DATA SCIENCE TESTING

Eric J. Ma

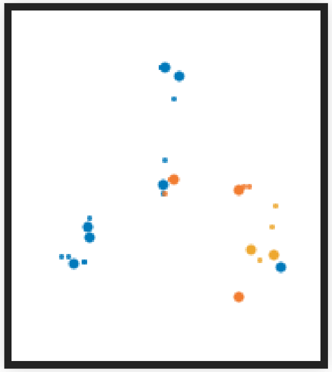
PyData Ann Arbor

2020-01-15

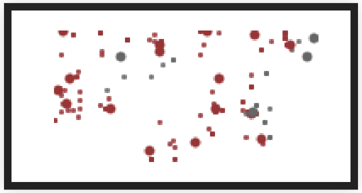
GET THE SLIDES!

https://ericmjl.github.io/testing-for-data-scientists/

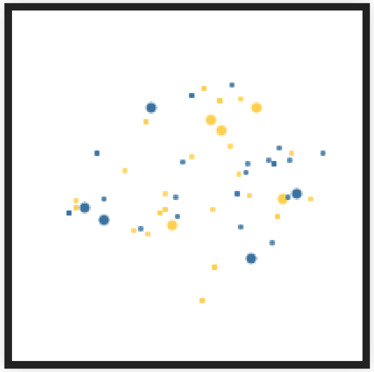
ABOUT ME



Investigator, Scientific Data Analysis



ScD, Biological Engineering, 2017



Bayesian Stats, ML, Network Science

HOW MANY OF YOU ARE DATA SCIENTISTS?

Keep your hands up...

HOW MANY OF YOU WRITE CODE IN A PROGRAMMING LANGUAGE >50% OF YOUR TIME?

Keep your hands up...

HOW MANY OF YOU WRITE TESTS FOR YOUR CODE?

HOW MANY OF YOU WRITE TESTS FOR YOUR CODE?

OK, please put your hands down.

FOR THOSE WHO DON'T WRITE TESTS, WHAT ARE YOUR REASONS?

FOR THOSE WHO DO WRITE TESTS, WHAT ARE YOUR REASONS?

TODAY'S MAIN POINT

If you build data products as a data scientist, investing time in writing tests today will:

TODAY'S MAIN POINT

If you build data products as a data scientist, investing time in writing tests today will:

- save you time later on,

TODAY'S MAIN POINT

If you build data products as a data scientist, investing time in writing tests today will:

- save you time later on,
- thus decreasing team frustration,

TODAY'S MAIN POINT

If you build data products as a data scientist, investing time in writing tests today will:

- save you time later on,
- thus decreasing team frustration,
- and increasing team productivity.

TODAY'S MAIN POINT

If you build data products as a data scientist, investing time in writing tests today will:

- save you time later on,
- thus decreasing team frustration,
- and increasing team productivity.

The same holds for basic software engineering skills in general.

TWO STORIES

TWO STORIES

1. How automated testing revealed weakspots

TWO STORIES

1. How automated testing revealed weakspots
2. How testing accelerated our data analysis workflow

HOW AUTOMATED TESTING REVEALED WEAKSPOTS

HIV DRUG RESISTANCE PREDICTION

HIV DRUG RESISTANCE PREDICTION

`MAKVLENPALEO` $\rightarrow$ 1.34

HIV DRUG RESISTANCE PREDICTION

`` `MAKVLENPALEO` `` $\rightarrow$ 1.34

`` `MADVLENPALEO` `` $\rightarrow$ 150

`MADVLENPALRO` $\rightarrow$ 39.3

MODEL TRAINING FUNCTIONS

```python
# utils.py

def read_protein(filename):
    sequence = ... # stuff happens
    return sequence

# Returns array 5 times the length of sequence.
def featurize(sequence):
    features = ... # stuff happens
    return features

# Return model predictions.
def predict(features):
    model = ... # load scikit-learn model
    return model.predict(features)
```

MODEL TRAINING FUNCTIONS

```python
# utils.py

def read_protein(filename):
    sequence = ... # stuff happens
    return sequence

# Returns array 5 times the length of sequence.
def featurize(sequence):
    features = ... # stuff happens
    return features

# Return model predictions.
def predict(features):
    model = ... # load scikit-learn model
    return model.predict(features)
```

What have we assumed here that probably ought to be tested?

LET'S WRITE A TEST FOR FEATURIZE!

```
from utils import featurize

def test_featurize():
    sequence = "MKALVIELQDPG..."  # something 99 amino acids l
    feats = featurize(sequence)

    assert feats.shape[0] == 1
    assert feats.shape[1] == len(sequence) * 5
```

LET'S WRITE A TEST FOR PREDICT!

```
from utils import predict

# An integration test for the predict function.
def test_predict():
    sequence = "MKALVIELQDPG..."  # something 99 amino acids l
    feats = featurize(sequence)
    preds = predict(feats)
```

Cool! Are we done?

CLEARLY NOT!

One **huge** assumption we made here was about the input string.

CLEARLY NOT!

One **huge** assumption we made here was about the input string.

Let's revisit that test.

If a user inputs a string that is not 99 letters long, the program should crash.

If a user inputs a string with invalid characters, the program should crash.

Let's make the code more robust.

Let's make the code more robust.

```
acceptable_letters = set('ACDEFGHIJKLMNPQRSTVWXY')
def featurize(sequence):
    if not len(sequence) == 99:
        raise ValueError("put informative error here.")
    if not set(sequence).issubset(acceptable_letters):
        raise Exception("put informative error here.")
    features = ... # stuff happens
    return features
```

Let's make the test more robust.

```python
from hypothesis import strategies as st, given
# other imports here...
acceptable_letters = set('ACDEFGHIJKLMNPQRSTVWXY')

@given(
    sequence=st.text(
        alphabet=acceptable_letters,
        min_size=0,
        max_size=200)
    )
)
def test_featurize(sequence):
    if len(sequence) != 99:
        with pytest.raises(ValueError):
            feats = featurize(sequence)
```

Let's make the test more robust.

```
from hypothesis import strategies as st, given
# other imports here...
acceptable_letters = set('ACDEFGHIJKLMNPQRSTVWXY')

@given(
    sequence=st.text(
        alphabet=acceptable_letters,
        min_size=0,
        max_size=200)
    )
)
def test_featurize(sequence):
    if len(sequence) != 99:
        with pytest.raises(ValueError):
            feats = featurize(sequence)
```

Doing the same for invalid characters is an exercise left for the reader (tm).

WE ARE IN A MUCH BETTER POSITION

WE ARE IN A MUCH BETTER POSITION

- Function defends against unexpected inputs.

WE ARE IN A MUCH BETTER POSITION

- Function defends against unexpected inputs.
- Tests help us catch breaking changes.

WE ARE IN A MUCH BETTER POSITION

- Function defends against unexpected inputs.
- Tests help us catch breaking changes.
- Your engineers are going to thank you.

IN PRACTICE...

We caught this issue by using `Hypothesis`, and worked backwards.

IN PRACTICE...

We caught this issue by using `Hypothesis`, and worked backwards.

Writing tests helps you catch bugs.

HOW TESTING ACCELERATED OUR DATA ANALYSIS WORKFLOW

PROTEIN ENGINEERING PLATFORM

PROTEIN ENGINEERING PLATFORM

- Large but simple codebase.

PROTEIN ENGINEERING PLATFORM

- Large but simple codebase.
- Many independent utilities with some function sharing.

PROTEIN ENGINEERING PLATFORM

- Large but simple codebase.
- Many independent utilities with some function sharing.

Focus on data access.

OUR DATA: COMPLEX, EVOLVING DATA MANAGEMENT REQUIREMENTS

OUR DATA: COMPLEX, EVOLVING DATA MANAGEMENT REQUIREMENTS

- As platform gets built out, data requirements change.

OUR DATA: COMPLEX, EVOLVING DATA MANAGEMENT REQUIREMENTS

- As platform gets built out, data requirements change.
- New enzyme = similar schema.

OUR DATA: COMPLEX, EVOLVING DATA MANAGEMENT REQUIREMENTS

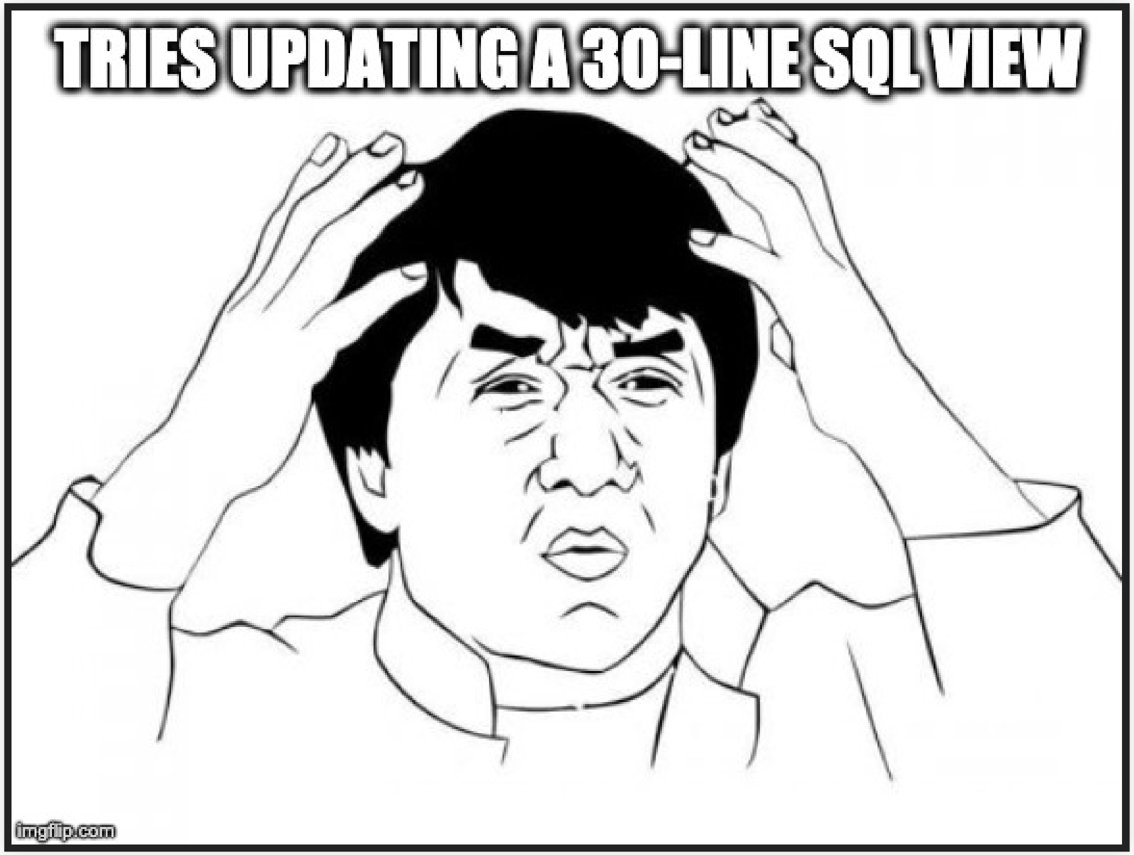
- As platform gets built out, data requirements change.
- New enzyme = similar schema.
- The database is as far normalized as possible

OUR DATA: COMPLEX, EVOLVING DATA MANAGEMENT REQUIREMENTS

- As platform gets built out, data requirements change.
- New enzyme = similar schema.
- The database is as far normalized as possible

Lots of joins needed to make get data in human-readable form.

WE USED TO HAVE TO UPDATE POSTGRES VIEWS...



...UNTIL WE SWITCHED TO CACHING OUR VIEWS AS DATAFRAMES.

THE SCHEMA IS THE DATA'S API!

Actually, data access APIs might sometimes be better...

THE SCHEMA IS THE DATA'S API!

Actually, data access APIs might sometimes be better...

...but that's another story

IF THE SCHEMA CHANGES EVEN SLIGHTLY...

...WE WANT TO KNOW ASAP.

TESTS FOR DATA

TESTS FOR DATA

- Column names

TESTS FOR DATA

- Column names
- Column data types

TESTS FOR DATA

- Column names
- Column data types
- Nullity

TESTS FOR DATA

- Column names
- Column data types
- Nullity
- Bounds

TESTS FOR DATA

- Column names
- Column data types
- Nullity
- Bounds
- ...more?

EXAMPLE OF TESTING DATA

```
def test_query_function():
    data = query_function()

    # Column tests:
    expected_columns = [...]
    assert set(expected_columns) == set(data.columns)

    # Null checks: this column __must__ be fully populated
    assert pd.isnull(df[column_name]).sum() == 0
```

TEST-ACCELERATED WORKFLOW

TEST-ACCELERATED WORKFLOW

Because of data caching and data testing...

TEST-ACCELERATED WORKFLOW

Because of data caching and data testing...

...a non-routine data query that may have taken half a day to get right...

TEST-ACCELERATED WORKFLOW

Because of data caching and data testing...

...a non-routine data query that may have taken half a day to get right...

...instead took 10 minutes to finish and be confident in.

(repeat this N times for new analyses)

HOW TO BUILD A REGULAR PRACTICE OF TESTING IN DATA SCIENCE

BE #UNBOSSED

BE #UNBOSSED

- Make opinionated CI configuration templates.

BE #UNBOSSED

- Make opinionated CI configuration templates.
- Push DevOps team for guidance.

#AUTOMATE EVERYTHING

Nobody argues against convenience!

#AUTOMATE EVERYTHING

- Set up a CI system that mandates checks on code.

Nobody argues against convenience!

#AUTOMATE EVERYTHING

- Set up a CI system that mandates checks on code.
- Don't allow code to be merged without review passing tests.

Nobody argues against convenience!

BUILD #CREDIBILITY

BUILD #CREDIBILITY

- Provide testimonials to your DevOps team (where applicable).

BUILD #CREDIBILITY

- Provide testimonials to your DevOps team (where applicable).
- Deliver talks about testing!

CONCLUSIONS

- If you depend on code, write tests for it.
- If you depend on data, write tests for it.

If you depend on it, write a test for it.

RESOURCES

- Essays on Data Science
- Personal Blog
- Data Science Manifesto
- Great Expectations
- Hypothesis
- `pytest`

HAVE FUN TESTING!

GRAB THE PDF HERE