

@ZipfianAcademy

Data Engineering 101: Building your first data product

May 4th, 2014

Jonathan Dinu
Co-Founder, Zipfian Academy
jonathan@zipfianacademy.com
@clearspandex





- **whoami**
- Nws Rdr (News Reader)
- The What, Why, and How of Data Products
- Data Engineering
- Building a Pipeline
- Productionizing the Products
- Q&A

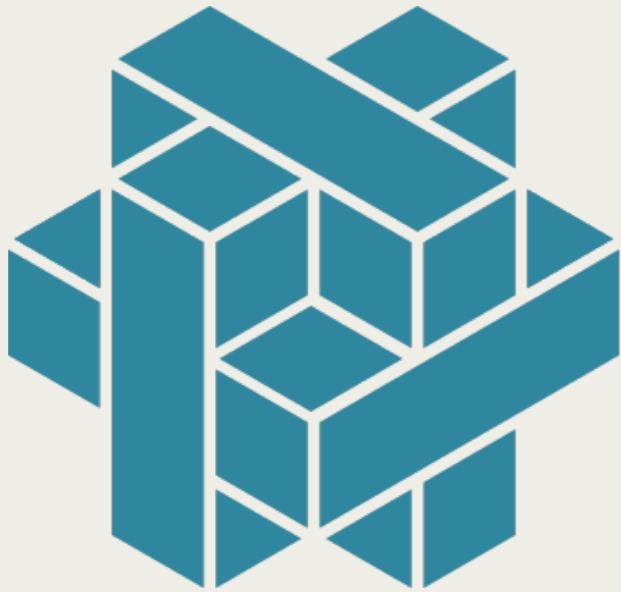
Formerly



Formerly



Currently



Zipfian
Academy

Today

Disclaimer:

All characters appearing in this presentation are fictitious. Any resemblance to real persons, living or dead, is purely coincidental.

Today

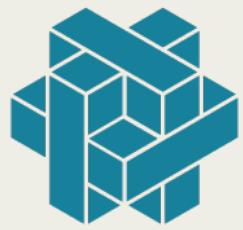
Disclaimer:

This presentation contains strong opinions that you may or may not agree with. All thoughts are my own.

Jonathan Dinu
Co-Founder, Zipfian Academy
jonathan@zipfianacademy.com
@clearspandex



- whoami
- Nws Rdr (News Reader)
- The What, Why, and How of Data Products
- Data Engineering
- Building a Pipeline
- Productionizing the Products
- Creating Value for Users
- Q&A



nwsrdr (News Reader)

The screenshot shows a web browser window with the following elements:

- Address Bar:** Shows the URL Technology News, Tips, Review....
- Toolbar:** Includes links for Microsoft News Center, Windows Home Server 2011, Xbox Live Account, gP, and others.
- Content Area:** Displays the text "nwsrdr".
- Left Sidebar:** A box titled "Get nwsrdr on your desktop" contains the text "Bookmark Button" and "Drag this button to your Bookmarks Bar." Below it is a yellow button labeled "+ nwsrdr".
- Right Sidebar:** Text "How to install the bo..." is partially visible.
- Bottom Right:** A yellow button labeled "+ nwsrdr".
- Bottom Left:** Text "When browsing the web simply click the +nwsrdr to save any page to nwsrdr".
- Bottom Center:** A screenshot of a Mac OS X desktop showing a browser window with the address bar set to getnews.com/bookmarklet. A red arrow points from the "+ nwsrdr" button in the sidebar to the bookmark bar, where a bookmark labeled "nwsrdr" is highlighted.
- Bottom Right:** Another yellow button labeled "+ nwsrdr".



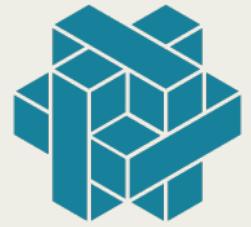
It's like Prismatic + Pocket + Google Reader (RIP) + Delicious!

- Auto-categorize Articles
- Find Similar Articles
- Recommend articles
- Suggest Feeds to Follow
- No Ads!

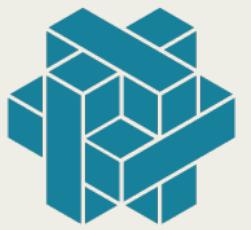


It's like Prismatic + Pocket + Google Reader (RIP) + Delicious!

- Naive Bayes (classification)
- Clustering (unsupervised learning)
- Collaborative Filtering
- Triangle Closing
- Real Business Model!

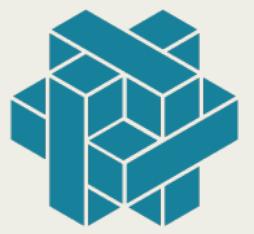


- whoami
- Nws Rdr (News Reader)
- The What, Why, and How of Data Products
- Data Engineering
- Building a Pipeline
- Productionizing the Products
- Q&A



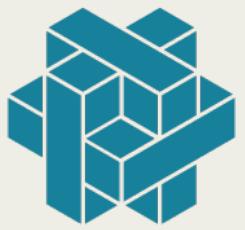
**Product Built on Data
(that you sell)**

OR



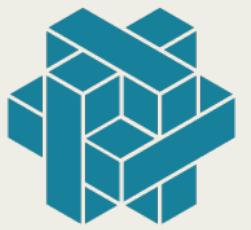
Product that Generates Data

OR



**Product that Generates Data
(that you sell)**

OR



**Product that Generates Data
(that you sell)**

OR

i.e. Facebook



Data Products

@goGIFGIF for updates.

GIFGIF

Search | Results | Data | About

Which better expresses **excitement?** [Change Question](#)



Info Share

NEITHER



Info Share

Achievements [Import](#) | [Export](#) | [Delete](#) Your votes: 44 **Analysis: Excitement** Global votes: 2,530,975

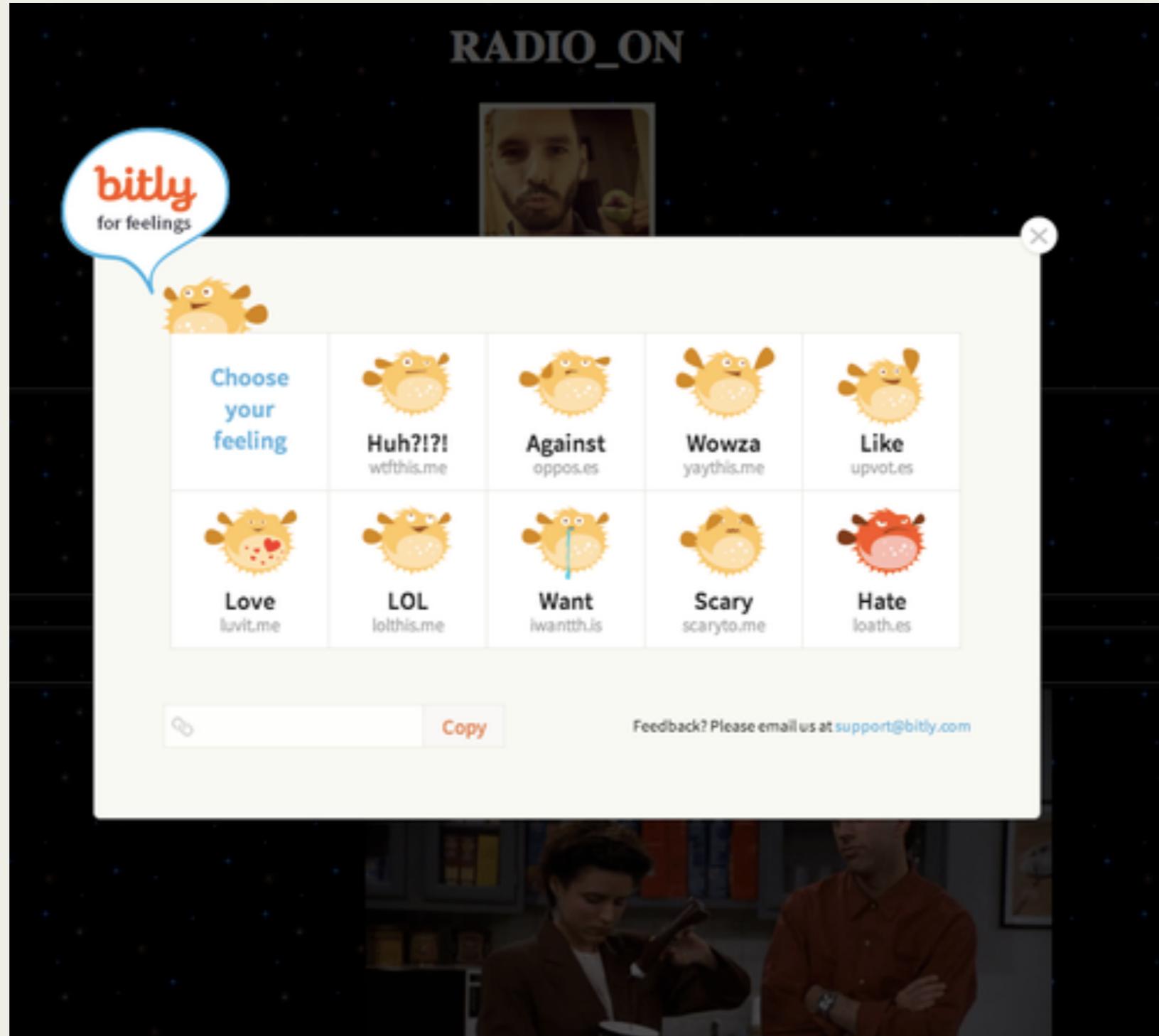
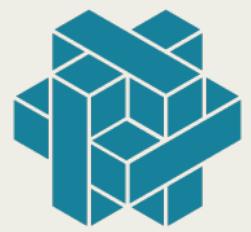
Best    Worst

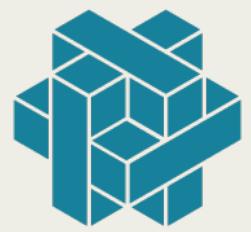


Data Products



Data Generating Products





Products that enhance a users' experience the more “data” a user provides

Ex: Recommender Systems



- whoami
- Nws Rdr (News Reader)
- The What, Why, and How of Data Products
- Data Engineering
- Building a Pipeline
- Productionizing the Products
- Q&A



Josh Wills

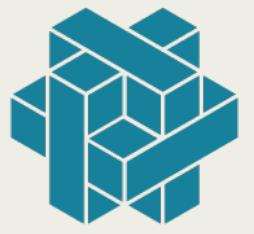
@josh_wills



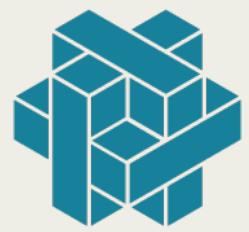
Follow

Data Scientist (n.): Person who is better at statistics than any software engineer and better at software engineering than any statistician.

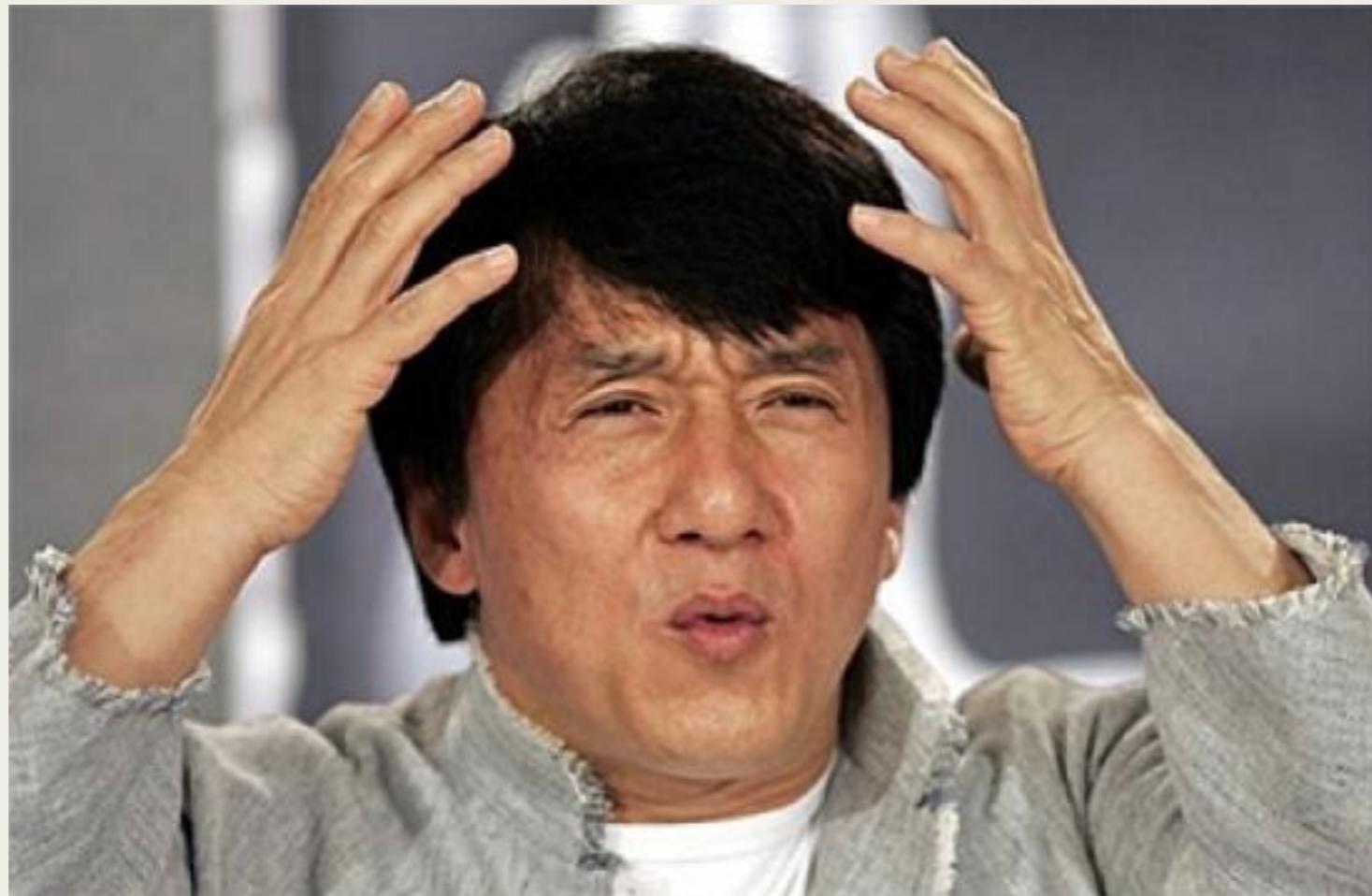
[Reply](#) [Retweet](#) [Favorite](#) [More](#)

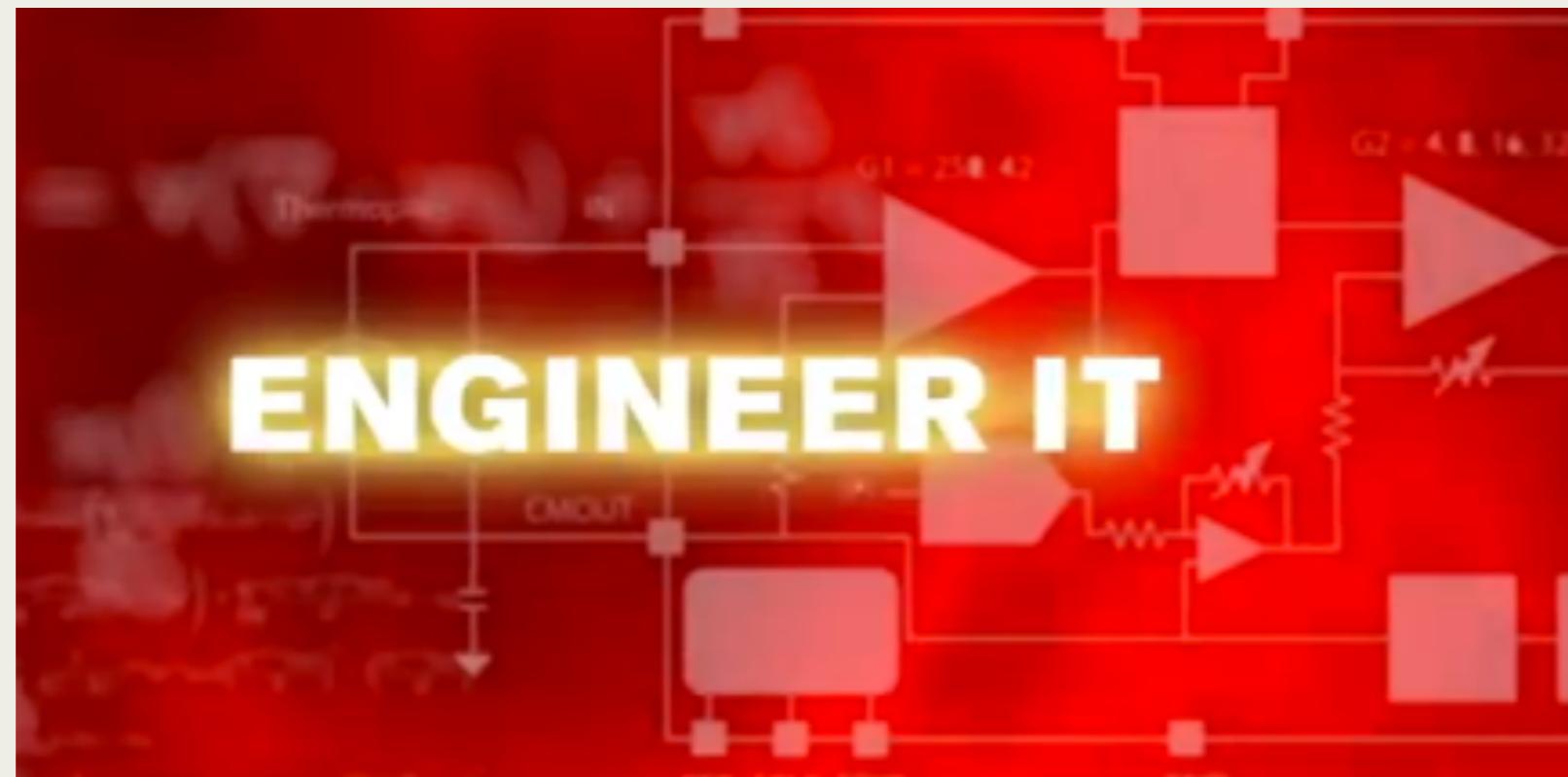
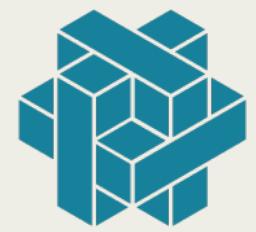


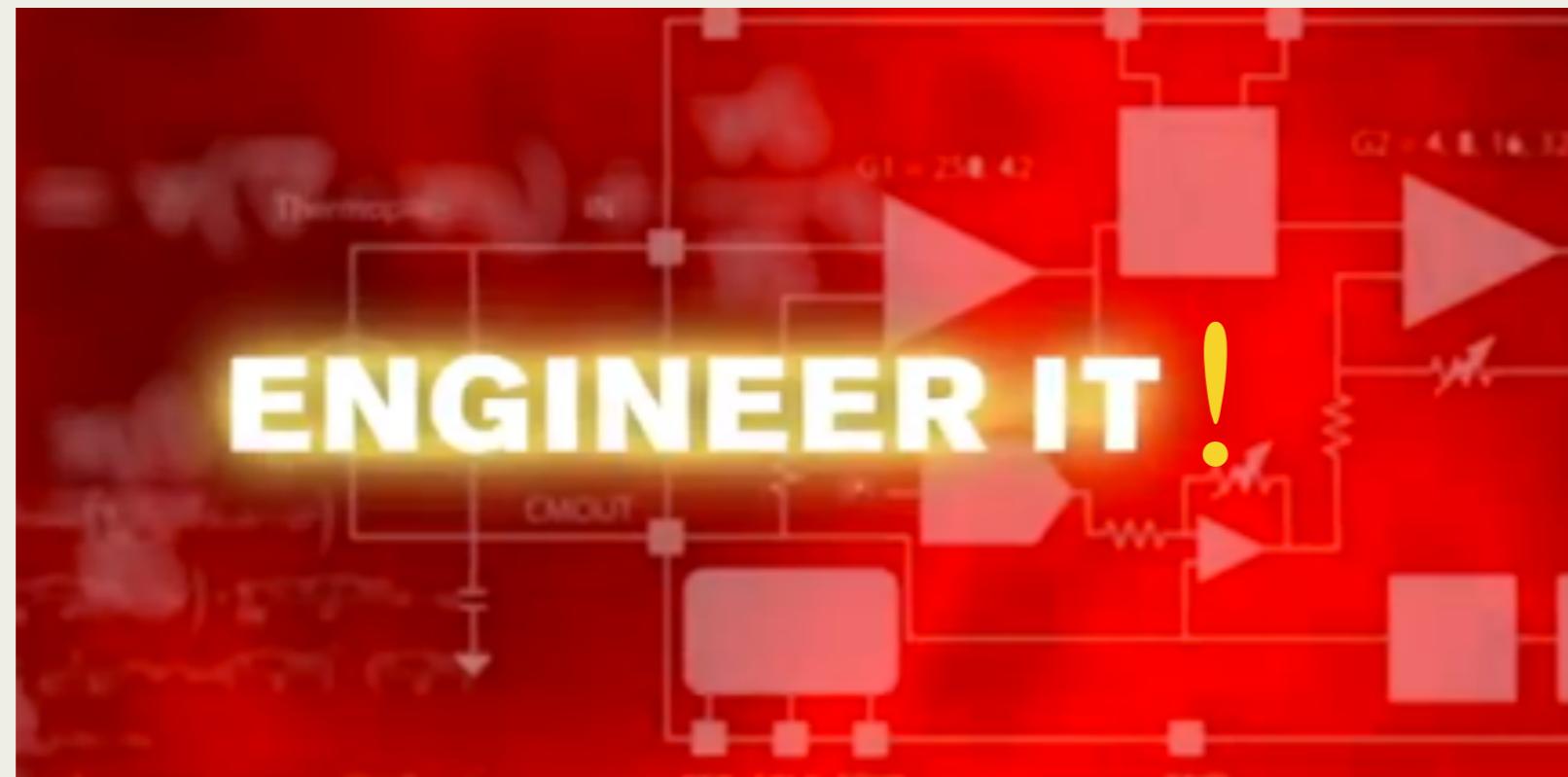
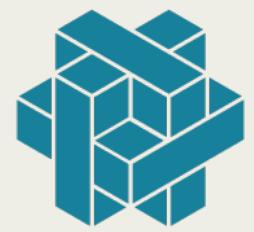
i.e. solve more problems than you create

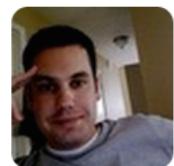
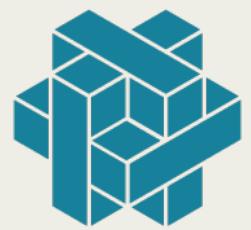


But.... How?!?!!?









Josh Wills

@josh_wills

Engineer

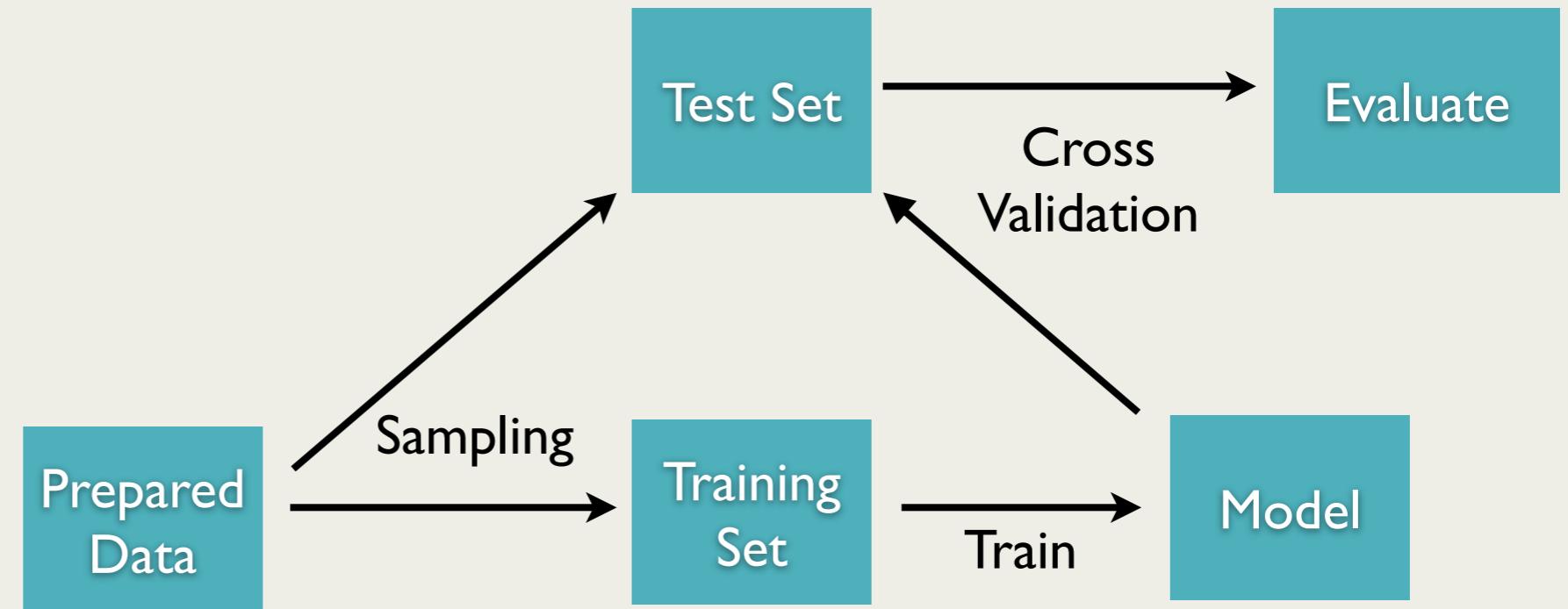
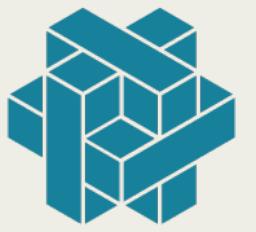


Follow

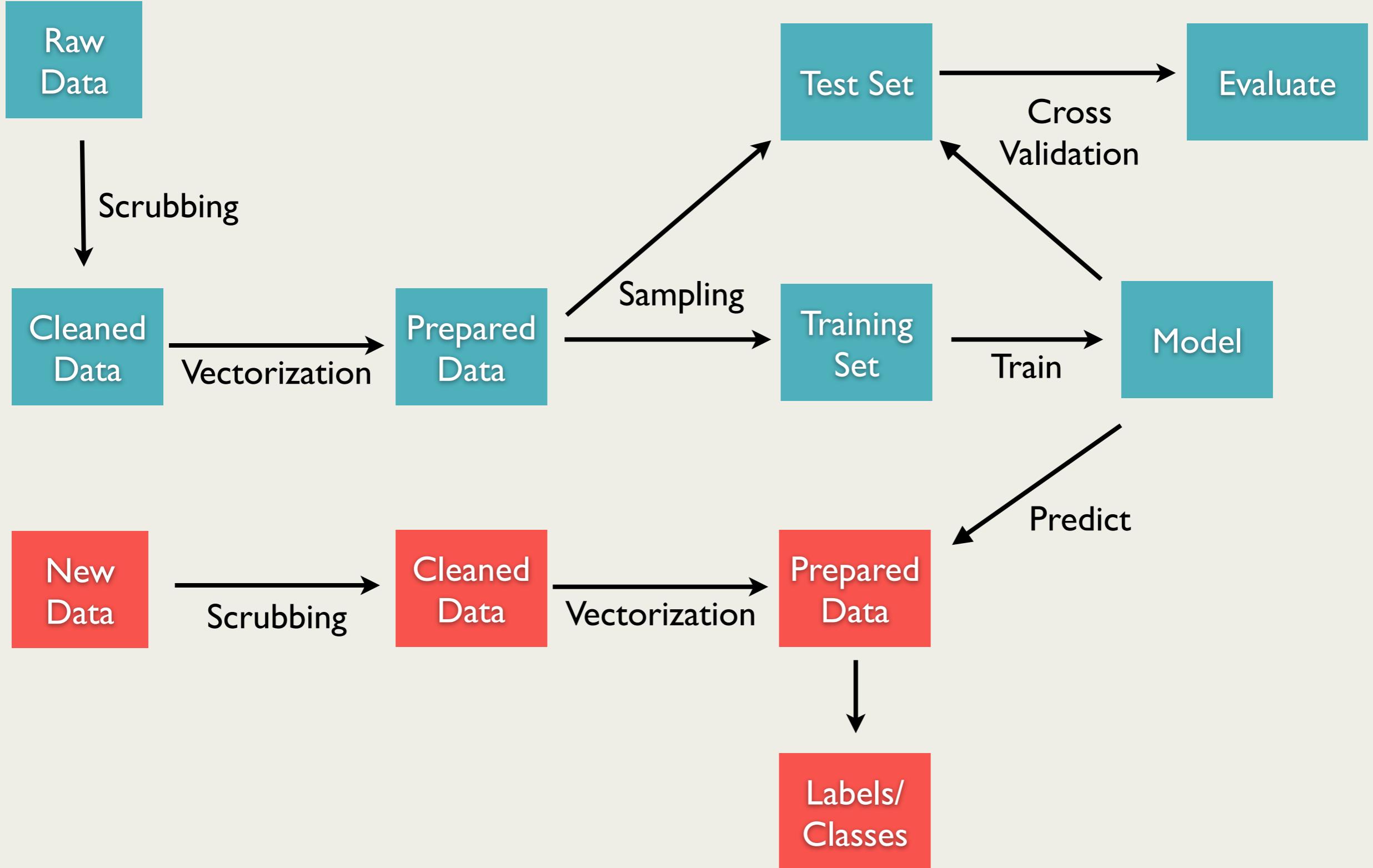
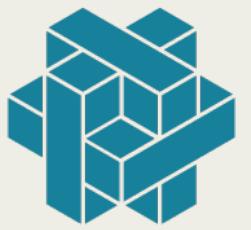
Data ~~Scientist~~ (n.): Person who is better at statistics than any software engineer and better at software engineering than any ~~statistician~~. Data Scientist

Reply Retweet Favorite More

Data Science



Data Engineering





- whoami
- Nws Rdr (News Reader)
- The What, Why, and How of Data Products
- Data Engineering
- Building a Pipeline
- Productionizing the Products
- Q&A



- Naive Bayes (classification)
- Clustering (unsupervised learning)
- Collaborative Filtering
- Triangle Closing
- Real Business Model



It's like Prismatic + Pocket + Google Reader (RIP) + Delicious!

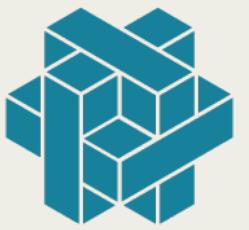
- Auto-categorize Articles
- Find Similar Articles
- Recommend articles
- No Ads!



It's like Prismatic + Pocket + Google Reader (RIP) + Delicious!

- Naive Bayes (classification)
- Clustering (unsupervised learning)
- Collaborative Filtering
- Triangle Closing
- Real Business Model!

How

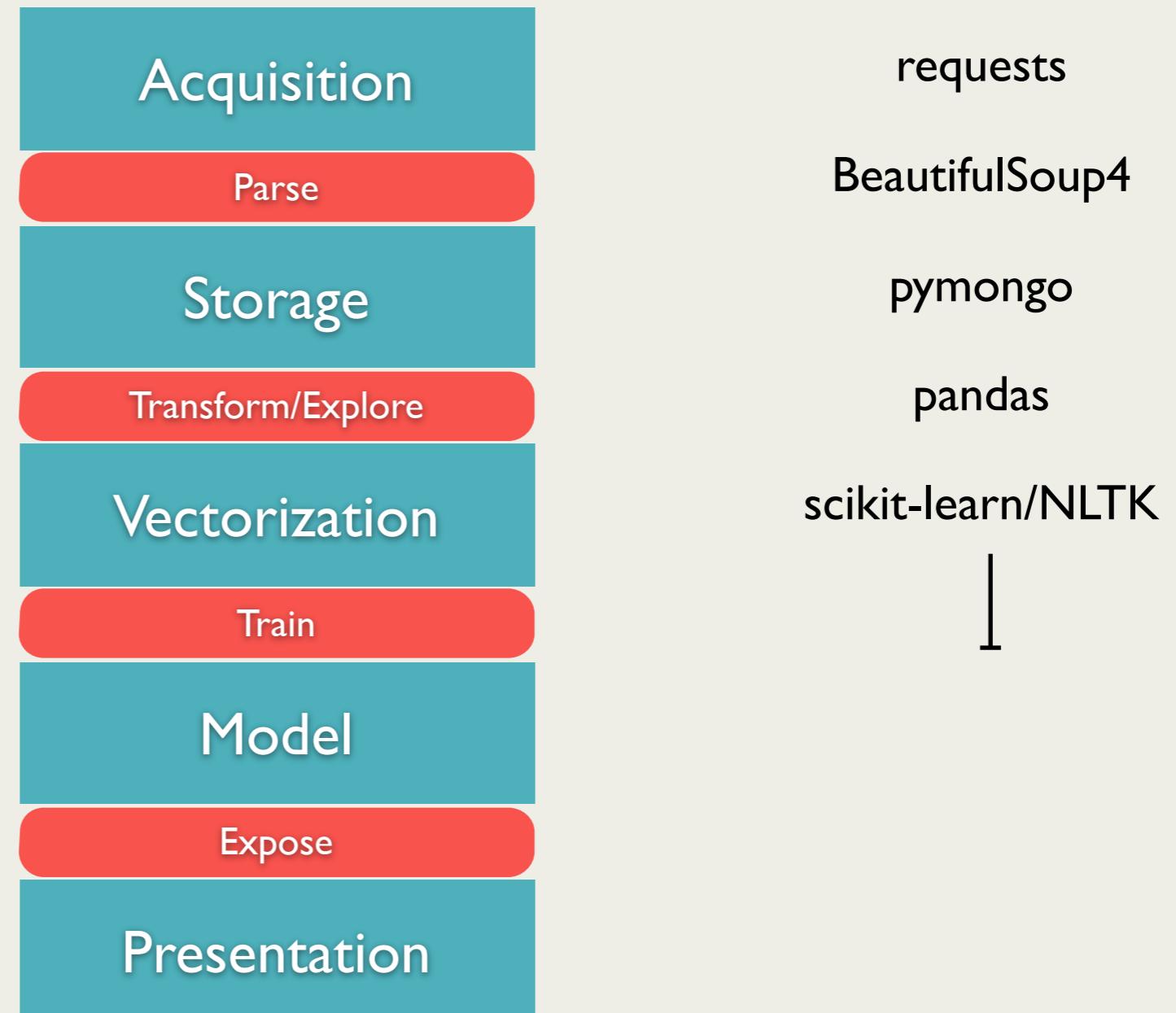


Abstraction (Cake)

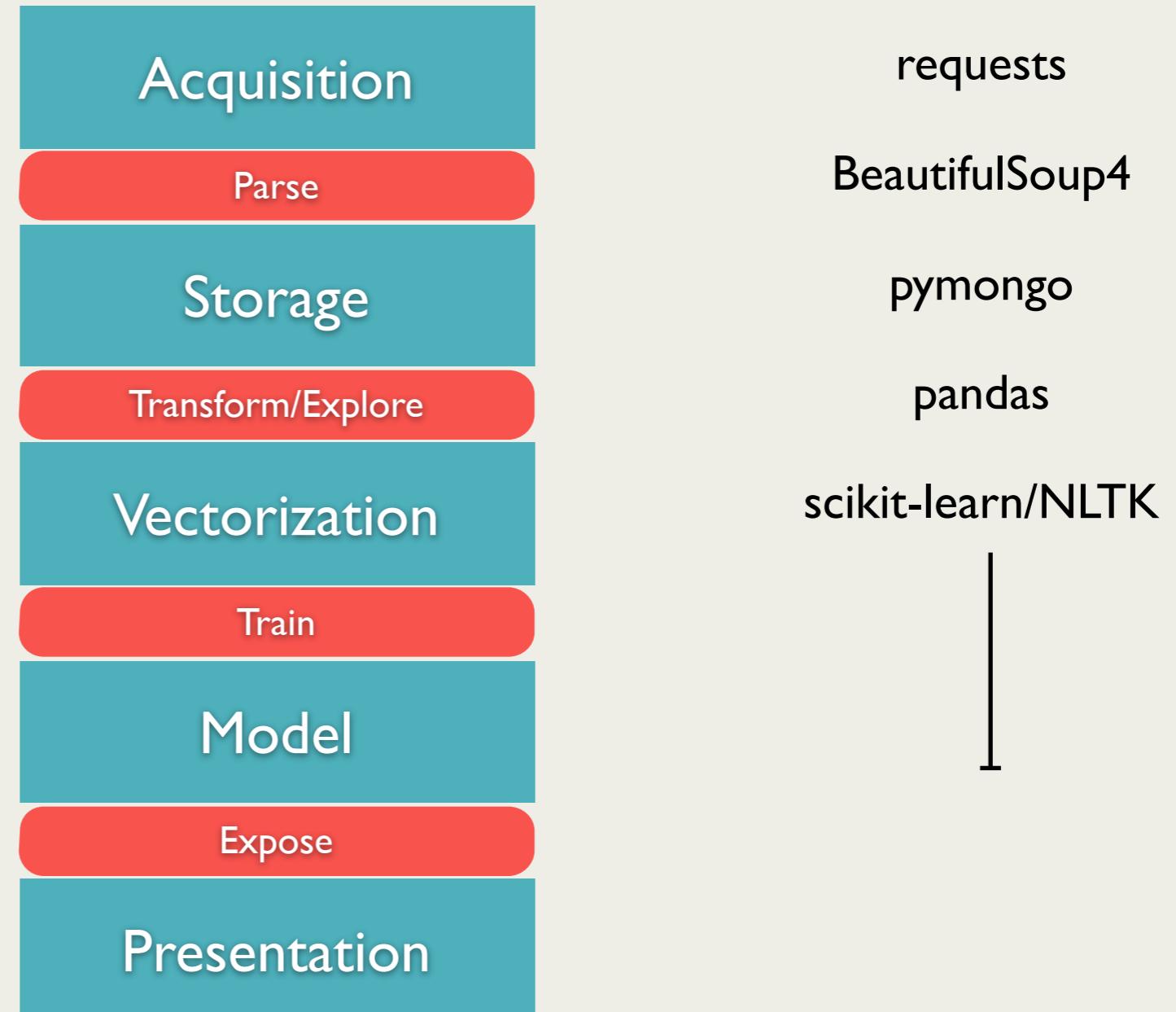
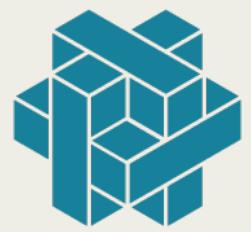


(ABK)

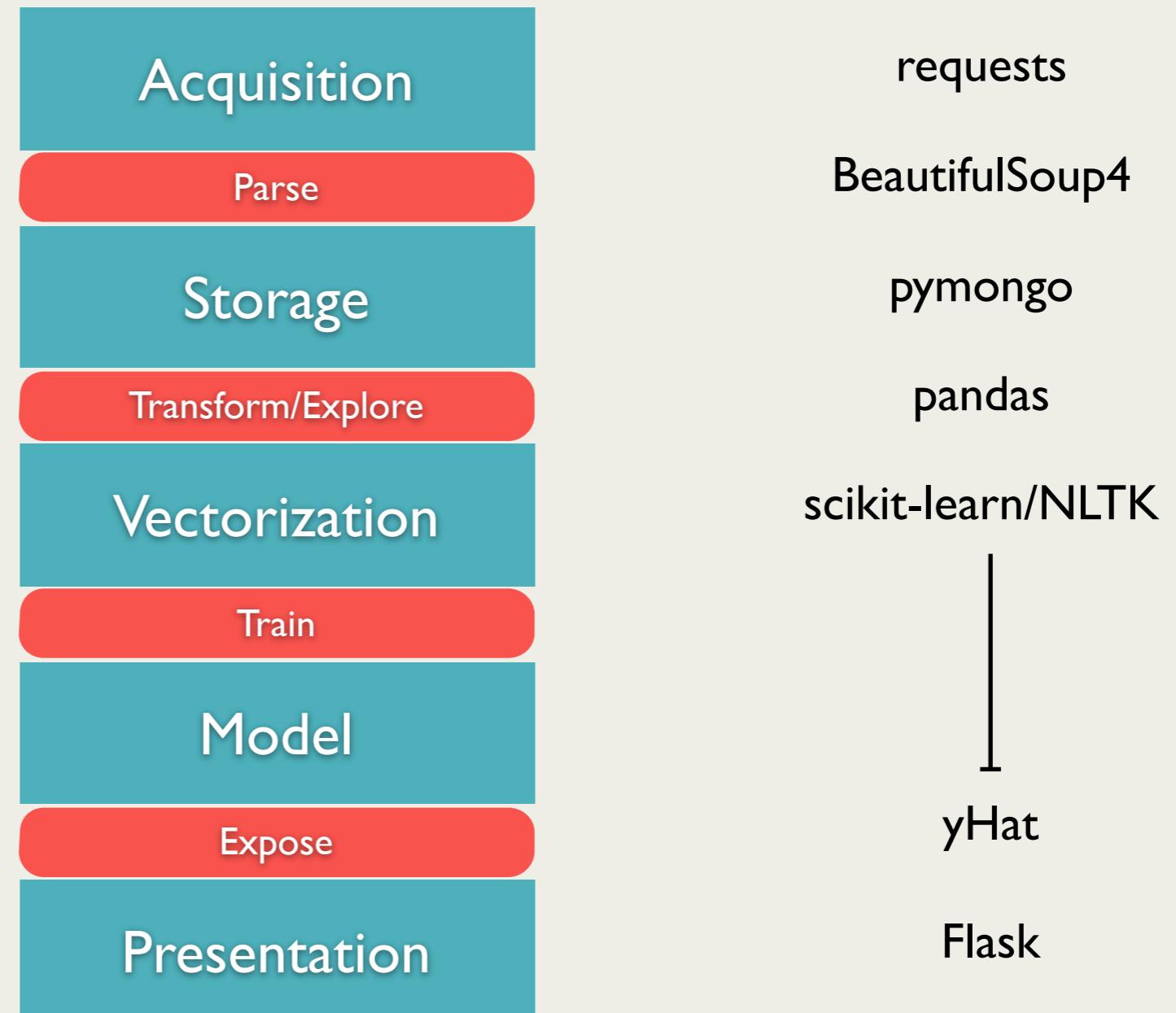
Obligatory Name Drop



Obligatory Name Drop



Obligatory Name Drop

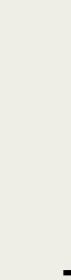


Obligatory Name Drop



At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)
Snakebite (HDFS)
mrjob or
Mortar (w/ Python UDF)
MLlib (pySpark)



yHat

Flask



Locally

requests
BeautifulSoup4
pymongo
pandas
scikit-learn/NLTK
yHat
Flask



Obligatory Name Drop



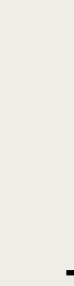
At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)

Snakebite (HDFS)

mrjob or
Mortar (w/ Python UDF)

MLlib (pySpark)



yHat

Flask



Locally

requests
BeautifulSoup4

pymongo

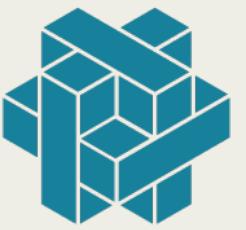
pandas

scikit-learn/NLTK

yHat

Flask

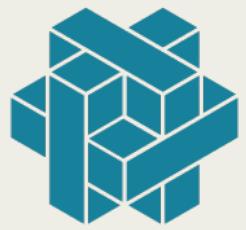




Iteration 0:

- Find out how much data
- Run locally
- Experiment

Acquire



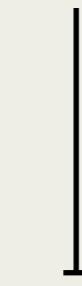
At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)

Snakebite (HDFS)

mrjob or
Mortar (w/ Python UDF)

MLlib (pySpark)



yHat

Flask



Locally

requests
BeautifulSoup4

pymongo

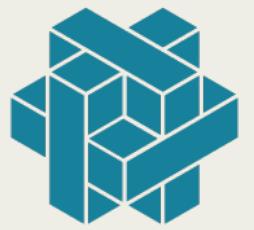
pandas

scikit-learn/NLTK

yHat

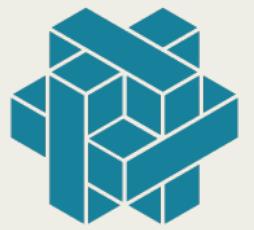
Flask





Retrieve Meta-data for **ALL** NYT articles

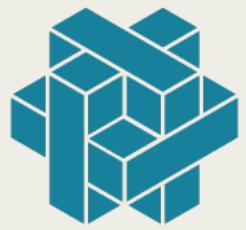
Acquire



```
api_key='xxxxxxxxxxxxxx'

url = 'http://api.nytimes.com/svc/search/v2/
articlesearch.json?fq=section_name.contains:( "Arts"
"Business Day" "Opinion" "Sports" "U.S."
"World")&sort=newest&api-key=' + api_key

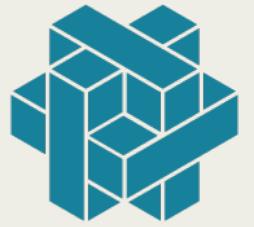
# make an API request
api = requests.get(url)
```



```
# parse resulting JSON and insert into a mongoDB collection
for content in api.json()['response']['docs']:
    if not collection.find_one(content):
        collection.insert(content)

# only returns 10 per page
"There are only %i docuemtns returned 0_o" % \
len(api.json()['response']['docs'])
```

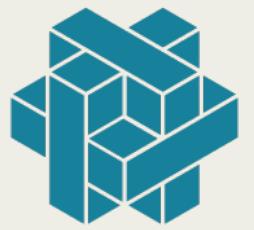
Acquire



```
# there are many more than 10 articles however
total_art = articles_left = api.json()['response']['meta']['hits']

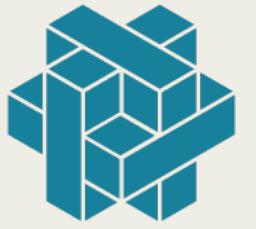
print "There are currently %s articles in the NYT archive" % total_art

#=> There are currently 15277775 articles in the NYT archive
```



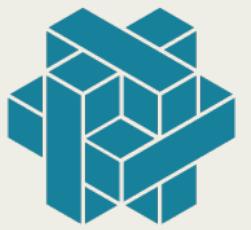
Gotchas!

- Rate Limiting
- Page Limiting

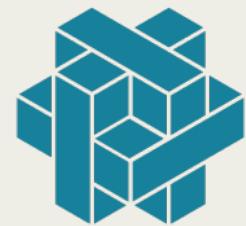


Iteration I:

- (Meaningful) Sample of Data
- Prototype — “Close the Loop”

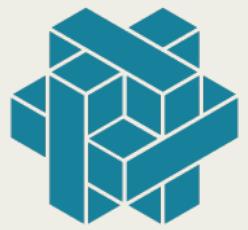


Retrieve Meta-data for **ALL** NYT articles (take 2)

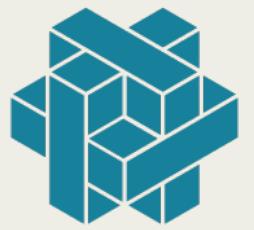


```
# let us loop (and hopefully not hit our rate limit)
while articles_left > 0 and page_count < max_pages:
    more_articles = requests.get(url + "&page=" + str(page) + "&end_date=" + str(last_date))
    print "Inserting page " + str(page)
    # make sure it was successful
    if more_articles.status_code == 200:
        for content in more_articles.json()['response']['docs']:
            latest_article = parser.parse(content['pub_date']).strftime("%Y%m%d")
            if not collection.find_one(content) and content['document_type'] == 'article':
                print "No dups"
                try:
                    print "Inserting article " + str(content['headline'])
                    collection.insert(content)
                except errors.DuplicateKeyError:
                    print "Duplicates"
                    continue
            else:
                print "In collection already"
    ...

```



```
articles_left -= 10
    page += 1
    page_count += 1
    cursor_count += 1
    final_page = max(final_page, page)
else:
    if more_articles.status_code == 403:
        print "Sleepy..."
        # account for rate limiting
        time.sleep(2)
    elif cursor_count > 100:
        print "Adjusting date"
        # account for page limiting
        cursor_count = 0
        page = 0
        last_date = latest_article
    else:
        print "ERRORS: " + str(more_articles.status_code)
        cursor_count = 0
        page = 0
        last_date = latest_article
```



Download HTML content of
articles from [NYT.com](#)
(and store in MongoDB™)

Acquire



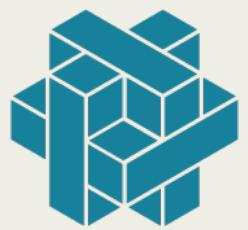
```
# now we can get some content!
#limit = 100
limit = 10000

for article in collection.find({'html' : {'$exists' : False}}):
    if limit and limit > 0:
        if not article.has_key('html') and article['document_type'] == 'article':
            limit -= 1
            print article['web_url']
            html = requests.get(article['web_url'] + "?smid=tw-nytimes")

            if html.status_code == 200:
                soup = BeautifulSoup(html.text)

                # serialize html
                collection.update({ '_id' : article['_id'] }, { '$set' :
                    { 'html' : unicode(soup), 'content' : [] }
                })

                for p in soup.find_all('div', class_='articleBody'):
                    collection.update({ '_id' : article['_id'] }, { '$push' :
                        { 'content' : p.get_text() }
                    })
```



At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)
Snakebite (HDFS)
mrjob or
Mortar (w/ Python UDF)
MLlib (pySpark)



yHat

Flask



Locally

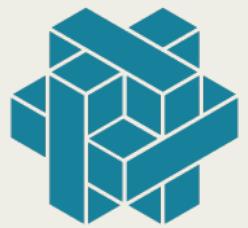
requests
BeautifulSoup4
pymongo
pandas
scikit-learn/NLTK
yHat
Flask





Parse HTML with BeautifulSoup and Extract the article Body (and store in MongoDB™)

Parse



```
# parse HTML content of articles
for article in collection.find({'html' : {'$exists' : True}}):
    print article['web_url']
    soup = BeautifulSoup(article['html'], 'html.parser')
    arts = soup.find_all('div', class_='articleBody')

    if len(arts) == 0:
        arts = soup.find_all('p', class_='story-body-text')
    ...

```



At Scale

scrapy

Hadoop Streaming
(w/ BeautifulSoup4)

Snakebite (HDFS)

mrjob or
Mortar (w/ Python UDF)

MLlib (pySpark)



yHat

Flask



Locally

requests

BeautifulSoup4

pymongo

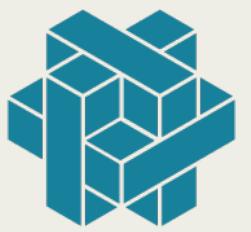
pandas

scikit-learn/NLTK

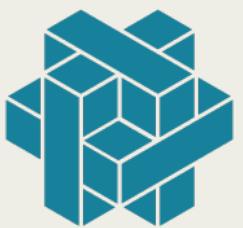


yHat

Flask



```
for p in arts:  
    collection.update({ '_id' : article['_id'] }, { '$push' :  
        { 'content' : p.get_text() }  
    })
```



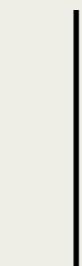
At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)

Snakebite (HDFS)

mrjob or
Mortar (w/ Python UDF)

MLlib (pySpark)



yHat

Flask



Locally

requests
BeautifulSoup4

pymongo

pandas

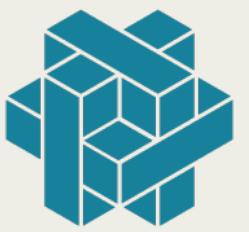
scikit-learn/NLTK

yHat

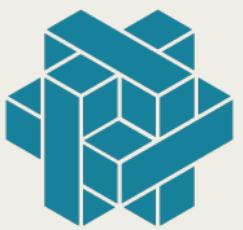
Flask



Explore



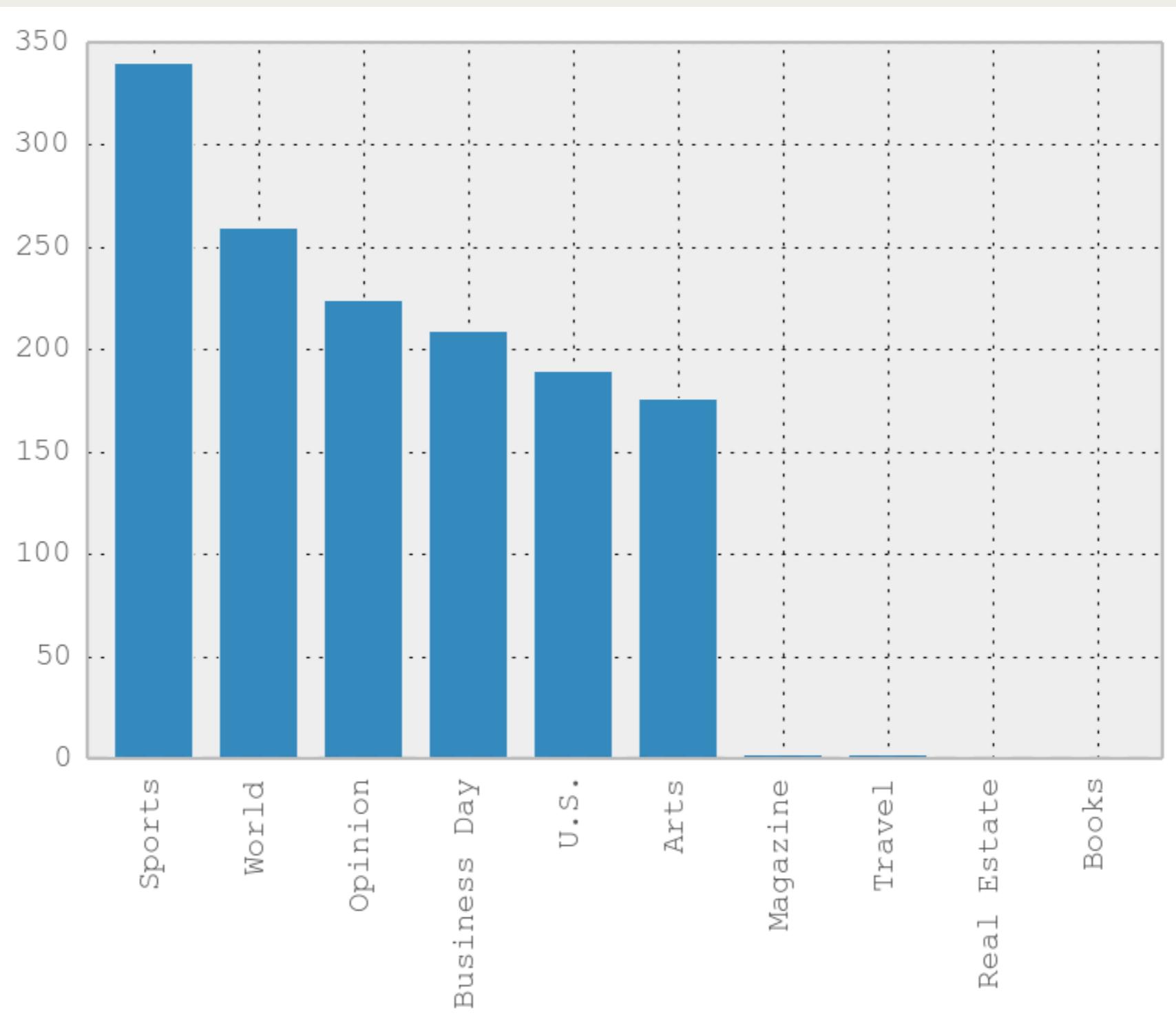
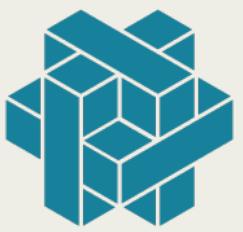
Exploratory Data Analysis with pandas



```
articles.describe()
#          text  section
# count      1405   1405
# unique     1397    10

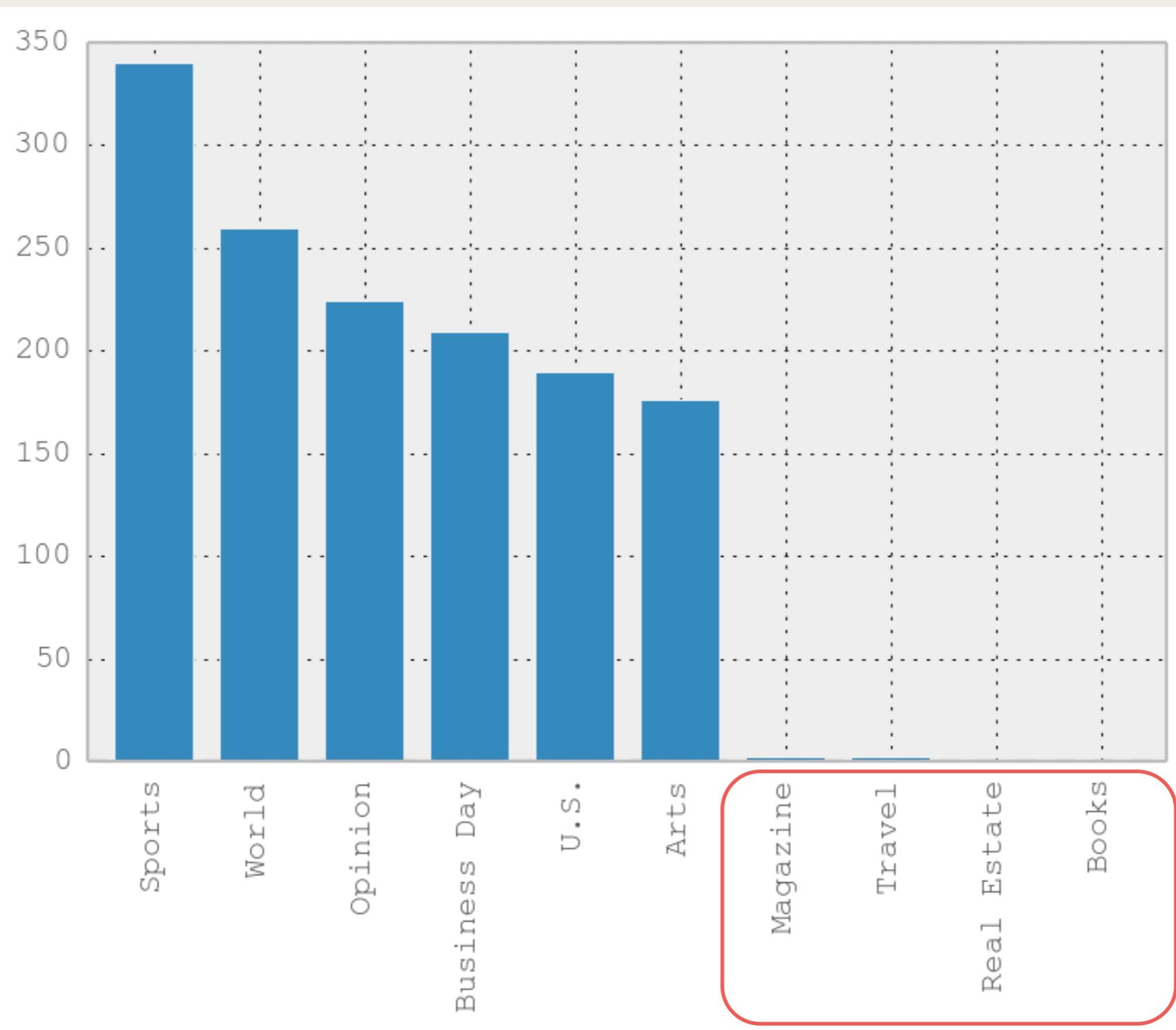
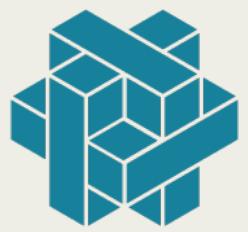
fig = plt.figure()
# histogram of section counts
articles['section'].value_counts().plot(kind='bar')
```

Explore



Questions? tweet @zipfianacademy #pydata

Explore



error with
NYT API

Explore

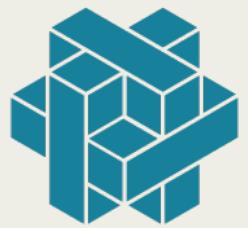


```
api_key='xxxxxxxxxxxxxx'
```

```
url = 'http://api.nytimes.com/svc/search/v2/
articlesearch.json?fq=section_name.contains:( "Arts"
"Business Day" "Opinion" "Sports" "U.S."
"World")&sort=newest&api-key=' + api_key
```

```
# make an API request
api = requests.get(url)
```

error with
NYT API



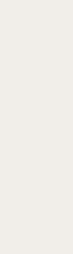
At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)

Snakebite (HDFS)

mrjob or
Mortar (w/ Python UDF)

MLlib (pySpark)



yHat

Flask



Locally

requests
BeautifulSoup4

pymongo

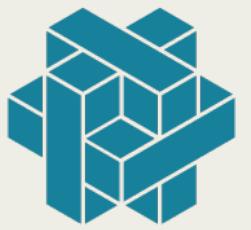
pandas

scikit-learn/NLTK

yHat

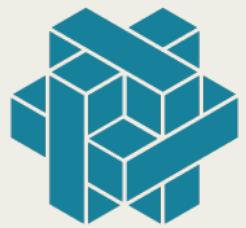
Flask





Tokenize article text and
create feature vectors with NLTK

Vectorize



```
wnl = nltk.WordNetLemmatizer()

def tokenize_and_normalize(chunks):
    words = [ tokenize.word_tokenize(sent) for sent in
tokenize.sent_tokenize("".join(chunks)) ]
    flatten = [ inner for sublist in words for inner in sublist ]
    stripped = [ ]

    for word in flatten:
        if word not in stopwords.words('english'):
            try:
                stripped.append(word.encode('latin-1').decode('utf8').lower())
            except:
                print "Cannot encode: " + word

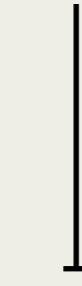
    no_punks = [ word for word in stripped if len(word) > 1 ]
    return [wnl.lemmatize(t) for t in no_punks]
```

Train



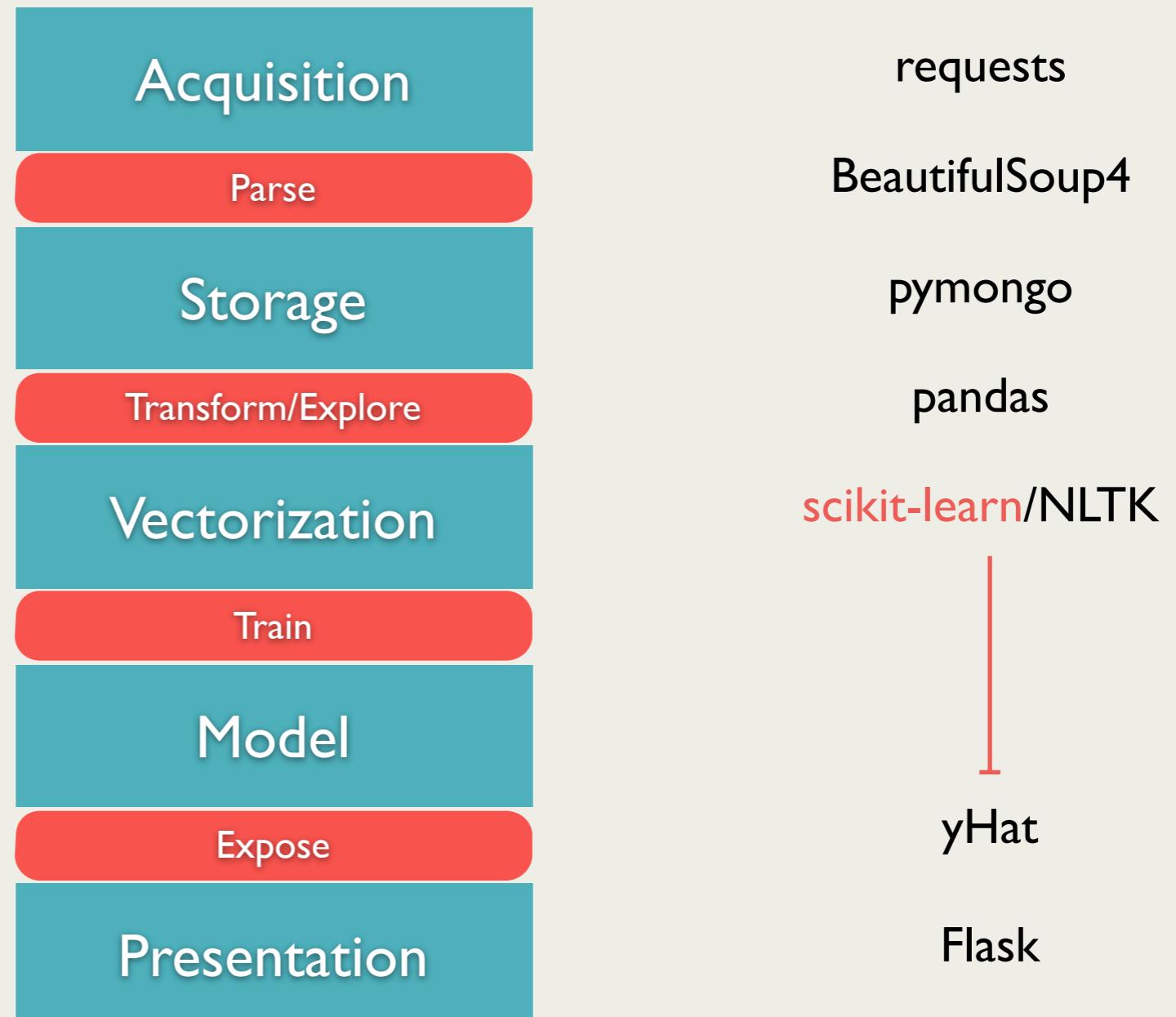
At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)
Snakebite (HDFS)
mrjob or
Mortar (w/ Python UDF)
MLlib (pySpark)



yHat

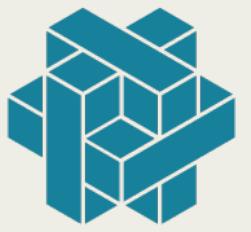
Flask



yHat

Flask

Train



Train and score a model with scikit-learn

Train

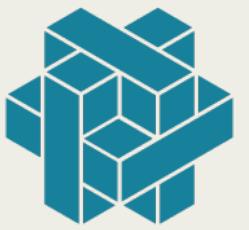


```
# cross validate
from sklearn.cross_validation import train_test_split

xtrain, xtest, ytrain, ytest =
    train_test_split(X, labels, test_size=0.3)

# train a model
alpha = 1
multi_bayes = MultinomialNB(alpha=alpha)

multi_bayes.fit(xtrain, ytrain)
multi_bayes.score(xtest, ytest)
```



Gotchas!

- Model only exists locally on Laptop
- Not Automated for realtime prediction

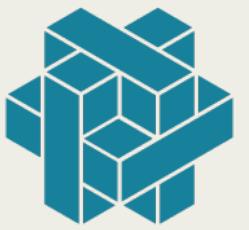


If a tree falls in the forest and nobody hears it,



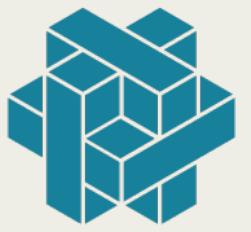
I'll only buy the soundtrack on vinyl

ICANHASCHEEZBURGER.COM



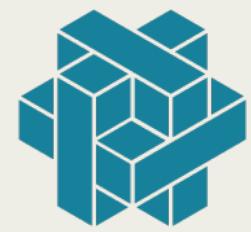
Iteration 2:

- Expose your model
- Automate your processes



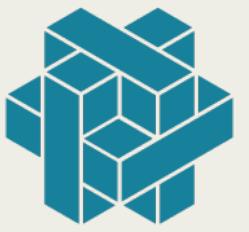
Getting that model off your lap(top)

Exposé



Questions? tweet @zipfianacademy #pydata

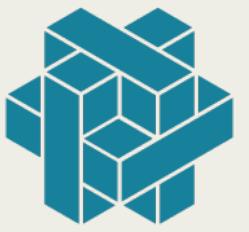
Source: http://pixel.nymag.com/imgs/daily/vulture/2012/03/09/09_joan-taylor.o.jpg_a_560x0.jpg



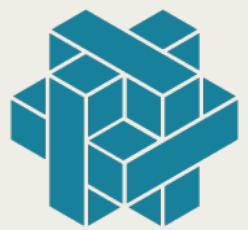
A model is just a function



Inputs...

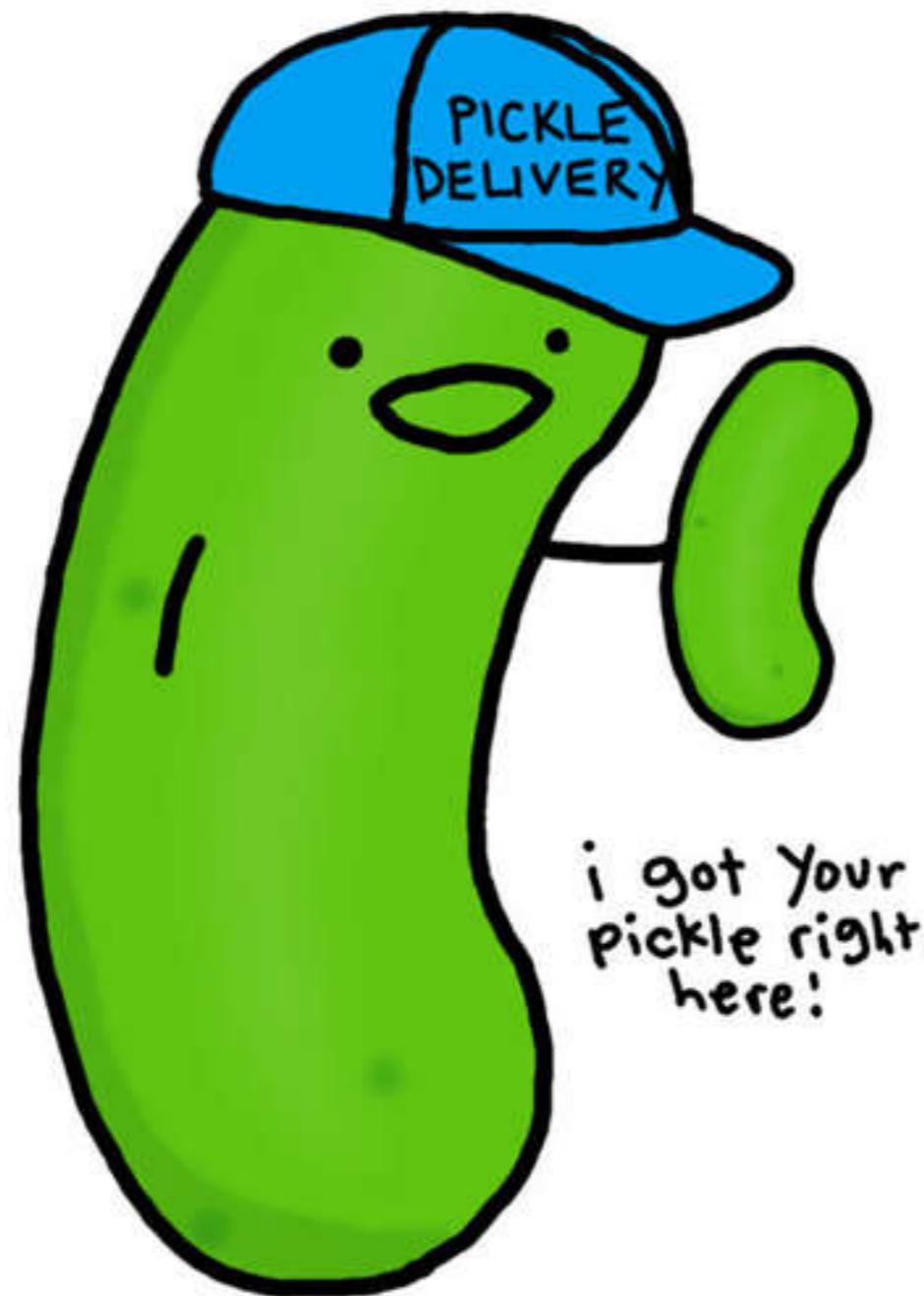
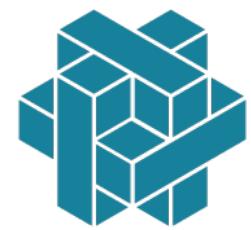


Outputs...



**Serialize your model with pickle
(or cPickle or joblib)**

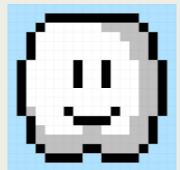
Persistence



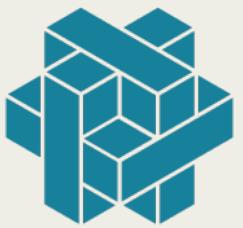


SerDes

- Disk
- Database
- Memory

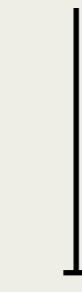


Exposé



At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)
Snakebite (HDFS)
mrjob or
Mortar (w/ Python UDF)
MLlib (pySpark)



yHat

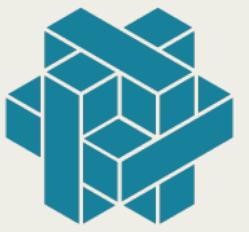
Flask



Locally

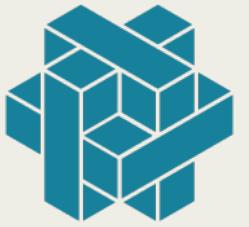
requests
BeautifulSoup4
pymongo
pandas
scikit-learn/NLTK
yHat
Flask





Deploy your Model to yHat

Exposé



```
class DocumentClassifier(YhatModel):
    @preprocess(in_type=dict, out_type=dict)
    def execute(self, data):
        featureBody = vectorizer.transform([data['content']])
        result = multi_bayes.predict(featureBody)
        list_res = result.tolist()
        return {"section_name": list_res}

clf = DocumentClassifier()
yh = Yhat("jonathan@zipfianacademy.com", "xxxxxx",
          "http://cloud.yhathq.com/")
yh.deploy("documentClassifier", DocumentClassifier, globals())
```

Present



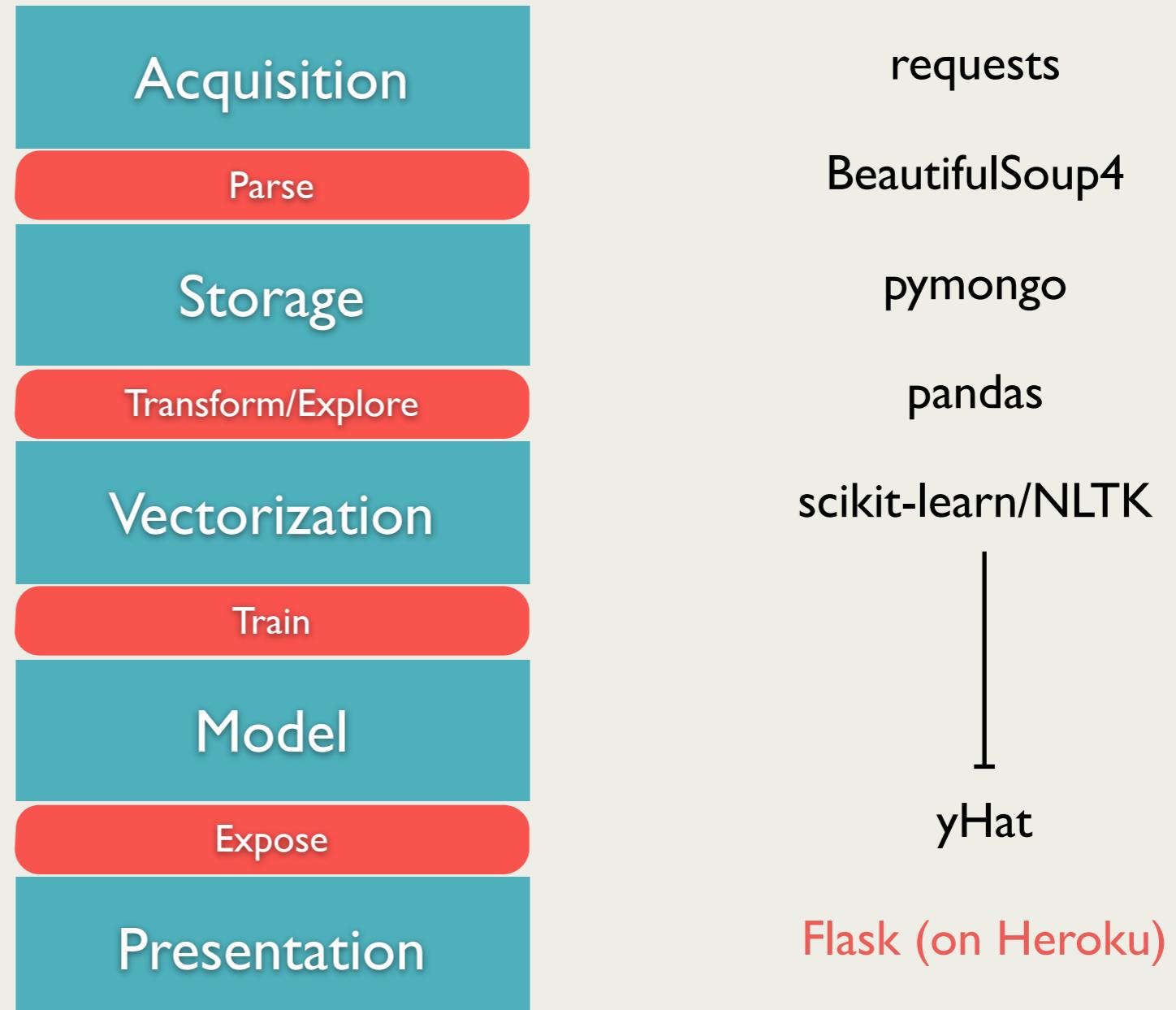
At Scale

scrapy
Hadoop Streaming
(w/ BeautifulSoup4)
Snakebite (HDFS)
mrjob or
Mortar (w/ Python UDF)
MLlib (pySpark)



yHat

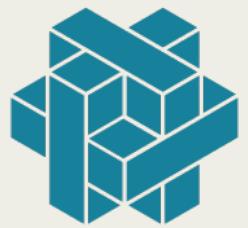
Flask





Create a Flask application to
expose your model on the web

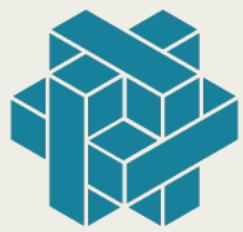
Present



```
yh = Yhat("<USERNAME>", "<API KEY>", "http://cloud.yhathq.com/")

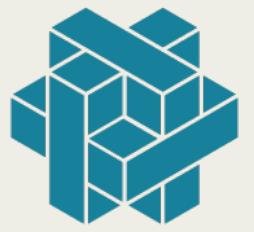
@app.route('/')
def index():
    return app.send_static_file('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    article = request.form['article']
    results = yh.predict("documentClf", { 'content': article })
    return jsonify({"results": results})
```



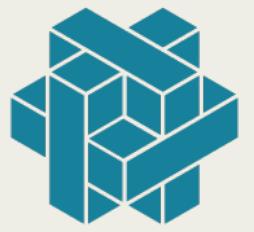
Only Data should Flow

Pipeline



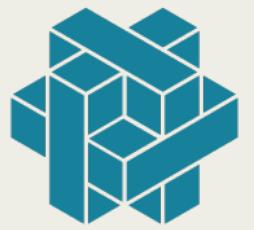
Remember to Remember
(Lineage)





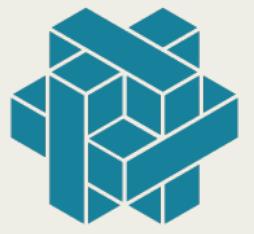
Immutable append only set of Raw Data

Computation is a view on data

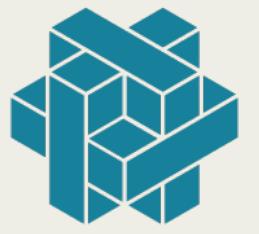


Functional Data Science

- Modularity
- Define interfaces
- Separate data from computation
- Data Lineage

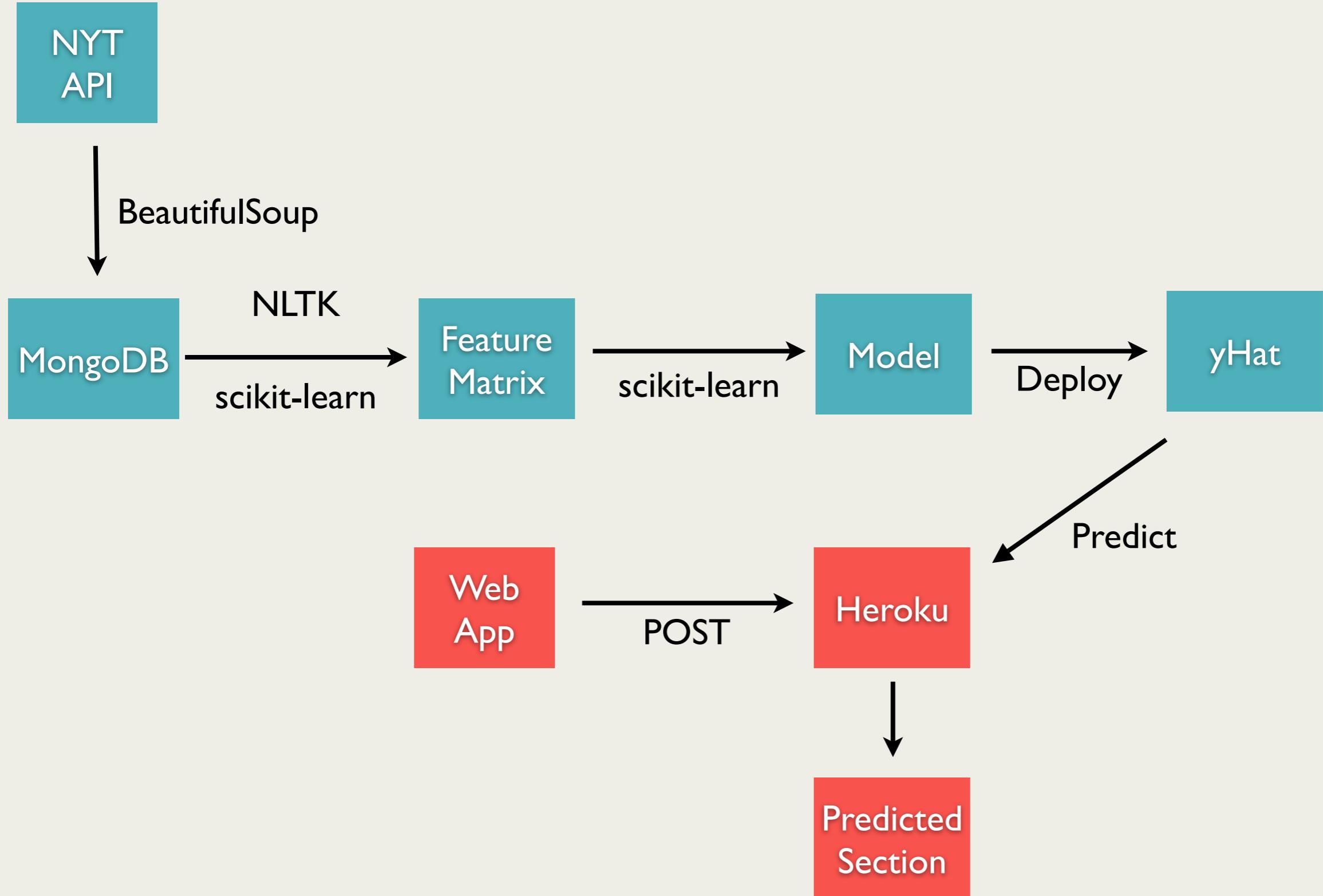
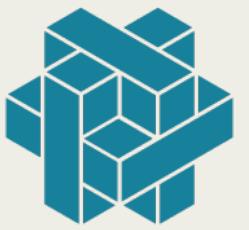


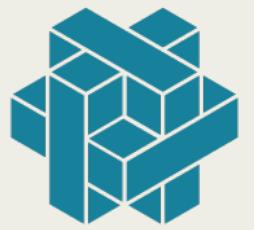
Need Robust and Flexible Pipeline!



Whatever you do, **DO NOT** cross the streams

Where we are





Gotchas!

- Only have a static subset of articles
- Pipeline not automated for re-training



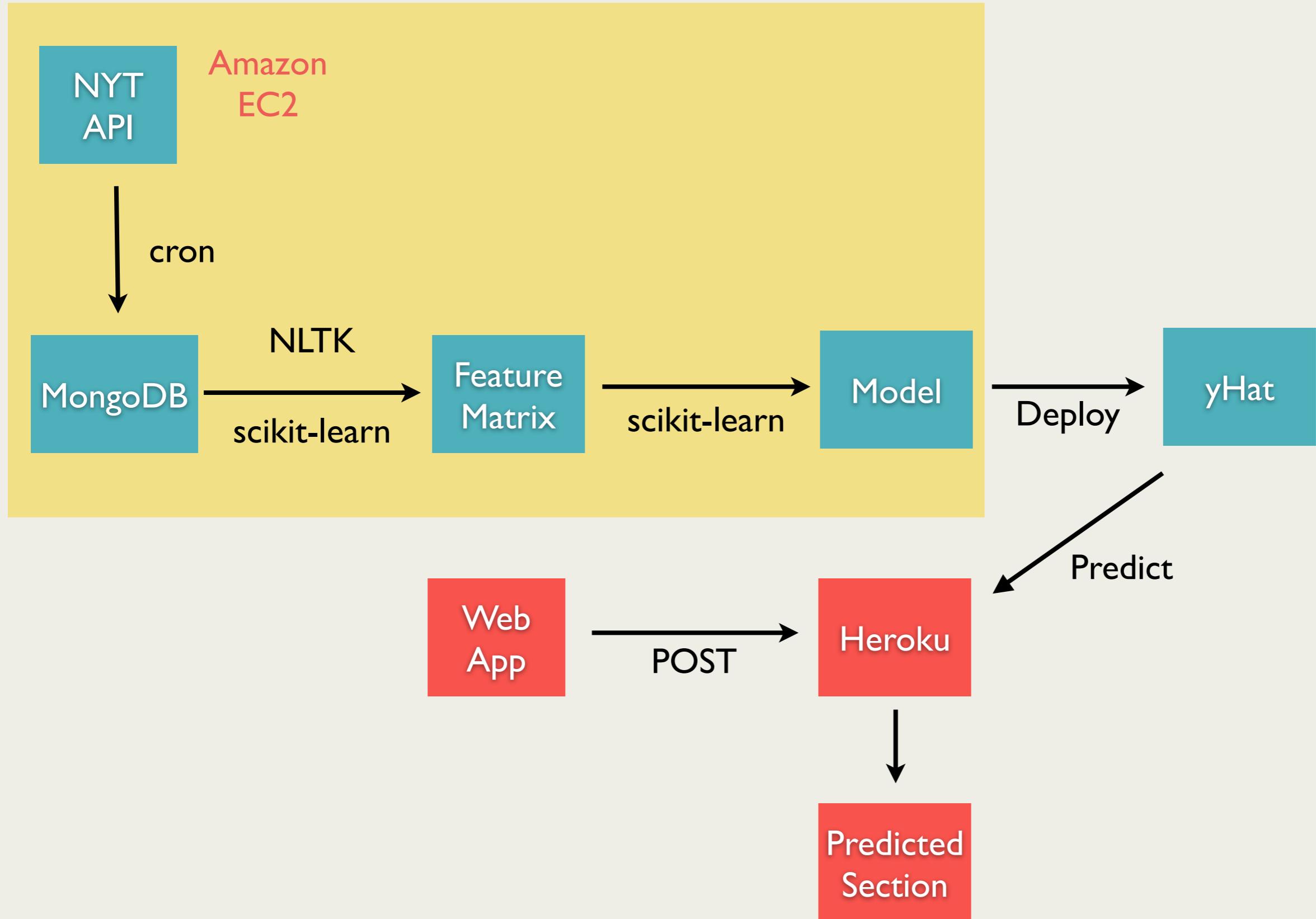
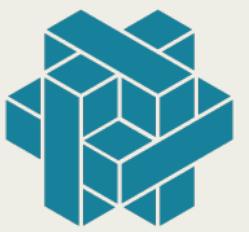
- whoami
- Nws Rdr (News Reader)
- The What, Why, and How of Data Products
- Data Engineering
- Building a Pipeline
- Productionizing the Products
- Q&A



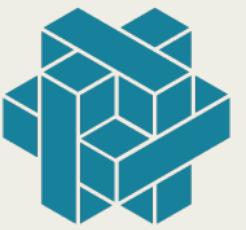
Iteration 3:



Where we are



How to Scale



Start small (data)
and fast
(development)

testing

Increase size of
data set

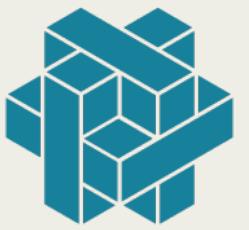
testing

Optimize and
productionize

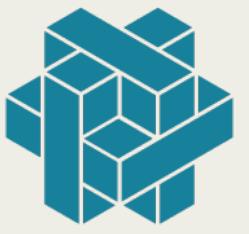
\$\$\$

PROFIT!

How to Scale



Products



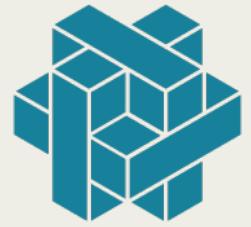
If you build it...

Products

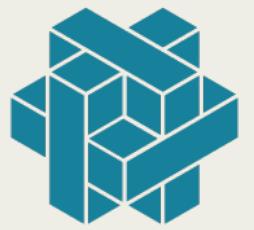


Questions? tweet @zipfianacademy #pydata

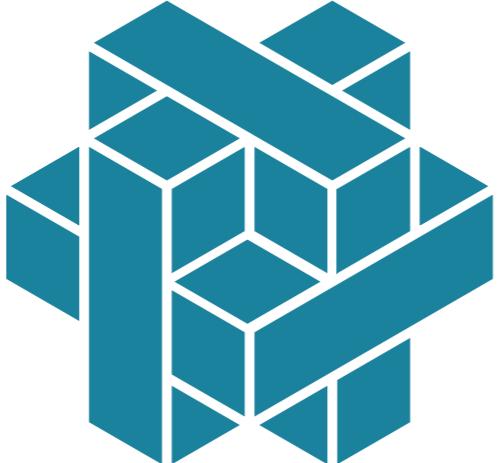
Source: <http://nateemery.com/wp-content/uploads/2013/05/field-of-dreams.jpg>



- whoami
- Nws Rdr (News Reader)
- The What, Why, and How of Data Products
- Data Engineering
- Building a Pipeline
- Productionizing the Products
- Q&A



Q & A



Zipfian
Academy

@ZipfianAcademy

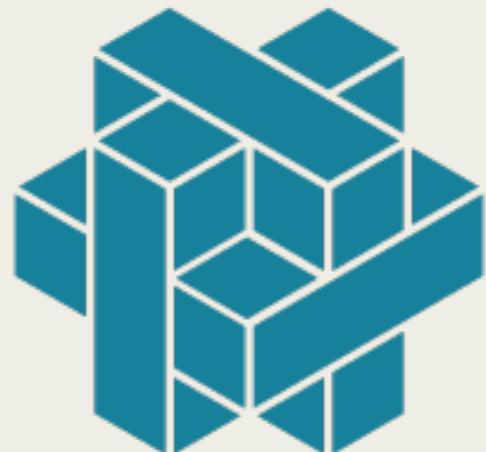
Data Science & Data Engineering
12-week Bootcamp (May 12th & Sep 8th)

<http://zipfianacademy.com/apply>

Weekend Workshops

<http://zipfianacademy.com/workshops>

Next: Interactive Visualizations w/ d3.js (June 7th)

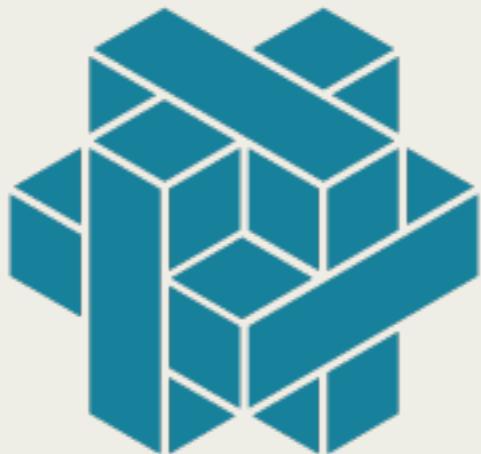


@ZipfianAcademy

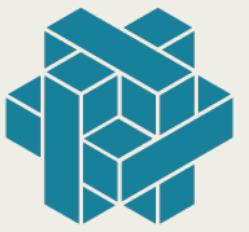
Thank You!

<http://zipfianacademy.com>

Jonathan Dinu
Co-Founder, Zipfian Academy
jonathan@zipfianacademy.com
@clearspandex

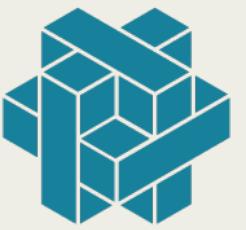


Appendix



Obtain (ranked by ease of use)

1. DaaS -- Data as a service
2. Bulk Download
3. APIs
4. Web Scraping



Data Sources

DaaS

(Data as a Service)

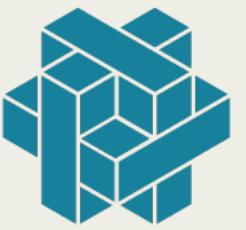
- Time Series/Numeric: [Quandl](#)
- Financial Modeling: [Quantopian](#)
- Email Contextualization: [Rapleaf](#)
- Location and POI: [Factual](#)



Bulk Download

(just like the good ol' days)

- File Transfer Protocol (FTP): [CDC](#)
- Amazon Web Services: [Public Datasets](#)
- Infochimps: [Data Marketplace](#)
- Academia: [UCI Machine Learning Repository](#)

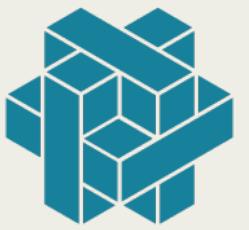


Data Sources

APIs

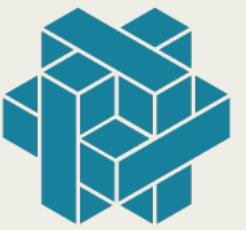
(if it's not RESTed, I'm not buying)

- Geographic: [Foursquare](#)
- Social: [Facebook](#)
- Audio: [Rdio](#)
- Content: [Tumblr](#)
- Realtime: [Twitter](#)
- Hidden: [Yahoo Finance](#)



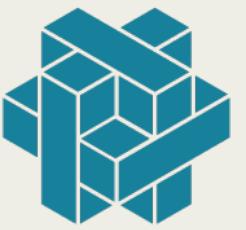
Web Scraping (DIY for life)

1. wget and curl
2. Web Spider/Crawler
3. API scraping
4. Manual Download



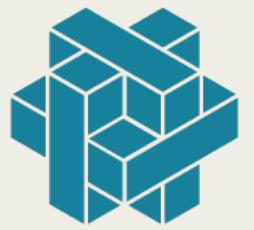
Data Formats

- Delimited Values
 - TSV
 - CSV
 - WSV
- JSON
- XML
- Ad Hoc Formats (avoid these if you can)



Data Formats

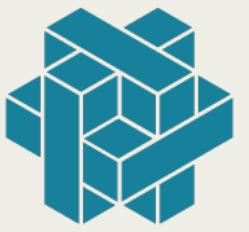
- JSON is made up of hash tables and arrays
 - Hash tables: { "foo" : 1, "bar" : 2, baz : "3" }
 - Arrays: [1, 2, 3]
 - Arrays of arrays: [[1, 2, 3], ['foo', 'bar', 'baz']]
 - Array of hashes: [{‘foo’:1, ‘bar’:2}, {‘baz’:3}]
 - Hashes of hashes: {‘foo’: {‘bar’: 2, ‘baz’: 3}}}



Data Formats

```
{"widget": {  
    "debug": "on",  
    "window": {  
        "title": "Sample Konfabulator Widget",  
        "name": "main_window",  
        "width": 500,  
        "height": 500  
    },  
    "image": {  
        "src": "Images/Sun.png",  
        "name": "sun1",  
        "hOffset": 250,  
        "vOffset": 250,  
        "alignment": "center"  
    },  
    "text": {  
        "data": "Click Here",  
        "size": 36,  
        "style": "bold",  
        "name": "text1",  
        "hOffset": 250,  
        "vOffset": 100,  
        "alignment": "center",  
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"  
    }  
}
```

Data Formats



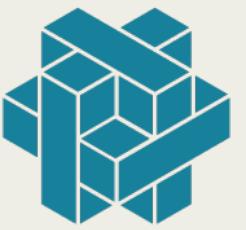
- XML is a recursive self-describing container

```
<container>
    <item>Foo</item>
    <item>Bar</item>
        <container>
            <item attr="SomethingAboutBaz">Baz</item>
        </container>
    </item>
<container>
```

Data Formats

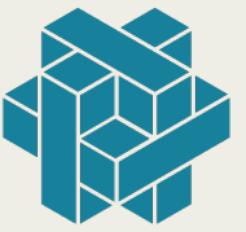


```
<widget>
    <debug>on</debug>
    <window title="Sample Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>
    <image src="Images/Sun.png" name="sun1">
        <hOffset>250</hOffset>
        <vOffset>250</vOffset>
        <alignment>center</alignment>
    </image>
    <text data="Click Here" size="36" style="bold">
        <name>text1</name>
        <hOffset>250</hOffset>
        <vOffset>100</vOffset>
        <alignment>center</alignment>
        <onMouseUp>
            sun1.opacity = (sun1.opacity / 100) * 90;
        </onMouseUp>
    </text>
</widget>
```



Data Formats

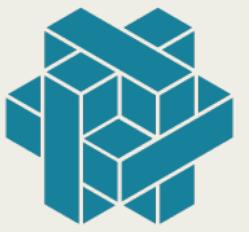
- Ad hoc data formats
 - Fixed-width (Census data)
 - Graph Edgelists
 - Voting records
 - etc.



Data Formats

- 7-5-5 format
 - Sam foo bar
 - Roger baz 6
 - Jane 314 99

Data Formats



- Directed Graph Format

1 2

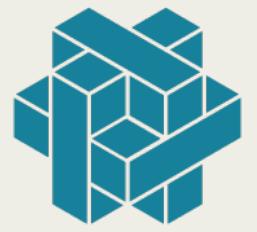
1 3

1 4

2 3

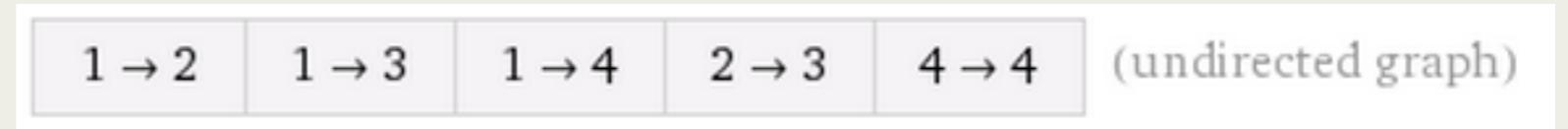
4 4

Data Formats



- Directed Graph Format

1 2

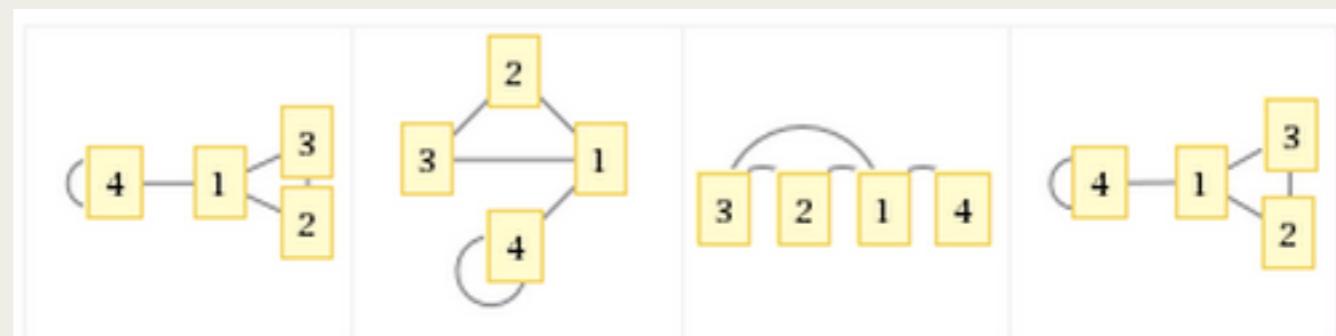


1 3

1 4

2 3

4 4





Programming languages like Python, Ruby, and R have built in parsers for data formats such as JSON and CSV. For other esoteric formats you will probably have to write your own