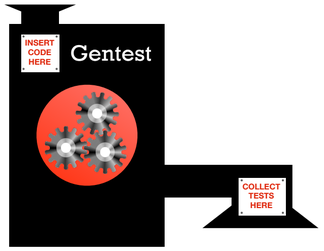
AUTOMATIC TEST GENERATION

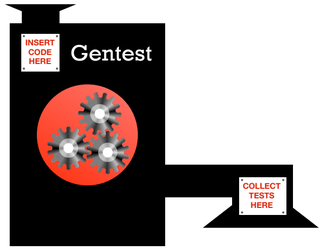


Record inputs

Capture as script

Record ("reference") outputs



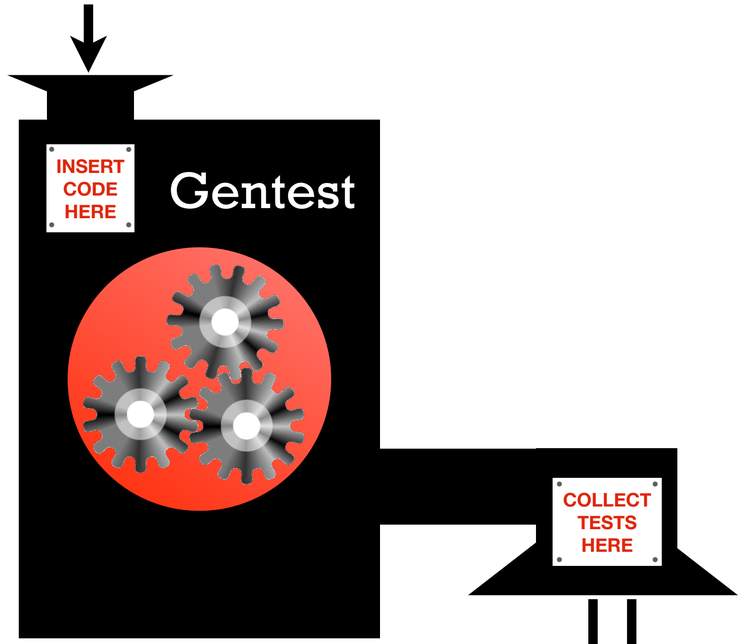


Develop a verification procedure (`diff`) and periodically rerun:
do the same inputs (still) produce the same or equivalent outputs?

GENTEST

`tdda gentest "sh classify.sh"`

`sh classify.sh`



`test_sh_classify_sh.py` *test script*

`ref/sh_classify_sh` *reference outputs*

GENTEST

stdout

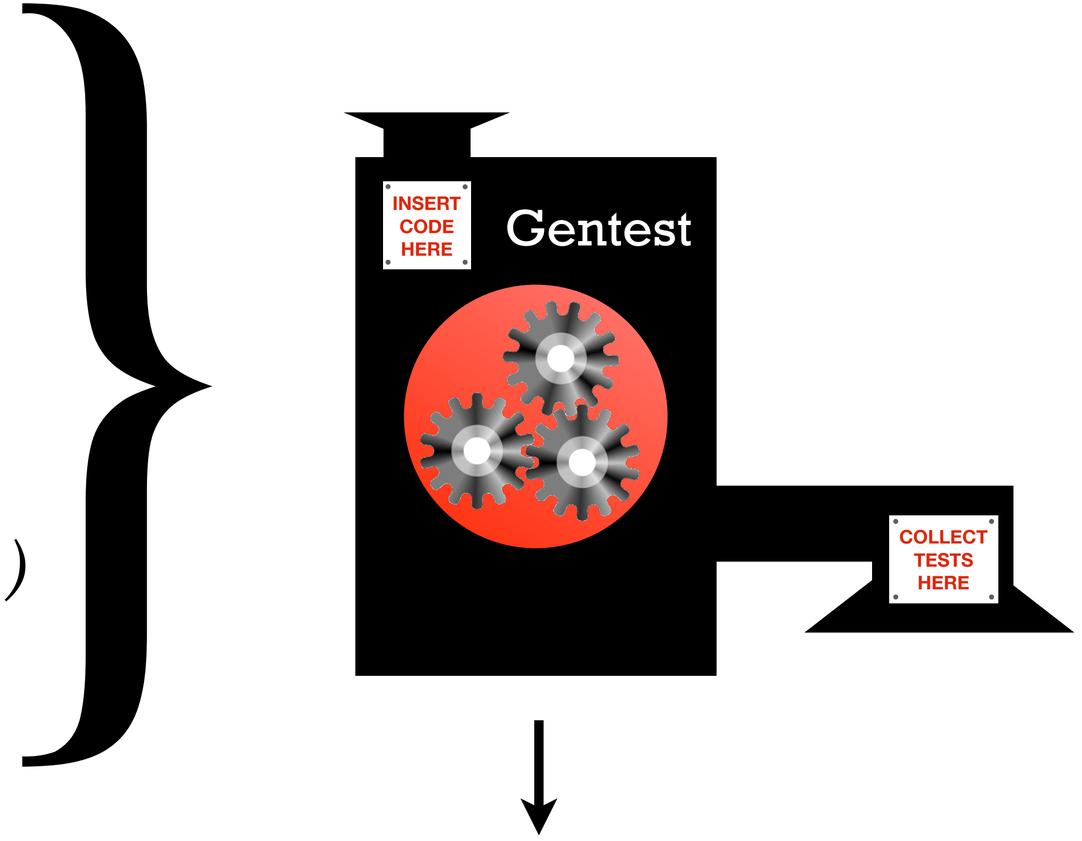
stderr

exit code

file system changes

environment (path, date, …)

differences between runs



“intelligent” decisions about how and what to test

Gentest is largely enabled by Rexpy!

GENTEST

stdout

stderr

exit code

file system changes

environment (path, data, ...)

differences between runs

"intelligent" decisions about how and what to test

Gentest is largely enabled by Rexpy!

GENERATIVE ARTIFICIAL INTELLIGENCE

2018

example2.sh

```
echo "Hey, cats!"
echo
echo "This is gentest, running on `hostname`"
echo
echo "I have to say, the weather was better in Münich!"
echo
echo "Today, `date` it's proper dreich here."
echo
echo "Let's have a file as well." > FILE1
echo "Have a number: $RANDOM" >> FILE1
echo "Random number written to $PWD/FILE1"
```

sh example2.sh

njr@book: ~

```
$
$ sh example2.sh
Hey, cats!

This is gentest, running on escher.local

I have to say, the weather was better in Münich!

Today, Thu  4 Jun 2026 10:53:49 BST it's proper dreich here.

Random number written to /Users/njr/tmp/gentest_examples/FILE1
$
```

Gentest Wizard

```
cd ~/tmp
tdda examples
cd gentest_examples
```

```
$ tdda gentest
Enter shell command to be tested: sh example2.sh
Enter name for test script [test_sh_example2_sh]:
Check all files written under $(pwd)?: [y]:
Check all files written under (gentest's) $TMPDIR?: [y]:
Enter other files/directories to be checked, one per line, then a blank line:

Check stdout?: [y]:
Check stderr?: [y]:
Exit code should be zero?: [y]:
Clobber (overwrite) previous outputs (if they exist)?: [y]:
Number of times to run script?: [2]:
```

WIZARD OUTPUT

```
Running command 'sh example2.sh' to generate output (run 1 of 2).
Saved (non-empty) output to stdout to /Users/njr/tmp/gentest_examples/ref/sh_example2_sh/STDOUT.
Saved (empty) output to stderr to /Users/njr/tmp/gentest_examples/ref/sh_example2_sh/STDERR.
Copied $(pwd)/FILE1 to $(pwd)/ref/sh_example2_sh/FILE1

Running command 'sh example2.sh' to generate output (run 2 of 2).
Saved (non-empty) output to stdout to /Users/njr/tmp/gentest_examples/ref/sh_example2_sh/2/STDOUT.
Saved (empty) output to stderr to /Users/njr/tmp/gentest_examples/ref/sh_example2_sh/2/STDERR.
Copied $(pwd)/FILE1 to $(pwd)/ref/sh_example2_sh/2/FILE1

Test script written as /Users/njr/tmp/gentest_examples/test_sh_example2_sh.py
Command execution took: 0.031s

SUMMARY:

Directory to run in:         /Users/njr/tmp/gentest_examples
Shell command:               sh example2.sh
Test script generated:       /Users/njr/tmp/gentest_examples/test_sh_example2_sh.py
Reference files:
    $(pwd)/FILE1
Check stdout:                yes (was 9 lines)
Check stderr:                yes (was empty)
Expected exit code:          0
Clobbering permitted:        yes
Number of times script ran: 2
Number of tests written:    5
```

GENERATED CODE

```
"""
test_sh_example2_sh.py: Automatically generated test code from tdda
gentest.

Generation command:

tdda gentest 'sh example2.sh' 'test_sh_example2_sh.py' '.'
"""

import os
import sys
import tempfile

from tdda.referencetest import ReferenceTestCase
from tdda.referencetest.gentest import exec_command

class Test_SH_EXAMPLE2(ReferenceTestCase):
    command = 'sh example2.sh'
    cwd = os.path.abspath(os.path.dirname(__file__))
    refdir = os.path.join(cwd, 'ref', 'sh_example2_sh')

    generated_files = [
        os.path.join(cwd, 'FILE1')
    ]

    @classmethod
    def setUpClass(cls):
        for path in cls.generated_files:
            if os.path.exists(path):
                os.unlink(path)
        (cls.output,
         cls.error,
         cls.exception,
         cls.exit_code,
         cls.duration) = exec_command(cls.command, cls.cwd)

    def test_no_exception(self):
        self.assertIsNone(self.exception)

    def test_exit_code(self):
        self.assertEqual(self.exit_code, 0)

    def test_stdout(self):
        substrings = [
            '/Users/njr/tmp/gentest_examples',
            '3 Jun 2026 18:40:48',
            'gardot.local',
        ]
        self.assertStringCorrect(self.output,
                                 os.path.join(self.refdir, 'STDOUT'),
                                 ignore_substrings=substrings)

    def test_stderr(self):
        self.assertStringCorrect(self.error,
                                 os.path.join(self.refdir, 'STDERR'))

    def test_FILE1(self):
        patterns = [
            r'^Have a number: [0-9]{4,5}$',
        ]
        self.assertTextFileCorrect(os.path.join(self.cwd, 'FILE1'),
                                   os.path.join(self.refdir, 'FILE1'),
                                   ignore_patterns=patterns,
                                   encoding='ascii')

if __name__ == '__main__':
    ReferenceTestCase.main()
```

Note exclusions for local context and run-to-run variability

SAVED FILES

```
$ ls ref/sh_example2_sh/
FILE1   STDERR   STDOUT

$ more ref/sh_example2_sh/FILE1
Let's have a file as well.
Have a number: 7856

$ more ref/sh_example2_sh/STDOUT
Hey, cats!

This is gentest, running on godel.local

I have to say, the weather was better in Münich!

Today, Tue  9 Apr 2019 17:45:49 BST it's proper dreich here.

Random number written to /Users/njr/tmp/gentest_examples/FILE1

$ more ref/sh_example2_sh/STDERR
$
```

(This file is empty)

RUNNING THE TESTS

```
$ python test_sh_example2_sh.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.018s

OK
```

RUNNING REPEATEDLY

If you run enough times, you will get a failure, because the exclusion is assuming the random number generated will always be four or five digits.

On the night, it didn't fail.

But after, I ran it another 33 times, and the last time it failed.

WHEN IT DOES FAIL

```
$ python test_sh_example2_sh.py
1 line is different, starting at line 2
Compare with:
    diff /Users/njr/tmp/gentest_examples/FILE1 /Users/njr/tmp/gentest_examples/ref/sh_example2_sh/FILE1

Note exclusions:
    ignore_patterns:
        ^Have a number\: \d{4,5}$
F....
======================================================================
FAIL: test_FILE1 (__main__.Test_SH_EXAMPLE2)
----------------------------------------------------------------------
Traceback (most recent call last):
  File “test_sh_example2_sh.py", line 61, in test_FILE1
    ignore_patterns=patterns)
  File "/Users/njr/python/tdda/tdda/referencetest/referencetest.py", line 857, in assertTextFileCorrect
    self._check_failures(failures, msgs)
  File "/Users/njr/python/tdda/tdda/referencetest/referencetest.py", line 1046, in _check_failures
    self.assert_fn(failures == 0, msgs.message())
AssertionError: 1 line is different, starting at line 2
Compare with:
    diff /Users/njr/tmp/gentest_examples/FILE1 /Users/njr/tmp/gentest_examples/ref/sh_example2_sh/FILE1

Note exclusions:
    ignore_patterns:
        ^Have a number\: \d{4,5}$

----------------------------------------------------------------------
Ran 5 tests in 0.018s

FAILED (failures=1)
```

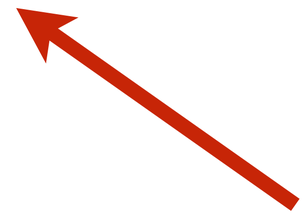
tdda diff

tdda diff

The tdda library also has a tool for doing visual diffs between dataframes stored in serial files (i.e. `.parquet`, `.csv`, `.psv` etc.)

```
cd ~/tmp
tdda examples
cd serial_examples
```

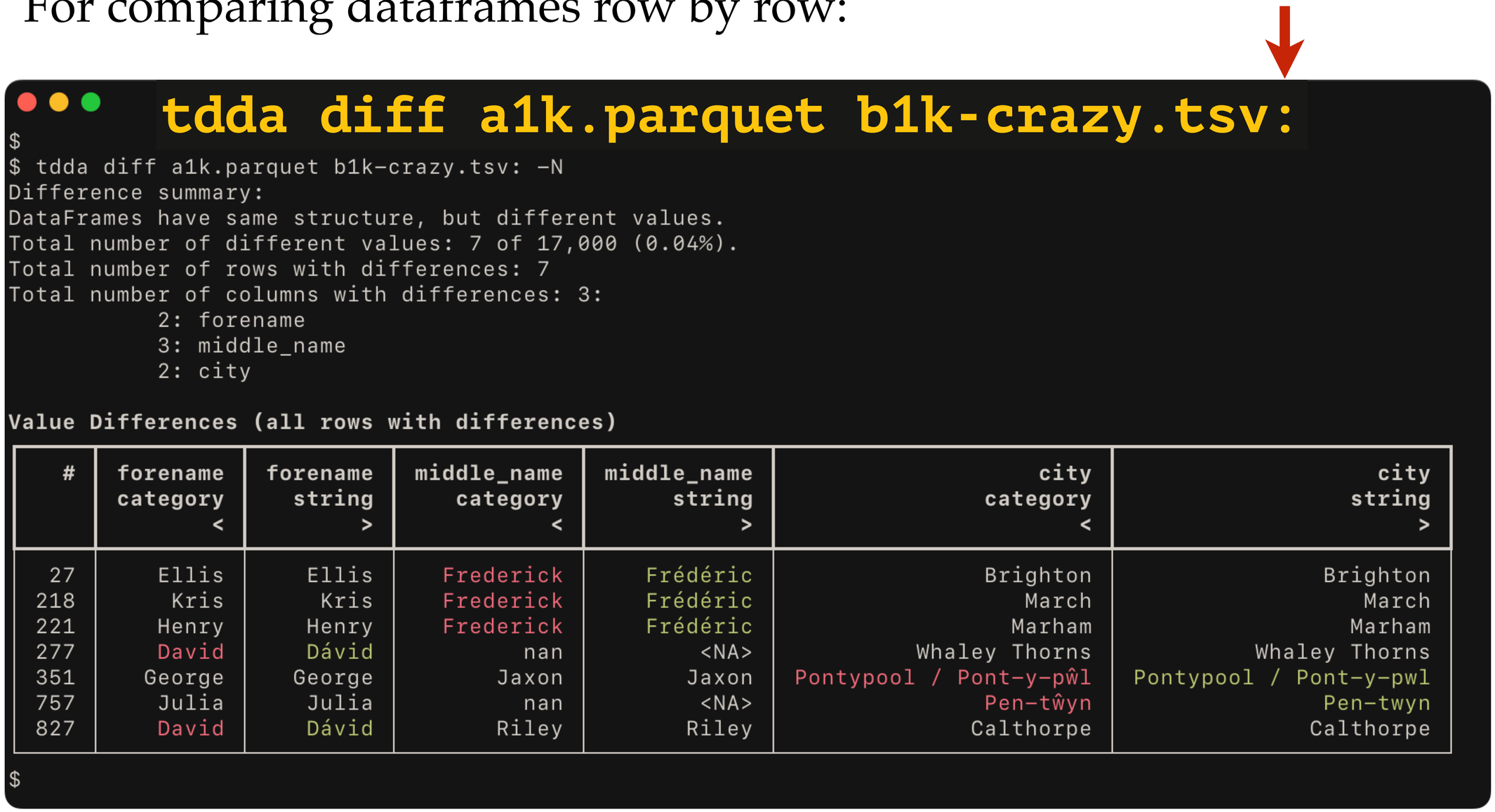
```
$ tdda diff a1k.csv: a1k-usd.csv:
$
```



No output: no differences!

tdda diff

For comparing dataframes row by row:



tdda.serial: trailing colon

All the `a999.csv` and `a1001.csv` files contain 999 and 1001 rows respectively

They have `.serial` files too, but they are identical to `a1k.csv.serial` (and could be shared)

They should ***pass*** if verified against `accounts.tdda`.

```
tdda verify a999.csv: accounts.tdda
```

tdda.serial: trailing colon

All the `b*` files are semantically different from the `a1k` files.

They can be used but will produce validation failures

They should *fail* if verified against `accounts.tdda`.

```
tdda verify b1k-crazy.tsv: accounts.tdda
```

tdda diff

For comparing dataframes using database-style join:

```
$
$
$
$ tdda diff a1k.parquet a1001.csv: --key account_number --vertical
Data frames have different column structure.
Data frames have different numbers of rows.
Actual records: 1,000; Expected records: 1,001
Difference summary:
DataFrames have same structure, but different values.
Total number of different values: 43 of 18,036 (0.24%).
Total number of rows with differences: 3
Total number of columns with differences: 15:
          3: open_date
          3: title
          3: forename
          1: middle_name
          3: surname
          3: address1
          3: city
          3: postcode
          3: home_tel
          3: mobile_tel
          3: email
          3: account_type
          3: overdraft_limit
          3: cash_card_number
          3: is_open

Value Differences (all rows with differences)
```

account_number	open_date datetime64 datetime64	title string string	forename string string	middle_name string string	surname string string	address1 string string	city string string	postcode string string	home_tel string string	mobile_tel string string	email string string	account_type string string	overdraft_limit Int64 Int64	cash_card_number string string	is_open boolean boolean
10073827 <	2008-09-23 00:00:00	Miss	Jasmine	Amelia	Davis	26 Mayfield Road	Stevenston	KA20 4MP	0549 088 3290	0701 011 0714	jasmine.davis4159@hotmail.com	premium	0	1202 8629 4416 6191	True
10073827 >															
12999375 <															
12999375 >	2016-03-26 00:00:00	Mrs	Elsie	<NA>	Marshall	16 Redwood Croft	Birmingham	B14 9ZU	0787 402 6472	0798 752 9615	jfvhly9h6yashfu@gmail.com	current	0	1202 8651 1713 9363	True
12999930 <															
12999930 >	2013-11-29 00:00:00	Mr	Kris	<NA>	Jenkins	30 Knight's Close	Macclesfield	SK11 4GN	0859 781 4609	0709 108 8265	kris.jenkins421@googlemail.co.uk	current	1700	1202 8643 7878 7800	True

```
$
```

A bit small; hard to see

tdda diff: selective fields

Horizontal `diff` (default)

```
$ tdda diff a1k.parquet a1001.csv:
$       --key account_number
$       --fields account_number forename surname
$
$ tdda diff a1k.parquet a1001.csv: --key account_number --fields
account_number forename surname
Data frames have different column structure.
Data frames have different numbers of rows.
Actual records: 1,000; Expected records: 1,001
Difference summary:
DataFrames have same structure, but different values.
Total number of different values: 6 of 4,008 (0.15%).
Total number of rows with differences: 3
Total number of columns with differences: 2:
          3: forename
          3: surname

Value Differences (all rows with differences)
```

account_number	forename string <	forename string >	surname string <	surname string >
10073827	Jasmine		Davis	
12999375		Elsie		Marshall
12999930		Kris		Jenkins

```
$
```

Vertical `diff` (`-V` or `--vertical`)

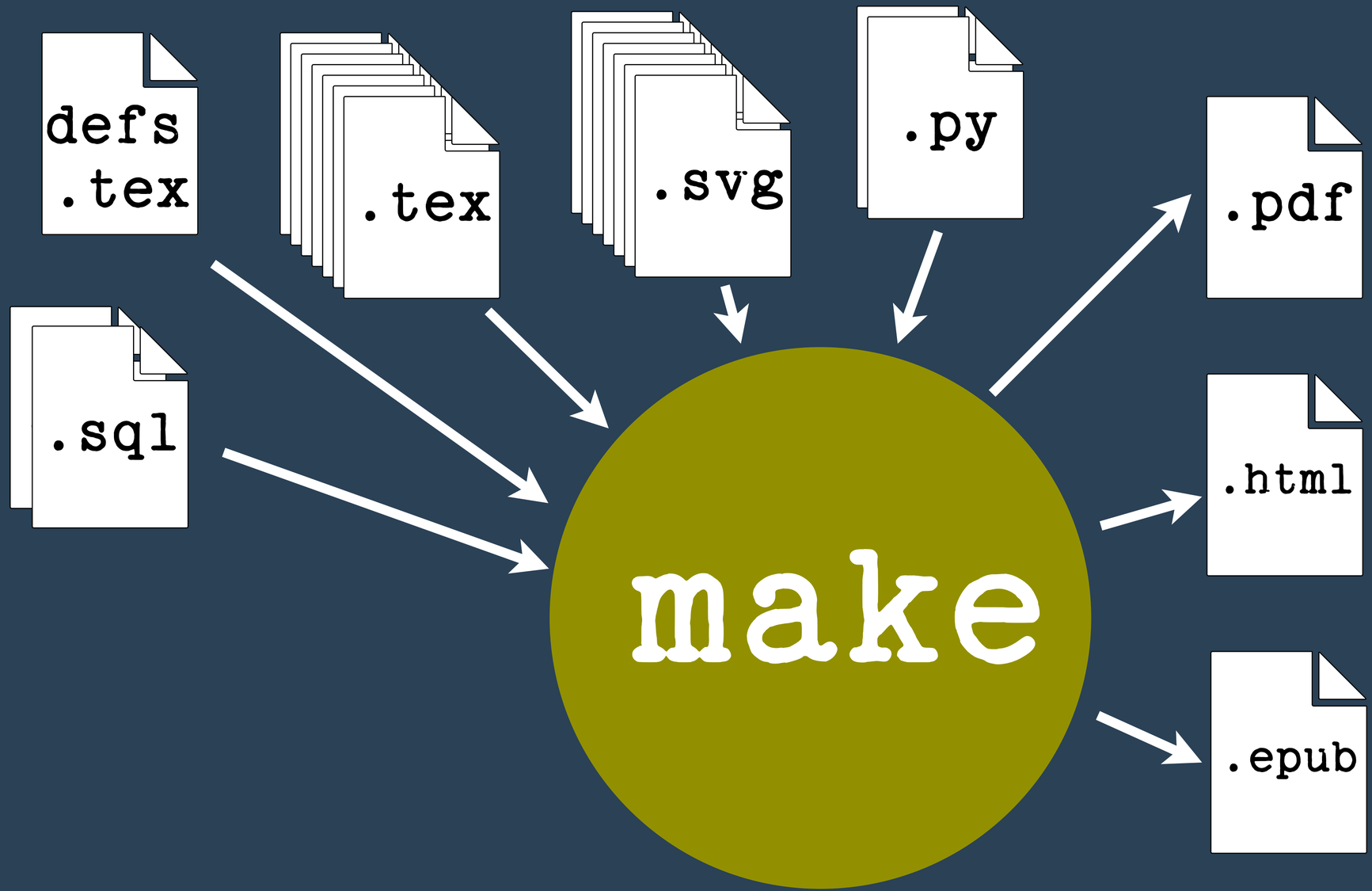
```
$ tdda diff a1k.parquet a1001.csv:
$       --key account_number
$       --fields account_number forename surname -V
$
$ tdda diff a1k.parquet a1001.csv: --key account_number --fields account_
number forename surname -V
Data frames have different column structure.
Data frames have different numbers of rows.
Actual records: 1,000; Expected records: 1,001
Difference summary:
DataFrames have same structure, but different values.
Total number of different values: 6 of 4,008 (0.15%).
Total number of rows with differences: 3
Total number of columns with differences: 2:
          3: forename
          3: surname

Value Differences (all rows with differences)
```

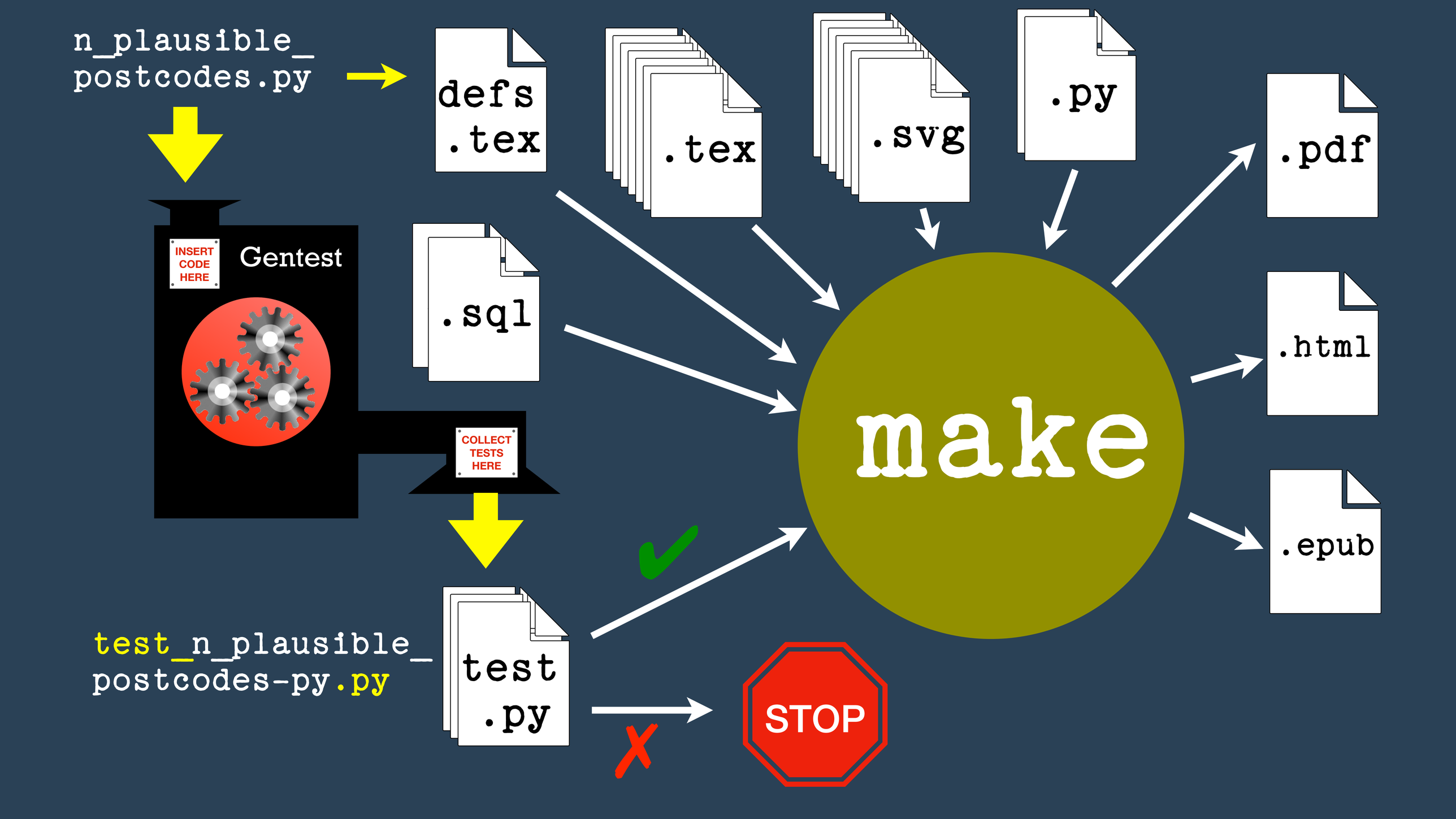
account_number	forename string string	surname string string
10073827 <	Jasmine	Davis
10073827 >		
12999375 <		
12999375 >	Elsie	Marshall
12999930 <		
12999930 >	Kris	Jenkins

```
$
```

TEST-DRIVEN DOCUMENT DEVELOPMENT (TDDD)

TYPESETTING THE BOOK AS AN ANALYTICAL PROCESS




n_plausible_
postcodes.py
defs
.tex
.tex
.svg
.py
.pdf
INSERT
CODE
HERE
Gentest
.sql
.html
COLLECT
TESTS
HERE
make
.epub
test_n_plausible_
postcodes-py.py
test
.py
STOP

TDDD

Results are generated by running tests

If tests fail (results changed; code rusted)
`BOOK WILL NOT BUILD`

Results are guaranteed to have passed tests*

Computational Documents

Results are generated by running code in doc

Qu…,
RM…wn,
Ju…ter,
…

If code rusts (but still runs)

RESULTS CHANGE

CO-RUSTING

If code rusts (but still runs) results & document change:

"co-rusting"

Extract from book

UK POSTCODES

```
^[A-Z]{1,2}[0-9]{1,2}[A-Z]? [0-9][A-Z]{2}$
```

This would be a good regular expression to substitute in.

We should be aware, however, that although this regular expression matches all valid postcodes, not everything that it matches is a valid postcode. For example, the letters at the start define the *postal area,* and there are only 124 postal areas in existence, while there are $26 + 26 \times 26 = 702$ possible 1- and 2-letter combinations. In fact, at the time of writing, there are only around 1 million postcodes, but 14,094,194,400 strings that match the regular expression above.

Extract from book

UK POSTCODES

```
^[A-Z]{1,2}[0-9]{1,2}[A-Z]? [0-9][A-Z]{2}$
```

This would be a good regular expression to substitute in.

We should be aware, however, that although this regular expression matches all valid postcodes, not everything that it matches is a valid postcode. For example, the letters at the start define the *postal area,* and there are only 124 postal areas in existence, while there are $26 + 26 \times 26 = 702$ possible 1- and 2-letter combinations. In fact, at the time of writing, there are only around 1 million postcodes, but 14,094,194,400 strings that match the regular expression above.

(Source for typesetter)

TeX

Book Source file: constraints.tex

```
\begin{Verbatim}
   ^[A-Z]{1,2}[0-9]{1,2}[A-Z]? [0-9][A-Z]{2}$
\end{Verbatim}
This would be a good regular expression to substitute in.

We should be aware, however, that although this regular
expression matches all valid postcodes, not everything that
it matches is a valid postcode. For example, the letters at
the start define the {\it postal
area,\/} and there are only 124 postal areas in existence,
while there are $26 + 26\times26 = 702$ possible 1- and
2-letter combinations. In fact, at the time of writing,
there are only around 1 million postcodes, but
\nPlausiblePostcodes{} strings that match the regular
expression above.
% see n_plausible_postcodes.py.
```

Includes: n-plausible-postcodes-defs.tex

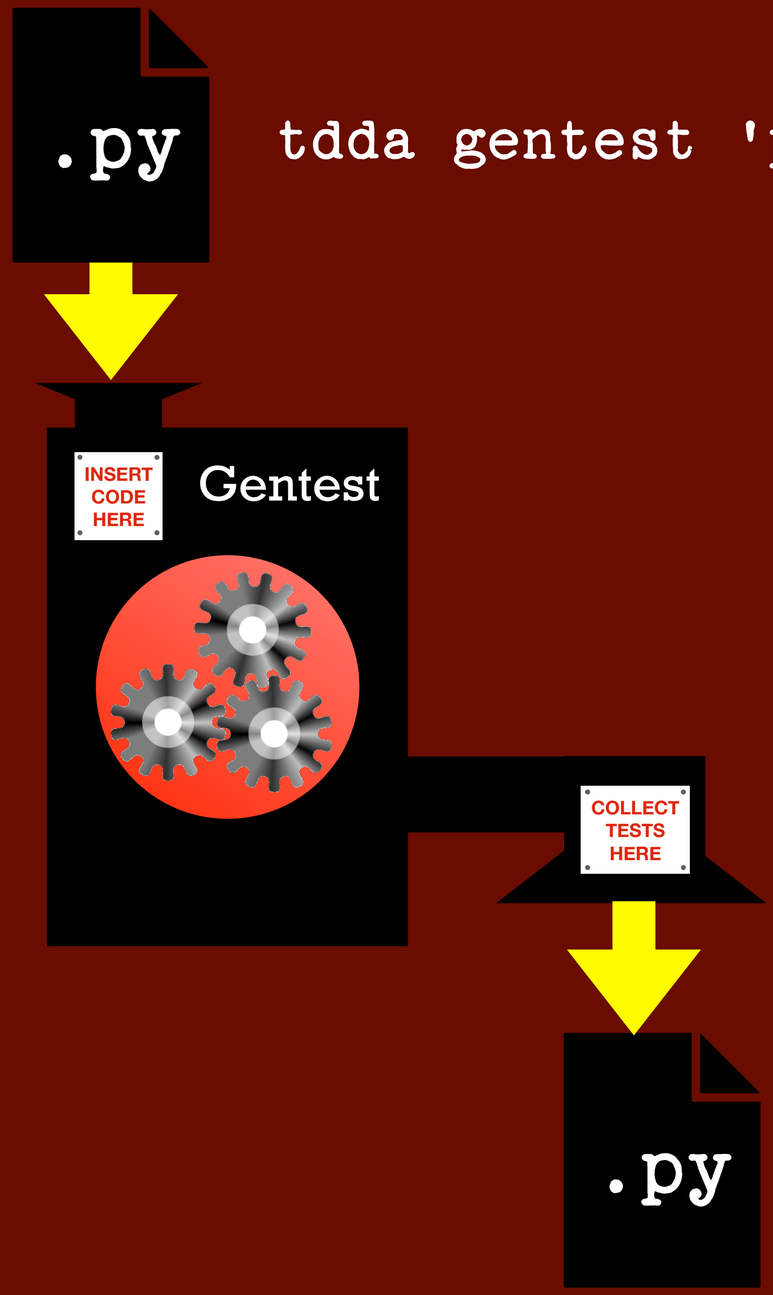
```
\def\nPlausiblePostcodes{14,094,194,400}
```

Python code: n_plausible_postcodes.py

Writes: n-plausuble-postcodes-defs.tex

```
import json
from tdda.utils import dict_to_tex_macros
RE = r'^[A-Z]{1,2}[0-9]{1,2}[A-Z]? [0-9][A-Z]{2}$'

def n_poss_postcodes_for_re():
    n_postal_areas = 26 + 26 * 26  # One or two letters
    n_postal_districts = 10 + 100  # Any one or two digit number
                                   # 0 and 0x aren't used, but match the regex
    n_subdistricts = 26 + 1        # Not all letters are used,
                                   # and only for some London codes,
                                   # but for our regex...
                                   # The + 1 is for ones not using a subdistrict

    n_outcodes = n_postal_areas * n_postal_districts * n_subdistricts
    n_incodes = 10 * 26 * 26       # Digit then two letters
    n_postcodes = n_outcodes * n_incodes

    return n_postcodes

if __name__ == '__main__':
    n = n_poss_postcodes_for_re()
    print(f'Number of postcode-like strings matching\n{re}:\n{n:,}\n')
    d = {'n_plausible_postcodes': f'{n:,}'}
    with open('n-plausible-postcodes-results.json', 'w') as f:
        json.dump(d, f)
    dict_to_tex_macros(d, 'n-plausible-postcodes-defs.tex', verbose=True)
```

n_plausible_
postcodes.py



tdda gentest 'python n_plausible_postcodes.py'

Gentest writes tests,
so you don't have to™

test_n_plausible_
postcodes_py.py

Python TDDA Gentest Reference test: test_python_n_plausible_postcodes_py.py

Runs: python_n_plausible_postcodes.py

```
"""
test_python_n_plausible_postcodes_py.py:
Automatically generated test code from tdda gentest.

Generation command:

tdda gentest 'python n_plausible_postcodes.py' 'test_python_n_plausible_postcodes_py.py' '.'
"""
import os, sys, tempfile
from tdda.referencetest import ReferenceTestCase
from tdda.referencetest.gentest import exec_command

class Test_PYTHON_N_PLAUSIBLE_POSTCODES(ReferenceTestCase):
    command = 'python n_plausible_postcodes.py'
    cwd = os.path.abspath(os.path.dirname(__file__))
    refdir = os.path.join(cwd, 'ref', 'python_n_plausible_postcodes_py')

    generated_files = [os.path.join(cwd, 'n-plausible-postcodes-defs.tex'),
                       os.path.join(cwd, 'n-plausible-postcodes-results.json')]
    @classmethod
    def setUpClass(cls):
        for path in cls.generated_files:
            if os.path.exists(path):
                os.unlink(path)
        (cls.output, cls.error,
         cls.exception, cls.exit_code,
         cls.duration) = exec_command(cls.command, cls.cwd)
```

Python TDDA Reference test: test_python_n_plausible_postcodes_py.py ctd

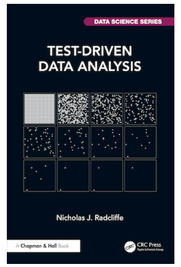
Checks output from python_n_plausible_postcodes.py

```
    def test_no_exception(self):
        self.assertIsNone(self.exception)

    def test_exit_code(self):
        self.assertEqual(self.exit_code, 0)

    def test_stdout(self):
        self.assertStringCorrect(self.output,
                                 os.path.join(self.refdir, 'STDOUT'))

    def test_stderr(self):
        self.assertStringCorrect(self.error,
                                 os.path.join(self.refdir, 'STDERR'))

    def test_n_plausible_postcodes_defs_tex(self):
        self.assertTextFileCorrect(os.path.join(self.cwd,    'n-plausible-postcodes-defs.tex'),
                                   os.path.join(self.refdir, 'n-plausible-postcodes-defs.tex'))

    def test_n_plausible_postcodes_results_json(self):
        self.assertTextFileCorrect(os.path.join(self.cwd, 'n-plausible-postcodes-results.json'),
                                   os.path.join(self.refdir, 'n-plausible-postcodes-results.json'))


if __name__ == '__main__':
    ReferenceTestCase.main()
```

Test fails if the generated TeX definition for `\nPlausiblePostcodes` *is different from the reference result*

https://stochasticsolutions.com/pdf/tdda-london-2026.pdf

https://stochasticsolutions.com



https://book.tdda.info *Book resources and free serialization*

https://www.routledge.com **26SMA1** *for* ***20% off*** *till* **2026-06-30**

https://tdda.info *Blog*

https://github.com/tdda *Source*

njr@stochasticsolutions.com

https://linkedin.com/in/njradcliffe

@njr & @tdda@mathstodon.xyz

Stochastic Solutions