

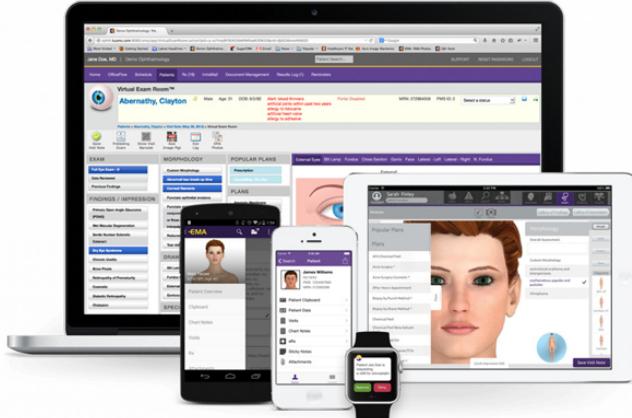


# Dynamic Healthcare Dataset Generation, Curation & Quality with PySpark

Aaron Richter, Modernizing Medicine®

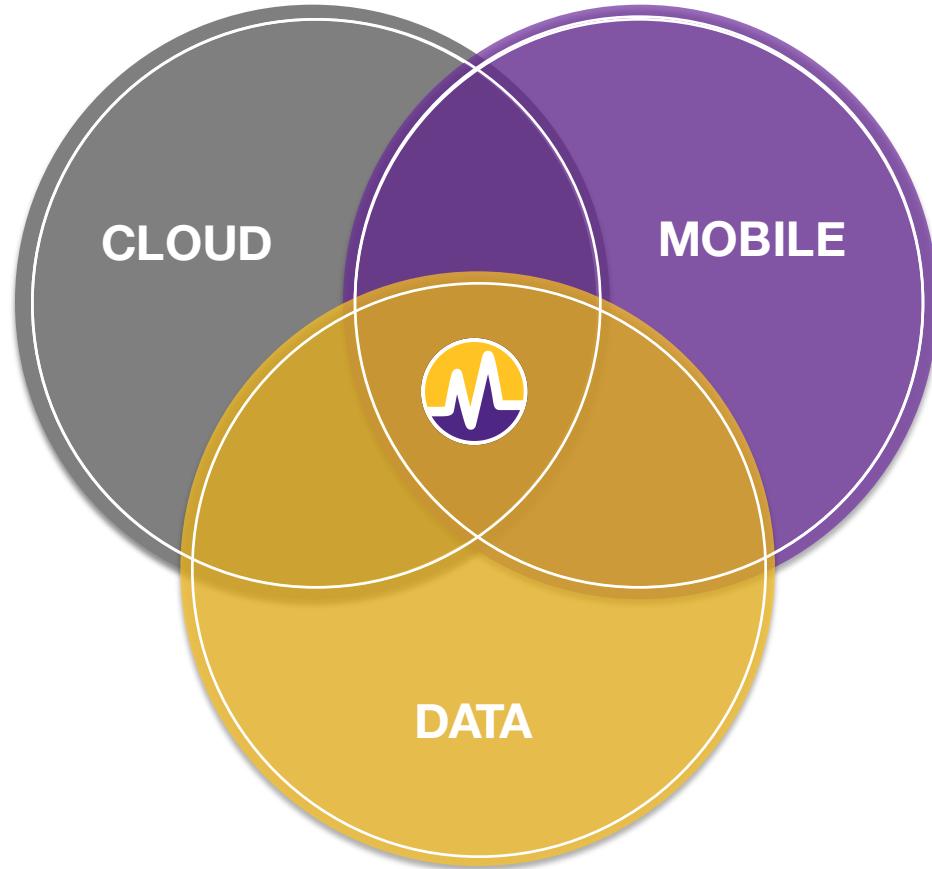
#Py3SAIS  
@rikturr

# MODERNIZING MEDICINE



- Electronic Health Record (EHR) system, EMA™
- Practice Management (PM)
- Revenue Cycle Management (RCM)
- Analytics
- Specialty-specific
- 14,000+ providers across the US

Transforming how healthcare information is created, consumed and utilized to increase practice efficiency and improve patient outcomes.



# DATA @ MODMED

BIG DATA? BIG DATA! BIG DATA!?



# DATA @ MODMED



# DATA @ MODMED

## What we do

- Data Warehouse
- De-identified Data Warehouse
- Data Extracts
  - Practice data
  - Population health research
  - Business intelligence
  - Data Science

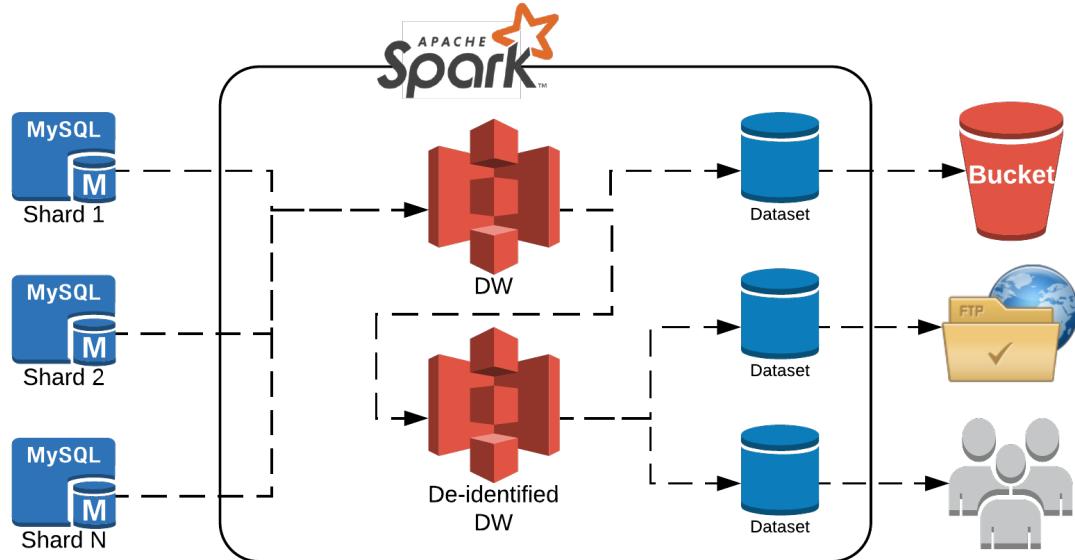
## How we do it



# DATA @ MODMED

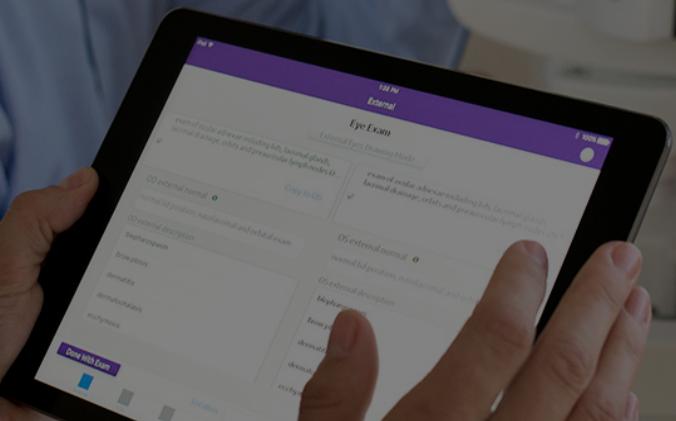
## Data Architecture

- Multi-tenant database shards
- Shard (MySQL)
  - 800+ tables
  - 600GB+
  - 2 largest tables 100GB each
- S3 data lake
  - Extract out of MySQL
  - Save to parquet

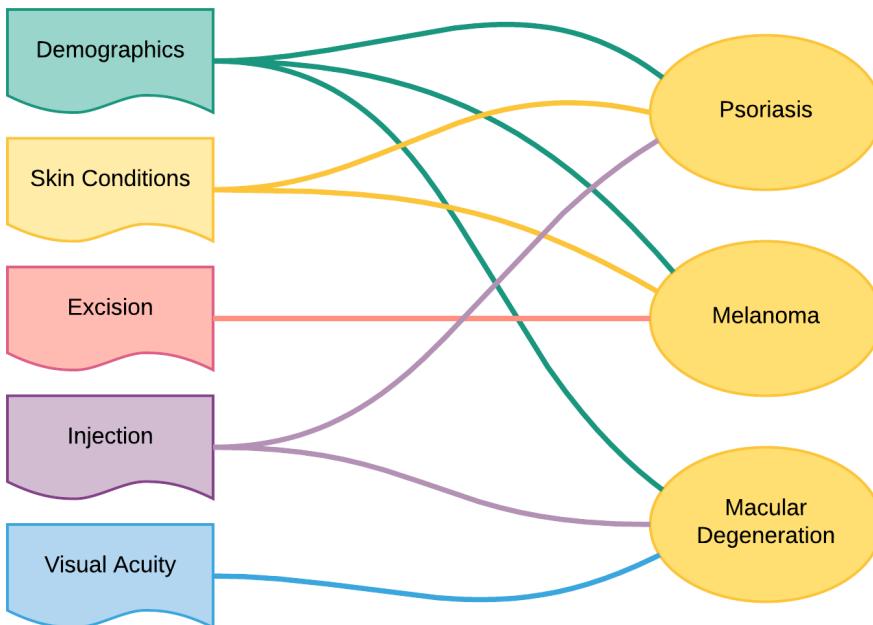


# DATASET GENERATION

THE SECRET SAUCE



# DYNAMIC DATASETS



# WHAT DID WE DO BEFORE?

- Impala SQL statements
  - Impala dies -> try Hive
- XML definitions of datasets
- SQL file for each table
- String replacements to inject variables
- Java program



# WHY DO WE USE PYTHON?

## Team Considerations

- Everyone already knows Python
- Too stubborn to learn Scala
- Not fans of JVM, JARs, Maven, etc.
- Infrastructure consistency (i.e. Airflow)

## Python ecosystem

- PyCharm
- Pandas, numpy, etc.
- Read/write JSON
- Write Excel



#Py3SAIS

# PYTHON, FTW!

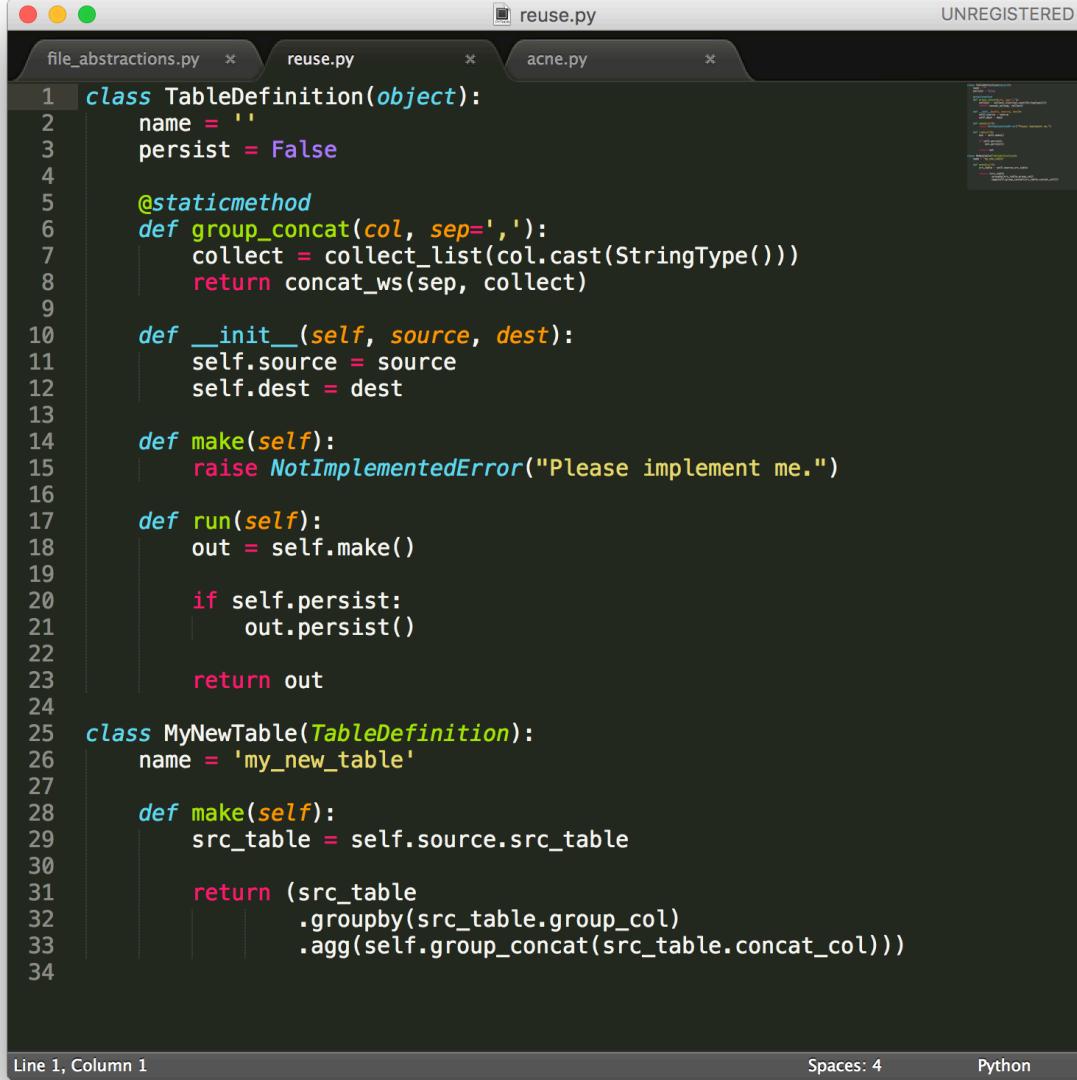
- Define helper methods for each type of source
- File presence, saving, loading
- Unique methods for sources
  - SQL indexes
  - Text file delimiters, separator, quote char
- Dot/dictionary access to tables

```
file_abstractions.py * reuse.py * file_abstractions.py
1 import os
2
3
4 class DataDirectory(object):
5     def __init__(self, spark, path):
6         self.spark = spark
7         self.path = path
8
9     def __getattr__(self, item):
10        return self.__getitem__(item)
11
12     def __getitem__(self, item):
13        return self.load_table(item)
14
15     def load_table(self, name):
16         table_path = os.path.join(self.path, name)
17         return self.spark.read.parquet(table_path)
18
19
20 class HiveDatabase(DataDirectory):
21     def load_table(self, name):
22         db_table = '{}.{}'.format(self.db, name)
23         return self.spark.read.table(db_table)
24
25
26 class TextFiles(DataDirectory):
27     def load_table(self, name):
28         table_path = os.path.join(self.path, name)
29         return self.spark.read.csv(table_path, sep=self.sep)
```

Line 1, Column 1      Spaces: 4      Python

# PYTHON, FTW!

- DataFrame/Column functions
- Python UDFs
- OOP
- Runtime arguments



```
file_abstractions.py * reuse.py * acne.py *
1  class TableDefinition(object):
2      name = ''
3      persist = False
4
5      @staticmethod
6      def group_concat(col, sep=','):
7          collect = collect_list(col.cast(StringType()))
8          return concat_ws(sep, collect)
9
10     def __init__(self, source, dest):
11         self.source = source
12         self.dest = dest
13
14     def make(self):
15         raise NotImplementedError("Please implement me.")
16
17     def run(self):
18         out = self.make()
19
20         if self.persist:
21             out.persist()
22
23         return out
24
25 class MyNewTable(TableDefinition):
26     name = 'my_new_table'
27
28     def make(self):
29         src_table = self.source.src_table
30
31         return (src_table
32                .groupby(src_table.group_col)
33                .agg(self.group_concat(src_table.concat_col)))
```

# PYTHON, FTW!

- Module per dataset
- Enumerate tables
- Parameters/inclusion criteria

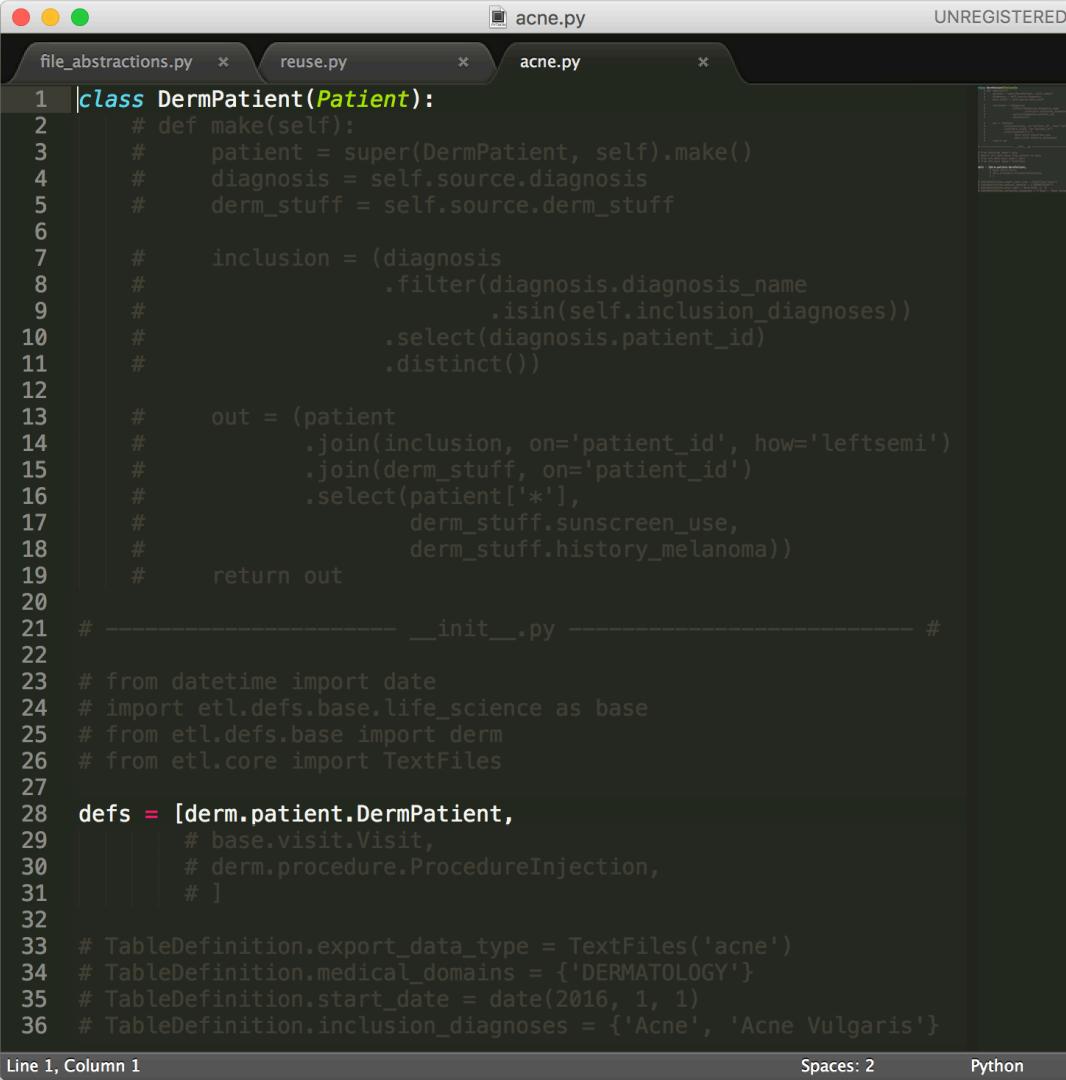
A screenshot of a Mac OS X desktop environment. At the top, there's a menu bar with 'UNREGISTERED'. Below it, a terminal window shows three tabs: 'file\_abstractions.py', 'reuse.py', and 'acne.py'. The 'acne.py' tab is active and displays the following Python code:

```
1  class DermPatient(Patient):
2      def make(self):
3          patient = super(DermPatient, self).make()
4          diagnosis = self.source.diagnosis
5          derm_stuff = self.source.derm_stuff
6
7          inclusion = (diagnosis
8              .filter(diagnosis.diagnosis_name
9                  .isin(self.inclusion_diagnoses))
10             .select(diagnosis.patient_id)
11             .distinct())
12
13          out = (patient
14              .join(inclusion, on='patient_id', how='leftsemi')
15              .join(derm_stuff, on='patient_id')
16              .select(patient['*'],
17                  derm_stuff.sunscreen_use,
18                  derm_stuff.history_melanoma))
19
20      return out
21
22  # ----- __init__.py ----- #
23
24  from datetime import date
25  import etl.defs.base.life_science as base
26  from etl.defs.base import derm
27  from etl.core import TextFiles
28
29  defs = [derm.patient.DermPatient,
30         base.visit.Visit,
31         derm.procedure.ProcedureInjection,
32         ]
33
34  TableDefinition.export_data_type = TextFiles('acne')
35  TableDefinition.medical_domains = {'DERMATOLOGY'}
36  TableDefinition.start_date = date(2016, 1, 1)
37  TableDefinition.inclusion_diagnoses = {'Acne', 'Acne Vulgaris'}
```

The code is written in Python, utilizing classes and various data processing libraries like pandas and etl. The code defines a class 'DermPatient' that inherits from 'Patient'. It implements the 'make' method to create a DataFrame ('out') by joining multiple datasets ('patient', 'inclusion', and 'derm\_stuff') based on specific inclusion criteria. The code also includes an \_\_init\_\_.py file with imports for datetime, etl.defs.base, etl.core, and specific modules from etl.defs.base (base.visit.Visit, derm.patient.DermPatient, derm.procedure.ProcedureInjection). The TableDefinition section specifies the export type as TextFiles('acne'), medical domains as DERMATOLOGY, and inclusion diagnoses as 'Acne' and 'Acne Vulgaris'.

# PYTHON, FTW!

- Module per dataset
- Enumerate tables
- Parameters/inclusion criteria



```

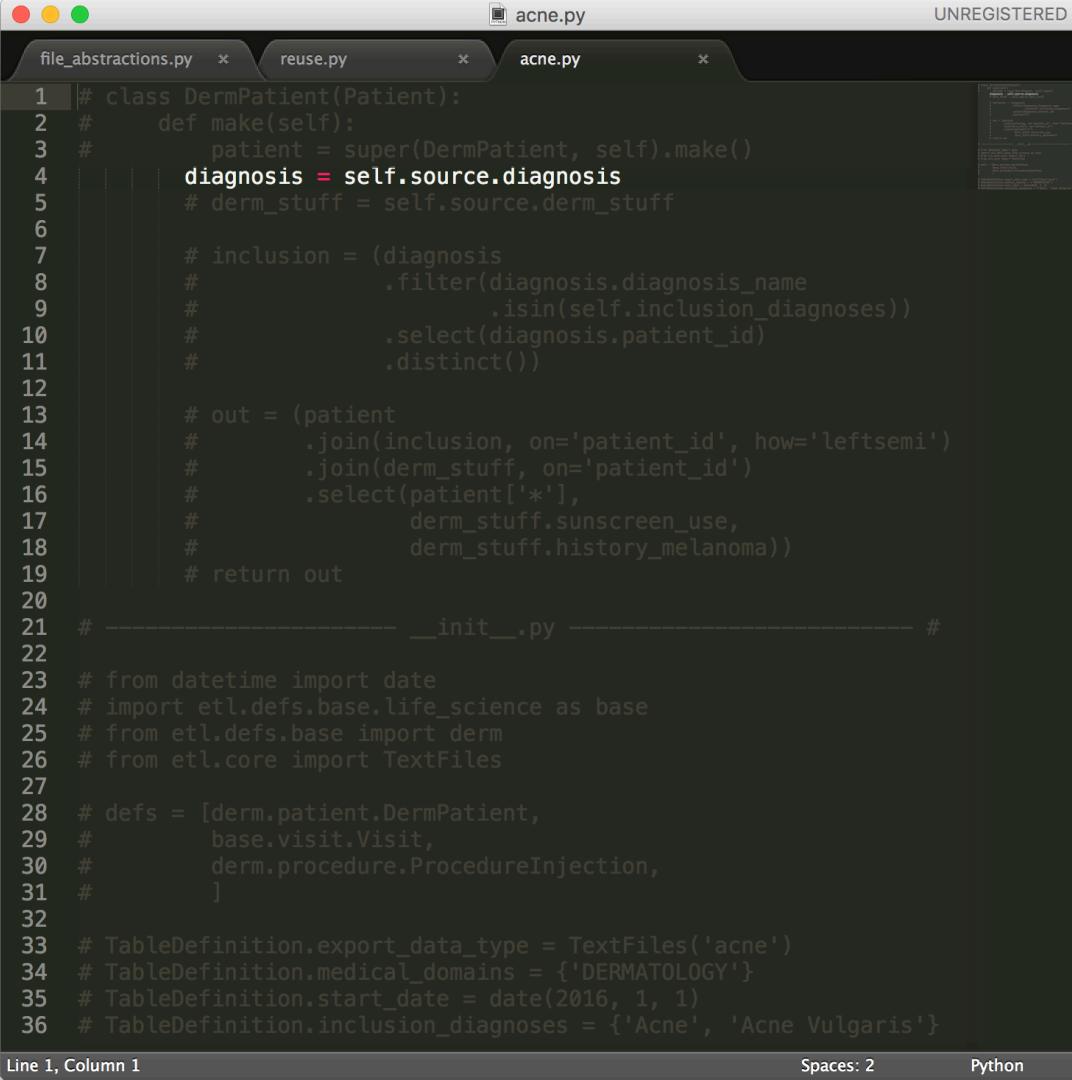
file_abstractions.py * reuse.py * acne.py *
1 |class DermPatient(Patient):
2 |    # def make(self):
3 |    #     patient = super(DermPatient, self).make()
4 |    #     diagnosis = self.source.diagnosis
5 |    #     derm_stuff = self.source.derm_stuff
6 |
7 |    #     inclusion = (diagnosis
8 |    #                     .filter(diagnosis.diagnosis_name
9 |    #                           .isin(self.inclusion_diagnoses))
10 |    #                      .select(diagnosis.patient_id)
11 |    #                      .distinct())
12 |
13 |    #     out = (patient
14 |    #             .join(inclusion, on='patient_id', how='leftsemi')
15 |    #             .join(derm_stuff, on='patient_id')
16 |    #             .select(patient['*'],
17 |    #                     derm_stuff.sunscreen_use,
18 |    #                     derm_stuff.history_melanoma))
19 |    #     return out
20 |
21 |# ----- __init__.py ----- #
22 |
23 |# from datetime import date
24 |# import etl.defs.base.life_science as base
25 |# from etl.defs.base import derm
26 |# from etl.core import TextFiles
27 |
28 |defs = [derm.patient.DermPatient,
29 |        # base.visit.Visit,
30 |        # derm.procedure.ProcedureInjection,
31 |        # ]
32 |
33 |# TableDefinition.export_data_type = TextFiles('acne')
34 |# TableDefinition.medical_domains = {'DERMATOLOGY'}
35 |# TableDefinition.start_date = date(2016, 1, 1)
36 |# TableDefinition.inclusion_diagnoses = {'Acne', 'Acne Vulgaris'}

```

Line 1, Column 1      Spaces: 2      Python

# PYTHON, FTW!

- Module per dataset
- Enumerate tables
- Parameters/inclusion criteria

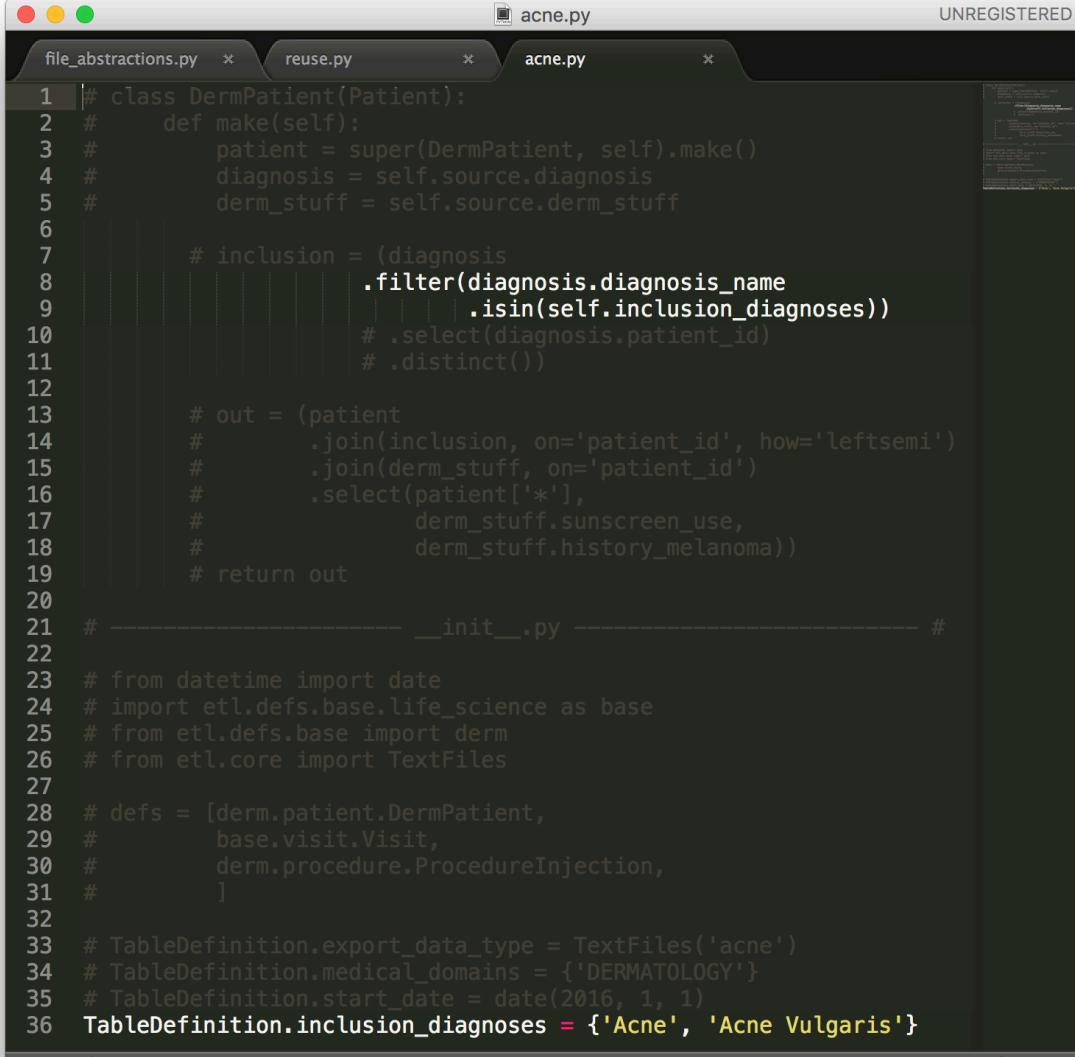


```
file_abstractions.py * reuse.py * acne.py * UNREGISTERED
1 # class DermPatient(Patient):
2 #     def make(self):
3 #         patient = super(DermPatient, self).make()
4 #         diagnosis = self.source.diagnosis
5 #         # derm_stuff = self.source.derm_stuff
6 #
7 #         # inclusion = (diagnosis
8 #         #                 .filter(diagnosis.diagnosis_name
9 #         #                         .isin(self.inclusion_diagnoses))
10 #         #                 .select(diagnosis.patient_id)
11 #         #                 .distinct())
12 #
13 #         # out = (patient
14 #         #                 .join(inclusion, on='patient_id', how='leftsemi')
15 #         #                 .join(derm_stuff, on='patient_id')
16 #         #                 .select(patient['*'],
17 #         #                         derm_stuff.sunscreen_use,
18 #         #                         derm_stuff.history_melanoma))
19 #         # return out
20 #
21 # ----- __init__.py ----- #
22
23 # from datetime import date
24 # import etl.defs.base.life_science as base
25 # from etl.defs.base import derm
26 # from etl.core import TextFiles
27
28 # defs = [derm.patient.DermPatient,
29 #         base.visit.Visit,
30 #         derm.procedure.ProcedureInjection,
31 #         ]
32
33 # TableDefinition.export_data_type = TextFiles('acne')
34 # TableDefinition.medical_domains = {'DERMATOLOGY'}
35 # TableDefinition.start_date = date(2016, 1, 1)
36 # TableDefinition.inclusion_diagnoses = {'Acne', 'Acne Vulgaris'}
```

Line 1, Column 1      Spaces: 2      Python

# PYTHON, FTW!

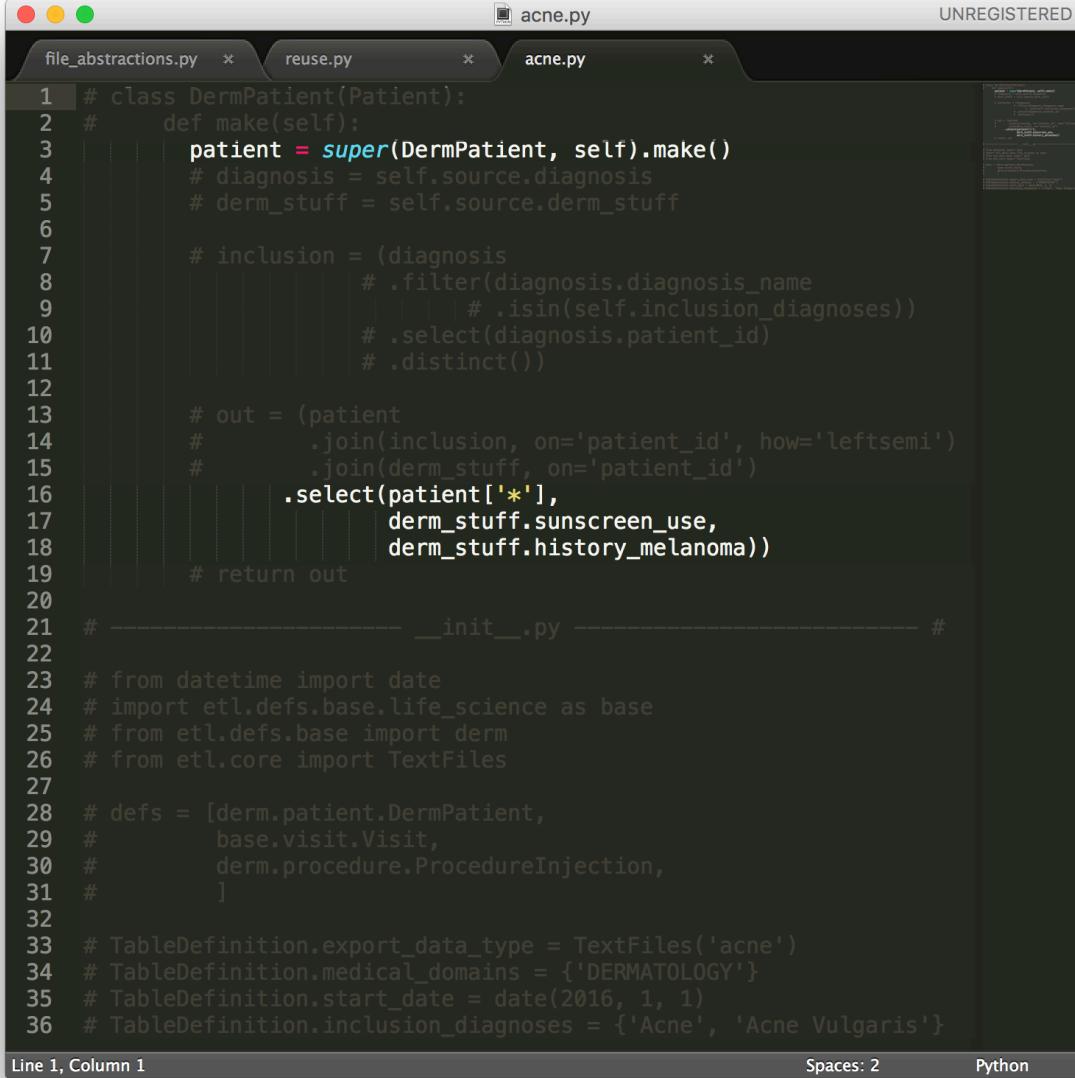
- Module per dataset
- Enumerate tables
- Parameters/inclusion criteria



```
file_abstractions.py * reuse.py * acne.py * acnepy
1 # class DermPatient(Patient):
2 #     def make(self):
3 #         patient = super(DermPatient, self).make()
4 #         diagnosis = self.source.diagnosis
5 #         derm_stuff = self.source.derm_stuff
6 #
7 #         inclusion = (diagnosis
8 #                         .filter(diagnosis.diagnosis_name
9 #                                 .isin(self.inclusion_diagnoses))
10 #                         # .select(diagnosis.patient_id)
11 #                         # .distinct())
12 #
13 #         out = (patient
14 #                 .join(inclusion, on='patient_id', how='leftsemi')
15 #                 .join(derm_stuff, on='patient_id')
16 #                 .select(patient['*'],
17 #                         derm_stuff.sunscreen_use,
18 #                         derm_stuff.history_melanoma))
19 #         return out
20 #
21 # ----- __init__.py ----- #
22 #
23 # from datetime import date
24 # import etl.defs.base.life_science as base
25 # from etl.defs.base import derm
26 # from etl.core import TextFiles
27 #
28 # defs = [derm.patient.DermPatient,
29 #         base.visit.Visit,
30 #         derm.procedure.ProcedureInjection,
31 #         ]
32 #
33 # TableDefinition.export_data_type = TextFiles('acne')
34 # TableDefinition.medical_domains = {'DERMATOLOGY'}
35 # TableDefinition.start_date = date(2016, 1, 1)
36 TableDefinition.inclusion_diagnoses = {'Acne', 'Acne Vulgaris'}
```

# PYTHON, FTW!

- Module per dataset
- Enumerate tables
- Parameters/inclusion criteria

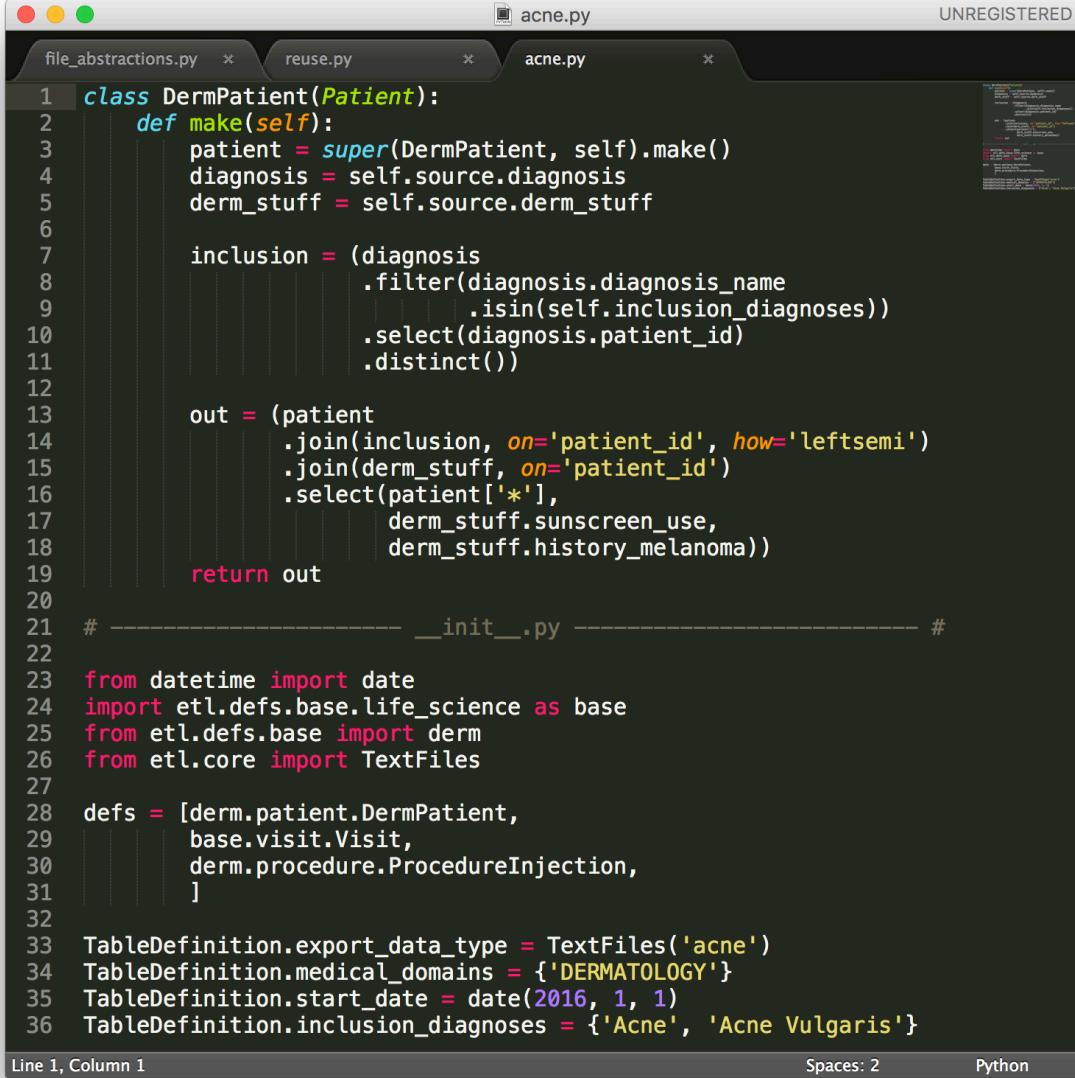


```
file_abstractions.py * reuse.py * acne.py * UNREGISTERED
1 # class DermPatient(Patient):
2 #     def make(self):
3 #         patient = super(DermPatient, self).make()
4 #         diagnosis = self.source.diagnosis
5 #         derm_stuff = self.source.derm_stuff
6 #
7 #         inclusion = (diagnosis
8 #                         .filter(diagnosis.diagnosis_name
9 #                                 .isin(self.inclusion_diagnoses))
10 #                         .select(diagnosis.patient_id)
11 #                         .distinct())
12 #
13 #         out = (patient
14 #                 .join(inclusion, on='patient_id', how='leftsemi')
15 #                 .join(derm_stuff, on='patient_id')
16 #                 .select(patient['*'],
17 #                         derm_stuff.sunscreen_use,
18 #                         derm_stuff.history_melanoma))
19 #         return out
20 #
21 # ----- __init__.py ----- #
22 #
23 # from datetime import date
24 # import etl.defs.base.life_science as base
25 # from etl.defs.base import derm
26 # from etl.core import TextFiles
27 #
28 # defs = [derm.patient.DermPatient,
29 #         base.visit.Visit,
30 #         derm.procedure.ProcedureInjection,
31 #         ]
32 #
33 # TableDefinition.export_data_type = TextFiles('acne')
34 # TableDefinition.medical_domains = {'DERMATOLOGY'}
35 # TableDefinition.start_date = date(2016, 1, 1)
36 # TableDefinition.inclusion_diagnoses = {'Acne', 'Acne Vulgaris'}
```

Line 1, Column 1      Spaces: 2      Python

# PYTHON, FTW!

- Module per dataset
- Enumerate tables
- Parameters/inclusion criteria



```
1  class DermPatient(Patient):
2      def make(self):
3          patient = super(DermPatient, self).make()
4          diagnosis = self.source.diagnosis
5          derm_stuff = self.source.derm_stuff
6
7          inclusion = (diagnosis
8              .filter(diagnosis.diagnosis_name
9                  .isin(self.inclusion_diagnoses))
10             .select(diagnosis.patient_id)
11             .distinct())
12
13          out = (patient
14              .join(inclusion, on='patient_id', how='leftsemi')
15              .join(derm_stuff, on='patient_id')
16              .select(patient['*'],
17                  derm_stuff.sunscreen_use,
18                  derm_stuff.history_melanoma))
19
20      return out
21
22  # ----- __init__.py ----- #
23
24  from datetime import date
25  import etl.defs.base.life_science as base
26  from etl.defs.base import derm
27  from etl.core import TextFiles
28
29  defs = [derm.patient.DermPatient,
30         base.visit.Visit,
31         derm.procedure.ProcedureInjection,
32         ]
33
34  TableDefinition.export_data_type = TextFiles('acne')
35  TableDefinition.medical_domains = {'DERMATOLOGY'}
36  TableDefinition.start_date = date(2016, 1, 1)
37  TableDefinition.inclusion_diagnoses = {'Acne', 'Acne Vulgaris'}
```

Line 1, Column 1      Spaces: 2      Python

# PYTHON, FTW!

- PyCharm
  - Autocomplete
  - Code checks
- Unit tests

The screenshot shows the PyCharm IDE interface. The top navigation bar indicates the file is named `test_functions.py` and is part of the `spark-etl` project. The left sidebar displays the project structure under `spark-etl`, including sub-directories like `etl` and files such as `__init__.py`, `confest.py`, `core.py`, `derby.log`, `dw.py`, `functions.py`, `quality.py`, `quality_rules.py`, `quality_schema.py`, `quality_test.py`, `test_core.py`, `test_functions.py`, `test_quality.py`, and `test_quality_rules.py`. The main code editor window contains the `test_functions.py` script, which includes imports for `pytest` and `etl`, and defines several test functions using `assert_df` and `hive_context.createDataFrame`. Below the code editor is the `Run` tool window, which shows the results of a test run: "All 11 tests passed - 14s 900ms". The bottom status bar shows the message "Tests Passed: 11 passed (a minute ago)".

```
import pytest
from etl import functions as CF
pytest.mark.usefixtures("spark_context", "hive_context", "assert_df")

def test_group_concat(hive_context, assert_df):
    input_data = hive_context.createDataFrame([(1, 'blah'),
                                              (1, 'foo'),
                                              (1, 'blah'),
                                              (2, 'foobar'),
                                              (3, 'foofoo'),
                                              (3, None),
                                              (4, None)], schema=['id', 'col'])

    expected = hive_context.createDataFrame([(1, 'blah,foo,blah'),
                                             (2, 'foobar'),
                                             (3, 'foofoo'),
                                             (4, '')], schema=['id', 'col'])

    expected_distinct = hive_context.createDataFrame([(1, 'blah,foo,blah'),
                                                      (2, 'foobar'),
                                                      (3, 'foofog'),
                                                      (4, '')], schema=['id', 'col'])

    # need two because set order is not guaranteed
    expected_distinct_alt = hive_context.createDataFrame([(1, 'foo,blah'),
                                                          (2, 'foobar'),
                                                          (3, 'foofoo'),
                                                          (4, '')], schema=['id', 'col'])

# Test Results 14s 900ms
# test_core.py 14s 900ms
#   test_data_dir 2ms
#   test_load 3s 835ms
#   test_load_file 92ms
#   test_load_ls 658ms
#   test_load_tat 12ms
#   test_save_tz 797ms
#   test_save_tz 169ms
#   test_load_ta 137ms
#   test_save 2s 792ms
# ===== test session starts =====
# platform darwin -- Python 2.7.13, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
# rootdir: /Users/aaron.richter/repos/spark-etl, inifile:
# collected 11 items

# etl/test_core.py Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
# Setting default log level to "WARN".
# To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
# 18/05/03 20:36:35 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
# /Users/aaron.richter/anaconda2/lib/python2.7/site-packages/pymysql/cursors.py:166: Warning: (1007, "Can't
# result = self._query(query)
```

A photograph showing a person's hands wearing a white lab coat. They are using a smartwatch with a dark strap and a light-colored face. The watch screen displays a menu with several icons and text labels. The background is blurred, suggesting a clinical or laboratory setting.

# CURATION & QUALITY

DATA ABOUT THE DATA

# CURATION & QUALITY

## Data Dictionaries

- JSON file for each dataset's schema + metadata
- Generate Excel files from JSON definitions
  - Version control
  - Source of truth
  - pandas + xlsxwriter

## Quality

- JSON file includes quality rules
  - Column names and types
  - Primary/foreign keys
  - Accepted values
- Processing in PySpark
- Future: Web-based content management system

Shoutout to *Smartsheet* for inspiration

<https://pages.databricks.com/Smartsheet-operationalized-Spark.html>

example.json UNREGISTERED

```
1 {  
2     "name": "Example Dataset",  
3     "_comment": "No patients were harmed in the making of this example",  
4     "tables": [  
5         {  
6             "group": "Patient",  
7             "name": "patient",  
8             "description": "patient table",  
9             "columns": [  
10                 {  
11                     "name": "patient_id",  
12                     "type": "string",  
13                     "pk": true,  
14                     "field_length": 11,  
15                     "description": "patient record identifier"  
16                 },  
17                 {  
18                     "name": "date_of_birth",  
19                     "type": "date"  
20                 },  
21                 {  
22                     "name": "sex",  
23                     "type": "string",  
24                     "field_length": 10,  
25                     "values": [  
26                         "MALE",  
27                         "FEMALE",  
28                         "UNKNOWN"  
29                     ]  
30                 }  
31             ]  
32         }  
33     ]  
34 }
```

Line 1, Column 1      Spaces: 2      JSON

example Search Sheet

Home Insert Page Layout Formulas Data Review View Conditional Formatting Format as Table Cell Styles Cells Editing

E1 A B C D E F

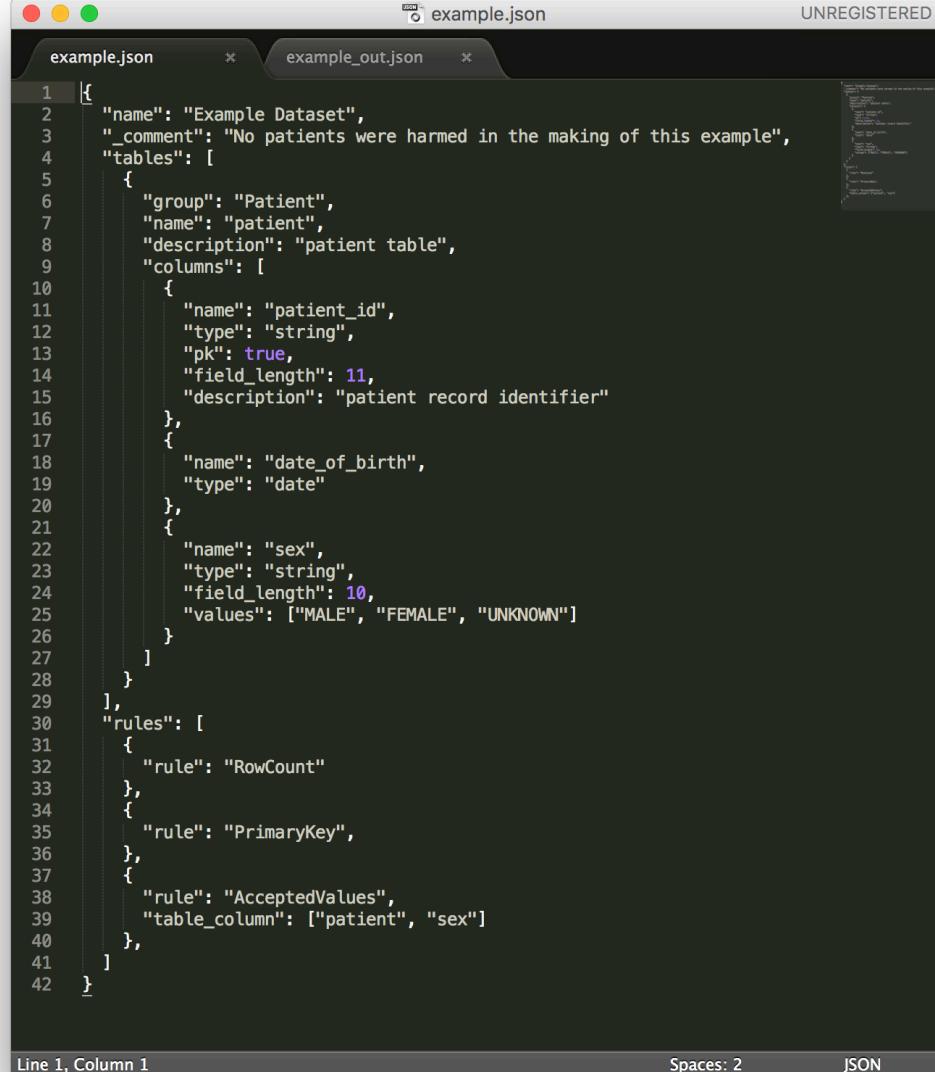
1	Example				
2	Data Dictionary				
3	3 May 2018				
5	Table	Column	Description	Data Type	Values
6	patient	patient_id	patient record identifier	string	
7	patient	date_of_birth		date	
8	patient	sex		string	MALE FEMALE UNKNOWN
9					
10					
11					

Tables Data Dictionary +

Ready 188%

# QUALITY

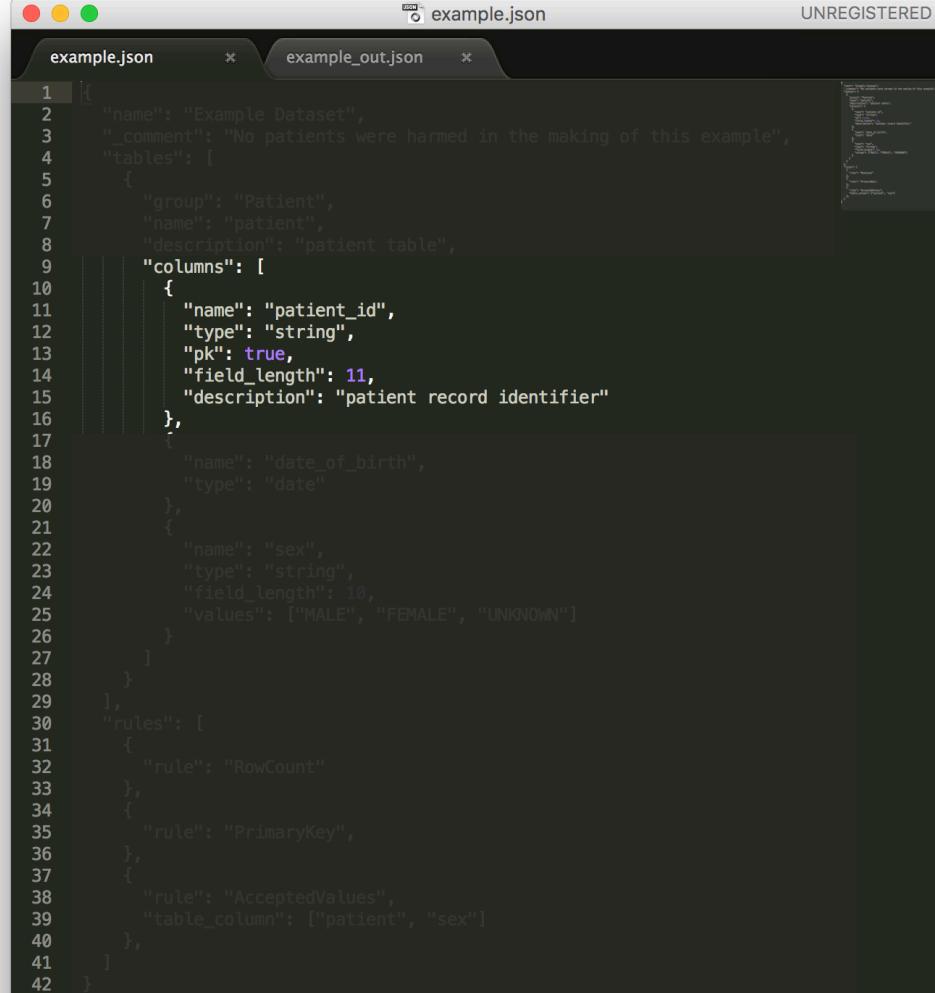
- Column properties trigger rules
- Specify table/column to execute rule for



```
example.json * example_out.json *
Line 1, Column 1
Spaces: 2
JSON
1 | {
2 |   "name": "Example Dataset",
3 |   "_comment": "No patients were harmed in the making of this example",
4 |   "tables": [
5 |     {
6 |       "group": "Patient",
7 |       "name": "patient",
8 |       "description": "patient table",
9 |       "columns": [
10 |         {
11 |           "name": "patient_id",
12 |           "type": "string",
13 |           "pk": true,
14 |           "field_length": 11,
15 |           "description": "patient record identifier"
16 |         },
17 |         {
18 |           "name": "date_of_birth",
19 |           "type": "date"
20 |         },
21 |         {
22 |           "name": "sex",
23 |           "type": "string",
24 |           "field_length": 10,
25 |           "values": ["MALE", "FEMALE", "UNKNOWN"]
26 |         }
27 |       ]
28 |     },
29 |     {
30 |       "rules": [
31 |         {
32 |           "rule": "RowCount"
33 |         },
34 |         {
35 |           "rule": "PrimaryKey",
36 |         },
37 |         {
38 |           "rule": "AcceptedValues",
39 |           "table_column": ["patient", "sex"]
40 |         }
41 |       ]
42 |     }
43 |   ]
44 | }
```

# QUALITY

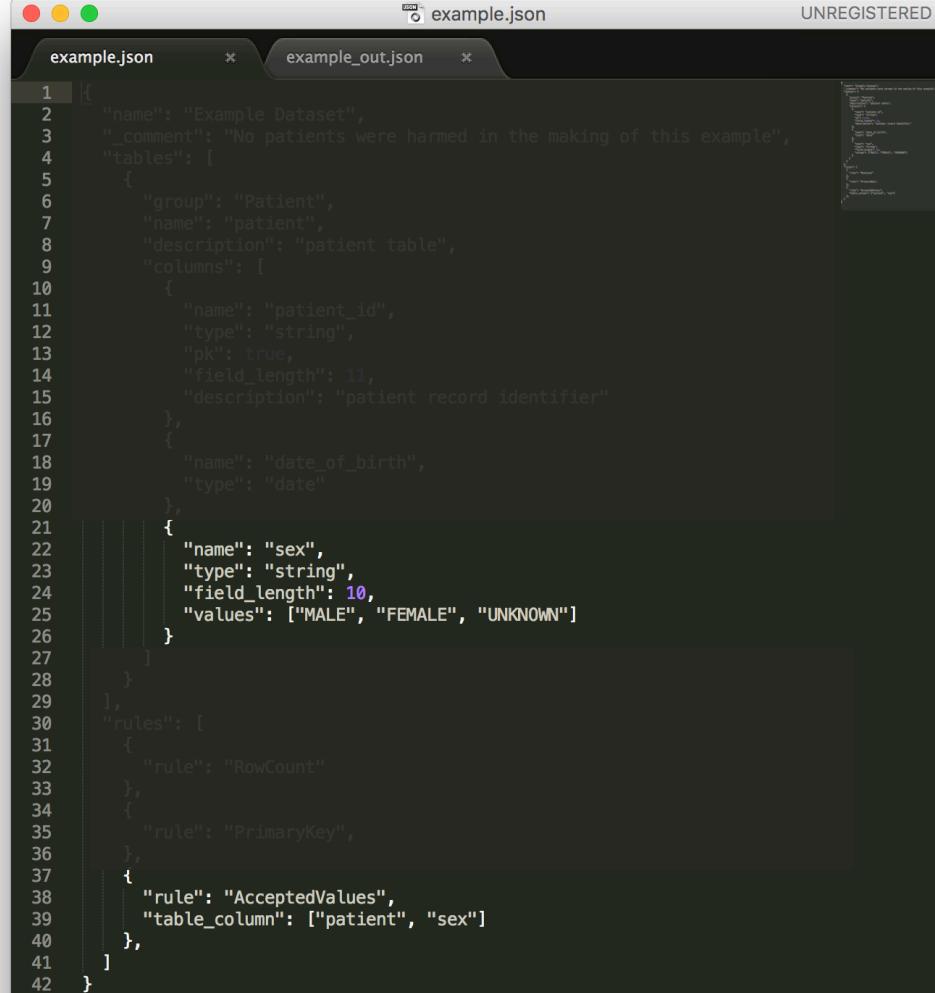
- Column properties trigger rules
- Specify table/column to execute rule for



```
example.json * example_out.json *
1 [
2   "name": "Example Dataset",
3   "_comment": "No patients were harmed in the making of this example",
4   "tables": [
5     {
6       "group": "Patient",
7       "name": "patient",
8       "description": "patient table",
9       "columns": [
10         {
11           "name": "patient_id",
12           "type": "string",
13           "pk": true,
14           "field_length": 11,
15           "description": "patient record identifier"
16         },
17         {
18           "name": "date_of_birth",
19           "type": "date"
20         },
21         {
22           "name": "sex",
23           "type": "string",
24           "field_length": 10,
25           "values": ["MALE", "FEMALE", "UNKNOWN"]
26         }
27       ]
28     ],
29   ],
30   "rules": [
31     {
32       "rule": "RowCount"
33     },
34     {
35       "rule": "PrimaryKey",
36     },
37     {
38       "rule": "AcceptedValues",
39       "table_column": ["patient", "sex"]
40     }
41   ]
42 ]
```

# QUALITY

- Column properties trigger rules
- Specify table/column to execute rule for



```
example.json * example_out.json *
1 [
2   "name": "Example Dataset",
3   "_comment": "No patients were harmed in the making of this example",
4   "tables": [
5     {
6       "group": "Patient",
7       "name": "patient",
8       "description": "patient table",
9       "columns": [
10         {
11           "name": "patient_id",
12           "type": "string",
13           "pk": true,
14           "field_length": 11,
15           "description": "patient record identifier"
16         },
17         {
18           "name": "date_of_birth",
19           "type": "date"
20         },
21         {
22           "name": "sex",
23           "type": "string",
24           "field_length": 10,
25           "values": ["MALE", "FEMALE", "UNKNOWN"]
26         }
27       ]
28     },
29   ],
30   "rules": [
31     {
32       "rule": "RowCount"
33     },
34     {
35       "rule": "PrimaryKey",
36     },
37     {
38       "rule": "AcceptedValues",
39       "table_column": ["patient", "sex"]
40     },
41   ]
42 }
```

# QUALITY

- Execution date
- Pass/fail for each rule
- Messages



The screenshot shows a Mac OS X application window with two tabs: "example.json" and "example\_out.json". The "example\_out.json" tab is active, displaying a JSON report. The report details a dataset named "Example Dataset" with a note about no patients being harmed. It includes a "tables" section with a single table named "patient". This table has a "rules" section containing a "RowCount" rule that passed with a result of 5677213. It also has a "columns" section with three columns: "patient\_id" (which is a Primary Key and passed), "date\_of\_birth" (which is a date type), and "sex" (which failed the AcceptedValues rule, resulting in 3 bad values). The JSON code is as follows:

```
1  {
2      "name": "Example Dataset",
3      "_comment": "No patients were harmed in the making of this example",
4      "datetime": "2017-05-03 13:12:14",
5      "tables": [
6          {
7              "group": "Patient",
8              "name": "patient",
9              "description": "patient table",
10             "rules": [
11                 {
12                     "rule": "RowCount",
13                     "result": 5677213
14                 }
15             ],
16             "columns": [
17                 {
18                     "name": "patient_id",
19                     "rules": [
20                         {
21                             "rule": "PrimaryKey",
22                             "pass": true
23                         }
24                     ]
25                 },
26                 {
27                     "name": "date_of_birth",
28                     "type": "date"
29                 },
30                 {
31                     "name": "sex",
32                     "rules": [
33                         {
34                             "rule": "AcceptedValues",
35                             "pass": false,
36                             "message": "3 records have bad values"
37                         }
38                     ]
39                 }
40             ]
41         }
42     ]
43 }
```

# QUALITY

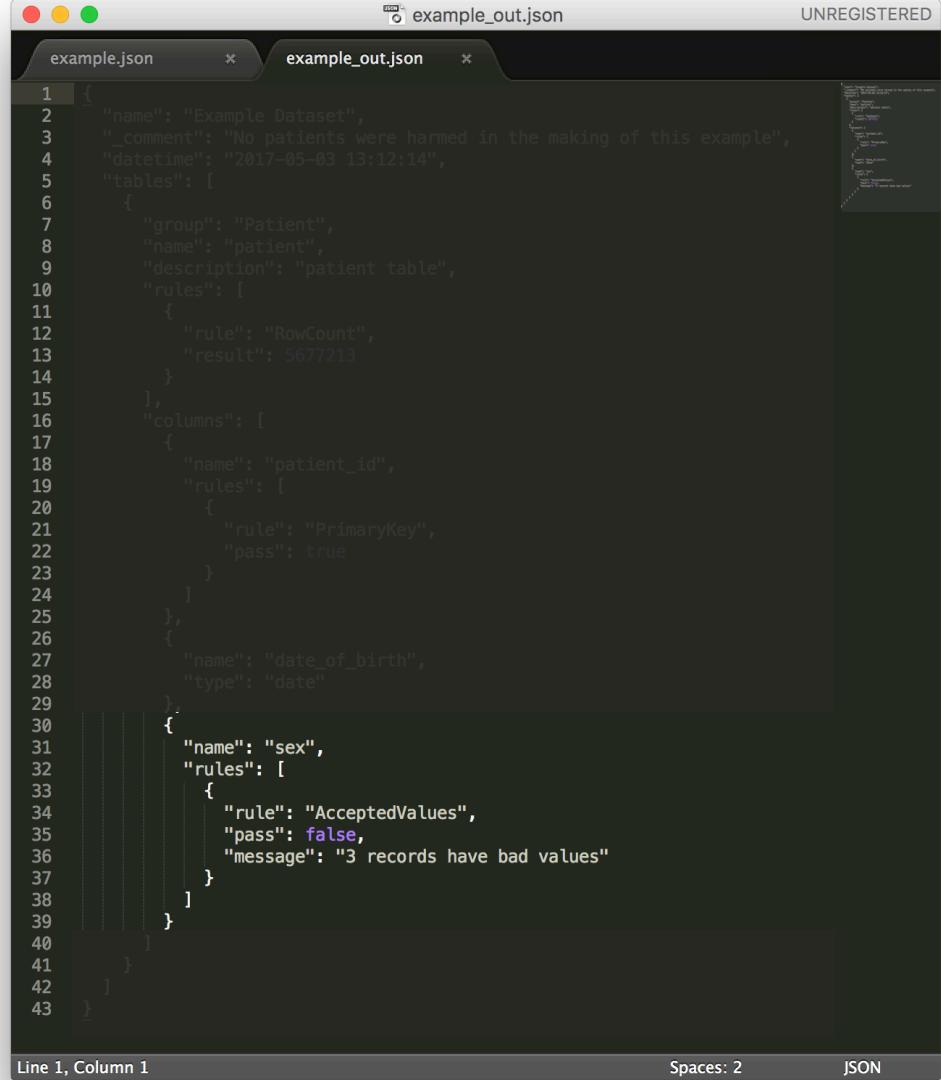
- Execution date
- Pass/fail for each rule
- Messages

```
example.json * example_out.json *
1 {
2   "name": "Example Dataset",
3   "_comment": "No patients were harmed in the making of this example",
4   "datetime": "2017-05-03 13:12:14",
5   "tables": [
6     {
7       "group": "Patient",
8       "name": "patient",
9       "description": "patient table",
10      "rules": [
11        {
12          "rule": "RowCount",
13          "result": 5677213
14        }
15      ],
16      "columns": [
17        {
18          "name": "patient_id",
19          "rules": [
20            {
21              "rule": "PrimaryKey",
22              "pass": true
23            }
24          ]
25        },
26        {
27          "name": "date_of_birth",
28          "type": "date"
29        },
30        {
31          "name": "sex",
32          "rules": [
33            {
34              "rule": "AcceptedValues",
35              "pass": false,
36              "message": "3 records have bad values"
37            }
38          ]
39        }
40      ]
41    }
42  ]
43 }
```

Line 1, Column 1      Spaces: 2      JSON

# QUALITY

- Execution date
- Pass/fail for each rule
- Messages



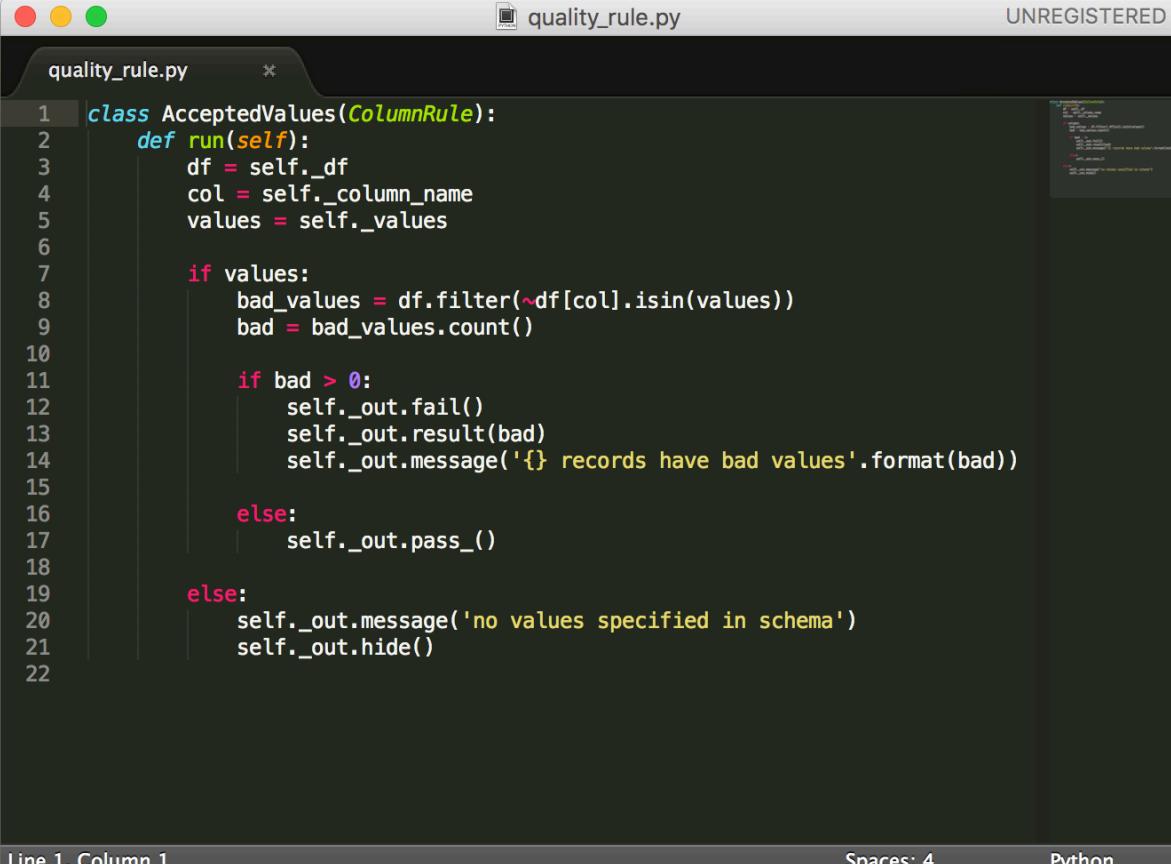
The screenshot shows a terminal window with two tabs: "example.json" and "example\_out.json". The "example\_out.json" tab is active, displaying a JSON object with execution details and quality rules.

```
Line 1, Column 1
example.json * example_out.json *
1 {
2   "name": "Example Dataset",
3   "_comment": "No patients were harmed in the making of this example",
4   "datetime": "2017-05-03 13:12:14",
5   "tables": [
6     {
7       "group": "Patient",
8       "name": "patient",
9       "description": "patient table",
10      "rules": [
11        {
12          "rule": "RowCount",
13          "result": 5677213
14        }
15      ],
16      "columns": [
17        {
18          "name": "patient_id",
19          "rules": [
20            {
21              "rule": "PrimaryKey",
22              "pass": true
23            }
24          ]
25        },
26        {
27          "name": "date_of_birth",
28          "type": "date"
29        }
30      ],
31      "rules": [
32        {
33          "rule": "AcceptedValues",
34          "pass": false,
35          "message": "3 records have bad values"
36        }
37      ]
38    }
39  ]
40 }
41 ]
42 }
43 }
```

The JSON output includes the dataset name, creation comment, timestamp, and a list of tables. Each table entry contains a "rules" section. For the "patient" table, there is a "RowCount" rule that passed with a result of 5677213. The "patient\_id" column has a "PrimaryKey" rule that also passed. The "date\_of\_birth" column is of type "date". The "sex" column has an "AcceptedValues" rule that failed, indicating 3 records have bad values.

# QUALITY

- Dataset rules
- Table rules
- Inter-column rules
- Column rules



```
1  class AcceptedValues(ColumnRule):
2      def run(self):
3          df = self._df
4          col = self._column_name
5          values = self._values
6
7          if values:
8              bad_values = df.filter(~df[col].isin(values))
9              bad = bad_values.count()
10
11             if bad > 0:
12                 self._out.fail()
13                 self._out.result(bad)
14                 self._out.message('{} records have bad values'.format(bad))
15
16             else:
17                 self._out.pass_()
18
19         else:
20             self._out.message('no values specified in schema')
21             self._out.hide()
22
```

Line 1, Column 1      Spaces: 4      Python

A photograph of two men in an office environment. One man, wearing a blue shirt and glasses, is leaning over a desk, looking at a laptop screen. The other man, wearing a grey suit jacket with a small orange logo on the lapel, is seated at the desk, also looking at the laptop. The laptop has an Apple logo on its back. In the background, there's a window showing a building and some greenery.

# PRODUCTION

LET THE AIRFLOW

Airflow - DAGs    Aaron

my-cool-airflow.com:8080

Airflow DAGs Data Profiling Browse Admin Docs About Quality 21:48 UTC ⚡

## DAGs

Show 25 entries Search:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
<input checked="" type="checkbox"/>		automation	@daily	operations	<span>1</span>	166		
<input checked="" type="checkbox"/>		such_data		Airflow				
<input checked="" type="checkbox"/>		so_spark	None	Airflow	<span>12</span>	2		
<input checked="" type="checkbox"/>		wow	0 11 ***	Airflow	<span>4</span>	1		
<input checked="" type="checkbox"/>		bad	None	Airflow	<span>2</span> <span>1</span>	17   17		

Quality - Data Quality - Airflow × Aaron

my-cool-airflow:8080/admin/dataquality/?ds=data&json=data\_001.json

Airflow DAGs Data Profiling Browse Admin Docs About Quality 21:58 UTC ⚡

## Jobs

Please select a job to begin quality:

DATA QUALITY IS COOL

## Data Statistics

Rule stats pertaining to current file selected:

Job Name: DATA Execution Date: 2018-05-22 02:00:05  
JSON Name: data\_001.json Total Failed: 3

Rules	Rule Count Total	Passed Rule Count	% Columns
CheckPrimaryKey	41	41	100.0%
FieldLength	187	187	100.0%
AcceptedValues	43	43	100.0%
ForeignKey	50	50	100.0%
Nullable	27	24	88.89%

# CONTRIBUTORS



Aaron Richter  
Data Scientist



Michael Crawford  
Data Engineer



Eric Golinko  
Data Scientist



Kevin Da Silva  
Data Engineering Intern



TELEPHONE

## AARON RICHTER

Data Scientist

PhD Candidate



@rikturr

# THANK YOU!

