https://stochasticsolutions.com/pdf/tdda-london-2024.pdf

TEST-DRIVEN DATA ANALYSIS

PyData London

2024-06-14T15:30:00+01:00

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

TDDA

Stochastic Solutions

OUTLINE

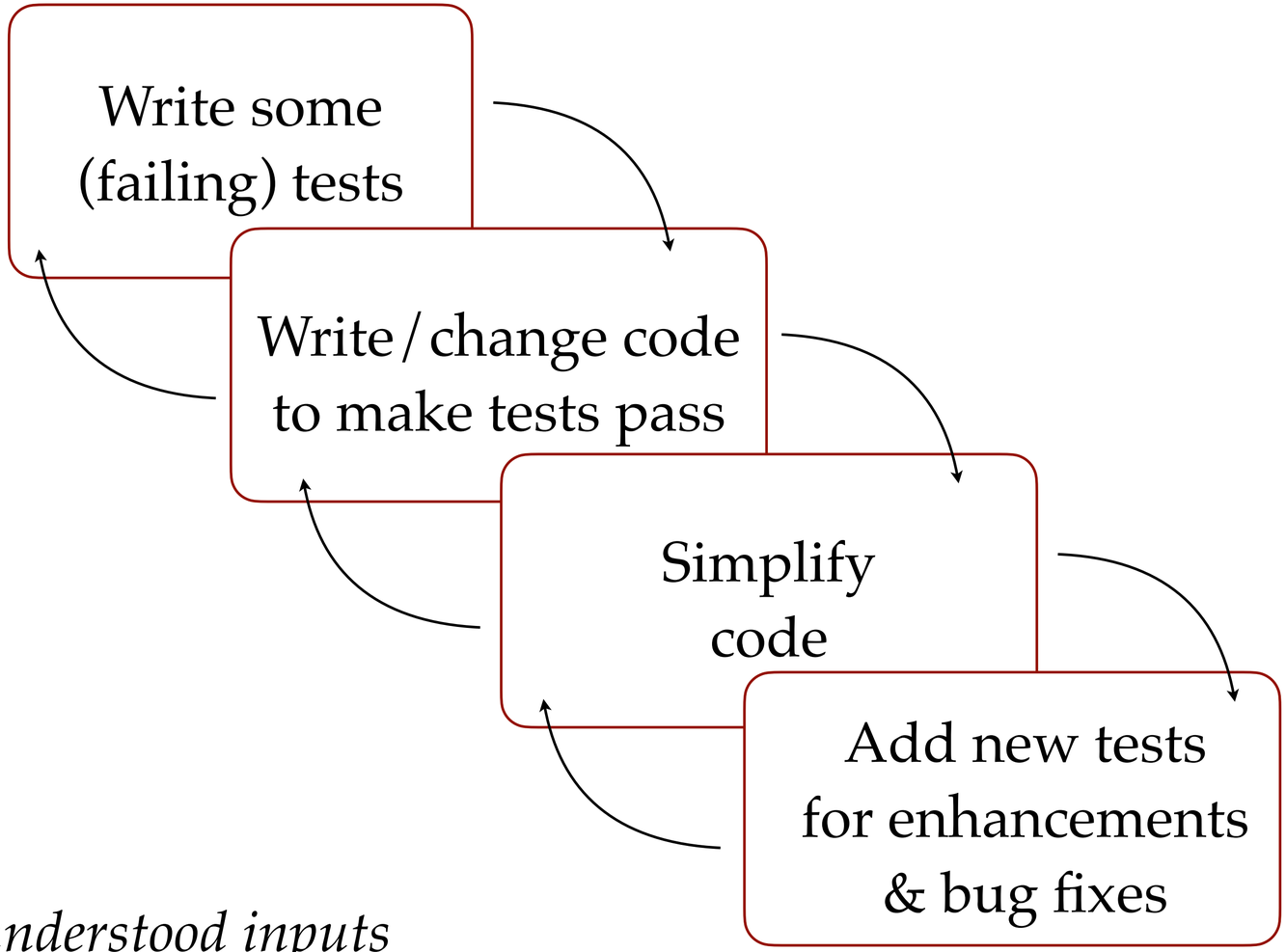
- TDDA Motivation & Methodology `assert sum(` 20 mins
- Checking data with constraints & `tdda` command line 20 mins
  - (Maybe) Rexpy (inferring regular expressions from examples) 5 mins
- Testing analytical code with `tdda` Python API 25 mins
  - (Possibly) Automatic test generation with gentest 10 mins
- Type VI Errors & wrap-up 10 mins`)`

TOTAL = 90 mins

✔

SOFTWARE DEVELOPMENT (WITH TDD*)

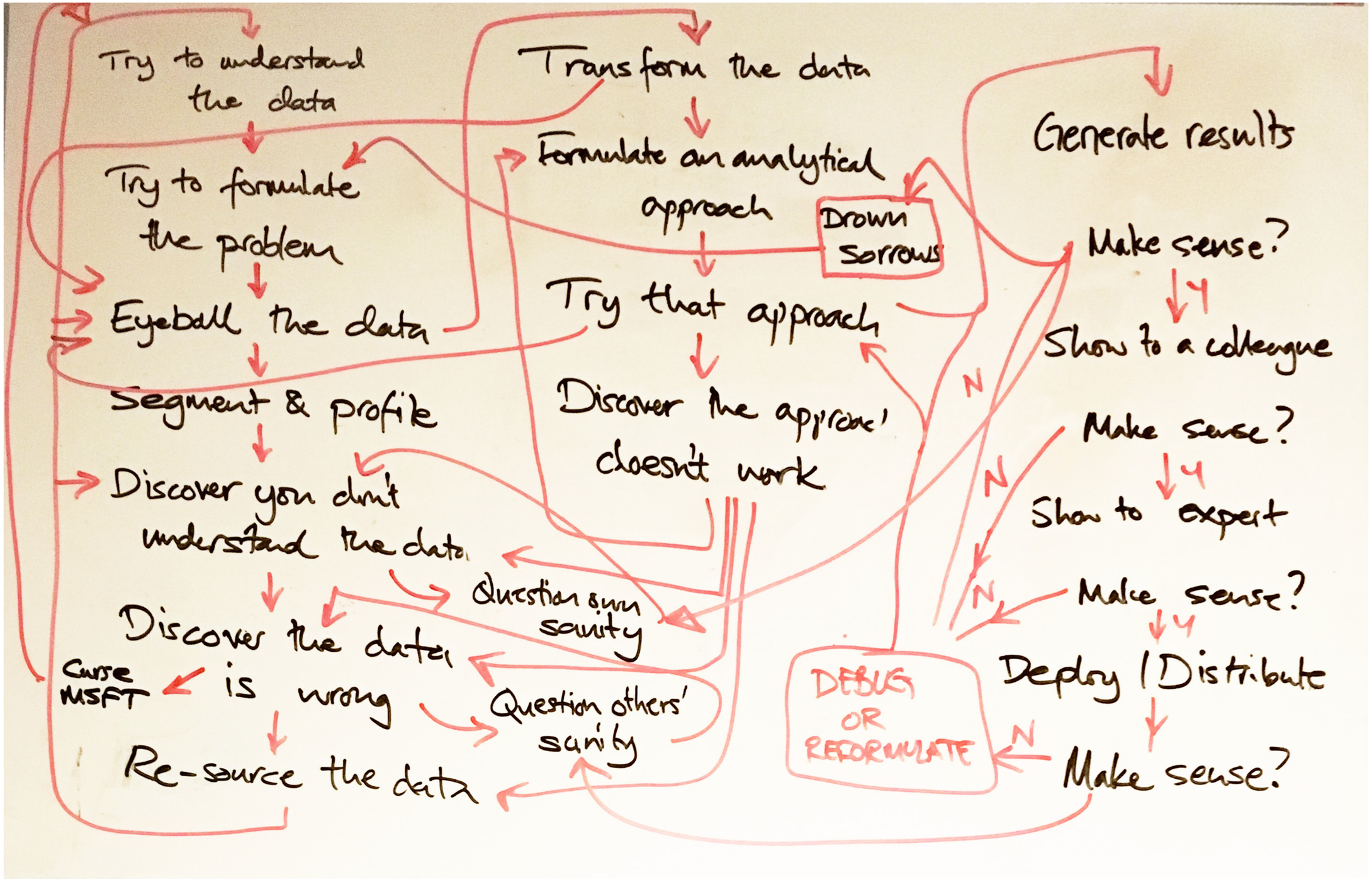
**test-driven development*



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
N
Make sense?
Y
Show to expert
N
Make sense?
Y
Deploy / Distribute
N
Make sense?

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"

Why is this lying bastard lying to me?

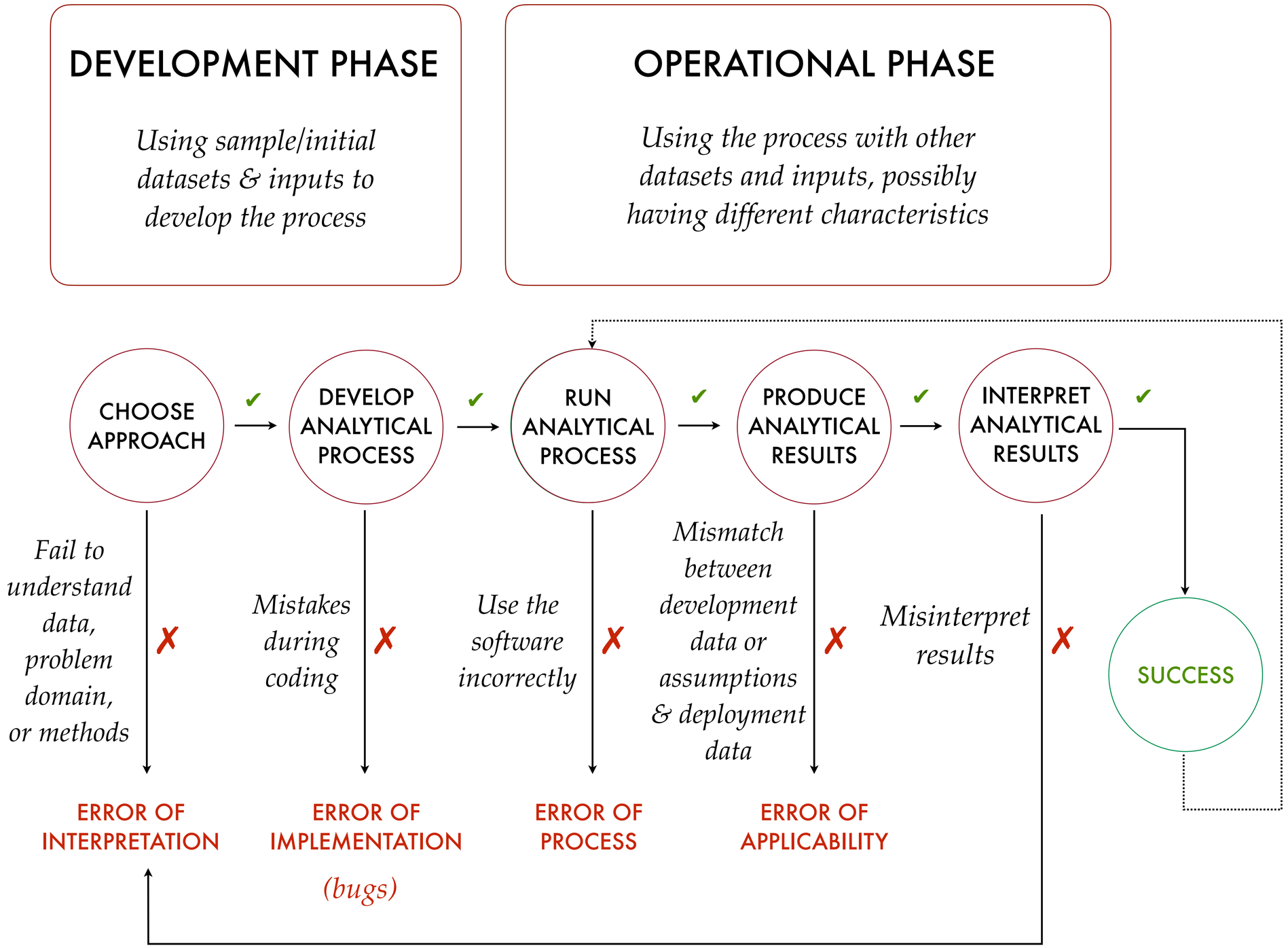
— Jeremy Paxman
(paraphrasing Louis Heron, paraphrasing unnamed mentor from Daily Worker)

How is this misleading data misleading me?

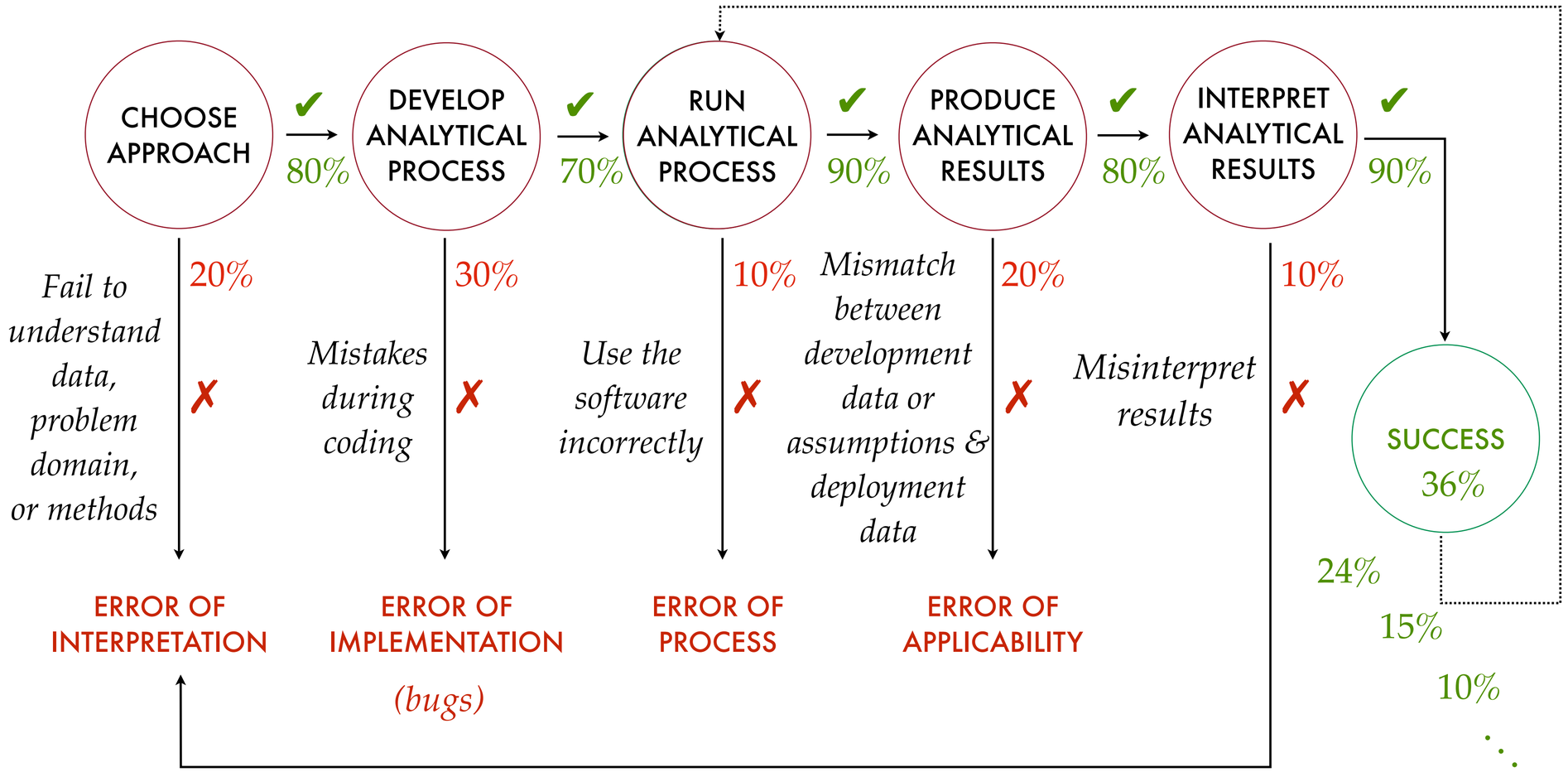
— Nick Radcliffe

TEST-DRIVEN DATA ANALYSIS IS …

- A methodology for attempting to improve the quality of data and analysis in data science (see blog `https://tdda.info`)
- An open source Python library (`pip install tdda`) with command-line tools and APIs for constraints, reference testing, automatic test generation and regular expression inference
- Available commercially as part of Miró, the data analysis suite from Stochastic Solutions, with enhanced capabilities in constraint generation, validation, reporting, synthetic data generation etc.


DEVELOPMENT PHASE
Using sample/initial datasets & inputs to develop the process
OPERATIONAL PHASE
Using the process with other datasets and inputs, possibly having different characteristics
CHOOSE APPROACH
DEVELOP ANALYTICAL PROCESS
RUN ANALYTICAL PROCESS
PRODUCE ANALYTICAL RESULTS
INTERPRET ANALYTICAL RESULTS
SUCCESS
Fail to understand data, problem domain, or methods
Mistakes during coding
Use the software incorrectly
Mismatch between development data or assumptions & deployment data
Misinterpret results
ERROR OF INTERPRETATION
ERROR OF IMPLEMENTATION
(bugs)
ERROR OF PROCESS
ERROR OF APPLICABILITY

If you buy into this model, it's sobering to attach probability estimates to each transition and calculate the probability of success after a few runs . . .



TDDA: MAIN IDEAS

1. Checking data: Constraint Discovery & Verification
   - A bit like "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 (currently in beta)

- Give **`tdda gentest`** a command/script to run.
- It generates tests for you.

TDDA LIBRARY

Install from PyPI (recommended)

```
pip install -U tdda
```

or (if your pip isn't connected to the specific Python binary you want to use)

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

or from Github (source)

```
git clone https://github.com/tdda/tdda.git
python setup.py install
```

TDDA LIBRARY

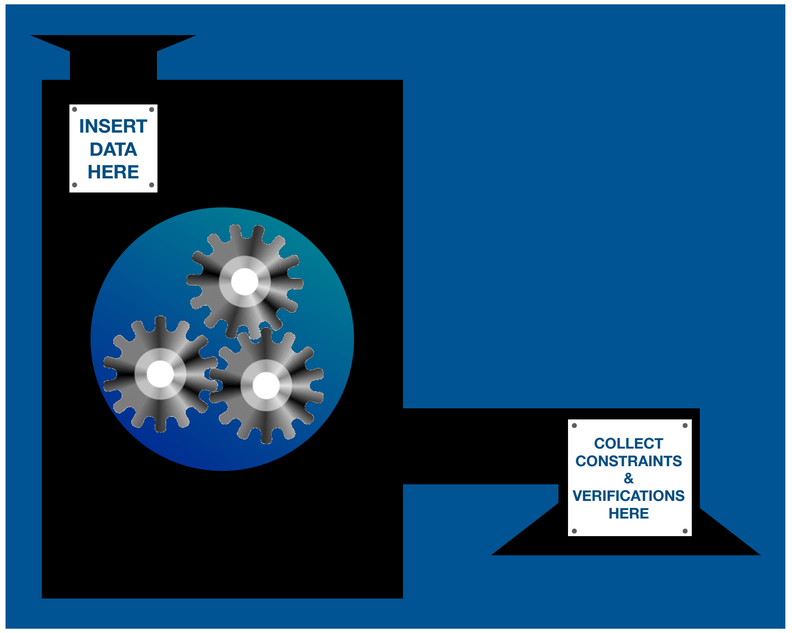
- Runs on Python (3), Mac, Linux & Windows, under `unittest` and `pytest`
- MIT Licensed
- Documentation:
  - Sphinx source in `doc` subdirectory
  - Built copy at `http://tdda.readthedocs.io`
- Quick reference:

  `http://www.tdda.info/pdf/tdda-quickref.pdf`

PYTHON TDDA LIBRARY (tdda)

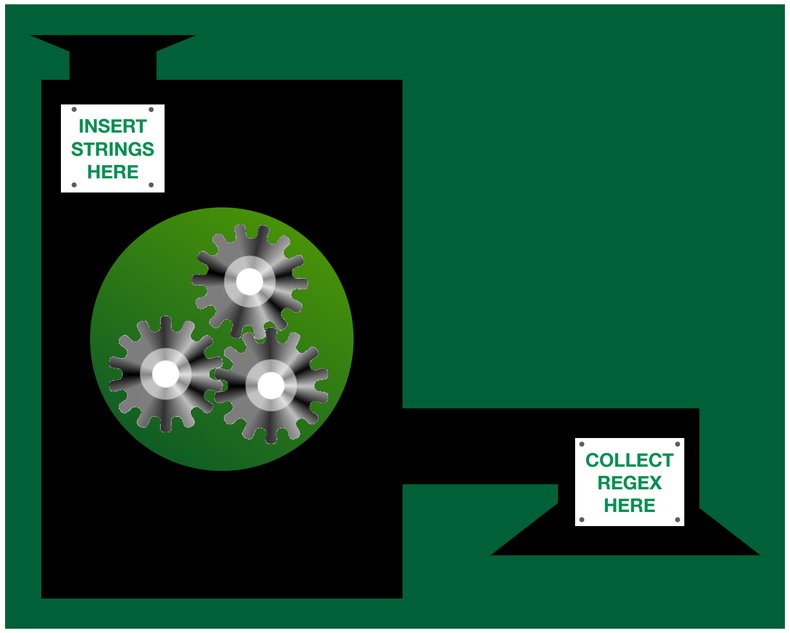
Discover • Verify • Detect

VERIFYING DATA
WITH
CONSTRAINTS



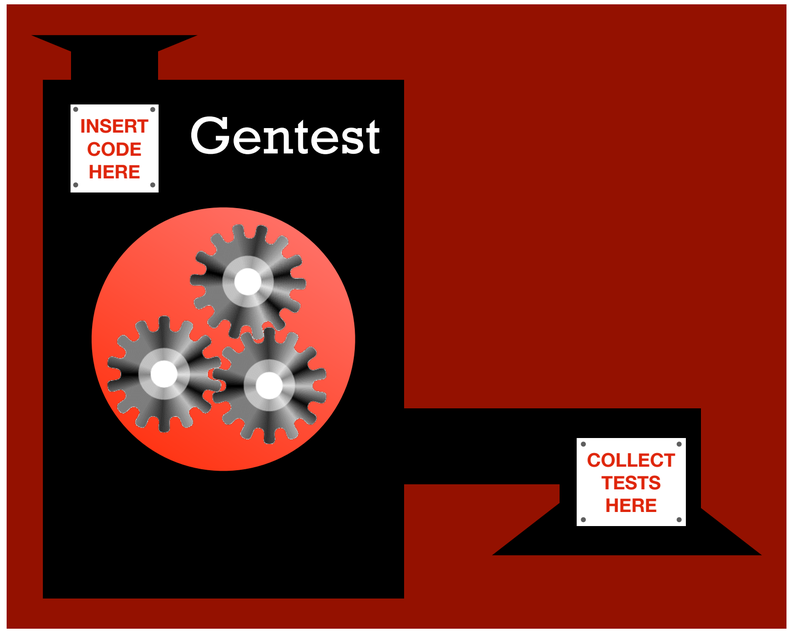
Rexpy

GENERATING
REGULAR EXPRESSIONS
FROM EXAMPLES

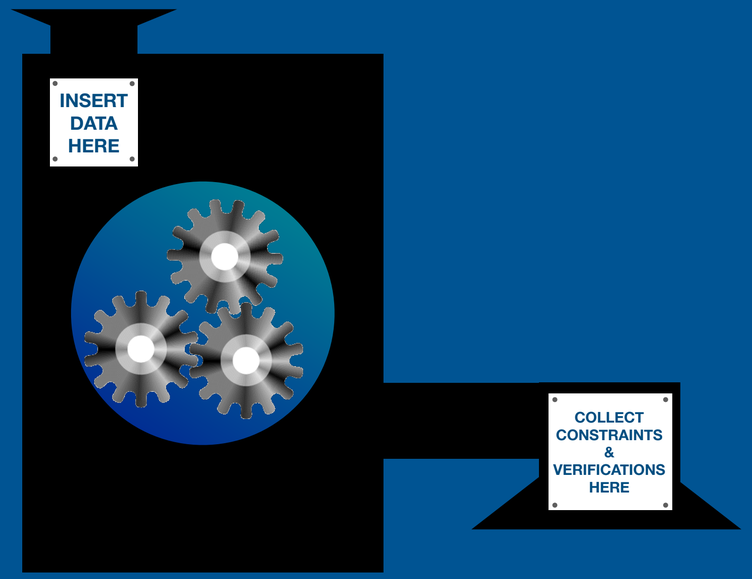


Reference Tests • Gentest

TESTING
DATA
PROCESSES



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



CONSTRAINT GENERATION, VERIFICATION & ANOMALY DETECTION

GIGO

Stochastic Solutions

EXCELLENT

AWFUL

NORMALITY
A.K.A. NEGLIGENCE

GARBAGE IN
•
GARBAGE OUT

ERROR DETECTION

GARBAGE IN
•
{WARNING / ERROR} OUT

ERROR CORRECTION

GARBAGE IN
•
GOLD OUT

CORRUPTION

GOLD IN
•
GARBAGE OUT

TDDA

DATA WORKERS SHOULD SIGN UP TO THE (ACTUAL) HYPOCRATIC OATH

First do no harm

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)

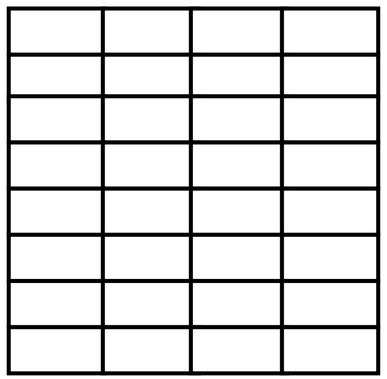
EXAMPLE CONSTRAINTS

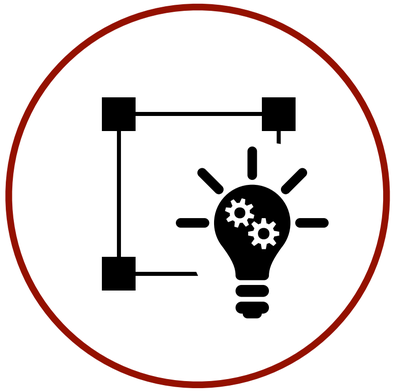
SINGLE FIELD CONSTRAINTS	DATASET CONSTRAINTS
Age ≤ 150	The data frame must contain field **CID**
type(Age) = int	Number of records must be 118
CID ≠ NULL	One field should be tagged O
CID unique *(in a customer table)*	**Date** should be sorted ascending
len(CardNumber) = 16	**MULTI-FIELD CONSTRAINTS**
Base in {"C", "G", "A", "T"}	StartDate ≤ EndDate
Vote ≠ "Trump"	AlmostEqual(F, m × a, 6)
StartDate < tomorrow()	sum(Favourite*) = 1
v < 2.97e8 (m/s)	minVal ≤ medianVal ≤ maxVal
Height ~ N(1.8, 0.2)	V ≤ H × W × 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.*	✗	✗	✗	✗	✔

AUTOMATIC CONSTRAINT GENERATION





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

TRAINING DATA

(believed to be "good")

AUTOMATIC DISCOVERY OF CONSTRAINTS

DISCOVERED CONSTRAINTS

(ideally, now refine by hand)

CONSTRAINT GENERATION

CSV

1. Copy examples somewhere:

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

2. Generate constraints from first 92 elements of periodic table (`testdata/elements92.csv`)

```
tdda discover -r testdata/elements92.csv elements92.tdda
```

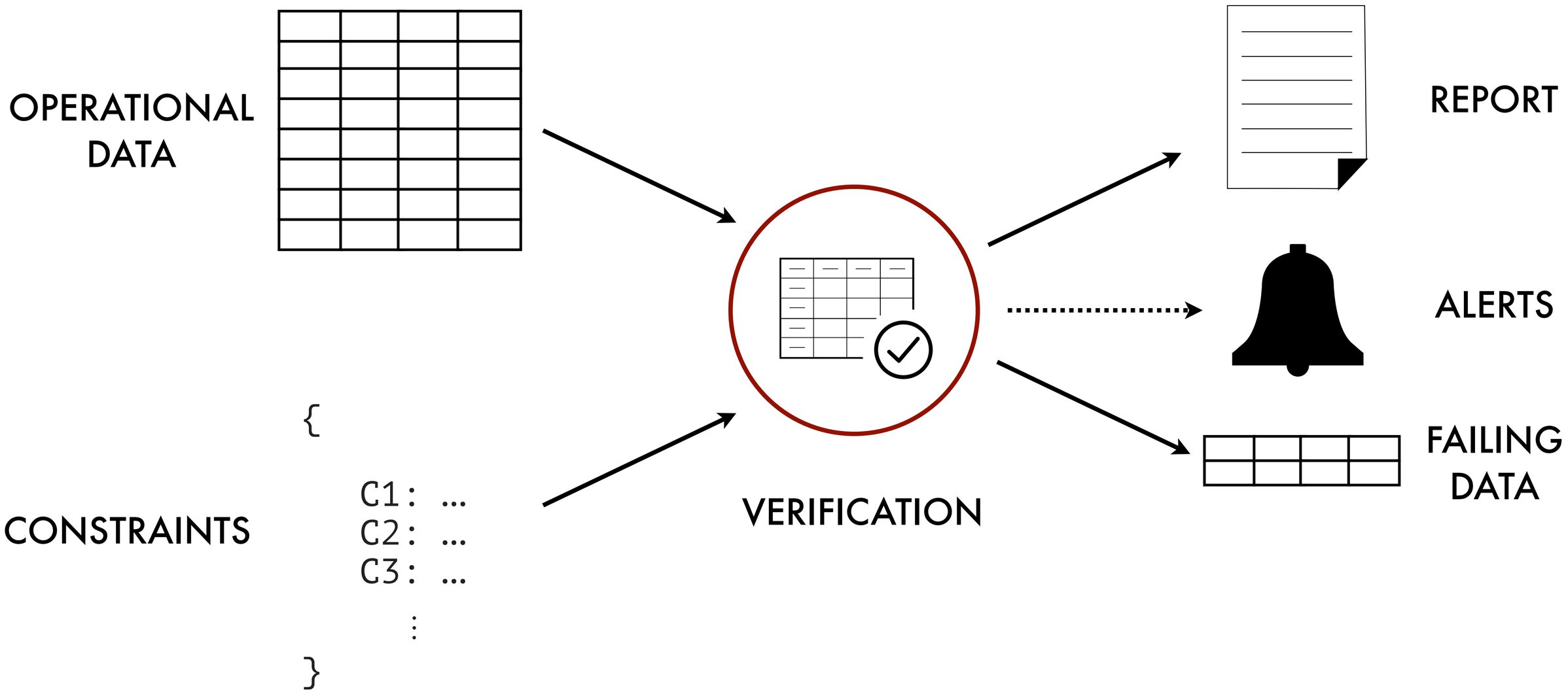
or `python elements_discover_92.py`

3. Examine output (`elements92.tdda`)

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

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

DATA VERIFICATION/DETECTION



CONSTRAINT VERIFICATION & ANOMALY DETECTION

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

```
tdda verify testdata/elements92.csv elements92.tdda
```

or `python elements_verify_92.py`

6. Now run verification of larger dataset (first 118 elements of periodic table) against the same constraints. Should **fail** (because, for example, atomic number now goes to 118).

```
tdda verify testdata/elements118.csv elements92.tdda
```

Or `python elements_verify_118_against_92.py`

You can use the `.parquet` *files instead of the CSV 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.*

FAILURE (ANOMALY) DETECTION

6. "Detect" the failing records

```
tdda detect testdata/elements118.parquet \
            elements92.tdda              \
            bads.csv                     \
            --per-constraint             \
            --output-fields              \
            --interleave
```

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

- 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)

ASIDE: PARQUET FILES

- Parquet is a cross-framework file format for storing typed data frames
- It was created by The Apache Foundation and effectively subsumes the feather file format, developed by Wes McKinney (Creator of Pandas) and Hadley Wycombe (Creator of HadleyVerse/TidyVerse in R)
- It's a good way to serialize DataFrames from many frameworks, including Pandas and Polars (and R!), to disk (i.e. to save them), and to deserialize them back into memory (i.e. to load them).
- It is broadly type-safe and efficient and has support for reading subsets of fields (columns) and records (rows)
- It is in `requirements.txt` for `tdda`.
- If you don't have it: `pip install pyarrow`

```
$ python
Python 3.11.0 (v3.11.0:deaf509e8f ...
Type "help", "copyright", "credits" or ...
>>> import pyarrow
>>> pyarrow.__version__
'15.0.0'

>>> import pandas as pd

>>> df = pd.DataFrame({'n': [0,1],
                       's': ['no', 'yes']})
>>> df
   n    s
0  0   no
1  1  yes
>>> df.to_parquet('foo.parquet', index=False)

>>> df2 = pd.read_parquet('foo.parquet')
>>> df2
   n    s
0  0   no
1  1  yes
>>> df2 = pd.read_parquet('foo.parquet')
```

CONSTRAINT GENERATION & VERIFICATION

7. Repeat verification of larger dataset (118 elements) against constraints generated against that same (118) data. Should **pass**.

   `tdda verify testdata/elements118.csv testdata/elements118.tdda`

   or `python elements_verify_118.py`

8. Finally, verify the constraints from 118 data against the 92 data. Should **pass**.

   `tdda verify testdata/elements92.csv elements118.tdda`

Note: fewer constraints are discovered for `elements118` *than for* `elements92` *(67 against 72). This is because there are nulls in some fields in the 118 data (the melting points, density etc.) but not in the 92 data.*

CONSTRAINT GENERATION & VERIFICATION PARQUET

1. Generate constraints from first 92 elements of periodic table

```
tdda discover -r testdata/elements92.parquet elements92p.tdda
```

2. Diff against `elements92.tdda`

```
diff elements92.tdda elements92p.tdda
```

```
3,4c3,4
<         "local_time": "2024-05-15 08:08:45",
<         "utc_time": "2024-05-15 07:07:45",
---
>         "local_time": "2024-05-15 08:08:23",
>         "utc_time": "2024-05-15 07:07:23",
5a6
>         "source": "testdata/elements92.csv",
7a9
>         "dataset": "elements92.csv",
9c11,12
<         "n_selected": 92
---
>         "n_selected": 92,
>         "tddafile": “elements92p.tdda”
```

Use `elements92.tdda`
or `elements92p.tdda`

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

```
tdda verify testdata/elements92.parquet elements92.tdda
```

EXAMPLE: `elements92.tdda`


```
{
    "fields": {
        "Z": {"type": "int", "min": 1, "max": 92, "sign": "positive", "max_nulls": 0, "no_duplicates": true},
        "Name": {"type": "string", "min_length": 3, "max_length": 12, "max_nulls": 0, "no_duplicates": true},
        "Symbol": {"type": "string", "min_length": 1,"max_length": 2, "max_nulls": 0, "no_duplicates": true},
        "Period": {"type": "int", "min": 1, "max": 7, "sign": "positive", "max_nulls": 0},
        "Group": {"type": "int", "min": 1, "max": 18,"sign": "positive"},
        "ChemicalSeries": {"type": "string", "min_length": 7, "max_length": 20, "max_nulls": 0,
                                          "allowed_values": ["Actinoid", "Alkali metal", "Alkaline earth metal”,
                                                              "Halogen", “Lanthanoid", "Metalloid", "Noble gas”,
                                                              "Nonmetal", "Poor metal", "Transition metal”]},
        "AtomicWeight": {"type": "real", "min": 1.007946, "max": 238.028914, "sign": "positive", "max_nulls": 0},
        "Etymology": {"type": "string", "min_length": 4, "max_length": 39, "max_nulls": 0},
        "RelativeAtomicMass": {"type": "real", "min": 1.007946, "max": 238.028914, "sign": “positive",
                                                 "max_nulls": 0},
        "MeltingPointC": {"type": "real", "min": -258.975, "max": 3675.0, "max_nulls": 1},
        "MeltingPointKelvin": {"type": "real", "min": 14.2, "max": 3948.0, "sign": "positive", "max_nulls": 1},
        "BoilingPointC": {"type": "real", "min": -268.93, "max": 5596.0, "max_nulls": 0},
        "BoilingPointF": {"type": "real", "min": -452.07, "max": 10105.0, "max_nulls": 0},
        "Density": {"type": "real", "min": 8.9e-05, "max": 22.610001, "sign": "positive", "max_nulls": 0},
        "Description": {"type": "string", "min_length": 1, "max_length": 83},
        "Colour": {"type": "string", "min_length": 4, "max_length": 80}
    }
}
```

EXAMPLE SUCCESSFUL VERIFICATION

constraints-examples — -bash — 113×40

```
0 godel:$ python elements_verify_92.py
FIELDS:

AtomicWeight: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓

Group: 0 failures  4 passes  type ✓  min ✓  max ✓  sign ✓

Name: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓

Density: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓

MeltingPointKelvin: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓

Symbol: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  no_duplicates ✓

Period: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓

Description: 0 failures  3 passes  type ✓  min_length ✓  max_length ✓

BoilingPointF: 0 failures  4 passes  type ✓  min ✓  max ✓  max_nulls ✓

Etymology: 0 failures  4 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓

ChemicalSeries: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  allowed_values ✓

MeltingPointC: 0 failures  4 passes  type ✓  min ✓  max ✓  max_nulls ✓

Z: 0 failures  6 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓  no_duplicates ✓

BoilingPointC: 0 failures  4 passes  type ✓  min ✓  max ✓  max_nulls ✓

Colour: 0 failures  3 passes  type ✓  min_length ✓  max_length ✓

RelativeAtomicMass: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓

SUMMARY:

Passes: 72
Failures: 0
0 godel:$
```

EXAMPLE UNSUCCESSFUL VERIFICATION

constraints-examples — -bash — 113×40

```
0 godel:$ python elements_verify_118_against_92.py
FIELDS:

AtomicWeight: 2 failures  3 passes  type ✓  min ✓  max ×  sign ✓  max_nulls ×

Group: 0 failures  4 passes  type ✓  min ✓  max ✓  sign ✓

Name: 1 failure  4 passes  type ✓  min_length ✓  max_length ×  max_nulls ✓  no_duplicates ✓

Density: 2 failures  3 passes  type ✓  min ✓  max ×  sign ✓  max_nulls ×

MeltingPointKelvin: 1 failure  4 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ×

Symbol: 1 failure  4 passes  type ✓  min_length ✓  max_length ×  max_nulls ✓  no_duplicates ✓

Period: 0 failures  5 passes  type ✓  min ✓  max ✓  sign ✓  max_nulls ✓

Description: 0 failures  3 passes  type ✓  min_length ✓  max_length ✓

BoilingPointF: 1 failure  3 passes  type ✓  min ✓  max ✓  max_nulls ×

Etymology: 2 failures  2 passes  type ✓  min_length ✓  max_length ×  max_nulls ×

ChemicalSeries: 0 failures  5 passes  type ✓  min_length ✓  max_length ✓  max_nulls ✓  allowed_values ✓

MeltingPointC: 1 failure  3 passes  type ✓  min ✓  max ✓  max_nulls ×

Z: 1 failure  5 passes  type ✓  min ✓  max ×  sign ✓  max_nulls ✓  no_duplicates ✓

BoilingPointC: 1 failure  3 passes  type ✓  min ✓  max ✓  max_nulls ×

Colour: 0 failures  3 passes  type ✓  min_length ✓  max_length ✓

RelativeAtomicMass: 2 failures  3 passes  type ✓  min ✓  max ×  sign ✓  max_nulls ×

SUMMARY:

Passes: 57
Failures: 15
0 godel:$
```

CONSTRAINTS API

DISCOVERY

```
from tdda.constraints.pdconstraints import discover_constraints
constraints = discover_constraints(df)
with open('constraints.tdda', 'w') as f:
    f.write(constraints.to_json())
```

VERIFICATION

```
from tdda.constraints.pdconstraints import verify_df
verification = verify_df(df, ‘constraints.tdda’) # (printable object)
constraints_df = verification.to_frame()         # (Pandas DataFrame)
```

OUTPUT OF `pd.DataFrame.to_frame()`

	field	failures	passes	type	min	min_length	max
0	AtomicWeight	2	3	True	True	NaN	False
2	Name	1	4	True	NaN	True	NaN
3	Density	2	3	True	True	NaN	False
4	MeltingPointKelvin	1	4	True	True	NaN	True
5	Symbol	1	4	True	NaN	True	NaN
7	BoilingPointF	1	3	True	True	NaN	True
8	Etymology	2	2	True	NaN	True	NaN
9	RelativeAtomicMass	2	3	True	True	NaN	False
11	MeltingPointC	1	3	True	True	NaN	True
12	Z	1	5	True	True	NaN	False
13	BoilingPointC	1	3	True	True	NaN	True

	max_length	sign	max_nulls	no_duplicates	allowed_values
0	NaN	True	False	NaN	NaN
2	False	NaN	True	True	NaN
3	NaN	True	False	NaN	NaN
4	NaN	True	False	NaN	NaN
5	False	NaN	True	True	NaN
7	NaN	NaN	False	NaN	NaN
8	False	NaN	False	NaN	NaN
9	NaN	True	False	NaN	NaN
11	NaN	NaN	False	NaN	NaN
12	NaN	True	True	True	NaN
13	NaN	NaN	False	NaN	NaN

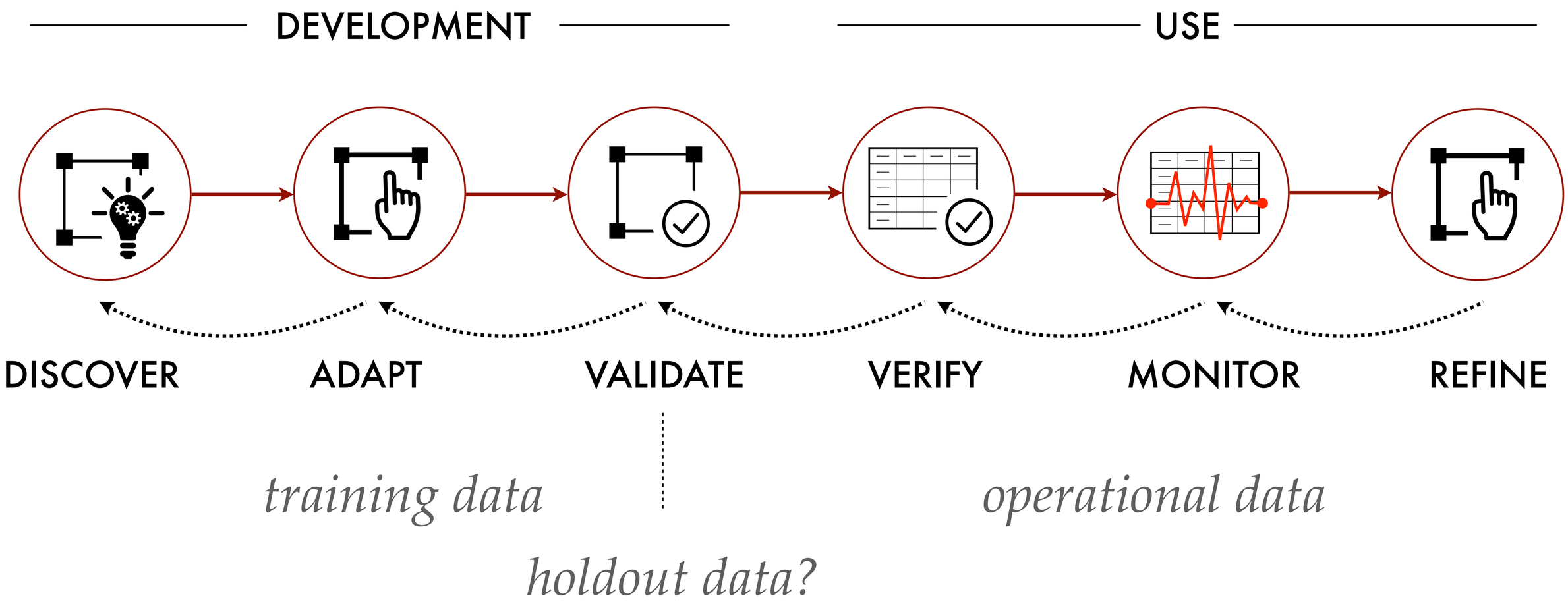
CONSTRAINTS

`True` *Satisfied*

`FALSE` *Not satisfied*

`NaN` *No constraint*

GENERATING CONSTRAINTS & VERIFYING DATA



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.

REFINING CONSTRAINTS

IDEAL PROCESS

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

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

* Tighten / Relax / Add / Delete / Choose Among

EXAMPLE CONSTRAINT DISCOVERY

account number	open date	close date	postcode	account type	overdraft limit
10074173	2004/05/07	∅	XZ97 6XC	current	0
10470530	2005/02/18	2011/11/14	BY1 7GK	current	6,600
10521429	2007/05/29	∅	IH2 6WE	current	4,800
10867373	2011/02/19	∅	NC53 0UZ	current	6,200
10956511	2006/02/08	2012/07/23	ZI60 8PG	current+	14,200
11156736	2009/01/08	∅	KM4 7BZ	current	0
11200644	2016/08/05	∅	GZ2 9UU	current	0
11586149	2011/04/07	∅	GQ66 7BN	current	0
11756979	2010/11/17	∅	VJ43 2NT	current	4,200
11935442	2012/03/14	∅	TB4 2CK	current	0
12011686	2013/12/30	2014/04/03	EA07 7GN	current+	0
12085703	2003/01/17	∅	OU45 2XC	current	1,700
12226724	2012/07/18	∅	VM44 6FL	current	0
12337790	2009/12/22	∅	PU63 0UJ	current	12,200
12350638	2004/10/03	∅	UY7 3YV	current+	16,800
12446447	2012/10/04	∅	RT1 8QO	current	11,300
12466957	2007/12/10	∅	VS84 2WY	current	13,700
12797926	2010/01/31	∅	LY9 2EQ	offset	0
12831336	2018/11/02	∅	EX31 8FM	current	16,600
12923415	2006/06/04	∅	IY62 6CN	current	6,600

```
$ tdda discover -r training.csv constraints.tdda
{
    "creation_metadata": {
        "local_time": "2019-03-07 08:08:56",
        "utc_time": "2019-03-07 08:08:56",
        "creator": "TDDA 1.0.21",
        "source": "data.csv",
        "host": "bartok.local",
        "user": "njr",
        "dataset": "data.csv",
        "n_records": 20,
        "n_selected": 20,
        "tddafile": "constraints.tdda"
    },
    "fields": {
        .
        .
        .
```

EXAMPLE CONSTRAINT DISCOVERY

account number	open date	close date	postcode	account type	overdraft limit
10074173	2004/05/07	∅	XZ97 6XC	current	0
10470530	2005/02/18	2011/11/14	BY1 7GK	current	6,600
10521429	2007/05/29	∅	IH2 6WE	current	4,800
10867373	2011/02/19	∅	NC53 0UZ	current	6,200
10956511	2006/02/08	2012/07/23	ZI60 8PG	current+	14,200
11156736	2009/01/08	∅	KM4 7BZ	current	0
11200644	2016/08/05	∅	GZ2 9UU	current	0
11586149	2011/04/07	∅	GQ66 7BN	current	0
11756979	2010/11/17	∅	VJ43 2NT	current	4,200
11935442	2012/03/14	∅	TB4 2CK	current	0
12011686	2013/12/30	2014/04/03	EA07 7GN	current+	0
12085703	2003/01/17	∅	OU45 2XC	current	1,700
12226724	2012/07/18	∅	VM44 6FL	current	0
12337790	2009/12/22	∅	PU63 0UJ	current	12,200
12350638	2004/10/03	∅	UY7 3YV	current+	16,800
12446447	2012/10/04	∅	RT1 8QO	current	11,300
12466957	2007/12/10	∅	VS84 2WY	current	13,700
12797926	2010/01/31	∅	LY9 2EQ	offset	0
12831336	2018/11/02	∅	EX31 8FM	current	16,600
12923415	2006/06/04	∅	IY62 6CN	current	6,600

```
"account_number": {
    "type": "int",
    "min": 10074173,
    "max": 12923415,
    "sign": "positive",
    "max_nulls": 0,
    "no_duplicates": true
},
"open_date": {
    "type": "date",
    "min": "2003-01-17 00:00:00",
    "max": "2018-11-02 00:00:00",
    "max_nulls": 0
},
"close_date": {
    "type": "date",
    "min": "2011-11-14 00:00:00",
    "max": "2014-04-03 00:00:00"
},
```

EXAMPLE CONSTRAINT DISCOVERY

account number	open date	close date	postcode	account type	overdraft limit
10074173	2004/05/07	∅	XZ97 6XC	current	0
10470530	2005/02/18	2011/11/14	BY1 7GK	current	6,600
10521429	2007/05/29	∅	IH2 6WE	current	4,800
10867373	2011/02/19	∅	NC53 0UZ	current	6,200
10956511	2006/02/08	2012/07/23	ZI60 8PG	current+	14,200
11156736	2009/01/08	∅	KM4 7BZ	current	0
11200644	2016/08/05	∅	GZ2 9UU	current	0
11586149	2011/04/07	∅	GQ66 7BN	current	0
11756979	2010/11/17	∅	VJ43 2NT	current	4,200
11935442	2012/03/14	∅	TB4 2CK	current	0
12011686	2013/12/30	2014/04/03	EA07 7GN	current+	0
12085703	2003/01/17	∅	OU45 2XC	current	1,700
12226724	2012/07/18	∅	VM44 6FL	current	0
12337790	2009/12/22	∅	PU63 0UJ	current	12,200
12350638	2004/10/03	∅	UY7 3YV	current+	16,800
12446447	2012/10/04	∅	RT1 8QO	current	11,300
12466957	2007/12/10	∅	VS84 2WY	current	13,700
12797926	2010/01/31	∅	LY9 2EQ	offset	0
12831336	2018/11/02	∅	EX31 8FM	current	16,600
12923415	2006/06/04	∅	IY62 6CN	current	6,600

```
"postcode": {
    "type": "string",
    "min_length": 7,
    "max_length": 8,
    "max_nulls": 0,
    "no_duplicates": true,
    "rex": ["^[A-Z]{2}\\d{1,2} \\d[A-Z]{2}$"]
},
"account_type": {
    "type": "string",
    "min_length": 6,
    "max_length": 8,
    "max_nulls": 0,
    "allowed_values": [
            "current",
            "current+",
            "offset”
    ],
    "rex": ["^[a-z]{6,7}$", "^current\\+$"]
},
```

EXAMPLE CONSTRAINT DISCOVERY

account number	open date	close date	postcode	account type	overdraft limit
10074173	2004/05/07	∅	XZ97 6XC	current	0
10470530	2005/02/18	2011/11/14	BY1 7GK	current	6,600
10521429	2007/05/29	∅	IH2 6WE	current	4,800
10867373	2011/02/19	∅	NC53 0UZ	current	6,200
10956511	2006/02/08	2012/07/23	ZI60 8PG	current+	14,200
11156736	2009/01/08	∅	KM4 7BZ	current	0
11200644	2016/08/05	∅	GZ2 9UU	current	0
11586149	2011/04/07	∅	GQ66 7BN	current	0
11756979	2010/11/17	∅	VJ43 2NT	current	4,200
11935442	2012/03/14	∅	TB4 2CK	current	0
12011686	2013/12/30	2014/04/03	EA07 7GN	current+	0
12085703	2003/01/17	∅	OU45 2XC	current	1,700
12226724	2012/07/18	∅	VM44 6FL	current	0
12337790	2009/12/22	∅	PU63 0UJ	current	12,200
12350638	2004/10/03	∅	UY7 3YV	current+	16,800
12446447	2012/10/04	∅	RT1 8QO	current	11,300
12466957	2007/12/10	∅	VS84 2WY	current	13,700
12797926	2010/01/31	∅	LY9 2EQ	offset	0
12831336	2018/11/02	∅	EX31 8FM	current	16,600
12923415	2006/06/04	∅	IY62 6CN	current	6,600

```
            "overdraft_limit": {
                "type": "int",
                "min": 0,
                "max": 16800,
                "sign": "non-negative",
                "max_nulls": 0
            }
        }
    }
```

CONFIRM THAT CONSTRAINTS PASS ON TRAINING DATA

```
$ tdda verify training.csv constraints.tdda
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 ✓
postcode:        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 ✓
Constraints passing: 30  Constraints failing: 0
```

CHECK WHETHER NEW DATA SATISFIES CONSTRAINTS

```
$ tdda verify operationaldata.csv constraints.tdda
account_number:   2 failures  4 passes
                  type ✓  min ✗  max ✗  sign ✓ max_nulls ✓  no_duplicates ✓
open_date:        1 failure  2 passes
                  type ✓  min ✗  max ✗  max_nulls ✓
close_date:       2 failures  1 pass
                  type ✓  min ✗  max ✗
postcode:         0 failures  6 passes
                  type ✓  min_length ✓  max_length ✓  max_nulls ✓
                  no_duplicates ✓  rex ✓
account_type:     3 failures  3 passes
                  type ✓  min_length ✗  max_length ✓ max_nulls ✓
                  allowed_values ✗  rex ✗
overdraft_limit:  1 failure  4 passes
                  type ✓  min ✓  max ✗  sign ✓  max_nulls ✓
Constraints passing: 21  Constraints failing: 9
```

FIND FAILING VALUES IN THE NEW DATA

```
$ tdda detect operationaldata.csv constraints.tdda failures.csv
account_number:   2 failures  4 passes
                  type ✓  min ✗  max ✗  sign ✓ max_nulls ✓  no_duplicates ✓
open_date:        1 failure  2 passes
                  type ✓  min ✗  max ✗  max_nulls ✓
close_date:       2 failures  1 pass
                  type ✓  min ✗  max ✗
postcode:         0 failures  6 passes
                  type ✓  min_length ✓  max_length ✓  max_nulls ✓
                  no_duplicates ✓  rex ✓
account_type:     3 failures  3 passes
                  type ✓  min_length ✗  max_length ✓ max_nulls ✓
                  allowed_values ✗  rex ✗
overdraft_limit:  1 failure  4 passes
                  type ✓  min ✓  max ✗  sign ✓  max_nulls ✓
Records passing: 76  Records failing: 24
```

account number	open date	close date	postcode	account type	overdraft limit	account number min ok	account number max ok	open date min ok	close date min ok	close date max ok	account type min ok	account type values ok	account type rex ok	overdraft limit max ok	nfailures
10033300	2005/02/08	∅	MO73 2YX	current	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	**1**
10050552	2009/02/24	∅	XK5 3NM	current	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	**1**
10066665	2003/02/16	∅	PI9 3BG	current+	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	**1**
10174458	2011/07/18	**2016/09/27**	SX5 5PV	current	0	✓	✓	✓	✓	✗	✓	✓	✓	✓	**1**
10278760	2004/05/15	**2007/11/20**	BA72 8XF	current	**18,000**	✓	✓	✓	✗	✓	✓	✓	✓	✗	**2**
10352931	2004/06/15	∅	WJ9 2OA	**basic**	0	✓	✓	✓	∅	∅	✗	✗	✗	✓	**3**
10440004	**2002/12/19**	∅	YC24 4UT	current+	4,800	✓	✓	✗	∅	∅	✓	✓	✓	✓	**1**
10476972	2018/01/27	∅	OE5 9UI	current	**17,400**	✓	✓	✓	∅	∅	✓	✓	✓	✗	**1**
10699455	2018/09/17	∅	GQ1 9IV	current	**19,200**	✓	✓	✓	∅	∅	✓	✓	✓	✗	**1**
10717064	2003/11/30	∅	VM1 8WR	current	**20,000**	✓	✓	✓	∅	∅	✓	✓	✓	✗	**1**
10824167	2008/05/21	∅	NI55 0OS	**basic**	1,400	✓	✓	✓	∅	∅	✗	✗	✗	✓	**3**
10902721	2005/10/30	∅	LL22 5UX	current	**17,100**	✓	✓	✓	∅	∅	✓	✓	✓	✗	**1**
10962316	2003/12/25	**2005/02/25**	XX9 2RP	current	4,000	✓	✓	✓	✗	✓	✓	✓	✓	✓	**1**
11005672	2007/06/10	∅	ZT64 3WP	**basic**	0	✓	✓	✓	∅	∅	✗	✗	✗	✓	**3**
11385380	2015/08/07	∅	WC47 7OA	current+	**19,900**	✓	✓	✓	∅	∅	✓	✓	✓	✗	**1**
11589140	2007/11/04	∅	PF53 9BM	**basic**	8,300	✓	✓	✓	∅	∅	✗	✗	✗	✓	**3**
11604974	2008/04/27	**2010/02/18**	XE76 8YA	current	2,800	✓	✓	✓	✗	✓	✓	✓	✓	✓	**1**
11705553	2014/05/02	**2018/05/05**	LK55 9TE	current	0	✓	✓	✓	✓	✗	✓	✓	✓	✓	**1**
11816734	2012/04/27	∅	SS73 8VO	**basic**	15,200	✓	✓	✓	∅	∅	✗	✗	✗	✓	**3**
11957115	2007/04/01	∅	WO8 7QE	current	19,500	✓	✓	✓	∅	∅	✓	✓	✓	✗	**1**
12086022	2013/05/29	**2016/10/28**	UA06 1CI	**premium**	0	✓	✓	✓	✓	✗	✓	✗	✓	✓	**2**
12899220	2014/09/08	**2015/06/08**	UX80 2RO	current	0	✓	✓	✓	✓	✗	✓	✓	✓	✓	**1**
12940182	2017/12/13	∅	WA93 4SW	current	0	✓	✗	✓	∅	∅	✓	✓	✓	✓	**1**
12987964	2015/08/27	∅	SD83 3CR	current	0	✓	✗	✓	∅	∅	✓	✓	✓	✓	**1**

account number	open date	close date	postcode	account type	overdraft limit	account number min ok	account number max ok	open date min ok	close date min ok	close date max ok	account type min ok	account type values ok	account type rex ok	overdraft limit max ok	nfailures
10033300	2005/02/08	∅	MO73 2YX	current	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	1
10050552	2009/02/24	∅	XK5 3NM	current	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	1
10066665	2003/02/16	∅	PI9 3BG	current+	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	1
10174458	2011/07/18	2016/09/27	SX5 5PV	current	0	✓	✓	✓	✓	✗	✓	✓	✓	✓	1
10962316	2003/12/25	2005/02/25	XX9 2RP	current	4,000	✓	✓	✓	✗	✓					2
11005672	2007/06/10	∅	ZT64 3WP	basic	0	✓	✓	✓	∅	∅					3
11385380	2015/08/07	∅	WC47 7OA	current+	19,900	✓	✓	✓	∅	∅					1
11589140	2007/11/04	∅	PF53 9BM	basic	8,300	✓	✓	✓	∅	∅					1
11604974	2008/04/27	2010/02/18	XE76 8YA	current	2,800	✓	✓	✓	✗	✓					1
11705553	2014/05/02	2018/05/05	LK55 9TE	current	0	✓	✓	✓	✓	✗	✓	✓	✓	✓	1
11816734	2012/04/27	∅	SS73 8VO	basic	15,200	✓	✓	✓	∅	∅	✗	✗	✗	✓	3
11957115	2007/04/01	∅	WO8 7QE	current	19,500	✓	✓	✓	∅	∅	✓	✓	✓	✗	1
12086022	2013/05/29	2016/10/28	UA06 1CI	premium	0	✓	✓	✓	✓	✗	✓	✗	✓	✓	2
12899220	2014/09/08	2015/06/08	UX80 2RO	current	0	✓	✓	✓	✓	✗	✓	✓	✓	✓	1
12940182	2017/12/13	∅	WA93 4SW	current	0	✓	✗	✓	∅	∅	✓	✓	✓	✓	1
12987964	2015/08/27	∅	SD83 3CR	current	0	✓	✗	✓	∅	∅	✓	✓	✓	✓	1

original data for failing records

indicator columns for each failing constraint

number of failures for each record

account number	open date	close date	postcode	account type	overdraft limit	account number min ok	account number max ok	open date min ok	close date min ok	close date max ok	account type min ok	account type values ok	account type rex ok	overdraft limit max ok	nfailures
10033300	2005/02/08	∅	MO73 2YX	current	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	1
10050552	2009/02/24	∅	XK5 3NM	current	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	1
10066665	2003/02/16	∅	PI9 3BG	current+	0	✗	✓	✓	∅	∅	✓	✓	✓	✓	1
10174458	2011/07/13	2016/09/27	SX5 5PV	current	0	✓	✓	✓	✓	✗	✓	✓	✓	✓	1

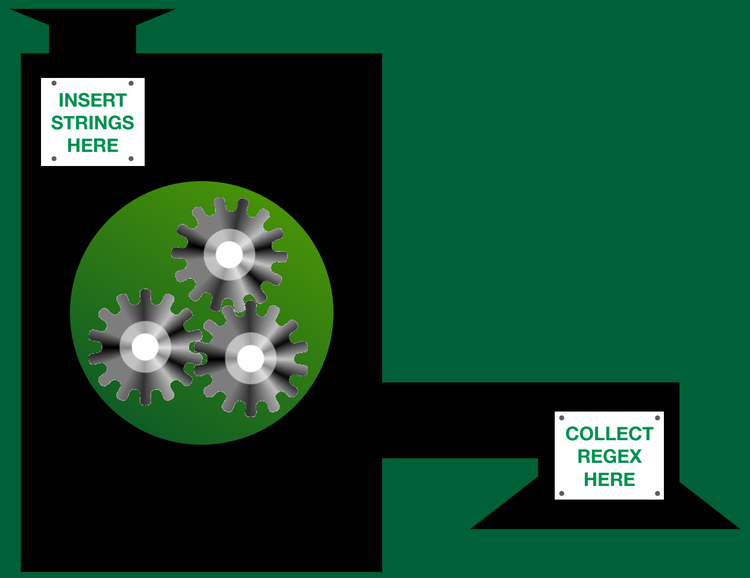
account number
10033300
10050552
10066665

account number min ok
✗
✗
✗

nfailures
1
1
1

11005672	2007/06/10	∅								∅	✗	✗	✗	✓	3
11385380	2015/08/07	∅								∅	✓	✓	✓	✗	1
11589140	2007/11/04	∅								∅	✗	✗	✗	✓	3
11604974	2008/04/27	2010/02								✓	✓	✓	✓	✓	1
11705553	2014/05/02	2018/05								✗	✓	✓	✓	✓	1
11816734	2012/04/27	∅								∅	✗	✗	✗	✓	3
11957115	2007/04/01	∅								∅	✓	✓	✓	✗	1
12086022	2013/05/29	2016/10								✗	✓	✗	✓	✓	2
12899220	2014/09/08	2015/06								✗	✓	✓	✓	✓	1
12940182	2017/12/13	∅								∅	✓	✓	✓	✓	1
12987964	2015/08/27	∅								∅	✓	✓	✓	✓	1

```
"account_number": {
    "type": "int",
    "min": 10074173,
    "max": 12923415,
    "sign": "positive",
    "max_nulls": 0,
    "no_duplicates": true
},
```



Rexpy

Automatic construction of regular expressions from data

REGULAR EXPRESSIONS

212-977-0331

totally specific (overfitted)

^212\-977\-0331$

^[12]{3}\-[7-9]{3}\-(0|1|3){4}$ *specific digits*

^\d{3}\-\d{3}\-\d{4}$ *What Rexpy produces*

^\d+\-\d+\-\d+$ + *means* ***"1 or more times"***

totally unspecific (underfitted)
(matches all strings)

^.*$

. *matches any char*

* *means* ***"0 or more times"***

REGULAR EXPRESSIONS

EH22 4EH

SW1A 1AA

B1 1BC

^[A-Z]{1,2}[0-9]{5}[A-Z]? [0-9]{5}[A-Z]{2}$

Some people, when confronted with a problem, think

"I know, I'll use regular expressions."

Now they have two problems.

— *Jamie Zawinski*
`comp.emacs.xemacs`, 1997

PROS	CONS
Powerful	*Ugly
Fast	Hard to write
Widely supported	Harder to read
	Harder still to debug
	Even harder to quote/escape†

---

*Extremely . . .

† `r'...'` is your friend

Why not let the computer do the work?

```
$ rexpy
212-988-0321
987-654-3210
476 123 8829
123 456 7890
701 734 9288
177 441 7712
```

```
^[0-9]{3}\-[0-9]{3}\-[0-9]{4}$
^[0-9]{3}\ [0-9]{3}\ [0-9]{4}$
```

Rexpy currently never groups white space with punctuation

COMMAND LINE

```
$ rexpy --help
Usage:
    rexpy [FLAGS] [input file [output file]]
or
    python -m tdda.rexpy.rexpy [FLAGS] [input file [output file]]
If input file is provided, it should contain one string per line; otherwise lines will be read from
standard input.
If output file is provided, regular expressions found will be written to that (one per line); otherwise
they will be printed.
FLAGS are optional flags. Currently::
  -h, --header      Discard first line, as a header.
  -?, --help        Print this usage information and exit (without error)
  -g, --group       Generate capture groups for each variable fragment of each regular expression
                    generated, i.e. surround variable components with parentheses
                        e.g.     '^([A-Z]+)\-([0-9]+)$'
                        becomes  '^[A-Z]+\-[0-9]+$'
  -u, --underscore  Allow underscore to be treated as a letter. Mostly useful for matching
                    identifiers. Also allow -_.
  -d, --dot         Allow dot to be treated as a letter. Mostly useful for matching identifiers.
                    Also -. --period.
  -m, --minus       Allow minus to be treated as a letter. Mostly useful for matching
                    identifiers. Also --hyphen or --dash.
  -v, --version     Print the version number.
```

PYTHON API

Get examples: `tdda examples rexpy`

ids.py:

```
from tdda import rexpy

corpus = ['123-AA-971', '12-DQ-802', '198-AA-045', '1-BA-834']
results = rexpy.extract(corpus)
print(f'Number of regular expressions found: {len(results)}')
for rex in results:
    print('    ' + rex)
```

RESULTS

```
$ python ids.py
Number of regular expressions found: 1
   ^\d{1,3}\-[A-Z]{2}\-\d{3}$
```

PYTHON API WITH PANDAS

pandas_ids.py:

```
import pandas as pd
import numpy as np
from tdda import rexpy

df = pd.DataFrame({'a3': ["one", "two", np.NaN],
                   'a45': ['three', 'four', 'five']})
re3 = rexpy.pdextract(df['a3'])
re45 = rexpy.pdextract(df['a45'])
re345 = rexpy.pdextract([df['a3'], df['a45']])
print(f'  re3: {re3}')
print(f' re45: {re45}')
print(f're345: {re345}')
```

RESULTS

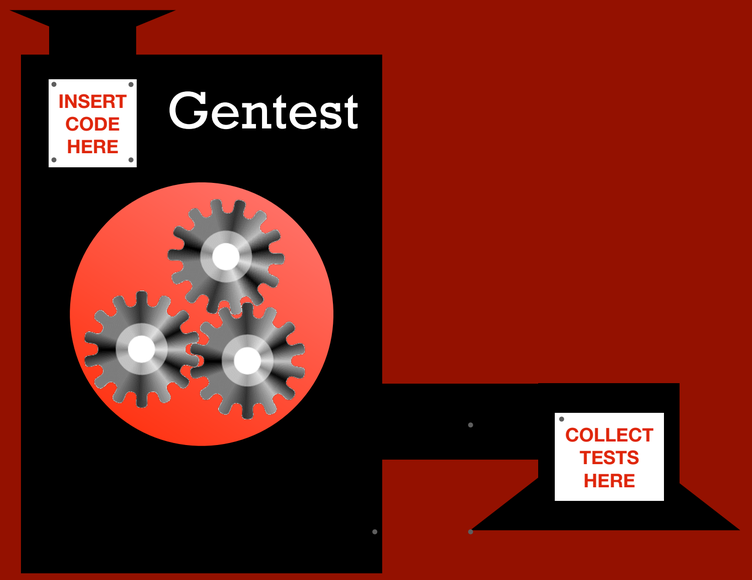
```
$ python pandas_ids.py
  re3: ['^[a-z]{3}$']
 re45: ['^[a-z]{4,5}$']
re345: ['^[a-z]{3,5}$']
```

Rexpy is intended for strings with structure

National insurance numbers
UUIDs
Zip codes
Postcodes
Credit card numbers
email addresses
car registration numbers
version numbers
identifiers
…

Feeding free text to Rexpy will be slow, frustrating, & useless

Free text
Novels
…



REFERENCE TESTS & AUTOMATIC TEST GENERATION WITH GENTEST

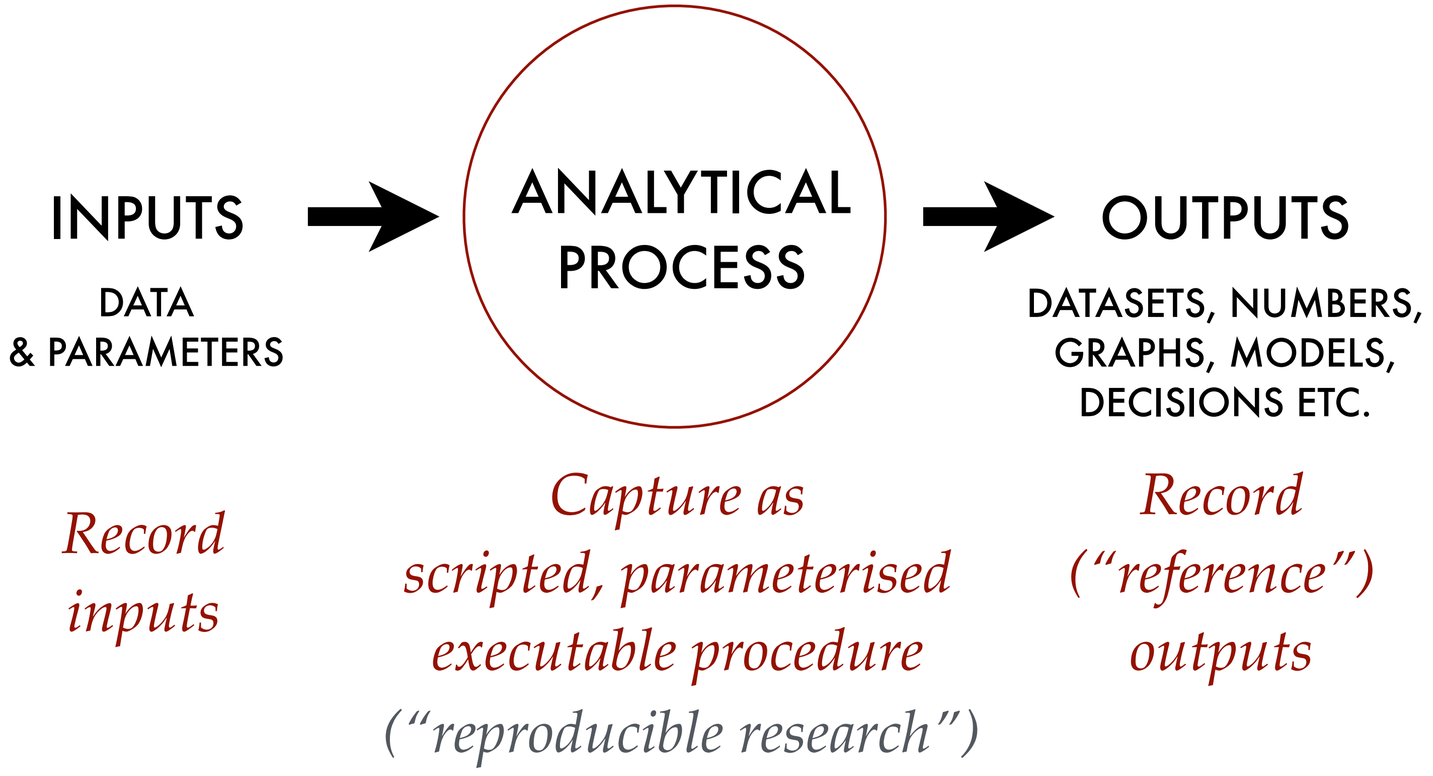
TESTING DATA PIPELINES

… 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.assertFileCorrect(actual_path, ref_path, . . .)
```

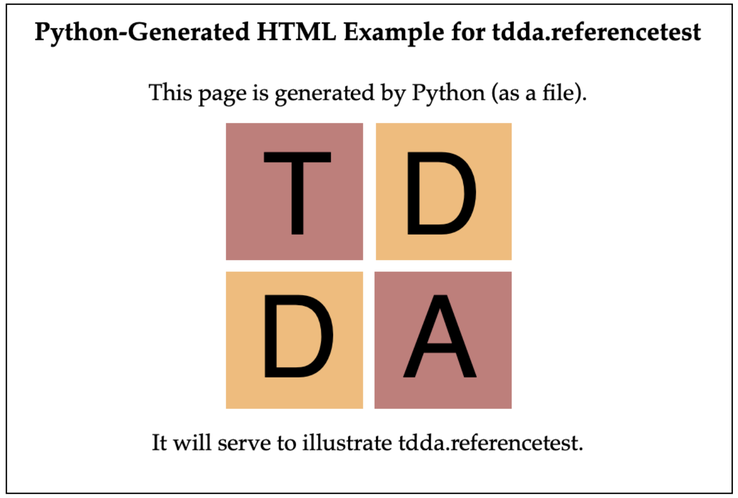
Check a multiple generated files against respective reference files:

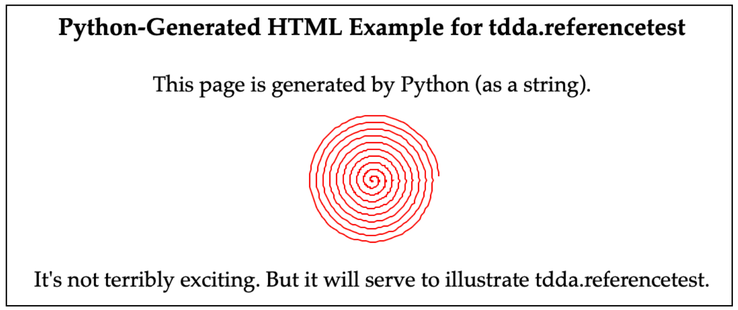
```
self.assertFilesCorrect(actual_paths, ref_paths, . . .)
```

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 looks 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.assertFileCorrect(outpath, 'file_result.html',
                       ignore_patterns=['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. Capitalise `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`
   or `(cd pytest; pytest)`

6. Check modified results in (reported) temporary directory are as expected; run the suggested `diff` command or something similar (`opendiff`, `fc`, …). Again, note that in addition to the changes you introduced, the Copyright and Version lines are different.

TAGGING TESTS TO RUN A SUBSET

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.*

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:

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

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

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

only tagged tests (`-1`)

re-write reference results (`-W`)

*** 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
- 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 whether to ignore actual data in particular columns
- 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.assertOnDiskDataFrameCorrect`(*actual_path, ref_path, . . .*)

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

`self.assertOnDiskDataFramesCorrect`(*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.assertDataFramesEqual`(*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) `reference_test_examples` directory

   ```
   cd ../reference_test_examples
   ```

2. Look at reference output:

   ```
   reference/dataframe_result.csv
   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())
```

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. Three tests should **fail**.
   `python unittest/test_using_referencetestcase.py; cd ..`
   or `(cd pytest; pytest)`

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

EXERCISE 2: DATAFRAME/CSV REFERENCE TESTS

II. RE-WRITE REFERENCE RESULTS; RE-RUN

7. 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 -s)`

8. 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!

AUTOMATIC TEST GENERATION

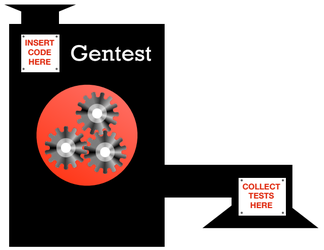
INPUTS → ANALYTICAL PROCESS → OUTPUTS

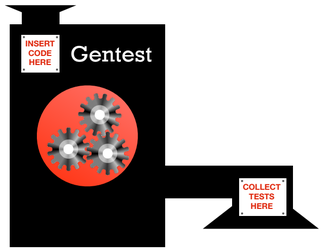
DATASETS, NUMBERS, GRAPHS, MODELS, DECISIONS ETC.

Record inputs

Capture as script

Record ("reference") outputs



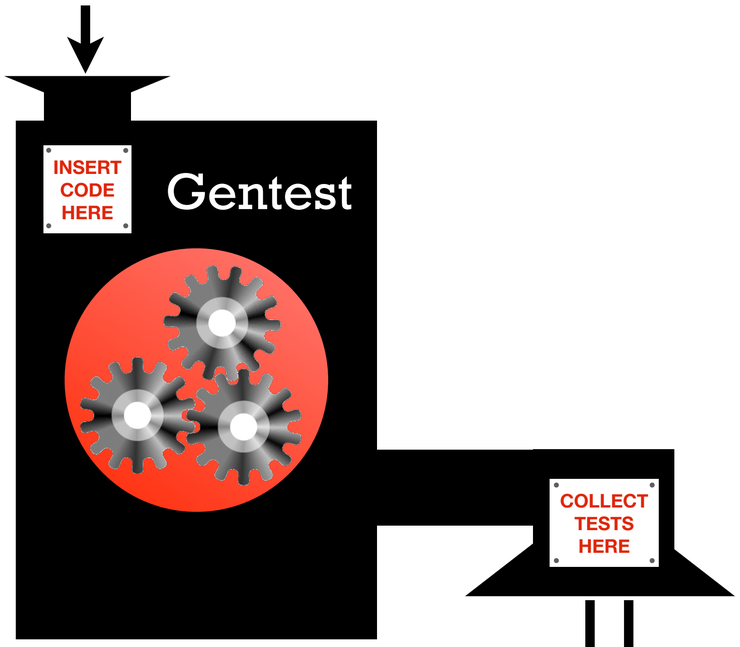


Develop a verification procedure (`diff`) and periodically rerun: do the same inputs (still) produce the same or equivalent outputs?

GENTEST

`tdda gentest "sh classify.sh"`

`sh classify.sh`



`test_sh_classify_sh.py` *test script*

`ref/sh_classify_sh` *reference outputs*

Really?

GENTEST

stdout

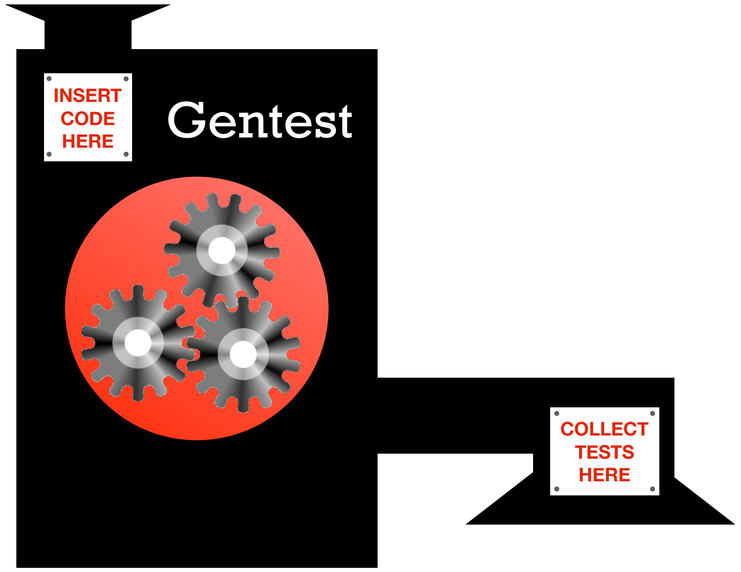
stderr

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, date, …)

differences between runs

"intelligent" decisions about how and what to test

Gentest is largely enabled by Rexpy!

ARTIFICIAL INTELLIGENCE

example1.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
echo "Have a number: $RANDOM" >> FILE1
```

TDDA Wizard

```
$ 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]:
Enter other files to be checked, one per line, then blank line:

Check stdout?: [y]:
Check stderr?: [y]:
Exit code should be zero?: [y]:
Number of times to run script?: [2]:
```

WIZARD OUTPUT

```
Running command 'sh example1.sh' to generate output (run 1 of 2).
Saved (non-empty) output to stdout to /Users/njr/tmp/pydata/ref/sh_example1_sh/STDOUT.
Saved (empty) output to stderr to /Users/njr/tmp/pydata/ref/sh_example1_sh/STDERR.
Copied $(pwd)/FILE1 to $(pwd)/ref/sh_example1_sh/FILE1
Running command 'sh example1.sh' to generate output (run 2 of 2).
Saved (non-empty) output to stdout to /Users/njr/tmp/pydata/ref/sh_example1_sh/2/STDOUT.
Saved (empty) output to stderr to /Users/njr/tmp/pydata/ref/sh_example1_sh/2/STDERR.
Copied $(pwd)/FILE1 to $(pwd)/ref/sh_example1_sh/2/FILE1
Test script written as /Users/njr/tmp/pydata/test_sh_example1_sh.py


Command execution took: 0.027s


SUMMARY:

Directory to run in:   /Users/njr/tmp/pydata
Shell command:         sh example1.sh
Test script generated: test_sh_example1_sh
Reference files:
    $(pwd)/FILE1
Check stdout:          yes (was 9 lines)
Check stderr:          yes (was empty)
Expected exit code:    0
```

GENERATED CODE

```
$ cat /Users/njr/tmp/pydata/test_sh_example2_sh.py
# -*- coding: utf-8 -*-

"""
test_sh_example1_sh.py: Automatically generated test code from tdda
gentest.

Generation command:

  tdda gentest 'sh example1.sh' ‘test_sh_example1_sh.py’ '.' STDOUT
STDERR
"""

from __future__ import absolute_import
from __future__ import print_function
from __future__ import division

import os
import sys

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

COMMAND = 'sh example1.sh'
CWD = os.path.abspath(os.path.dirname(__file__))
REFDIR = os.path.join(CWD, 'ref', 'sh_example1_sh')
```

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

```
class TestAnalysis(ReferenceTestCase):
    @classmethod
    def setUpClass(cls):
        (cls.output,
         cls.error,
         cls.exc,
         cls.exit_code,
         cls.duration) = exec_command(COMMAND, CWD)

    def test_no_exception(self):
        msg = 'No exception should be generated'
        self.assertEqual((str(self.exc), msg), ('None', msg))

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

    def test_stdout(self):
        substrings = [
            'godel.local',
            '9 Apr 2019 17:45:49',
        ]
        self.assertStringCorrect(self.output,
                                 os.path.join(REFDIR, 'STDOUT'),
                                 ignore_substrings=substrings)

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

    def test_FILE1(self):
        patterns = [
            r'^Have a number\: \d{4,5}$',
        ]
        self.assertFileCorrect(os.path.join(CWD, 'FILE1'),
                               os.path.join(REFDIR, 'FILE1'),
                               ignore_patterns=patterns)

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

SAVED FILES

```
$ ls ref/sh_example1_sh/
2  FILE1 STDERR  STDOUT

$ more ref/sh_example1_sh/FILE1
Let's have a file as well.
Have a number: 9310

$ more ref/sh_example1_sh/STDOUT
Hello, Edinburgh PyData!

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.

$ more ref/sh_example1_sh/STDERR
$
```

(This file is empty)

RUNNING THE TESTS

```
$ python test_sh_example1_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.

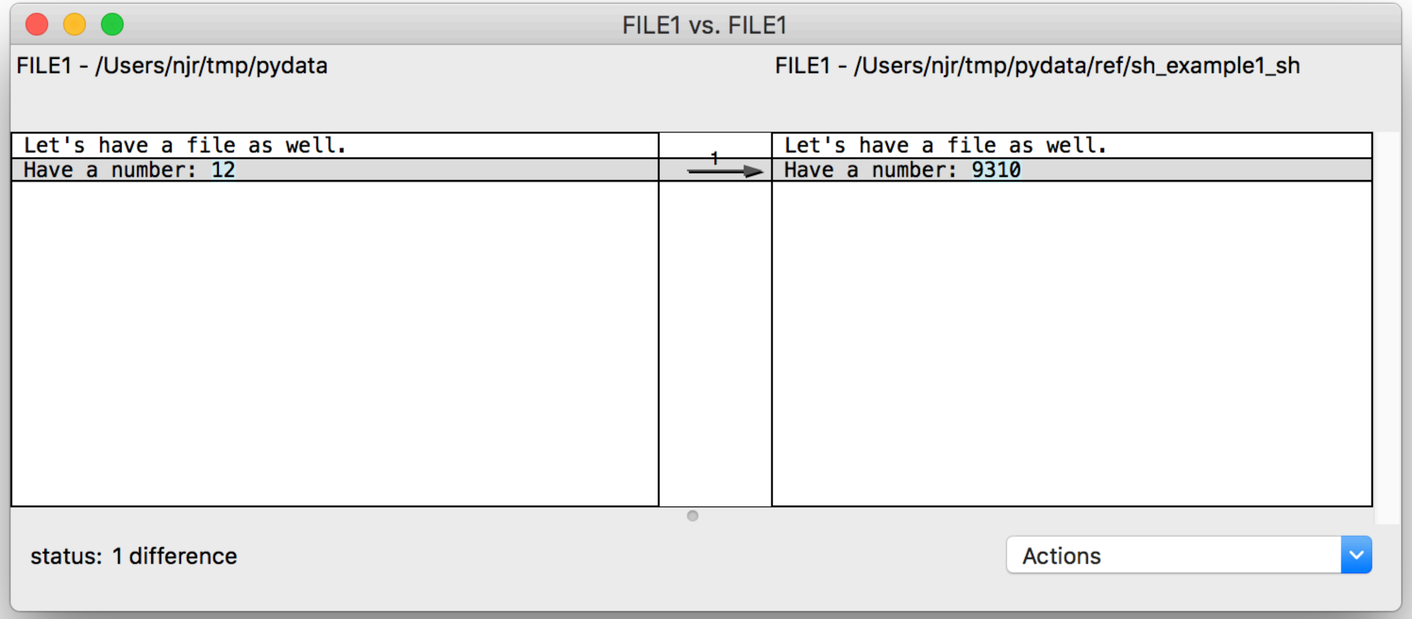
But after, I ran it another 33 times, and the last time it failed.

WHEN IT DOES FAIL

```
$ python test_sh_example1_sh.py
1 line is different, starting at line 2
Compare with:
    diff /Users/njr/tmp/pydata/FILE1 /Users/njr/tmp/pydata/ref/sh_example1_sh/FILE1

Note exclusions:
    ignore_patterns:
        ^Have a number\: \d{4,5}$
F....
======================================================================
FAIL: test_FILE1 (__main__.TestAnalysis)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_sh_example1_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/pydata/FILE1 /Users/njr/tmp/pydata/ref/sh_example1_sh/FILE1

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

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

FAILED (failures=1)
```

AND IF YOU RUN THE DIFF:

```
$ opendiff /Users/njr/tddacourse/FILE1 /Users/njr/tmp/tddacourse/ref/sh_example1_sh/FILE1
```



It is indeed that `\d{4,5}` *is too specific to capture all the variation.*
(In this case, it's just two digits!)
Easily fixed by hand.

ERRORS OF INTERPRETATION
(a.k.a. TYPE VI ERRORS)

Mars Climate Orbiter

NASA (SI)
Newton-seconds

v.

Lockheed Martin (FPS)
Pounds (force)
-seconds

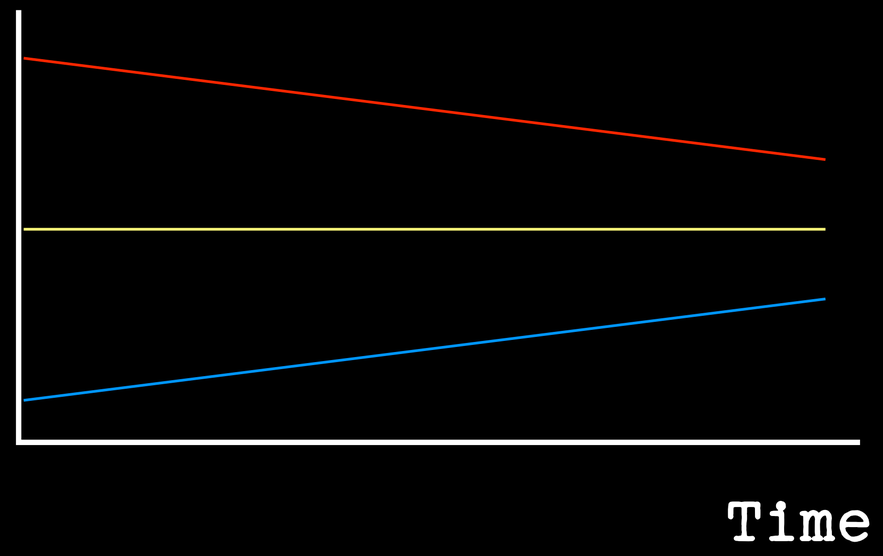
m	metres
m	miles
m	milli
m	million
M	Million (Mega)
M	Thousand
Mi	2^{20} (1,048,576)
MM	Million
k	Thousand
K	2^{10} (1024)
K	Kelvin

bn	10^9
bn	10^{12}
B	10^9; 10^{12}
90°	π/2
90°	45% alcohol
90°	nearly boiling (°C)
90°	wear suscreen (°F)
calories · Calories · kcal	
pt	20 fl oz · 16oz
pt	1/72.27” · 1/72”

Which class are we predicting?

99.9983%

Regression to the mean



Clustering Considered Harmful

http://www.scientificmarketer.com/2009/03/clustering-considered-harmful-i-outline.html

Unsupervised

Often unstable

People mostly look at cluster names

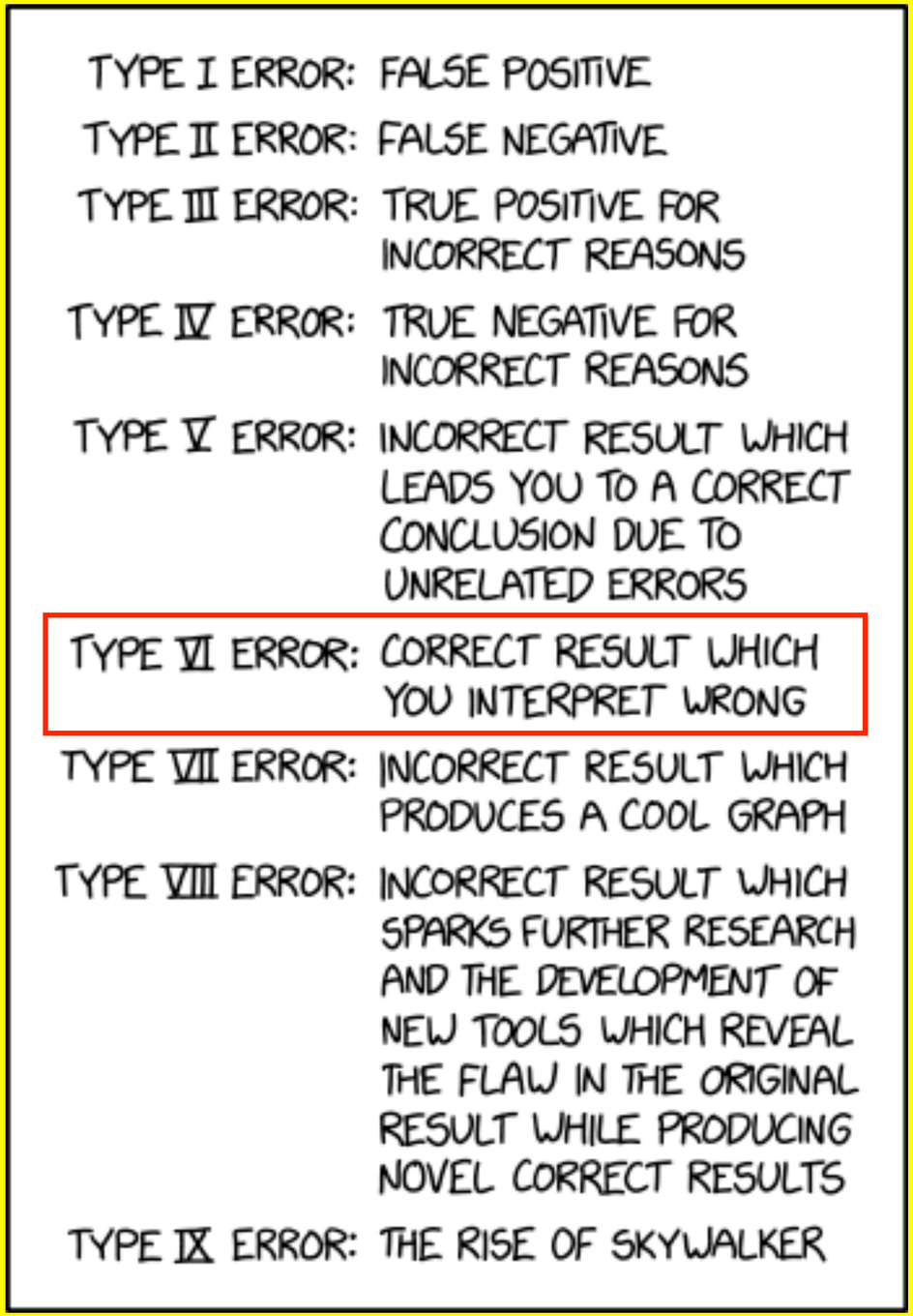
Distance function defines clusters

Curse of dimensionality

Non-commensurate variables

DATA SCIENTISTS: JUST SAY NO!

“Type I” & “Type II” Errors



https://xkcd.com/2303

01/02/12

Significant Figures & Spurious Precision

Table 2: World Water

	km^3	Per cent
Fresh Water		
Clouds	20,000	0
Continental Water	9,000,000	1
Ice	30,000,000	2
Salt Water		
Oceans	1,300,000,000	97
Total water	1,339,020,000	100

Source: Not: Sustainability: A Systems Approach. A M H Clayton & N J Radcliffe

Table 2: World water

	km³	*Per cent*
Fresh water		
Clouds	20,000	0
Continental water	9,000,000	1
Ice	30,000,000	2
Salt water		
Oceans	1,300,000,000	97
Total water	c.1,300,000,000	100

Source: Sustainability: A Systems Approach. A M H Clayton & N J Radcliffe

Percentage Changes

“Relative vs. Absolute risk”

Increase

1% → 1.1%

+10%

+0.1pp

JUNK CHARTS

Dual Axis

Non-uniform scale

False zero*

False zero colour*

Area, Volume

Inverted

Bezos charts

Unclear labels

Unclear tick labels

No units

Questionable lines of best fit

* When zero is meaningful

GRAPHING BEST PRACTICES

Annotate

Maximize Data Ink

Minimize chart junk

Direct labelling

Error bars

Pie charts are OK!

Units

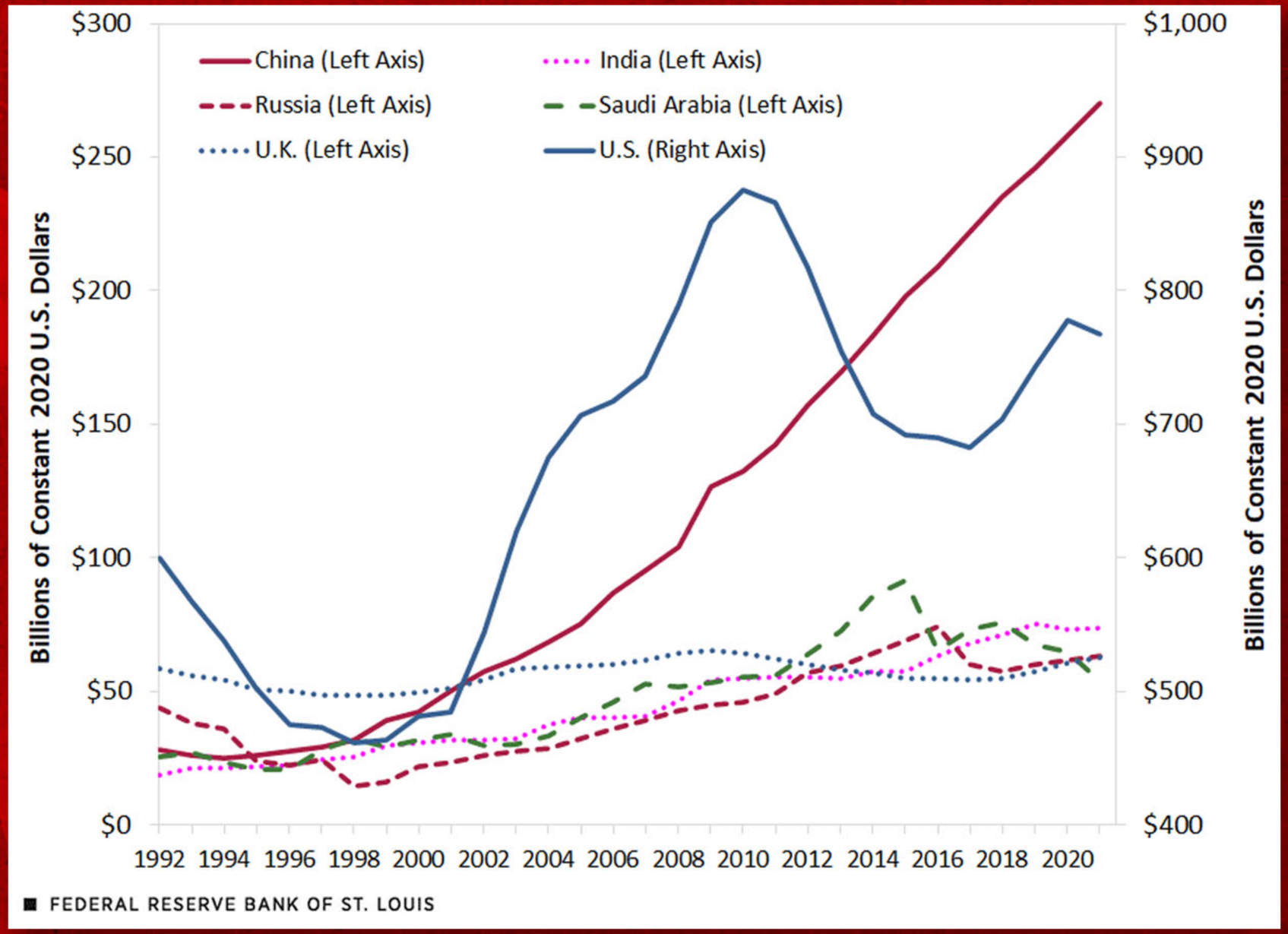
Zoomed sections for detail & context

Broken axes where required

```
>>> datetime.date(2101,12,2).strftime('%y/%d/%m')
'01/02/12'
```

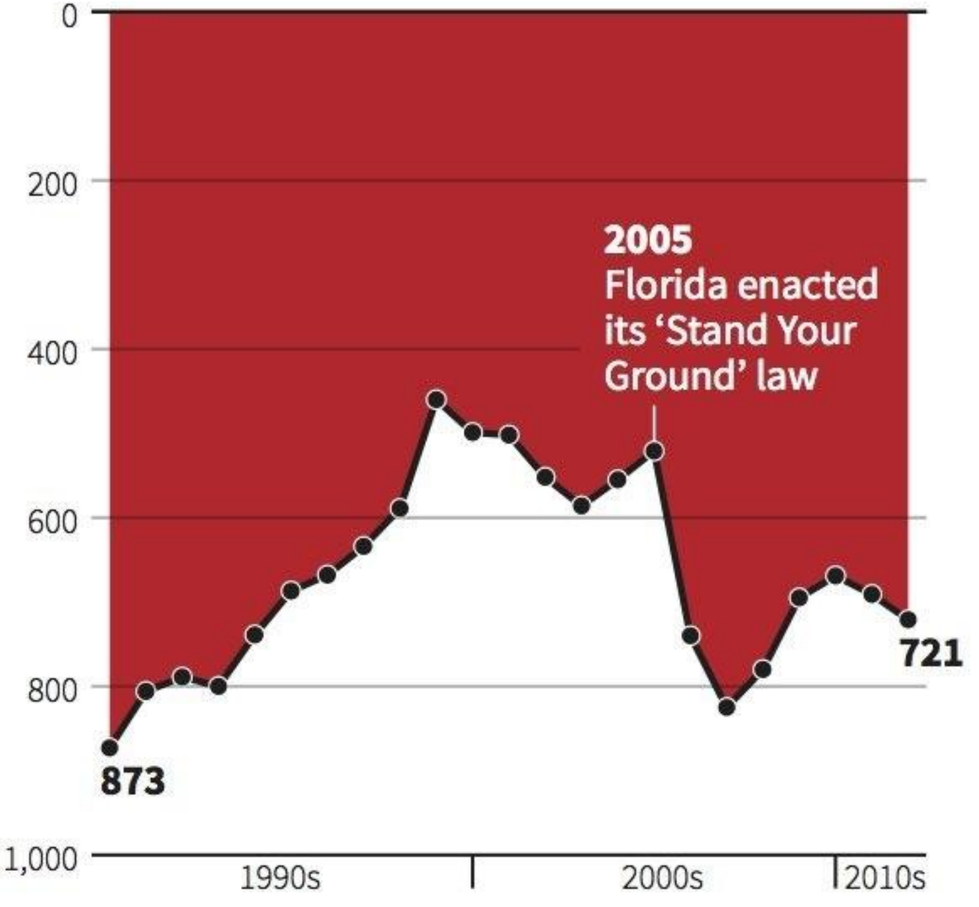
01/02/12

(2 Dec 2101)


China (Left Axis)
India (Left Axis)
Russia (Left Axis)
Saudi Arabia (Left Axis)
U.K. (Left Axis)
U.S. (Right Axis)
Billions of Constant 2020 U.S. Dollars
$300
$250
$200
$150
$100
$50
$0
$1,000
$900
$800
$700
$600
$500
$400
1992 1994 1996 1998 2000 2002 2004 2006 2008 2010 2012 2014 2016 2018 2020
FEDERAL RESERVE BANK OF ST. LOUIS

Gun deaths in Florida

Number of murders committed using firearms



Source: Florida Department of Law Enforcement

C. Chan 16/02/2014

REUTERS

https://stochasticsolutions.com/pdf/tdda-london-2024.pdf

http://stochasticsolutions.com

http://tdda.info

https://github.com/tdda

njr@stochasticsolutions.com

http://linkedin.com/in/njradcliffe

#tdda*

*tweet (DM) us email address for invitation
Or email me.

@njr@zirk.us @tdda@mathstodon.xyz

Stochastic Solutions