

Using MongoDB and Python for data analysis pipeline

Eoin Brazil, PhD, MSc
Proactive Technical Services, MongoDB

Github repo for this talk: http://github.com/braz/pycon2015_talk/

A photograph of a complex industrial piping system made of polished stainless steel. Several large horizontal pipes run across the frame, with smaller vertical pipes and components branching off. Two circular pressure gauges with white faces and red needles are visible on the upper pipes. The background shows more industrial equipment and a concrete floor. The text "From once off to real scale production" is overlaid in large white font.

From once off
to real scale
production

What this talk will cover

Pipelines

All about building



Systems

All about tools



Speed

Making it all hum



Challenges for an operational pipeline:

- Combining
- Cleaning / formatting
- Supporting free flow

Reproducibility

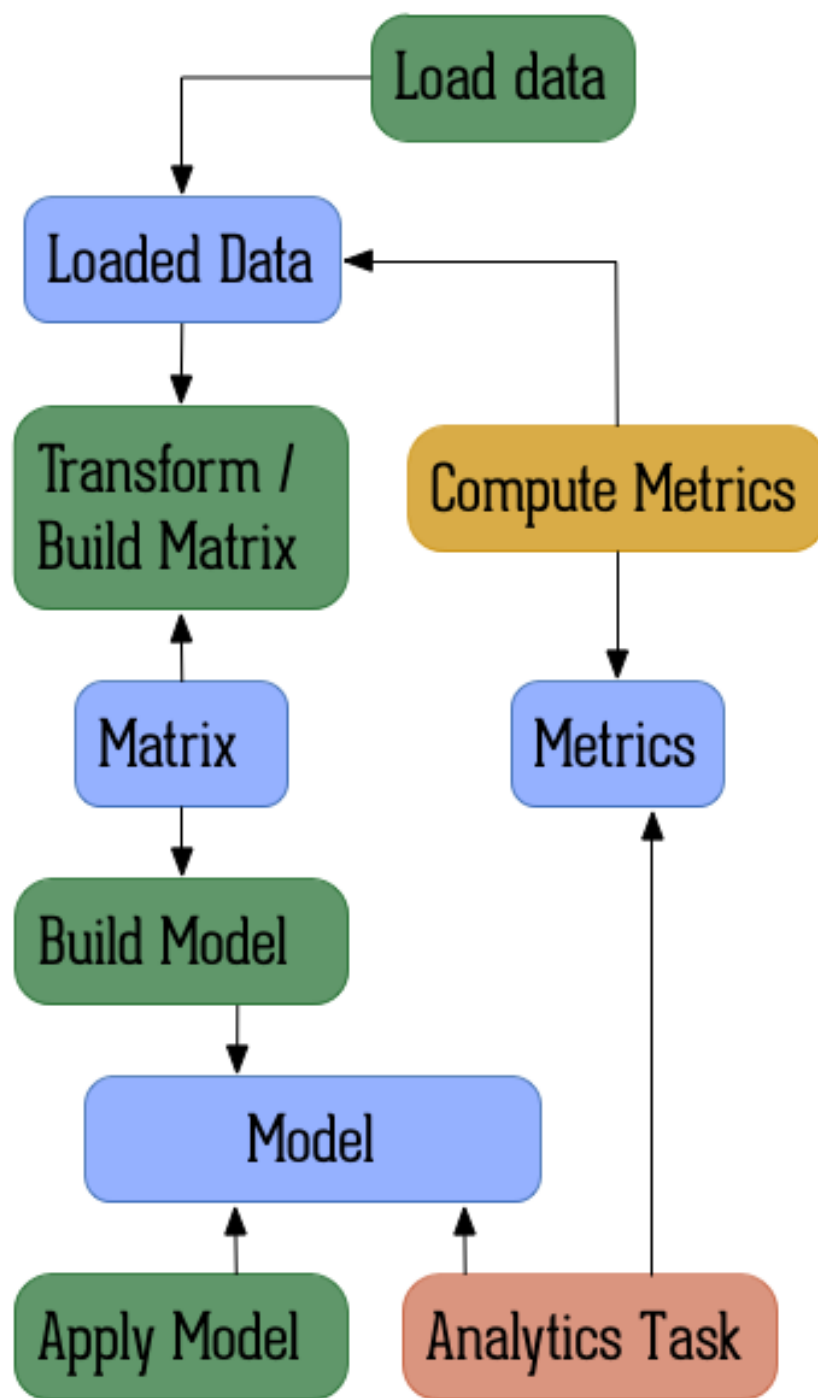


Production



An example data pipeline:

- Data
- State
- Operations / Transformation



Averaging a data set

- Python dictionary ~12 million numbers per second
- Python List 110 million numbers per second
- `numpy.ndarray` 500 million numbers per second

***ndarray** or n-dimensional array, provides high-performance c-style arrays uses built-in maths libraries.*



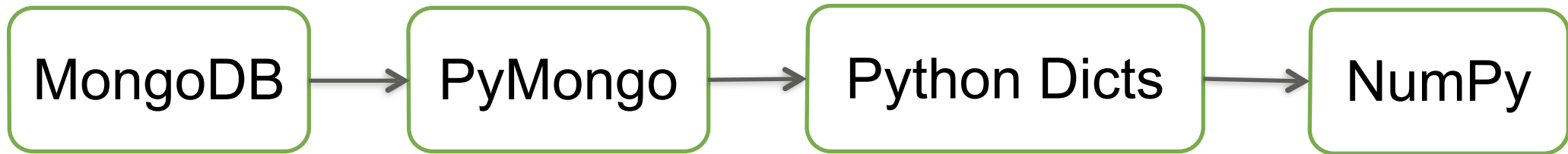
Data Transformations

C	Zero Mean	{	1 - Predictors - M	{	SS	M-Dim Sphere
S	One SD				PCA	
T	$\sqrt{\log}$ INV				PLS	

Correlation, Dummy Variables, Filtering

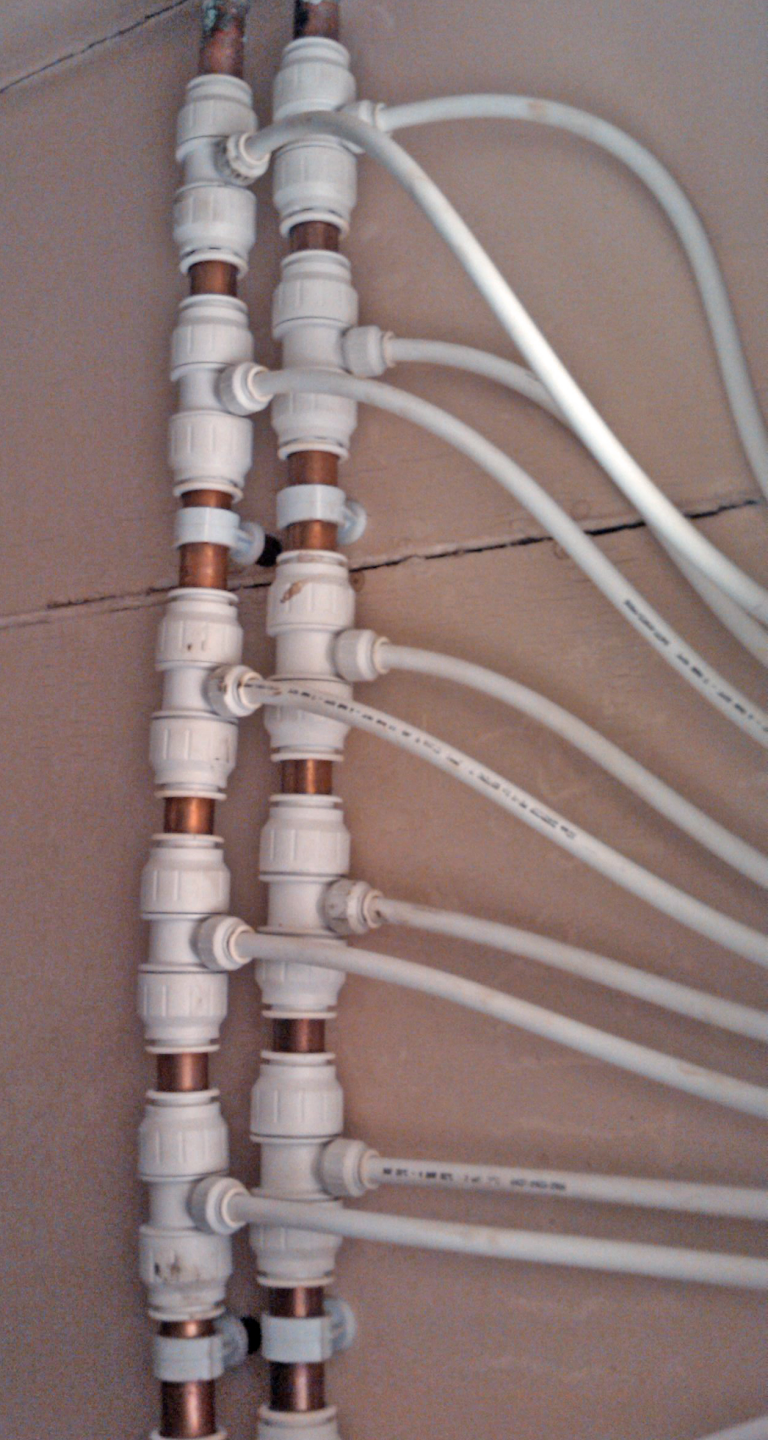
Workflows to / from MongoDB

PyMongo Workflow: ~150,000 documents per second



Monary Workflow: 1,700,000 documents per second





An example of connecting the pipes

- Monary
- MongoDB
- Python
- Airflow

Firstly dive into MongoDB's Aggregation & Monary

Data set and Aggregation

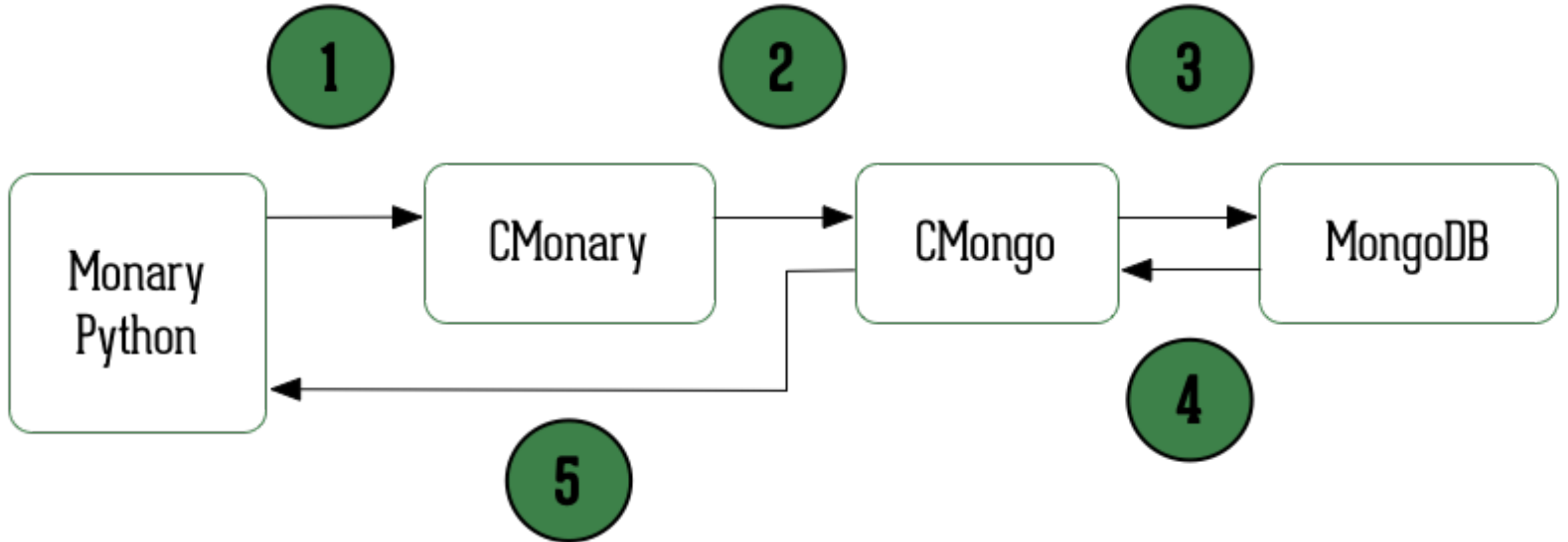
```
zip> db.data.findOne()  
{ "_id" : "01001",  
  "city" : "AGAWAM",  
  "loc" : [  
    -72.622739, 42.070206  
  ],  
  "pop" : 15338,  
  "state" : "MA"  
}
```

- ID
- City name
- Lat/Long
- Population
- State

```
pipeline = [{"$group" :  
  {"_id" : "$state",  
   "totPop" : {"$sum" :  
    "$pop"}}}]
```

1. Group documents by state
2. Sum the population value for each individual city to give a state population total

Monary Query



Monary Query

```
>>> from monary import Monary
>>> m = Monary()
>>> pipeline = [{"$group" : {"_id" :
"$state", "totPop" : {"$sum" : "$pop"}}}]
>>> states, population =
m.aggregate("zips", "data", pipeline,
["_id", "totpop"], ["string:2", "int64"])
```


Monary Query

Database

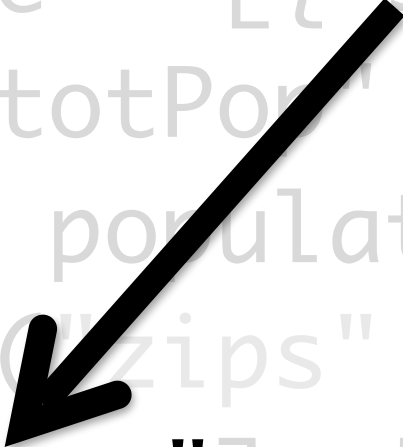
```
>>> from monary import Monary
>>> m = Monary()
>>> pipeline = [{"$group": {"_id":
"$state", "totPop": {"$sum": "$pop"}}}]
>>> states, population =
m.aggregate("zips", "data", pipeline,
["_id", "totpop"], ["string:2", "int64"])
```



Monary Query

Field Name

```
>>> from monary import Monary
>>> m = Monary()
>>> pipeline = [{"$group": {"_id":
"$state", "totPop": {"$sum": "$pop"}}}]
>>> states, population =
m.aggregate("zip", "data", pipeline,
["_id", "totpop"], ["string:2", "int64"])
```



Monary Query

```
>>> from monary import Monary
>>> m = Monary()
>>> pipeline = [{"$group": {"_id":
"$state", "totPop": {"$sum": "$pop"}}}]
>>> states, population =
m.aggregate("zips", "data", pipeline,
["_id", "totpop"], ["string:2", "int64"])
```

Return type



Aggregation Result

```
[u'WA: 4866692', u'HI: 1108229', u'CA: 29754890', u'OR: 2842321', u'NM: 1515069', u'UT: 1722850', u'OK: 3145585', u'LA: 4217595', u'NE: 1578139', u'TX: 16984601', u'MO: 5110648', u'MT: 798948', u'ND: 638272', u'AK: 544698', u'SD: 695397', u'DC: 606900', u'MN: 4372982', u'ID: 1006749', u'KY: 3675484', u'WI: 4891769', u'TN: 4876457', u'AZ: 3665228', u'CO: 3293755', u'KS: 2475285', u'MS: 2573216', u'FL: 12686644', u'IA: 2776420', u'NC: 6628637', u'VA: 6181479', u'IN: 5544136', u'ME: 1226648', u'WV: 1793146', u'MD: 4781379', u'GA: 6478216', u'NH: 1109252', u'NV: 1201833', u'DE: 666168', u'AL: 4040587', u'CT: 3287116', u'SC: 3486703', u'RI: 1003218', u'PA: 11881643', u'VT: 562758', u'MA: 6016425', u'WY: 453528', u'MI: 9295297', u'OH: 10846517', u'AR: 2350725', u'IL: 11427576', u'NJ: 7730188', u'NY: 17990402']
```

Aggregation Result

```
[ u'WA: 4866692', u'HI: 1108229', u'CA: 29754890', u'OR: 2842321', u'NM: 1515069', u'UT: 1722850', u'OK: 3145585', u'LA: 4217595', u'NE: 1578139', u'TX: 16984601', u'MO: 5110648', u'MT: 798948', u'ND: 638272', u'AK: 544698', u'SD: 695397', u'DC: 606900', u'MN: 4372982', u'ID: 1006749', u'KY: 3675484', u'WI: 4891769', u'TN: 4876457', u'AZ: 3665228', u'CO: 3293755', u'KS: 2475285', u'MS: 2573216', u'FL: 12686644', u'IA: 2776420', u'NC: 6628637', u'VA: 6181479', u'IN: 5544136', u'ME: 1226648', u'WV: 1793146', u'MD: 4781379', u'GA: 6478216', u'NH: 1109252', u'NV: 1201833', u'DE: 666168', u'AL: 4040587', u'CT: 3287116', u'SC: 3486703', u'RI: 1003218', u'PA: 11881643', u'VT: 562758', u'MA: 6016425', u'WY: 453528', u'MI: 9295297', u'OH: 10846517', u'AR: 2350725', u'IL: 11427576', u'NJ: 7730188', u'NY: 17990402' ]
```


NumPy

Monary

**Scikit -
learn**



mongoDB
FOR GIANT IDEAS

Python

Cron

Airflow

Luigi

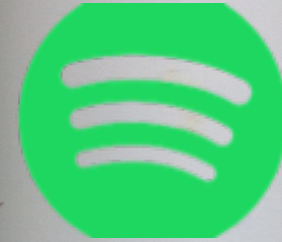
Pandas

Matlibplot

PyTables

Fitting your pipelines together:

- Schedule/Repeatable
- Monitoring
- Checkpoints
- Dependencies



**What have these companies
done to improve their
workflows for data pipelines ?**

localhost:8080/admin/

Airflow DAGs Data Profiling Browse Admin Docs 14:12 UTC

DAGs

Show 100 entries Search:

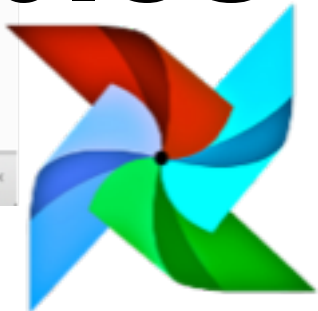
	DAG	Owner	Statuses	Links
On	example_bash_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄
On	example_branch_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄
On	example_http_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄
	example_monary_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄
	example_pymongo_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄
On	example_python_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄
On	example_xcom	airflow		🔍 📊 📈 📉 ⚙️ 🔄
On	tutorial	airflow		🔍 📊 📈 📉 ⚙️ 🔄

Showing 1 to 8 of 8 entries

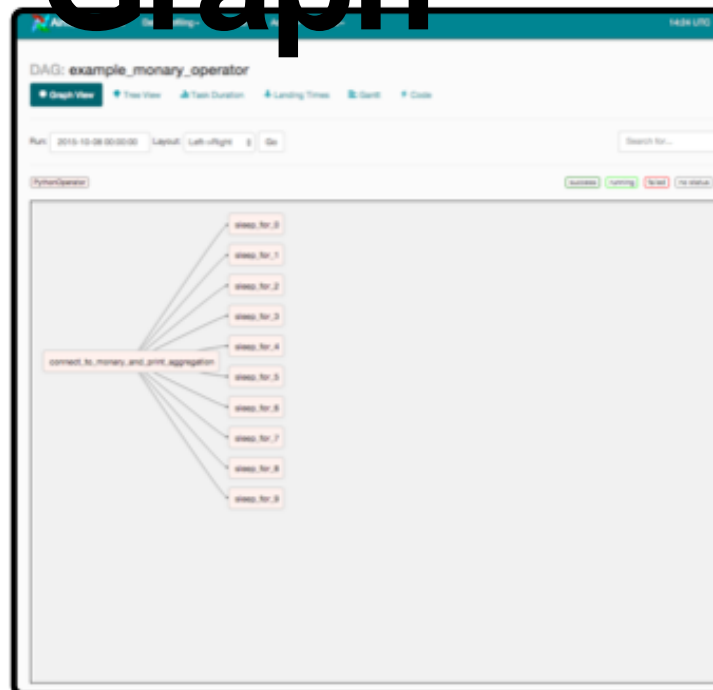
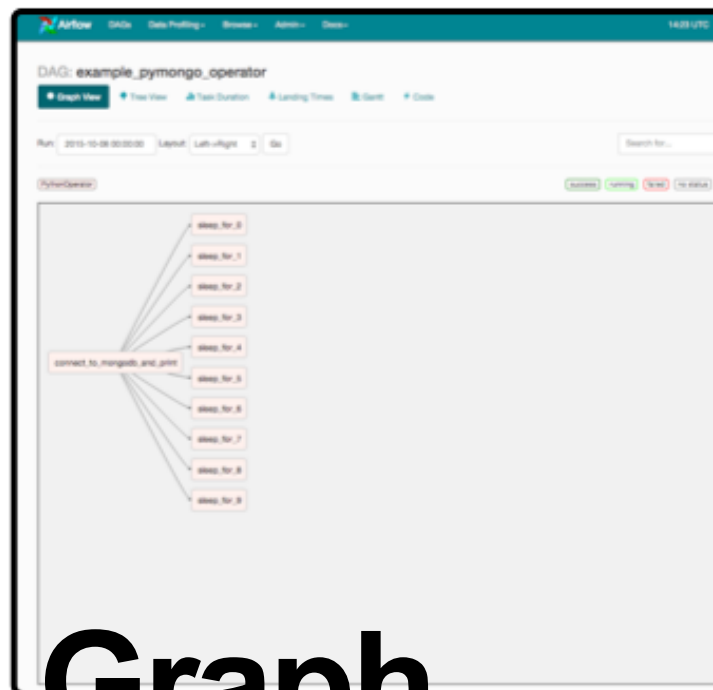
Previous 1 Next

	example_monary_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄
	example_pymongo_operator	airflow		🔍 📊 📈 📉 ⚙️ 🔄

Two Python/MongoDB Examples



Visual Graph



Code



DAG: example_pymongo_operator

Graph View

Tree View

Task Duration

Landing Times

Save

Code

14:04 UTC

example_dags/example_pymongo_operator.py

```

1 from __future__ import print_function
2 from datetime import datetime
3 from airflow.operators import MongoOperator
4 from airflow.models import DAG
5 from datetime import datetime, timedelta
6
7 import time
8 from pymongo import MongoClient
9
10 server_days_ago = datetime.utcnow().date() - timedelta(days=1)
11
12 client = MongoClient()
13
14 ops = [
15     "insert", "update",
16     "insert_many", "update_many",
17 ]
18
19 dag = DAG(dag_id="example_pymongo_operator", default_args=default_args)
20
21
22 def sleep_random_base():
23     """This is a function that will run within the DAG execution"""
24     time.sleep(random_base)
25
26
27 def connect_to_mongodb_and_print_info, sleep_random():
28     db = MongoClient().ops
29     collection = db.command("listCollections")
30     print(collection)
31     return "Whatever you return gets printed on the logs"
32
33 run_this = PythonOperator(
34     task_id="connect_to_mongodb_and_print",
35     provide_context=True,
36     python_callable=connect_to_mongodb_and_print,
37     dag=dag)
38
39 for i in range(10):
40     """
41     generating 10 sleeping task, sleeping from 0 to 8 seconds
42     respectively
43     """
44     task = PythonOperator(
45         task_id="sleep_{}".format(i),
46         python_callable=sleep_random,
47         sleep_time=(random_base + i),
48         dag=dag)
49
50 task.set_upstream(run_this)

```

 Airflow

Tools Profiling

Browse

Admin

Docs

Graph View

Tree View

Task Duration

Launching Times

Logs

Code

dag: example_monetary_operator

```
1 from ...Network import print_function
2 from ...utils import range
3 from airflow.operators import PythonOperator
4 from airflow.models import DAG
5 from datetime import datetime, timedelta
6 import time
7 from money import Money
8
9 seven_days_ago = datetime.combine(datetime.today() - timedelta(7),
10                                  datetime.min.time())
11
12 default_args = {
13     "owner": "airflow",
14     "start_date": seven_days_ago,
15     "retries": 3,
16     "retry_delay": timedelta(minutes=5),
17 }
18
19 dag = DAG(dag_id="example_monetary_operator", default_args=default_args)
20
21 def my_sleeping_function(random_base):
22     """Sleep as a function that will run within the DAG execution"""
23     time.sleep(random_base)
24
25
26 def connect_to_monery_and_print_aggregateinfo, mysleeping):
27     s = MyRange()
28     explore = [{"group": "I", "id": "baker", "month": "January", "year": 2000},
29               {"id": "apple", "year": 2000, "month": "I", "id": "baker", "year": 2000}, {"id": "I", "year": 2000, "month": "I", "id": "apple"},
30               {"id": "apple", "year": 2000, "month": "I", "id": "apple"}]
31     str = list(map(lambda x: x["id"], explore))
32     result = list("Yes No" * len(str)) and for str in explore, mysleeping()
33     print(result)
34     return "Whatever you return gets printed on the logs"
35
36 run_this = PythonOperator(
37     task_id="connect_to_monery_and_print_aggregate",
38     provide_context=True,
39     python_callable=connect_to_monery_and_print_aggregate,
40     dag=dag)
41
42
43 for i in range(10):
44     ...
45     Generating 10 sleeping task, sleeping from 0 to 9 seconds
46     respectively
47     ...
48     task = PythonOperator(
49         task_id="sleep_for_10seconds",
50         python_callable=sleeping_function,
51         kwargs={"random_base": i},
52         dag=dag)
53     task.set_upstream(run_this)
```

example_monary_operator.py

```
from __future__ import print_function
from builtins import range
from airflow.operators import PythonOperator
from airflow.models import DAG
from datetime import datetime, timedelta
import time
from monary import Monary

seven_days_ago = datetime.combine(datetime.today() - timedelta(7),
                                  datetime.min.time())

default_args = {
    'owner': 'airflow',
    'start_date': seven_days_ago,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(dag_id='example_monary_operator', default_args=default_args)

def my_sleeping_function(random_base):
    '''This is a function that will run within the DAG execution'''
    time.sleep(random_base)
```


example_monary_operator.py

```
from __future__ import print_function
from builtins import range
from airflow.operators import PythonOperator
from airflow.models import DAG
from datetime import datetime, timedelta
import time
from monary import Monary
```

IMPORTS

```
seven_days_ago = datetime.combine(datetime.today() - timedelta(7),
                                   datetime.min.time())

default_args = {
    'owner': 'airflow',
    'start_date': seven_days_ago,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(dag_id='example_monary_operator', default_args=default_args)

def my_sleeping_function(random_base):
    '''This is a function that will run within the DAG execution'''
    time.sleep(random_base)
```

example_monary_operator.py

```
from __future__ import print_function
from builtins import range
from airflow.operators import PythonOperator
from airflow.models import DAG
from datetime import datetime, timedelta
import time
from monary import Monary
```

```
seven_days_ago = datetime.combine(datetime.today() - timedelta(7),
                                   datetime.min.time())
```

```
default_args = {
    'owner': 'airflow',
    'start_date': seven_days_ago,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}
```

```
dag = DAG(dag_id='example_monary_operator', default_args=default_args)
```

```
def my_sleeping_function(random_base):
    '''This is a function that will run within the DAG execution'''
    time.sleep(random_base)
```

SETTINGS

example_monary_operator.py

```
from __future__ import print_function
from builtins import range
from airflow.operators import PythonOperator
from airflow.models import DAG
from datetime import datetime, timedelta
import time
from monary import Monary

seven_days_ago = datetime.combine(datetime.today() - timedelta(7),
                                  datetime.min.time())

default_args = {
    'owner': 'airflow',
    'start_date': seven_days_ago,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(dag_id='example_monary_operator', default_args=default_args)

def my_sleeping_function(random_base):
    '''This is a function that will run within the DAG execution'''
    time.sleep(random_base)
```

DAG &
Functions

example_monary_operator.py

```
from __future__ import print_function
from builtins import range
from airflow.operators import PythonOperator
from airflow.models import DAG
from datetime import datetime, timedelta
import time
from monary import Monary

seven_days_ago = datetime.combine(datetime.today() - timedelta(7),
                                   datetime.min.time())

default_args = {
    'owner': 'airflow',
    'start_date': seven_days_ago,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(dag_id='example_monary_operator', default_args=default_args)

def my_sleeping_function(random_base):
    '''This is a function that will run within the DAG execution'''
    time.sleep(random_base)
```

example_monary_operator.py

```
def connect_to_monary_and_print_aggregation(ds, **kwargs):
    m = Monary()
    pipeline = [{"$group": {"_id": "$state", "totPop": {"$sum":
"$pop"}}}]
    states, population = m.aggregate("zips", "data", pipeline, ["_id",
"totPop"], ["string:2", "int64"])
    strs = list(map(lambda x: x.decode("utf-8"), states))
    result = list("%s: %d" % (state, pop) for (state, pop) in
zip(strs, population))
    print (result)
    return 'Whatever you return gets printed in the logs'

run_this = PythonOperator(
    task_id='connect_to_monary_and_print_aggregation',
    provide_context=True,
    python_callable=connect_to_monary_and_print_aggregation,
    dag=dag)
```

example_monary_operator.py

AGGREGATION

```
def connect_to_monary_and_print_aggregation(ds, **kwargs):  
    m = Monary()  
    pipeline = [{"$group": {"_id": "$state", "totPop": {"$sum":  
"$pop"}}}]  
    states, population = m.aggregate("zips", "data", pipeline, ["_id",  
"totPop"], ["string:2", "int64"])  
    strs = list(map(lambda x: x.decode("utf-8"), states))  
    result = list("%s: %d" % (state, pop) for (state, pop) in  
zip(strs, population))  
    print (result)  
    return 'Whatever you return gets printed in the logs'  
  
run_this = PythonOperator(  
    task_id='connect_to_monary_and_print_aggregation',  
    provide_context=True,  
    python_callable=connect_to_monary_and_print_aggregation,  
    dag=dag)
```


example_monary_operator.py

```
def connect_to_monary_and_print_aggregation(ds, **kwargs):
    m = Monary()
    pipeline = [{"$group": {"_id": "$state", "totPop": {"$sum":
"$pop"}}}]
    states, population = m.aggregate("zips", "data", pipeline, ["_id",
"totPop"], ["string:2", "int64"])
    strs = list(map(lambda x: x.decode("utf-8"), states))
    result = list("%s: %d" % (state, pop) for (state, pop) in
zip(strs, population))
    print (result)
    return 'Whatever you return gets printed in the logs'
```

DAG SETUP

```
run_this = PythonOperator(
    task_id='connect_to_monary_and_print_aggregation',
    provide_context=True,
    python_callable=connect_to_monary_and_print_aggregation,
    dag=dag)
```

example_monary_operator.py

```
def connect_to_monary_and_print_aggregation(ds, **kwargs):
    m = Monary()
    pipeline = [{"$group": {"_id": "$state", "totPop": {"$sum":
"$pop"}}}]
    states, population = m.aggregate("zips", "data", pipeline, ["_id",
"totPop"], ["string:2", "int64"])
    strs = list(map(lambda x: x.decode("utf-8"), states))
    result = list("%s: %d" % (state, pop) for (state, pop) in
zip(strs, population))
    print (result)
    return 'Whatever you return gets printed in the logs'

run_this = PythonOperator(
    task_id='connect_to_monary_and_print_aggregation',
    provide_context=True,
    python_callable=connect_to_monary_and_print_aggregation,
    dag=dag)
```

example_monary_operator.py

```
for i in range(10):  
    '''  
        Generating 10 sleeping tasks, sleeping from 0 to 9  
seconds  
respectively  
    '''  
    task = PythonOperator(  
        task_id='sleep_for_'+str(i),  
        python_callable=my_sleeping_function,  
        op_kwargs={'random_base': i},  
        dag=dag)  
    task.set_upstream(run_this)
```


example_monary_operator.py

LOOP

```
for i in range(10):  
    '''
```

```
    Generating 10 sleeping tasks, sleeping from 0 to 9  
seconds  
    respectively  
    '''
```

```
    task = PythonOperator(  
        task_id='sleep_for_'+str(i),  
        python_callable=my_sleeping_function,  
        op_kwargs={'random_base': i},  
        dag=dag)  
    task.set_upstream(run_this)
```

example_monary_operator.py

```
for i in range(10):  
    '''
```

```
        Generating 10 sleeping tasks, sleeping from 0 to 9  
seconds  
        respectively  
    '''
```

```
    task = PythonOperator(  
        task_id='sleep_for_'+str(i),  
        python_callable=my_sleeping_function,  
        op_kwargs={'random_base': i},  
        dag=dag)  
    task.set_upstream(run_this)
```

DAG SETUP

example_monary_operator.py

```
for i in range(10):  
    '''  
        Generating 10 sleeping tasks, sleeping from 0 to 9  
seconds  
respectively  
    '''  
    task = PythonOperator(  
        task_id='sleep_for_'+str(i),  
        python_callable=my_sleeping_function,  
        op_kwargs={'random_base': i},  
        dag=dag)  
    task.set_upstream(run_this)
```


example_monary_operator.py

```
$ airflow backfill example_monary_operator -s 2015-01-01 -e 2015-01-02
```

```
2015-10-08 15:08:09,532 INFO - Filling up the DagBag from /Users/braz/airflow/dags
```

```
2015-10-08 15:08:09,532 INFO - Importing /usr/local/lib/python2.7/site-packages/airflow/example_dags/example_branch_operator.py
```

```
2015-10-08 15:08:09,533 INFO - Loaded DAG <DAG: example_branch_operator>
```

```
2015-10-08 15:08:09,533 INFO - Importing /usr/local/lib/python2.7/site-packages/airflow/example_dags/example_http_operator.py
```

```
2015-10-08 15:08:09,534 INFO - Loaded DAG <DAG: example_http_operator>
```

```
2015-10-08 15:08:09,534 INFO - Importing /usr/local/lib/python2.7/site-packages/airflow/example_dags/example_monary_operator.py
```

```
2015-10-08 15:08:09,535 INFO - Loaded DAG <DAG: example_monary_operator>
```

```
2015-10-08 15:08:09,535 INFO - Importing /usr/local/lib/python2.7/site-packages/airflow/example_dags/example_pymongo_operator.py
```

```
2015-10-08 15:08:09,719 INFO - Loaded DAG <DAG: example_pymongo_operator>
```

```
2015-10-08 15:08:09,719 INFO - Importing /usr/local/lib/python2.7/site-packages/airflow/example_dags/example_python_operator.py
```

```
2015-10-08 15:08:09,738 INFO - Loaded DAG <DAG: example_python_operator>
```

```
2015-10-08 15:08:09,738 INFO - Importing /usr/local/lib/python2.7/site-packages/airflow/example_dags/example_xcom.py
```

```
2015-10-08 15:08:09,739 INFO - Loaded DAG <DAG: example_xcom>
```

```
2015-10-08 15:08:09,739 INFO - Importing /usr/local/lib/python2.7/site-packages/airflow/example_dags/tutorial.py
```

```
2015-10-08 15:08:09,739 INFO - Loaded DAG <DAG: tutorial>
```

```
2015-10-08 15:08:09,740 INFO - Adding to queue: airflow run example_monary_operator connect_to_monary_and_print_aggregation 2015-01-02T00:00:00 --local -sd DAGS_FOLDER/example_dags/
```

```
example_monary_operator.py -s 2015-01-01T00:00:00
```

```
2015-10-08 15:08:09,865 INFO - Adding to queue: airflow run example_monary_operator connect_to_monary_and_print_aggregation 2015-01-01T00:00:00 --local -sd DAGS_FOLDER/example_dags/
```

```
example_monary_operator.py -s 2015-01-01T00:00:00
```

```
2015-10-08 15:08:14,765 INFO - [backfill progress] waiting: 22 | succeeded: 0 | kicked_off: 2 | failed: 0 | skipped: 0
```

```
2015-10-08 15:08:19,765 INFO - commandairflow run example_monary_operator connect_to_monary_and_print_aggregation 2015-01-02T00:00:00 --local -sd DAGS_FOLDER/example_dags/
```

```
example_monary_operator.py -s 2015-01-01T00:00:00
```

```
Logging into: /Users/braz/airflow/logs/example_monary_operator/connect_to_monary_and_print_aggregation/2015-01-02T00:00:00
```

```
[u'WA': 1000002', u'IL': 1100220', u'CA': 20751000', u'OR': 2012021', u'NM': 1515000', u'UT': 1722050', u'OK': 0115505', u'LA': 1217505', u'NE': 1570100', u'TX': 10001001', u'MO': 5110010', u'MT': 700010', u'ND': 000212',  
u'AK': 544698', u'SD': 695397', u'DC': 606900', u'MN': 4372982', u'ID': 1006749', u'KY': 3675484', u'WI': 4891769', u'TN': 4876457', u'AZ': 3665228', u'CO': 3293755', u'KS': 2475285', u'MS': 2573216', u'FL': 12686644', u'IA':  
0770100', u'IN': 0000007', u'VA': 0101170', u'NV': 5511100', u'ME': 1000010', u'WV': 1700110', u'MD': 1701070', u'GA': 0170010', u'NH': 1100050', u'NY': 1001000', u'DE': 000100', u'AL': 1010507', u'CT': 0007110', u'QO:
```

Building your pipeline

DAG: example_pymongo_and_aggregate_operator

Graph View Tree View Task Duration Landing Times Gantt Code

Run: 2015-10-09 00:00:00 Layout: Left->Right Go Search for...

EmailOperator PythonOperator success running failed no status

```
graph LR; A[connect_to_mongodb_and_aggregate_day] --> B[connect_to_mongodb_and_aggregate_hour]; B --> C[send_email_notification_flow_successful];
```

```
pipeline = [{"$project": {'page': '$PAGE', 'time': {'y': {'$year': '$DATE'}, 'm': {'$month': '$DATE'}, 'day': {'$dayOfMonth': '$DATE'}}}}, {'$group': {'_id': {'p': '$page', 'y': '$time.y', 'm': '$time.m', 'd': '$time.day'}, 'daily': {'$sum': 1}}}], {'$out': tmp_created_collection_per_day_name}]
```

Building your pipeline

```
mongoexport -d test -c page_per_day_hits_tmp --type=csv -  
f=_id,daily -o page_per_day_hits_tmp.csv
```

```
_id.d,_id.m,_id.y,_id.p,daily
```

```
3,2,2014,cart.do,115
```

```
4,2,2014,cart.do,681
```

```
5,2,2014,cart.do,638
```

```
6,2,2014,cart.do,610
```

```
....
```

```
3,2,2014,cart/error.do,2
```

```
4,2,2014,cart/error.do,14
```

```
5,2,2014,cart/error.do,23
```

Building your pipeline

CONVERSION

```
mongoexport -d test -c page_per_day_hits_tmp --type=csv -  
f=_id,daily -o page_per_day_hits_tmp.csv
```

```
_id.d,_id.m,_id.y,_id.p,daily
```

```
3,2,2014,cart.do,115
```

```
4,2,2014,cart.do,681
```

```
5,2,2014,cart.do,638
```

```
6,2,2014,cart.do,610
```

```
....
```

```
3,2,2014,cart/error.do,2
```

```
4,2,2014,cart/error.do,14
```

```
5,2,2014,cart/error.do,23
```


Building your pipeline

```
mongoexport -d test -c page_per_day_hits_tmp --type=csv -  
f=_id,daily -o page_per_day_hits_tmp.csv
```

```
_id.d,_id.m,_id.y,_id.p,daily  
3,2,2014,cart.do,115  
4,2,2014,cart.do,681  
5,2,2014,cart.do,638  
6,2,2014,cart.do,610  
....  
3,2,2014,cart/error.do,2  
4,2,2014,cart/error.do,14  
5,2,2014,cart/error.do,23
```

CSV FILE CONTENTS

Building your pipeline

```
mongoexport -d test -c page_per_day_hits_tmp --type=csv -  
f=_id,daily -o page_per_day_hits_tmp.csv
```

```
_id.d,_id.m,_id.y,_id.p,daily
```

```
3,2,2014,cart.do,115
```

```
4,2,2014,cart.do,681
```

```
5,2,2014,cart.do,638
```

```
6,2,2014,cart.do,610
```

```
....
```

```
3,2,2014,cart/error.do,2
```

```
4,2,2014,cart/error.do,14
```

```
5,2,2014,cart/error.do,23
```

Visualising the results

```
In [1]: import pandas as pd
In [2]: import numpy as np
In [3]: import matplotlib.pyplot as plt
In [4]: df1 = pd.read_csv('page_per_day_hits_tmp.csv', names=['day', 'month',
'year', 'page', 'daily'], header=0)
```

Out[4]:

	day	month	year	page	daily
0	3	2	2014	cart.do	115
1	4	2	2014	cart.do	681
...	...	...	...	...	...
103	10	2	2014	stuff/logo.ico	3

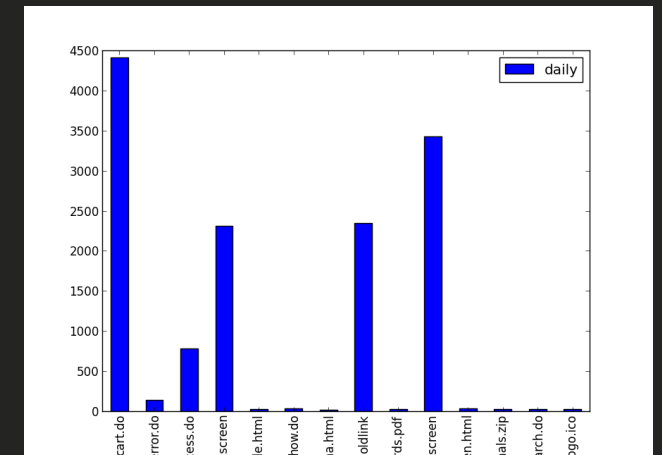
[104 rows x 5 columns]

```
In [5]: grouped = df1.groupby(['page'])
```

Out[5]: <pandas.core.groupby.DataFrameGroupBy object at 0x10f6b0dd0>

```
In [6]: grouped.agg({'daily': 'sum'}).plot(kind='bar')
```

Out[6]: <matplotlib.axes.AxesSubplot at 0x10f8f4d10>



Scikit-learn churn data

```
['State', 'Account Length', 'Area Code', 'Phone', "Int'l Plan", 'VMail Plan',  
'VMail Message', 'Day Mins', 'Day Calls', 'Day Charge', 'Eve Mins', 'Eve  
Calls', 'Eve Charge', 'Night Mins', 'Night Calls', 'Night Charge', 'Intl Mins',  
'Intl Calls', 'Intl Charge', 'CustServ Calls', 'Churn?']
```

	State	Account Length	Area Code	Phone	Intl Plan	VMail Plan	\
0	KS	128	415	382-4657	no	yes	
1	OH	107	415	371-7191	no	yes	
2	NJ	137	415	358-1921	no	no	
3	OH	84	408	375-9999	yes	no	

	Night Charge	Intl Mins	Intl Calls	Intl Charge	CustServ Calls	Churn?
0	11.01	10.0	3	2.70	1	False.
1	11.45	13.7	3	3.70	1	False.
2	7.32	12.2	5	3.29	0	False.
3	8.86	6.6	7	1.78	2	False.

Scikit-learn churn example

```
from __future__ import division
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json

from sklearn.cross_validation import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier as RF
%matplotlib inline
churn_df = pd.read_csv('churn.csv')
col_names = churn_df.columns.tolist()

print "Column names:"
print col_names

to_show = col_names[:6] + col_names[-6:]
```

Scikit-learn churn example

```
from __future__ import division
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json

from sklearn.cross_validation import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier as RF
%matplotlib inline

churn_df = pd.read_csv('churn.csv')
col_names = churn_df.columns.tolist()

print "Column names:"
print col_names

to_show = col_names[:6] + col_names[-6:]
```

IMPORTS

Scikit-learn churn example

```
from __future__ import division
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json
```

```
from sklearn.cross_validation import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier as RF
```

```
%matplotlib inline
```

```
churn_df = pd.read_csv('churn.csv')
col_names = churn_df.columns.tolist()
```

```
print "Column names:"
print col_names
```

```
to_show = col_names[:6] + col_names[-6:]
```

LOAD FILE / EXPLORE DATA

Scikit-learn churn example

```
from __future__ import division
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json

from sklearn.cross_validation import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier as RF
%matplotlib inline
churn_df = pd.read_csv('churn.csv')
col_names = churn_df.columns.tolist()

print "Column names:"
print col_names

to_show = col_names[:6] + col_names[-6:]
```


Scikit-learn churn example

```
print "\nSample data:"
churn_df[to_show].head(2)
# Isolate target data
churn_result = churn_df['Churn?']
y = np.where(churn_result == 'True.', 1, 0)
to_drop = ['State', 'Area Code', 'Phone', 'Churn?']
churn_feat_space = churn_df.drop(to_drop, axis=1)
# 'yes'/'no' has to be converted to boolean values
# NumPy converts these from boolean to 1. and 0. later
yes_no_cols = ["Int'l Plan", "VMail Plan"]
churn_feat_space[yes_no_cols] = churn_feat_space[yes_no_cols] == 'yes'

# Pull out features for future use
features = churn_feat_space.columns
X = churn_feat_space.as_matrix().astype(np.float)
scaler = StandardScaler()
X = scaler.fit_transform(X)
print "Feature space holds %d observations and %d features" % X.shape
print "Unique target labels:", np.unique(y)
```

Scikit-learn churn example

```
print "\nSample data:"
churn_df[to_show].head(2)
# Isolate target data
churn_result = churn_df['Churn?']
y = np.where(churn_result == 'True.', 1, 0)
to_drop = ['State', 'Area Code', 'Phone', 'Churn?']
churn_feat_space = churn_df.drop(to_drop, axis=1)
# 'yes'/'no' has to be converted to boolean values
# NumPy converts these from boolean to 1. and 0. later
yes_no_cols = ["Int'l Plan", "VMail Plan"]
churn_feat_space[yes_no_cols] = churn_feat_space[yes_no_cols] == 'yes'

# Pull out features for future use
features = churn_feat_space.columns
X = churn_feat_space.as_matrix().astype(np.float)
scaler = StandardScaler()
X = scaler.fit_transform(X)
print "Feature space holds %d observations and %d features" % X.shape
print "Unique target labels:", np.unique(y)
```

FORMAT
DATA FOR
USAGE

Scikit-learn churn example

```
print "\nSample data:"
churn_df[to_show].head(2)
# Isolate target data
churn_result = churn_df['Churn?']
y = np.where(churn_result == 'True.', 1, 0)
to_drop = ['State', 'Area Code', 'Phone', 'Churn?']
churn_feat_space = churn_df.drop(to_drop, axis=1)
# 'yes'/'no' has to be converted to boolean values
# NumPy converts these from boolean to 1. and 0. later
yes_no_cols = ["Int'l Plan", "VMail Plan"]
churn_feat_space[yes_no_cols] = churn_feat_space[yes_no_cols] == 'yes'

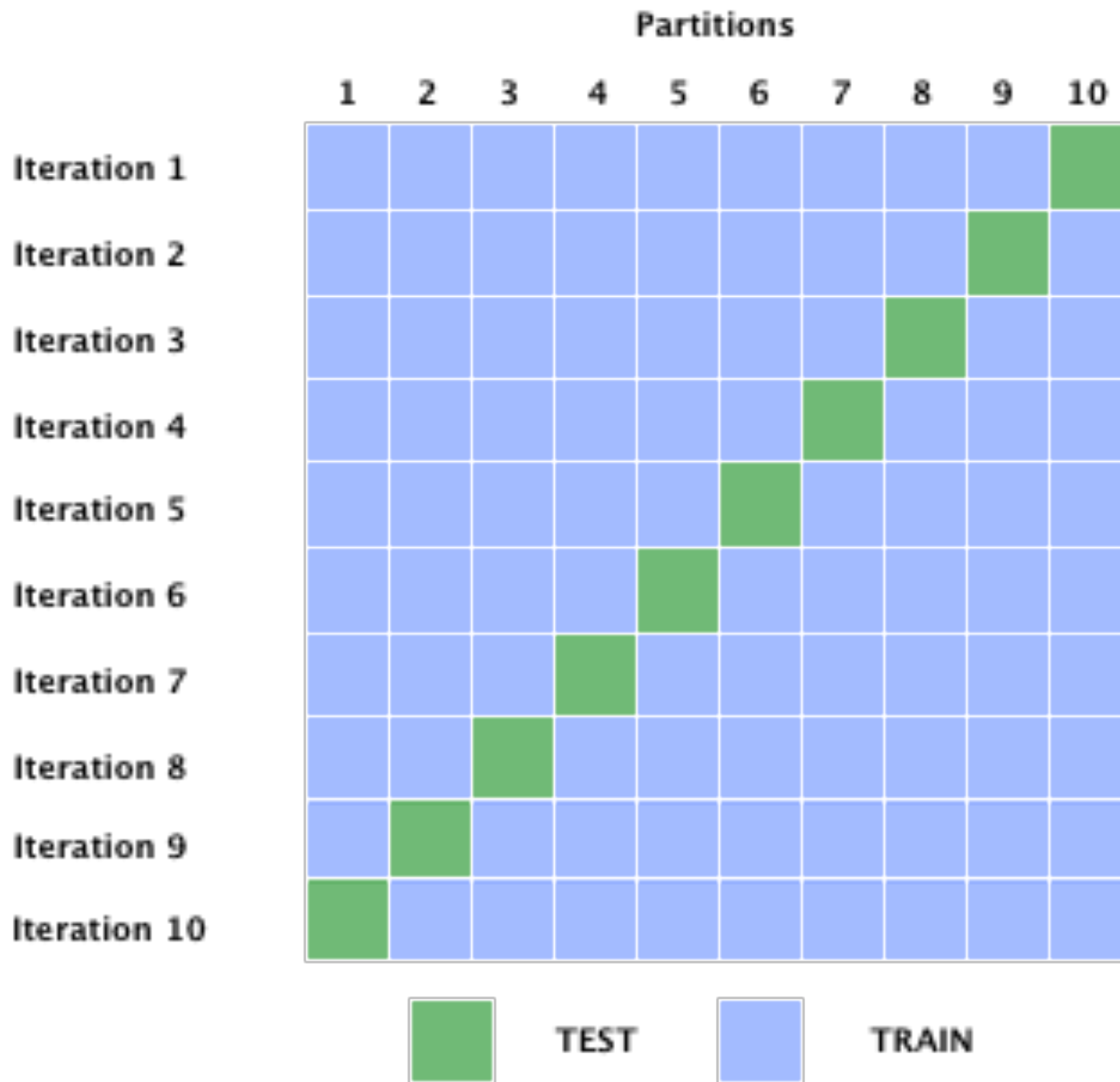
# Pull out features for future use
features = churn_feat_space.columns
X = churn_feat_space.as_matrix().astype(np.float)
scaler = StandardScaler()
X = scaler.fit_transform(X)
print "Feature space holds %d observations and %d features" % X.shape
print "Unique target labels:", np.unique(y)
```

FORMAT
DATA FOR
USAGE

Scikit-learn churn example

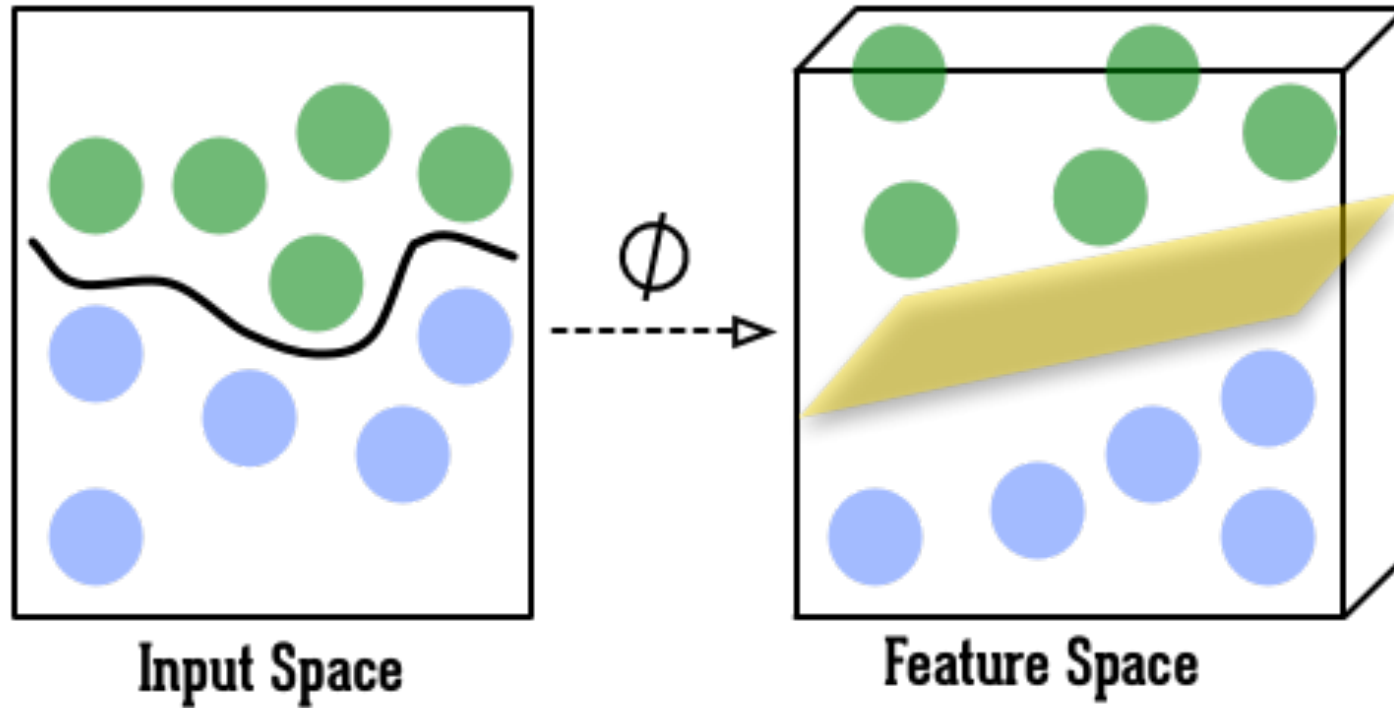
```
print "\nSample data:"
churn_df[to_show].head(2)
# Isolate target data
churn_result = churn_df['Churn?']
y = np.where(churn_result == 'True.', 1, 0)
to_drop = ['State', 'Area Code', 'Phone', 'Churn?']
churn_feat_space = churn_df.drop(to_drop, axis=1)
# 'yes'/'no' has to be converted to boolean values
# NumPy converts these from boolean to 1. and 0. later
yes_no_cols = ["Int'l Plan", "VMail Plan"]
churn_feat_space[yes_no_cols] = churn_feat_space[yes_no_cols] == 'yes'

# Pull out features for future use
features = churn_feat_space.columns
X = churn_feat_space.as_matrix().astype(np.float)
scaler = StandardScaler()
X = scaler.fit_transform(X)
print "Feature space holds %d observations and %d features" % X.shape
print "Unique target labels:", np.unique(y)
```

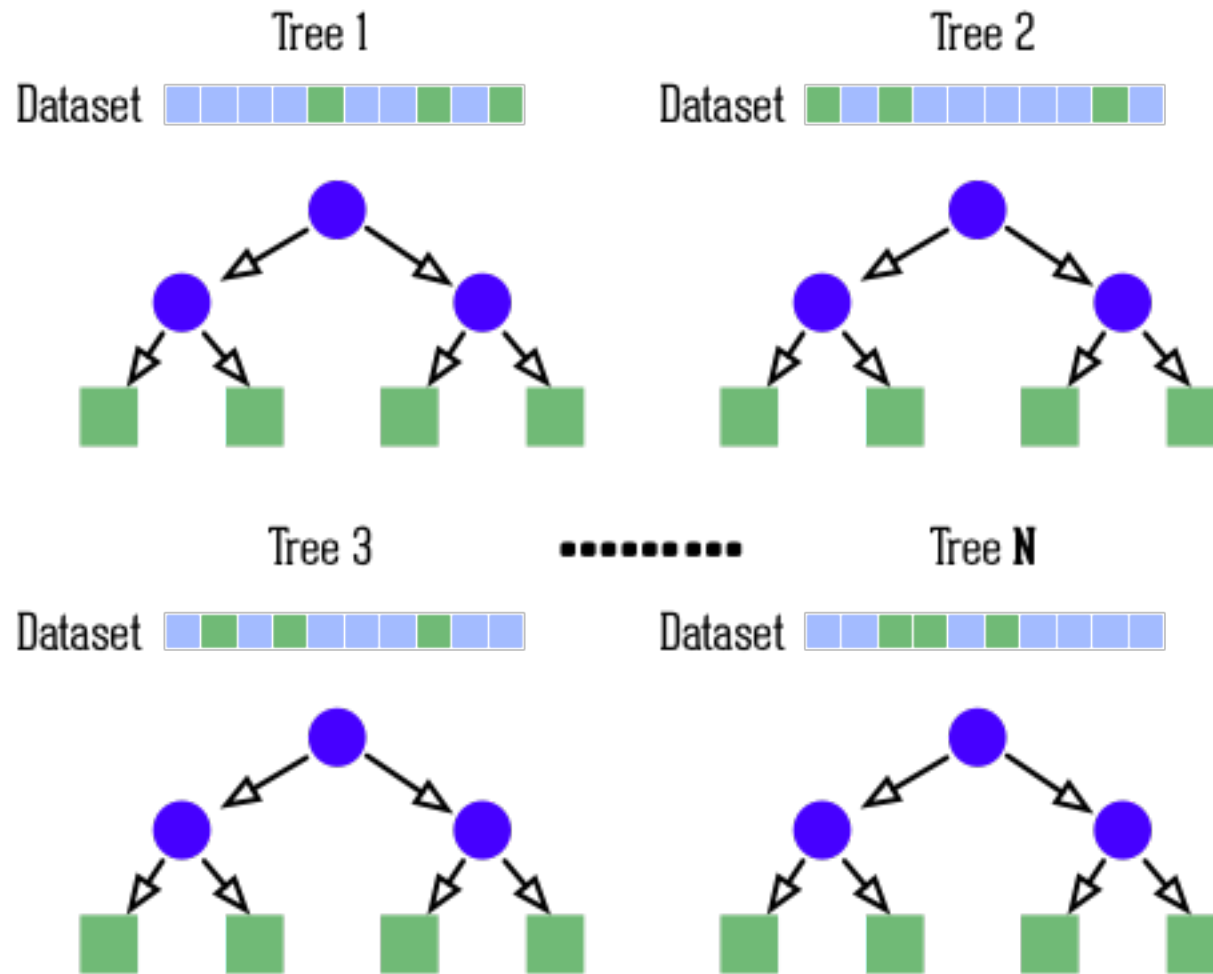
**10
Cross
Fold**

Support Vector Machine



A list of number [a n -dimensional vector] and transform the points into higher dimensions so it is easier to separate them using a $[n-1]$ dimensional hyperplane.

Random Forest



Traverse each tree and at each node in a tree:

- i. Select **m** random predictor variables from available set
- ii. Use the variable with best split [use objective function]
- iii. Move to next node in tree

Scikit-learn churn example

```
from sklearn.svm import SVC
from sklearn.ensemble import
RandomForestClassifier as RF
from sklearn.metrics import
average_precision_score
from sklearn.cross_validation import
KFold
```

```
def accuracy(y_true, y_pred):
    # NumPy interpretes True and
    False as 1. and 0.
    return np.mean(y_true == y_pred)
```

```
def run_cv(X, y, clf_class, **kwargs):
    # Construct a kfolds object
    kf =
    KFold(len(y), n_folds=3, shuffle=True)
    y_pred = y.copy()
```

```
# Iterate through folds
```

```
        for train_index, test_index in
kf:
            X_train, X_test =
X[train_index], X[test_index]
            y_train = y[train_index]
            clf = clf_class(**kwargs)
            clf.fit(X_train, y_train)
            y_pred[test_index] =
clf.predict(X_test)
        return y_pred
```

```
print "Support vector machines:"
print "%.3f" % accuracy(y,
run_cv(X, y, SVC))
print "Random forest:"
print "%.3f" % accuracy(y,
run_cv(X, y, RF))
```

Scikit-learn churn example

```
from sklearn.svm import SVC
from sklearn.ensemble import
RandomForestClassifier as RF
from sklearn.metrics import
average_precision_score
from sklearn.cross_validation import
KFold
```

```
def accuracy(y_true, y_pred):
    # NumPy interpretes True and
    False as 1. and 0.
    return np.mean(y_true == y_pred)
```

```
def run_cv(X, y, clf_class, **kwargs):
    # Construct a kfolds object
    kf =
    KFold(len(y), n_folds=3, shuffle=True)
    y_pred = y.copy()

    # Iterate through folds
```

```
        for train_index, test_index in
kf:
            X_train, X_test =
X[train_index], X[test_index]
            y_train = y[train_index]
            clf = clf_class(**kwargs)
            clf.fit(X_train, y_train)
            y_pred[test_index] =
clf.predict(X_test)
        return y_pred
```

```
print "Support vector machines:"
print "%.3f" % accuracy(y,
run_cv(X, y, SVC))
print "Random forest:"
print "%.3f" % accuracy(y,
run_cv(X, y, RF))
```

Cross Fold
K=3

Scikit-learn churn example

```
from sklearn.svm import SVC
from sklearn.ensemble import
RandomForestClassifier as RF
from sklearn.metrics import
average_precision_score
from sklearn.cross_validation import
KFold
```

```
def accuracy(y_true,y_pred):
    # NumPy interpretes True and
    False as 1. and 0.
    return np.mean(y_true == y_pred)
```

```
def run_cv(X,y,clf_class,**kwargs):
    # Construct a kfolds object
    kf =
    KFold(len(y),n_folds=3,shuffle=True)
    y_pred = y.copy()
```

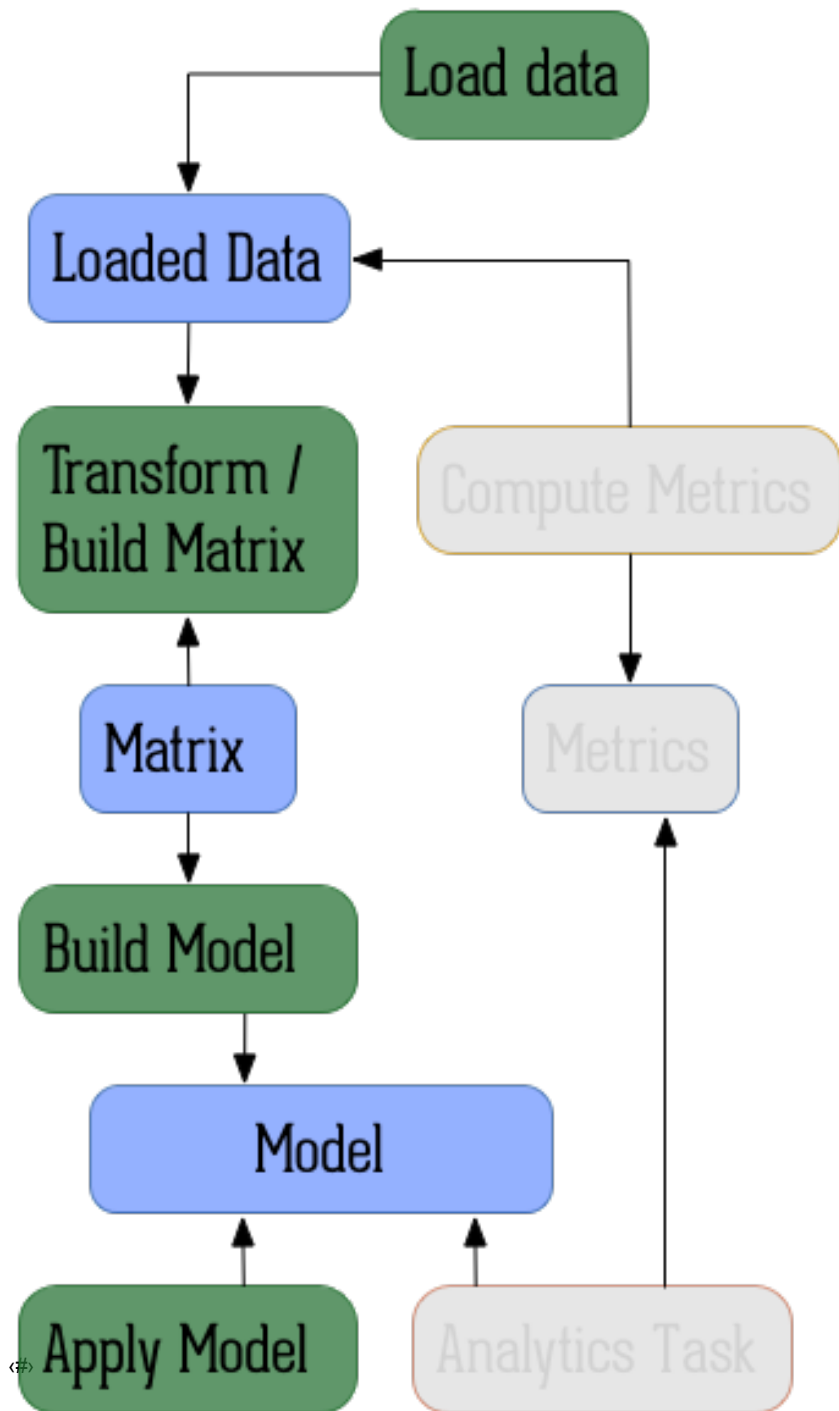
```
# Iterate through folds
```

```
        for train_index, test_index in
kf:
            X_train, X_test =
X[train_index], X[test_index]
            y_train = y[train_index]
            clf = clf_class(**kwargs)
            clf.fit(X_train,y_train)
            y_pred[test_index] =
clf.predict(X_test)
        return y_pred
```

```
print "Support vector machines:"
print "%.3f" % accuracy(y,
run_cv(X,y,SVC))
print "Random forest:"
print "%.3f" % accuracy(y,
run_cv(X,y,RF))
```

An example data pipeline:

- Data
- State
- Operations / Transformation



A wide-angle photograph of a distillery interior. The room is filled with rows of large, polished copper stills, each with a bulbous base and a curved neck. They are connected by a network of blue-painted metal pipes. The floor is a smooth, light grey concrete. In the foreground, a white metal safety railing with a blue handrail runs across the frame. The background shows more stills receding into the distance under a high, vaulted ceiling with industrial lighting. The overall atmosphere is one of a well-maintained, functional industrial facility.

Bringing it all
together

Systems



Speed



Photo Credits



<https://www.flickr.com/photos/rcbodden/2725787927/in>, Ray Bodden



<https://www.flickr.com/photos/iqremix/15390466616/in>, iqremix



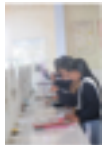
<https://www.flickr.com/photos/storem/129963685/in>, storem



<https://www.flickr.com/photos/diversey/15742075527/in>, Tony Webster



<https://www.flickr.com/photos/acwa/8291889208/in>, PEO ACWA



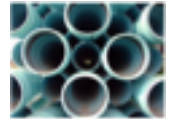
<https://www.flickr.com/photos/rowfoundation/8938333357/in>, Rajita Majumdar



<https://www.flickr.com/photos/54268887@N00/5057515604/in>, Rob Pearce



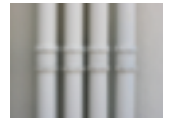
<https://www.flickr.com/photos/seeweb/6115445165/in>, seeweb



<https://www.flickr.com/photos/98640399@N08/9290143742/in>, Barta IV



<https://www.flickr.com/photos/aisforangie/6877291681/in>, Angie Harms



<https://www.flickr.com/photos/jakerome/3551143912/in>, Jakerome



<https://www.flickr.com/photos/ifyr/1106390483/>, Jack Shainsky



<https://www.flickr.com/photos/rioncm/4643792436/in>, rioncm



<https://www.flickr.com/photos/druidsnectar/4605414895/in>, druidsnectar

Thanks!

Questions?

Eoin Brazil
eoin.brazil@mongodb.com