Datajudge

22/08/17

Kevin Klein

@kevkle

Me

- Computer Science at ETH Zurich and University of Washington
- Broad interest in Discrete Maths, Machine Learning and Software Engineering
- Currently Data Scientist/Machine Learning Engineer at QuantCo

quantco

- Data Science solutions
- Mostly fraud detection, pricing and demand forecasting
- join@quantco.com

The Data

channel	watch time	stream time	peak viewers	average viewers	followers	followers gained	
xQcOW	6196161750	215250	222720	27716	3246298	1734810	9
summit1g	6091677300	211845	310998	25610	5310163	1370184	8
Gaules	5644590915	515280	387315	10976	1767635	1023779	10
ESL_CSGO	3970318140	517740	300575	7714	3944850	703986	10
Tfue	3671000070	123660	285644	29602	8938903	2068424	7
...	...	...	...	...	...	...	

~1000 rows, 11 columns (7 numeric, 4 categorical)

- Assumptions
    - We rely on this data.
        - Could be a ML systems recommending whom to sponsor.
        - Could be a deterministic system at Twitch, computing earnings for creators.
    - We receive monthly data deliveries.
    - Last month's delivery has been vetted manually.
- The task
    - We just received a new data delivery and want to use it for production purposes.
- The solution
    - We just use the new data as is. Since the old version of the data has been vetted, the new version of the data must still be valid!
    - Right!
    - ... Right?

THIS IS FINE.

Using our domain knowledge to validate the new data

- Expectation: The `language` column should only contain values containing a non-empty sequence of the latin alphabet.
- Code:

```python
from datajudge import WithinRequirement

# Defining a data source.
within_requirement = WithinRequirement.from_table(
    table_name="twitch_v2",
    schema_name=schema_name,
    db_name=db_name,
)

# Defining a constraint on the data source.
within_requirement.add_varchar_regex_constraint(
    column="language",
    regex="^[a-zA-Z]+$",
)
```

Using old data to validate the new data

- Expectation: The `language` column should contain the same unique values we already encountered before.
- Code:

```python
from datajudge import BetweenRequirement

between_requirement_version = BetweenRequirement.from_tables(
    db_name1=db_name,
    db_name2=db_name,
    schema_name1=schema_name,
    schema_name2=schema_name,
    table_name1="twitch_v1",
    table_name2="twitch_v2",
)

between_requirement_version.add_uniques_equality_constraint(
    columns1=["language"],
    columns2=["language"],
)
```

Using old data to validate the new data

- Expectation: The column structure should be equal for both versions.
- Code:

```
between_requirement_version.add_column_subset_constraint()
between_requirement_version.add_column_superset_constraint()
```

Using old data to validate the new data

- Expectation: The static features of the rows in the old data should be present and unchanged in the new data.
- Code:

```
columns = ["channel", "partnered", "mature"]
between_requirement_version.add_row_subset_constraint(
    columns1=columns, columns2=columns, constant_max_missing_fraction=0
)
```

- Note: `constant_max_missing_fraction`, a tolerance parameter

Using new data to validate new data

- Expectation: The distribution of `average_viewers` should follow the same underlying data generating process in both versions.
- Code: 2-sample Kolmogorov Smirnov hypothesis test

```
between_requirement_version.add_ks_2sample_constraint(
    column1="average_viewers",
    column2="average_viewers",
    significance_level=0.05,
)
```

Using new data to validate new data

- Expectation: `average_viewers` of mature channels shouldn't deviate too much from overall mean
- Code:

```
between_requirement_columns = BetweenRequirement.from_tables(
    db_name1=db_name,
    db_name2=db_name,
    schema_name1=schema_name,
    schema_name2=schema_name,
    table_name1="twitch_v2",
    table_name2="twitch_v2",
)

between_requirement_columns.add_numeric_mean_constraint(
    column1="average_viewers",
    column2="average_viewers",
    condition1=None,
    condition2=Condition(raw_string="mature IS TRUE"),
    max_absolute_deviation=0.1,
)
```

Execution of tests

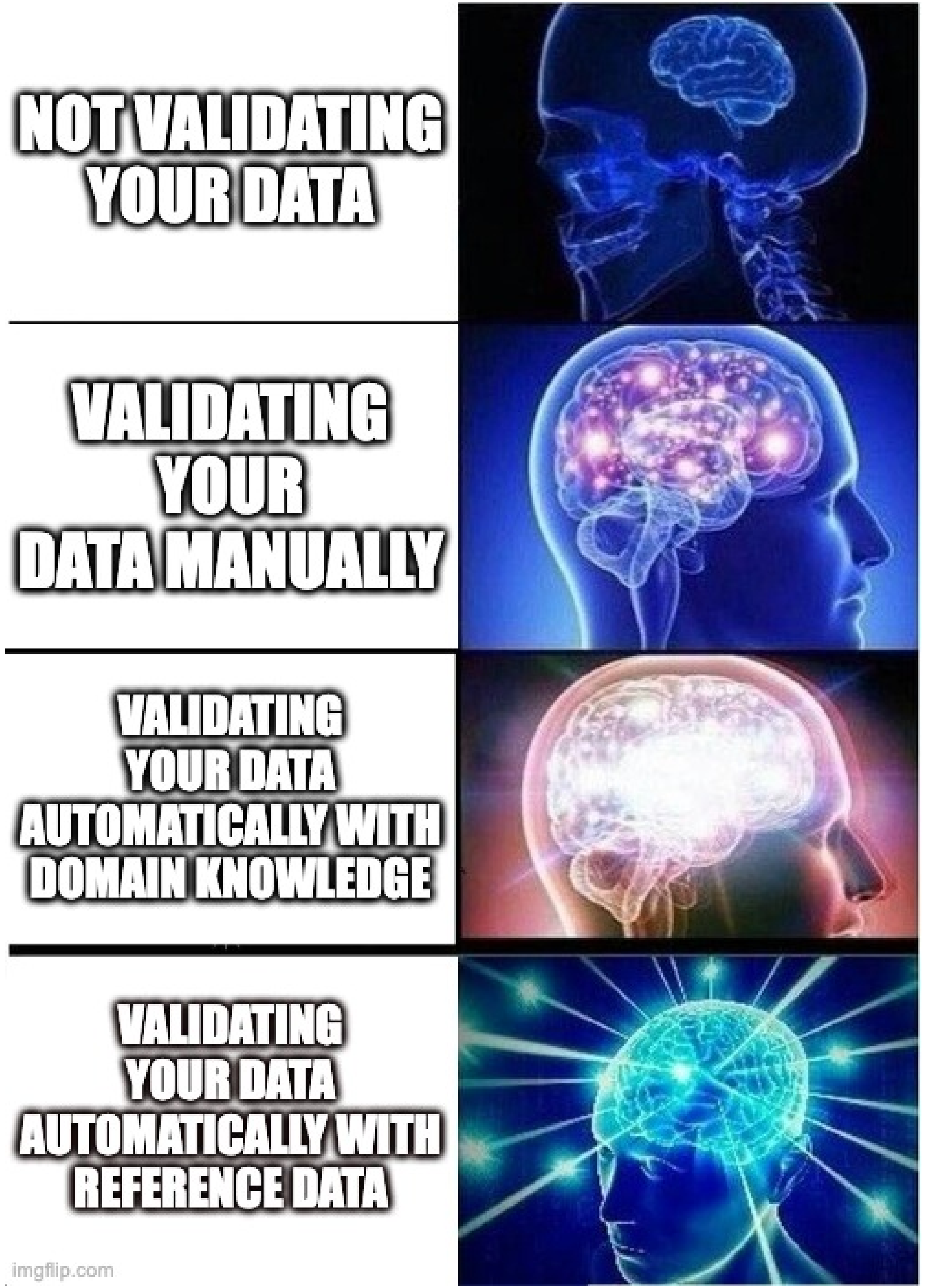
- All of the previous code simply goes into a python file.
- Add few lines of boilerplate code:

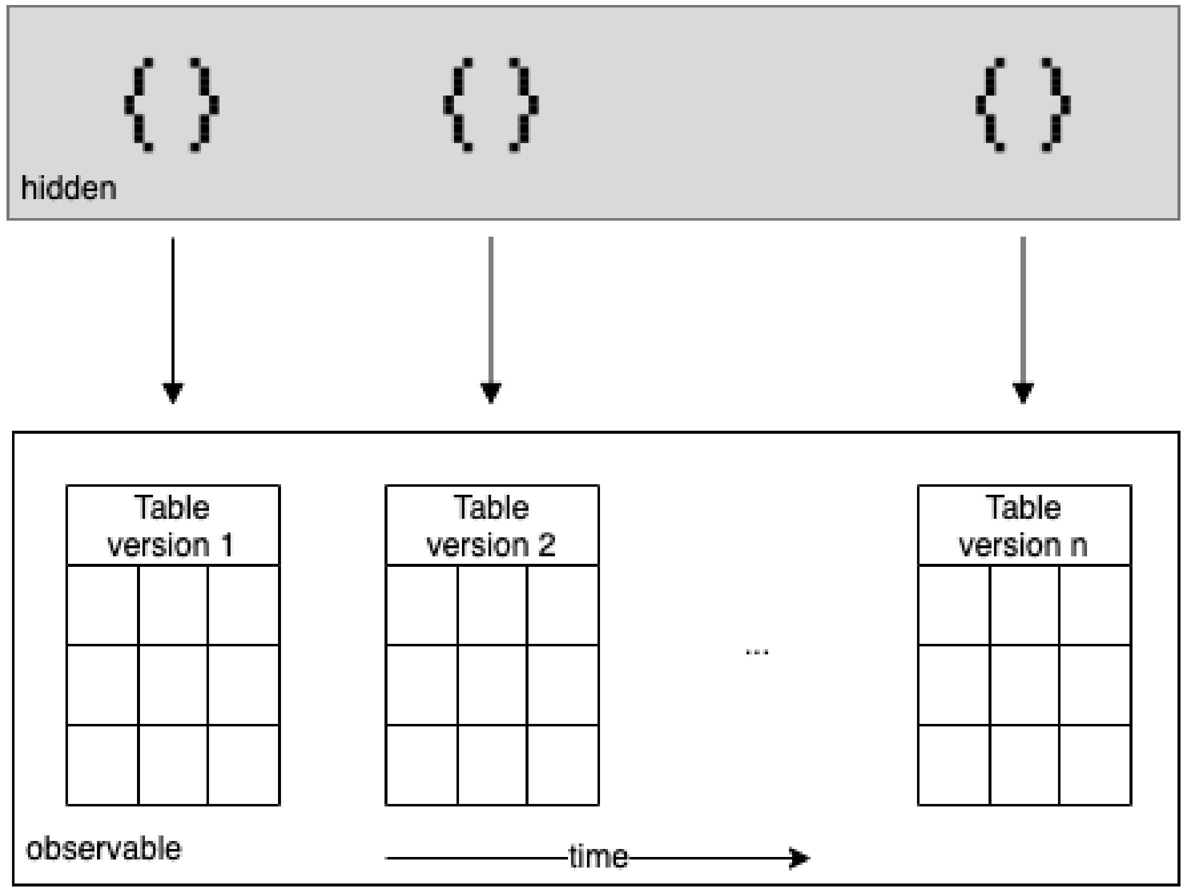
```
    within_requirement,
    between_requirement_version,
    between_requirement_columns,
  ]
  test_func = collect_data_tests(requirements)
```

- Run it with pytest: `pytest specification.py`

```
_________________ test_func[UniquesEquality::public.twitch_v1 | public.twitch_v2] __________________

constraint = <datajudge.constraints.uniques.UniquesEquality object at 0x108087e20>
datajudge_engine = Engine(postgresql://datajudge:***@localhost:5432/datajudge)

@pytest.mark.parametrize(
    "constraint", all_constraints, ids=Constraint.get_description
)
def test_constraint(constraint, datajudge_engine):
test_result = constraint.test(datajudge_engine)
>       assert test_result.outcome, test_result.failure_message
E       AssertionError: tempdb.public.twitch_v1's column(s) 'language' doesn't have
        the element(s) '{'Sw3d1zh'}' when compared with the reference values.

/usr/local/Caskroom/.../lib/python3.10/site-packages/datajudge/pytest_integration.py:25:
AssertionError
```

```
===================================== short test summary info ======================================
FAILED twitch_specification.py::test_func[VarCharRegex::tempdb.public.twitch_v2] - AssertionError...
FAILED twitch_specification.py::test_func[KolmogorovSmirnov2Sample::public.twitch_v1 | public.twitch_v2]
FAILED twitch_specification.py::test_func[UniquesEquality::public.twitch_v1 | public.twitch_v2]
FAILED twitch_specification.py::test_func[NumericMean::public.twitch_v2 | public.twitch_v2] - Ass...
=================================== 4 failed, 4 passed in 1.80s ====================================
```


NOT VALIDATING YOUR DATA
VALIDATING YOUR DATA MANUALLY
VALIDATING YOUR DATA AUTOMATICALLY WITH DOMAIN KNOWLEDGE
VALIDATING YOUR DATA AUTOMATICALLY WITH REFERENCE DATA
imgflip.com

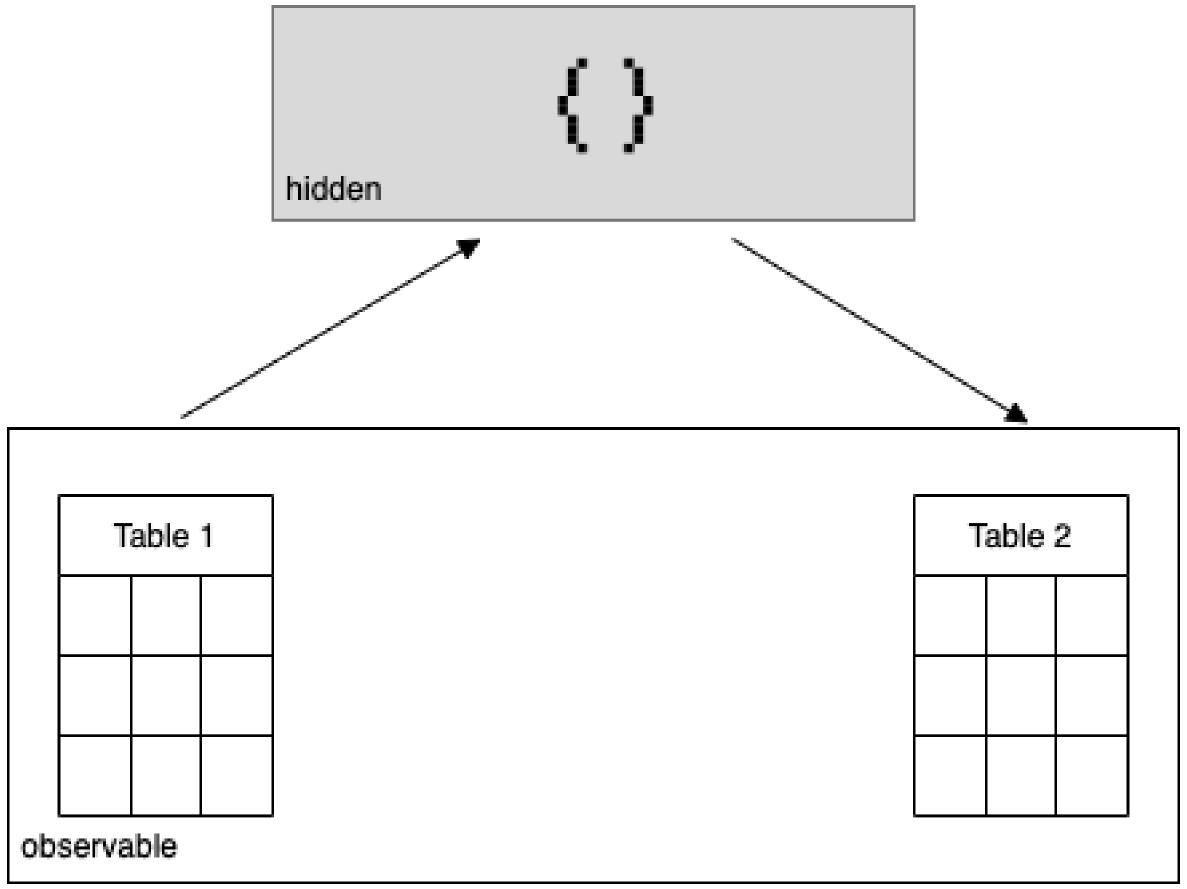


Comparisons between data sources: Across time

- Tables with equal structure and semantics.
- Tables from different points in time.
- Compare content with respect to expected change.
- Relevant use case: Obfuscated data-generating code between pipeline runs.

Comparisons between data sources: Across space

- Tables of different structure.
- Tables could have an input-output relationship.
- Compare tables with respect to invariants.
  - E.g. Min in summary stats table should be equal to min in unit table.
- Relevant use case: Obfuscated data-generating code between steps of a pipeline.



Datajudge: Good to know

- Only relational databases are supported.
  - Currently test against Postgres, Snowflake and Mssql.
- Heavy lifting happens in database, only test results are fetched to memory.
  - Can be *very* advantageous in terms of memory consumption and runtime.
- Datajudge generates sql queries from high-level API.
  - It relies on the SQLAlchemy Language Expression API to be dialect-agnostic.
  - Generated queries are logged and can be used for data debugging.
- Many more constraints exist.
  - E.g.: constraints for date columns, useful to validate historization of database

Questions?

How we use Datajudge at QuantCo

- Generate html test reports with a pytest plugin.
    - `pytest specification.py --html=report.html`
    - Eases collaboration and archiving.
- Parametrize data sources.
    - `pytest specification.py --new_db new_db --old_db old_db`
- Subselect tests
    - `pytest specification.py -k varchar_constraint`
- Integration in CI.

Why not just use database constraints?

- Not available across dialects/dbms.
- No error tolerance.
- No conditioning.
- No/hardly any comparisons between data sources.
- Only operate on a row-level.
    - E.g. constraining the mean of a column is not possible.
- Different workflow: pre-transaction vs. post-transaction.

Why not use Great Expectations?

	Datajudge	**Great Expectations**
Comparisons between data	First class citizen	Tricky :/
Data sources	Relational databases	Databases, files, in-memory data
Adoption	Fast startup time	Complex ecosystem
Objective	Testing	Exploration, monitoring, testing
Sql logic	By framework	By user/by framework