

# Deploying and Evaluating Data Products

<https://db.tt/JIYOoiPu>

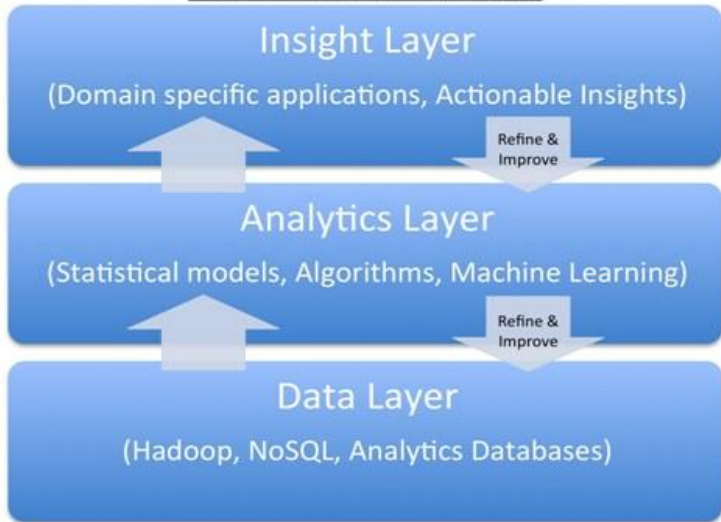
Josh Levy, PhD

**vast**

# About Vast

Data and analytics for considered purchases (vehicles, homes, ...)

## Full – Stack Analytics



White Label Marketplaces, Market Reports, Sales Apps

Pricing, Supply, Demand, Recommendations, Behavior

Inventory (rapid churn), Consumer Behavior

Graphic: Chip Hazard (Flybridge Capital Partners)

<http://www.kdnuggets.com/2014/05/stacking-deck-next-wave-opportunity-big-data.html>

# Maturity levels for turning "models" into "products"

1. Can we deploy one model into a production environment?
2. Given two models that perform similar functions can we evaluate which is better?
3. Can we operationalize model training?

# Goals for Mature Data Products

New version of a model automatically

- trained
- deployed
- evaluated

Traffic automatically routed to top performer



# Outline for this talk

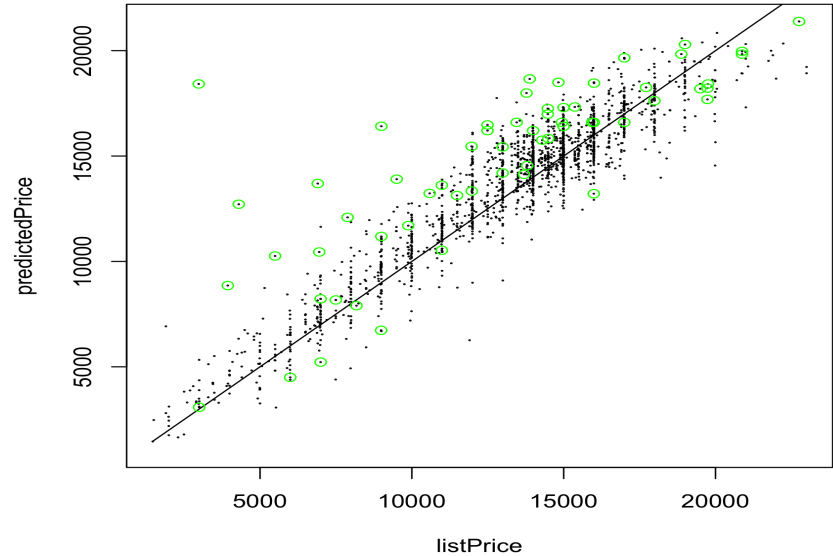
1. Can we deploy one model into a production environment?
2. Given two models that perform similar functions can we evaluate which is better?

# Deploying models is hard

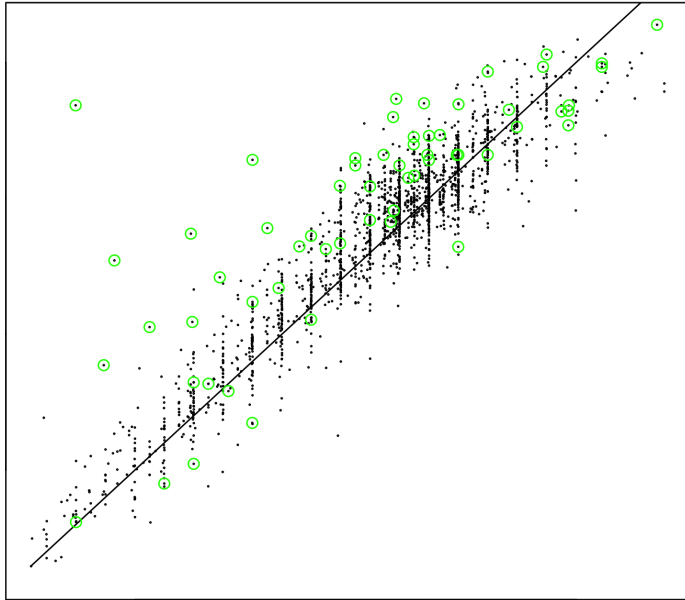
**Conway's Law** helps to explain why

*Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations*

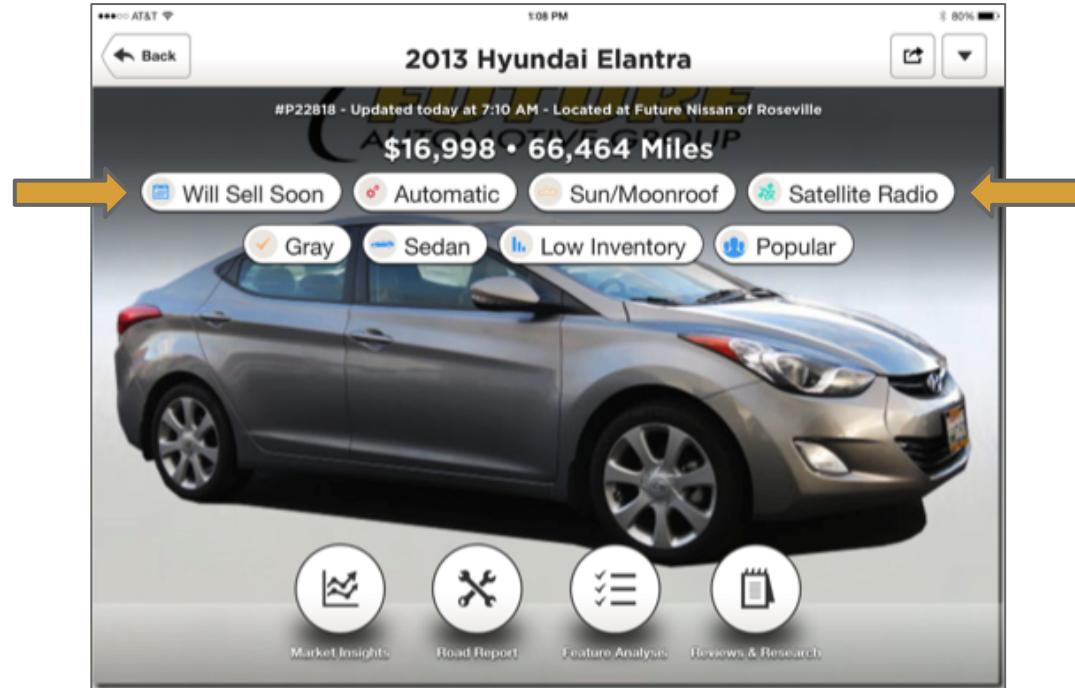
Vast has put together  
a Data Science team  
that thinks about  
training and validating  
**models**, running  
**experiments**



Models don't exist  
in isolation.



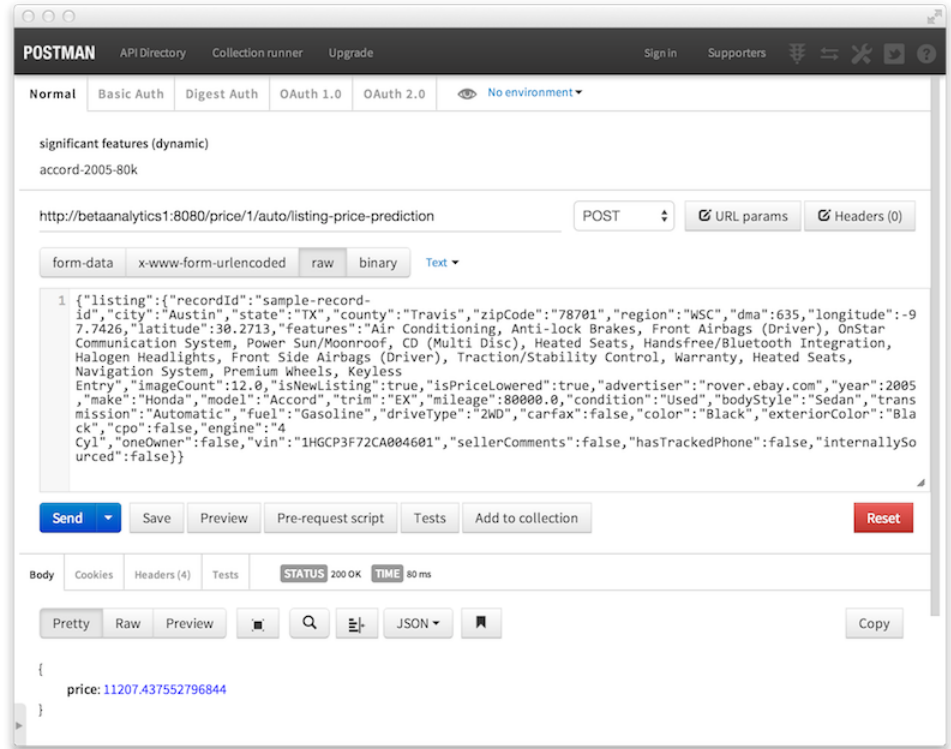
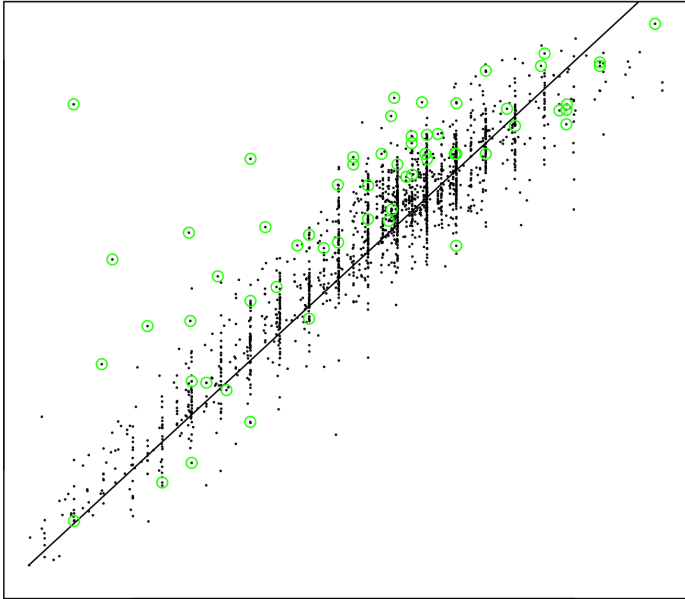
Add value when  
exposed in a product.



A screenshot of a mobile application interface for a car listing. The title is "2013 Hyundai Elantra". Below the title, it says "#P22818 - Updated today at 7:10 AM - Located at Future Nissan of Roseville". The price is "\$16,998" and the mileage is "66,464 Miles". There are several feature tags: "Will Sell Soon", "Automatic", "Sun/Moonroof", "Satellite Radio", "Gray", "Sedan", "Low Inventory", and "Popular". Below the tags is a photo of a silver Hyundai Elantra. At the bottom, there are four circular icons representing "Market Insights", "Road Report", "Feature Analysis", and "Reviews & Research". Two orange arrows point from the left and right sides towards the feature tags.

# Models don't exist in isolation.

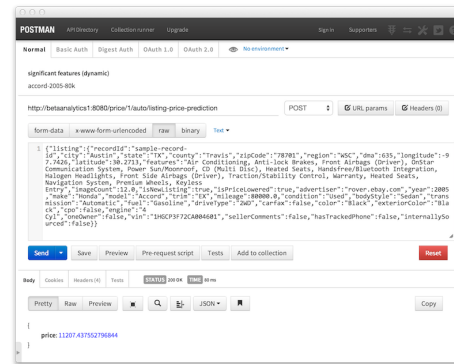
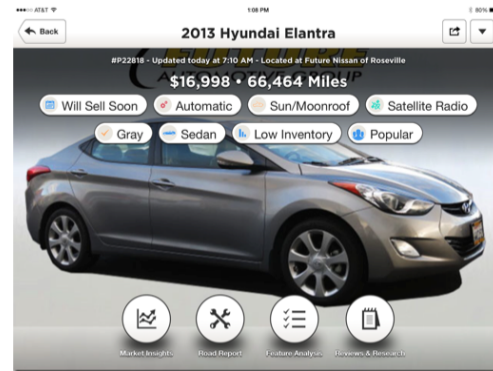
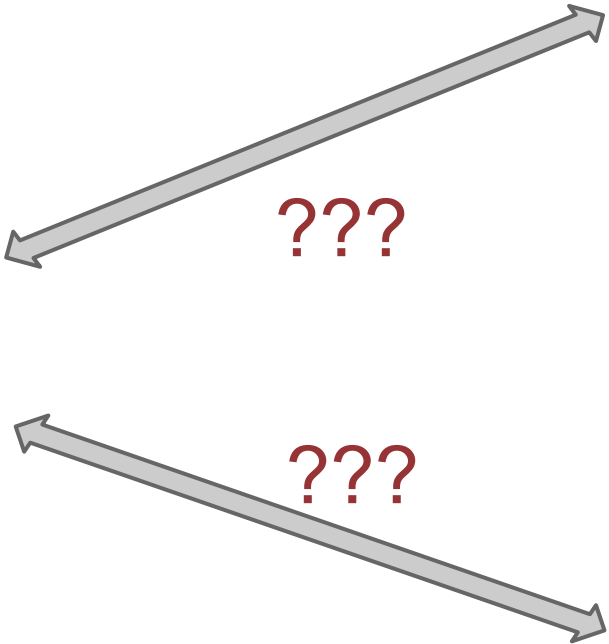
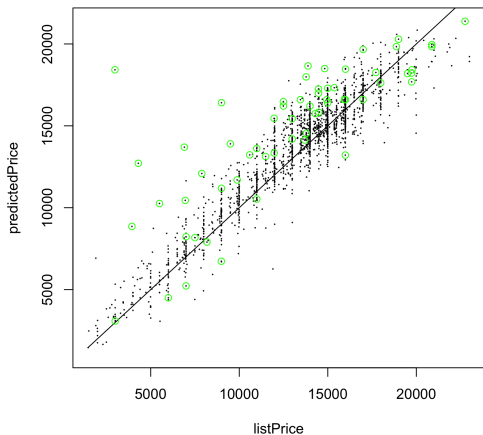
# Add value when exposed as a product.



The image shows a screenshot of the Postman API client interface. The top navigation bar includes "POSTMAN", "API Directory", "Collection runner", "Upgrade", "Sign in", and "Supporters". The main interface is divided into several sections:

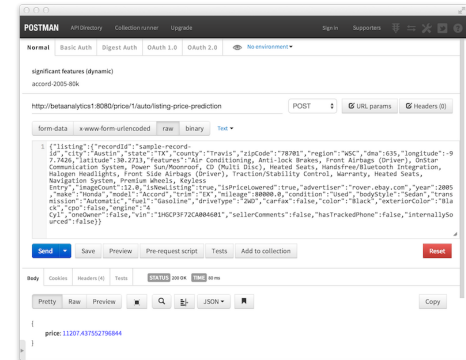
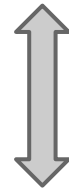
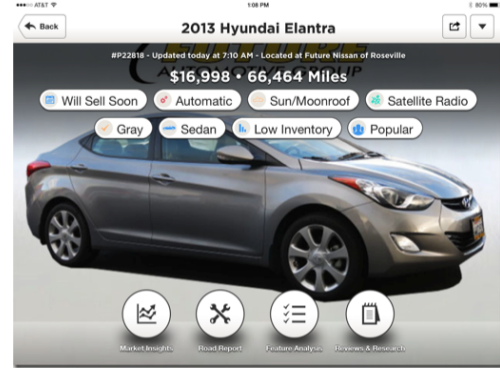
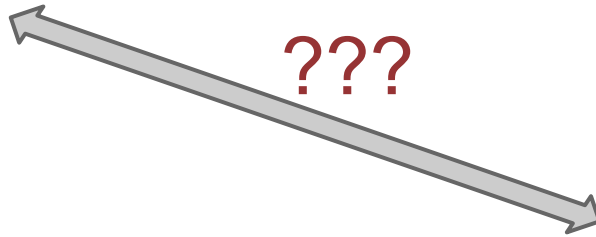
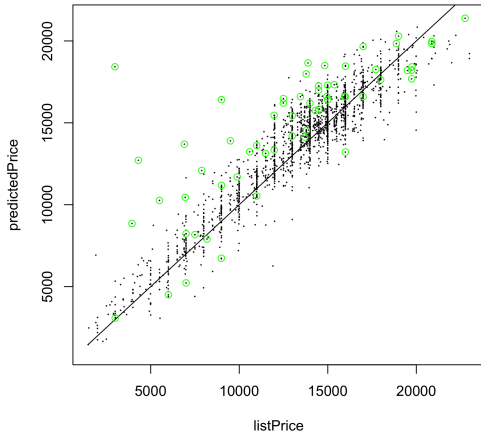
- Environment:** "Normal" is selected, with "Basic Auth", "Digest Auth", "OAuth 1.0", and "OAuth 2.0" as options. The environment is set to "No environment".
- Request:** The URL is `http://betaanalytics1:8080/price/1/auto/listing-price-prediction`. The method is "POST". There are "URL params" and "Headers (0)" buttons.
- Body:** The "raw" tab is selected, showing a JSON body with a single object containing a "listing" field with detailed car specifications.
- Response:** The "Body" tab is selected, showing a "STATUS 200 OK" and "TIME 80 ms". The response body is a JSON object with a "price" field: `{ price: 11207.437552796844 }`.

# Communication barriers between these teams

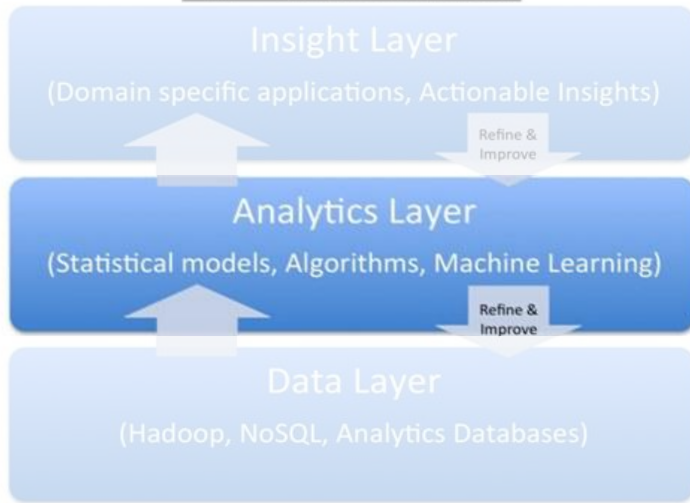


# Simplified Problem

Expose models as services  
internal apps and external  
customers can access.



## Full – Stack Analytics



Data Science and Engineering collaborate to build Analytics Layer

```
betaanalytics 1.8080/docs
POST /insight/3/auto/insight-insight
GET /insight/3/auto/listing-insight-lookup/search
GET /insight/3/auto/listing-insight-lookup/inventory
GET /insight/3/auto/listing-insight-lookup

/similarity
POST /similarity/1/auto/similar-criteria
POST /similarity/1/auto/score-listings
POST /similarity/1/auto/similar-listings/search
POST /similarity/1/auto/similar-listings/inventory
POST /similarity/1/auto/similar-listings
POST /similarity/1/re/similar-criteria
POST /similarity/1/re/score-listings
POST /similarity/1/re/similar-listings/search
POST /similarity/1/re/similar-listings

/supply
POST /supply/1/auto/listing-supply

/demand
POST /demand/1/auto/listing-demand
POST /demand/1/auto/listing-demand-drivers
GET /demand/1/auto/listing-demand-drivers-lookup/search
GET /demand/1/auto/listing-demand-drivers-lookup/inventory
GET /demand/1/auto/listing-demand-drivers-lookup

/price
POST /price/1/auto/listing-price-drivers
GET /price/1/auto/listing-price-drivers-lookup/search
GET /price/1/auto/listing-price-drivers-lookup/inventory
GET /price/1/auto/listing-price-drivers-lookup
POST /price/1/auto/listing-price-prediction
GET /price/1/auto/listing-price-prediction-lookup/search
GET /price/1/auto/listing-price-prediction-lookup/inventory
GET /price/1/auto/listing-price-prediction-lookup

/market
GET /market/1/auto/listing-statistics-bspr/search
GET /market/1/auto/listing-statistics-bspr/inventory
GET /market/1/auto/listing-statistics-bspr
GET /market/1/auto/lead-statistics-bspr/search
GET /market/1/auto/lead-statistics-bspr/inventory
GET /market/1/auto/lead-statistics-bspr
GET /market/1/auto/lead-statistics-mm/search
GET /market/1/auto/lead-statistics-mm/inventory
```



## POST /price/1/auto/listing-price-drivers

Retrieve price drivers for the auto listing in the POST body.

### Query Parameters

- modelName  
(optional) the name of the analytical model to be used [default]
- includeModelContext  
(optional) whether or not to include model context in the response [true, false]
- callerId  
(optional) a user-friendly identifier for the originating application
- partnerId  
(optional) a user-friendly identifier for the originating partner

### Samples

[Request JSON](#)  
[Response JSON](#)

analytics-service-server-5.8.0-86652

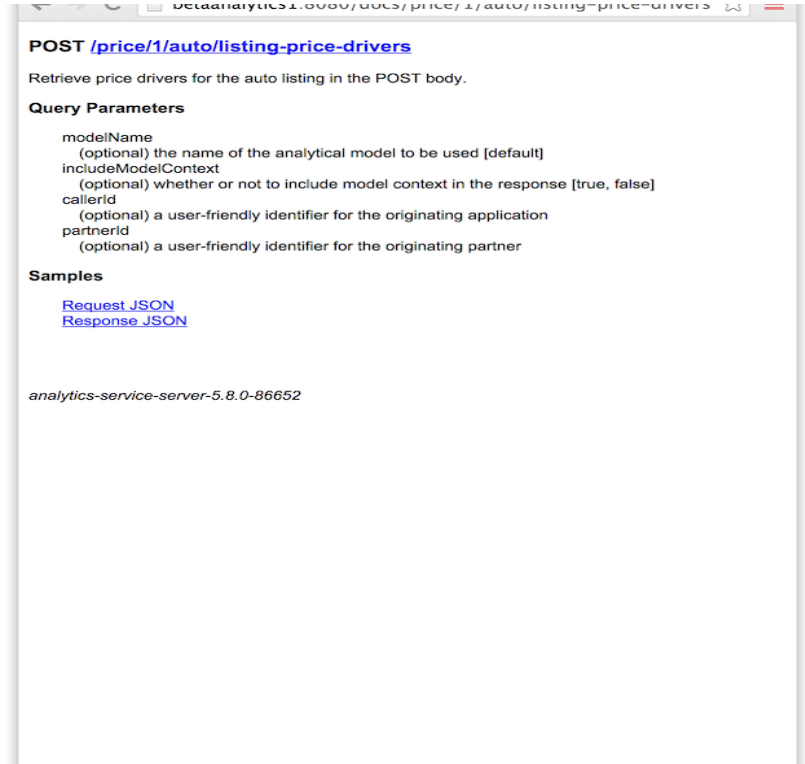
```
{
  "listing": {
    "recordId": "sample-record-id",
    "city": "Austin",
    "state": "TX",
    "county": "Travis",
    "zipCode": "78701",
    "region": "WSC",
    "dma": 635,
    "longitude": -97.7426,
    "latitude": 30.2713,
    "features": "Air Conditioning, Anti-lock Brakes, Front Airbags (Driver), OnStar Communication System, Power Sun/Moonroof, CD (Multi Disc), Heated Seats, Handsfree/Bluetooth Integration, Halogen Headlights, Front Side Airbags (Driver), Traction/Stability Control, Warranty, Heated Seats, Navigation System, Premium Wheels, Keyless Entry",
    "imageCount": 12,
    "isNewListing": true,
    "isPriceLowered": true,
    "advertiser": "rover.ebay.com",
    "price": 20000,
    "year": 2010,
    "make": "Honda",
    "model": "Accord",
    "trim": "EX",
    "mileage": 80000,
    "condition": "Used",
    "bodyStyle": "Sedan",
    "transmission": "Automatic",
    "fuel": "Gasoline",
    "driveType": "2WD",
    "carfax": false,
    "color": "Black",
    "exteriorColor": "Black",
    "cpo": false,
    "engine": "4 Cyl",
    "oneOwner": false
  }
}
```

```
{
  "featureName": "Wheel Type",
  "featureValue": "Premium",
  "baselineValue": "Alloy",
  "featurePrice": 671
},
{
  "featureName": "Navigation System",
  "featurePrice": 258.4500477481595
},
{
  "featureName": "Exterior Color",
  "featureValue": "Black",
  "featurePrice": 0
}
}
```

# Exposing Models as Services

Communications  
challenges between  
scientists and engineers

- Human language & concepts
- Technology platforms



The screenshot shows a REST client interface for a POST endpoint. The URL is `betaanalytics.1.0.0.0/docs/price/1/auto/listing-price-drivers`. The description states: "Retrieve price drivers for the auto listing in the POST body." Under "Query Parameters", there are four parameters: `modelName` (optional, default), `includeModelContext` (optional, true/false), `callerId` (optional), and `partnerId` (optional). Under "Samples", there are links for "Request JSON" and "Response JSON". At the bottom, the version `analytics-service-server-5.8.0-86652` is noted.

POST [/price/1/auto/listing-price-drivers](#)

Retrieve price drivers for the auto listing in the POST body.

**Query Parameters**

- `modelName`  
(optional) the name of the analytical model to be used [default]
- `includeModelContext`  
(optional) whether or not to include model context in the response [true, false]
- `callerId`  
(optional) a user-friendly identifier for the originating application
- `partnerId`  
(optional) a user-friendly identifier for the originating partner

**Samples**

- [Request JSON](#)
- [Response JSON](#)

*analytics-service-server-5.8.0-86652*

# Communication between Humans

Looking for common ground between

Data Scientists  
thinking about  
Experiments,  
Training and  
Validation

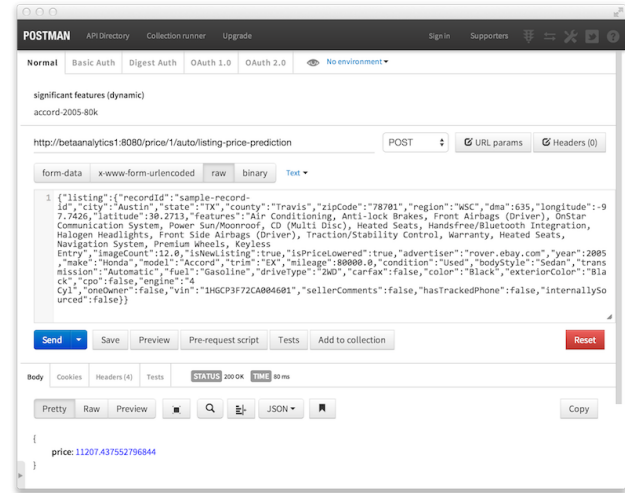
Engineers thinking  
about Scalability,  
Deployment,  
Reliability, and  
Monitoring

# Technology Platforms

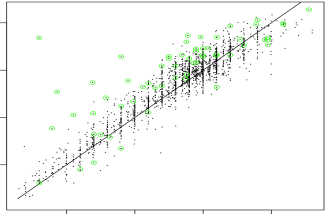


## Back-end engineering

- Typically JVM (Scala)
- Apps and other platforms can call JSON over HTTP services



# Technology Platforms



IP[y]: IPython  
Interactive Computing



Data Science: "Use the most comfortable tool for the job."

- Typically Python (sklearn) or R for models trained on inventory (millions of rows)
- Hadoop (scalding or streaming) when working with user behavior, systems exhaust

# Most comfortable tool for the job

DS team started CS PhD-heavy, DIY mindset

Growth from MS in Business Analytics program

- Strong stats background, productive in R
- Different hiring standards than back end engineering

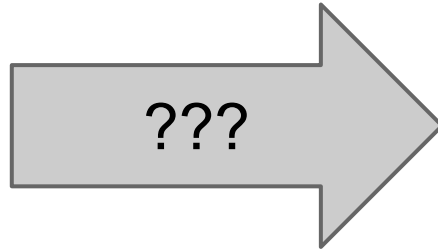
# **Expectations for MS in Business Analytics**

We're learning to push "Data Janitor" work onto engineering team.

Assuming MSBAs take clean data in, and produce models.

Those models need to become products, but MSBAs don't need to write production scala.

IP[y]:



Answers from Vast in chronological order

1. Rewrite scoring function for JVM
2. Export as PMML
3. Expose as JSON over HTTP or WebSockets from Python or R



# 1. Rewriting Scoring Function

## Pro

- DevOps is a freebie
- Additional implementations of an interface can be cheap

## Con

- First version of a model is expensive
  - Back to Conway's law - difficult communication
- Worry about transcription errors

# 1. Rewriting Scoring Function

## **Best Practice**

Scientist writes code to generate some model representation.

Engineer writes code to read model and score live data; Exposes that code as service.

## 2. Export as PMML

### Pro

- Cheaper, easier than rewrite
  - Scientist generates PMML
  - Engineer plugs in off the shelf runner
- DevOps still free

### Con

- Limited to doing things that can be expressed in PMML
- Jumped through hoops for feature transformations

# PMML Example: Auto Pricing Model

Training: split according to domain knowledge

- (make, model)
- (age, location)
- (completeness of data)

LASSO on each group

- Predicts total price
- Decomposes price by features

# Auto Pricing Model

One PMML Decision Tree  
per make, model

- Internal nodes from manual splits
- ~500 leaf nodes. Each is a regression model.



< Back

2009 Honda CR-V EXL

★ Vitals

▬ Market Insights

🕒 Road Report

📈 Fea

▬ Market Insights



Heated Seats

*about \$200 more*

Added value of about \$200 above vehicles without this feature.



Automatic

*Popular Feature*

Online car shoppe



Premium Sound System

*about \$150 more*

Added value of about \$150 above vehicles without this feature.



CD (Multi Di

*Popular Feature*

Online car shoppe



One Owner

*about \$50 more*

Added value of about \$50 above vehicles without this feature.



Crystal Blac

*Popular Feature*

Online car shoppe



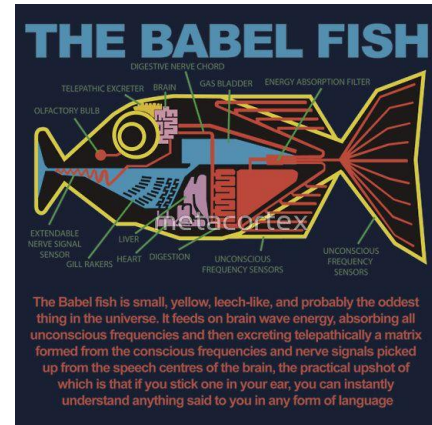
# 3. Expose as JSON over HTTP

## Pro

- Everyone speaks in their native tongue
- More natural data xform than PMML
- automate this: yhat, Wolfram, Azure

## Con

- Dev/Ops needs to harden something new



```
yhat_models - vim - 94x42
import json
from numpy import array
from operator import itemgetter
from yhat import Yhat, YhatModel, preprocess

class WeightedPercentileModel(YhatModel):

    @preprocess(in_type=dict, out_type=dict)
    def execute(self, data):
        d = sorted(data['data'], key=itemgetter('value'))
        values = array(map(itemgetter('value'), d))
        weights = array(map(itemgetter('weight'), d), dtype=float)
        weights /= weights.sum()
        cs = weights.cumsum()
        results = {'00': values[0], '100': values[-1]}

        for w0, v0, w1, v1 in zip(cs, values, cs[1:], values[1:]):
            print "%0.6f\t%0.6f\t%0.6f\t%0.6f" % (w0, v0, w1, v1)
            if ('75' not in results):
                if (w0 < 0.75) and (w1 >= 0.75):
                    results['75'] = v1
                elif (w0 >= 0.75):
                    results['75'] = v0
            if ('50' not in results):
                if (w0 < 0.50) and (w1 >= 0.50):
                    results['50'] = v1
                elif (w0 >= 0.50):
                    results['50'] = v0
            if ('25' not in results):
                if (w0 < 0.25) and (w1 >= 0.25):
                    results['25'] = v1
                elif (w0 >= 0.25):
                    results['25'] = v0
        else:
            break
        return results

if __name__ == "__main__":
    yh = Yhat(USERNAME, API_KEY, "http://yhat.oak.vast.com")
    result = yh.deploy("WeightedPercentileModel", WeightedPercentileModel, globals(), True)
:
```

4 lines of boilerplate to wrap python code in YhatModel

2 more lines to deploy model



## WeightedPercentileModel

Stage	Version	Language	Last Updated On	Status
Staging	1e89e2a	python	Fri, Oct 03 2014 18:51:24 UTC	online
Production	1e89e2a	python	Fri, Oct 03 2014 18:51:24 UTC	online

[Versions](#) [Scoring](#) [Environment Variables](#) [Logs](#) [Settings](#)

Consuming Your Model

[Input / Output](#)

[Batch Scoring](#)

## REST

<http://yhat.oak.vast.com/levy/models/WeightedPercentileModel/>

```
$ curl -X POST -H "Content-Type: application/json" \  
  --user levy:2baf8a9ad9974afafe8a859ff9ab9528 \  
  --data '{"your data": "goes here"}' \  
  http://yhat.oak.vast.com/levy/models/WeightedPercentileModel/
```

## Websockets

## WeightedPercentileModel

Stage	Version	Language	Last Updated On	Status
Staging	1e89e2a	python	Fri, Oct 03 2014 18:51:24 UTC	online
Production	1e89e2a	python	Fri, Oct 03 2014 18:51:24 UTC	online

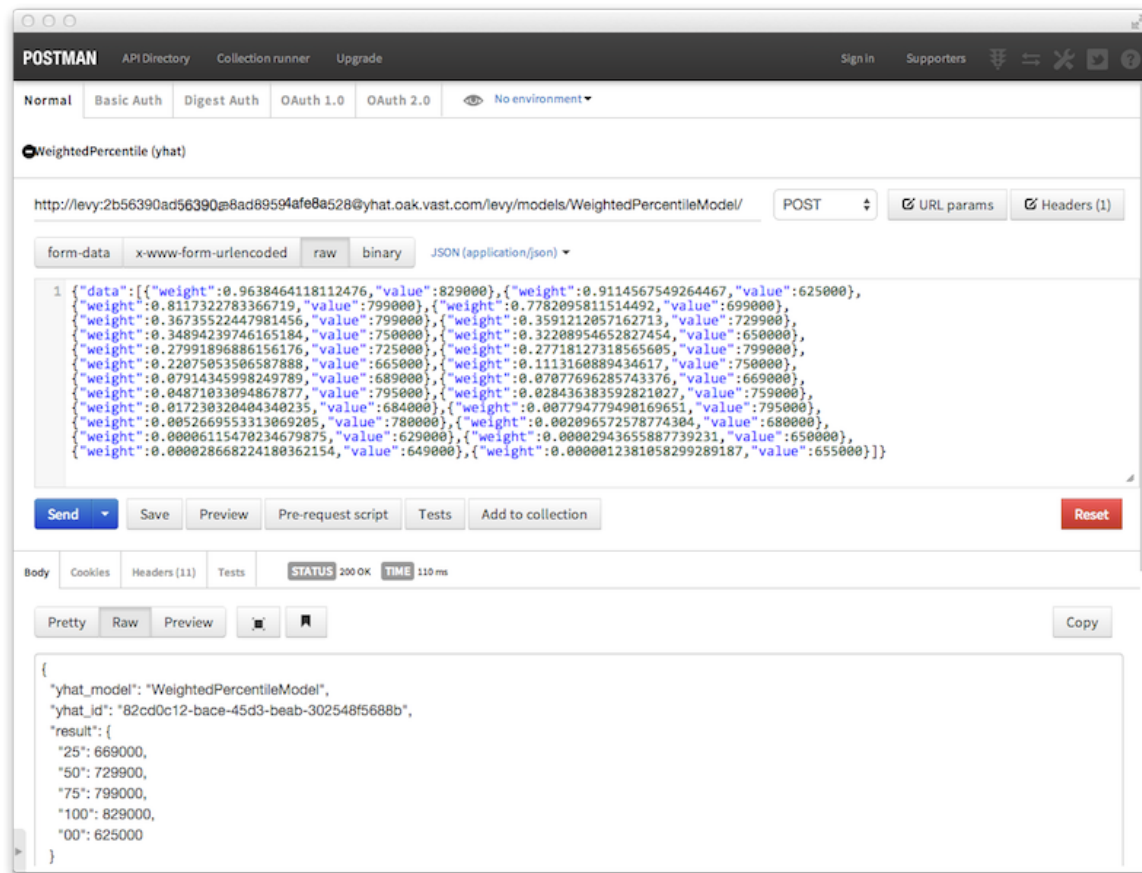
[Versions](#)[Scoring](#)[Environment Variables](#)[Logs](#)[Settings](#)[Consuming Your Model](#)[Input / Output](#)[Batch Scoring](#)

### Example Input

```
{
  "data": [
    {
      "weight": 1,
      "value": 100
    },
    {
      "weight": 10,
      "value": 50
    }
  ]
}
```

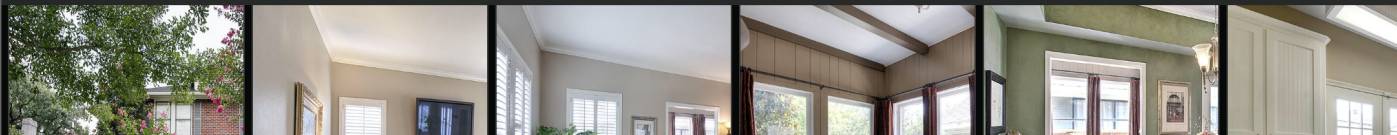
### Example Output

```
{
  "yhat_model": "WeightedPercentileModel",
  "yhat_id": "c14b5b73-ffa1-4b1e-b6cd-bab82e0cac84",
  "result": {
    "25": 50,
    "50": 50,
    "75": 50,
    "100": 100,
    "00": 0
  }
}
```



Deployed model exists as a service, could be integrated into apps

Our engineers wrap it in another service layer so they can control authentication, logging, ...



RARE

### Sunroom

Few houses like this one have this amenity.



RARE

### Playground

Few houses like this one have this amenity.



## Comparisons

Comparing to:

Similar Homes



3 BEDROOMS

### Average

This residential has a typical number of bedrooms as compared to similar properties.



18% HIGHER

### Above Average Price/ Sqft

This residential's price/sqft is 18% higher than similar properties.



AVERAGE

### Average Lot Size

This residential's lot has a typical size.



9% SMALLER

### Below Average Sqft

This residential's square footage is 9% smaller than similar properties.



10% MORE EXPENSIVE

### Above Average Price

This residential is 10% more expensive than similar properties.



3 BATHROOMS

### Average

This residential has a typical number of bathrooms as compared to similar properties.



# Deployment Recommendations

**New Projects:** yhat from the start

**Existing Projects:** Tempting to continue using PMML or Rewrite

- Want to deploy from python / R as soon as you want to do something new
- But then someone has to support both

Better to deploy directly from training code

# Deployment Recommendations

Always name your models. Allow multiple models that perform the same function to coexist.

- Sometimes want different customers on different instances of the model
  - trained on public data only
  - trained on public data + proprietary from customer X
- Allows competition

# Outline for this talk

1. Can we deploy one model into a production environment?
2. Given two models that perform similar functions can we evaluate which is better?

# Three types of evaluation

Depending on the nature of the model use

1. Direct evaluation against ground truth
2. Indirect evaluation against business metrics
3. Human judgment



# Evaluating price predictions against ground truth

Vehicle listings come off the feed with a price.

Given a choice between two models, choose the one that minimizes mean absolute error.

# Direct Evaluation Environment

Summary of global measures

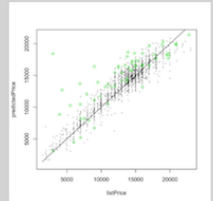
Drill-down graphs

Statistic	Value	Count
Mean		
Absolute Error	1023	1874
Mean		
Absolute Error (w/mileage)	973	1814
Mean		
Absolute Error (no mileage)	2534	60
Outlier threshold (w/mileage)	2500	

### Predicted Price vs. Price

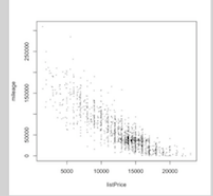
Plot of list (current) price versus predicted price for Toyota Corolla listings that appeared in inventory between 2014-09-07 and 2014-09-10 with 4 days of updates/deltas applied.

- Green circles indicate Toyota Corolla listings that did not specify a mileage.



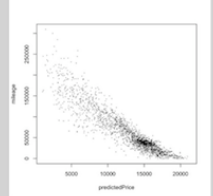
### Price vs. Mileage

Plot of list price versus mileage for Toyota Corolla listings that appeared in inventory between 2014-09-07 and 2014-09-10 with 4 days of updates/deltas applied.

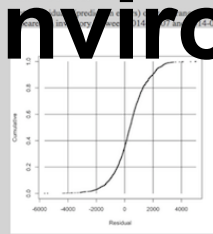


### Predicted Price vs. Mileage

Plot of predicted price versus mileage for Toyota Corolla listings that appeared in inventory between 2014-09-07 and 2014-09-10 with 4 days of updates/deltas applied.

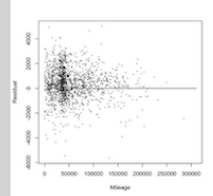


### Residual Distribution



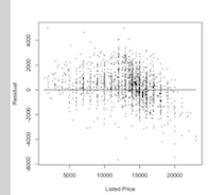
### Residual Distribution versus Mileage

Plot of residuals versus mileage for Toyota Corolla listings that appeared in inventory between 2014-09-07 and 2014-09-10 with 4 days of updates/deltas applied. Shows how residuals (+/-) are distributed over the range of mileages.



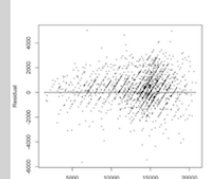
### Residual Distribution versus Listed Price

Plot of residuals versus listed price for Toyota Corolla listings that appeared in inventory between 2014-09-07 and 2014-09-10 with 4 days of updates/deltas applied. Shows how residuals (+/-) are distributed over the range of listed prices.



### Residual Distribution versus Predicted Price

Plot of residuals versus predicted price for Toyota Corolla listings that appeared in inventory between 2014-09-07 and 2014-09-10 with 4 days of updates/deltas applied. Shows how residuals (+/-) are distributed over the range of predicted prices.



- Residual graphs
- CDF
- Drill-down

Date/VIN	Prediction Error
<a href="#">2014-09-08-4: INXBR32E93Z103371</a>	-5639
<a href="#">2014-09-07-4: 2T1BU4EE9BC551738</a>	-5438
<a href="#">2014-09-08-4: 2T1BR32E84C260490</a>	5021
<a href="#">2014-09-07-4: SYFBU4EE0DP119963</a>	4958
<a href="#">2014-09-07-4: 2T1BU4EESAC354421</a>	4644
<a href="#">2014-09-09-4: 2T1BU4EE0CC822669</a>	-4343
<a href="#">2014-09-09-4: SYFBU4EE0DP090142</a>	-4224
<a href="#">2014-09-09-4: SYFBU4EE6DP157066</a>	4111
<a href="#">2014-09-10-4: SYFBURHE3EP033914</a>	4095
<a href="#">2014-09-09-4: 2T1BURHE0EC114305</a>	-4070
<a href="#">2014-09-09-4: SYFBU4EESDP159455</a>	4034
<a href="#">2014-09-07-4: INXBR32E88Z021820</a>	-3899
<a href="#">2014-09-10-4: 2T1BR32E66C627776</a>	3866
<a href="#">2014-09-09-4: 2T1BURHE2EC133602</a>	-3772
<a href="#">2014-09-10-4: SYFBURHE6EP010613</a>	-3689
<a href="#">2014-09-10-4: JTDDBR32E0600065495</a>	3589
<a href="#">2014-09-09-4: SYFBURHE4EP016152</a>	-3506
<a href="#">2014-09-08-4: 2T1BU4EE7BC642622</a>	3449
<a href="#">2014-09-07-4: SYFBU4EE4DP121621</a>	3430
<a href="#">2014-09-09-4: INXBU4EEXAZ199444</a>	3360
<a href="#">2014-09-09-4: SYFBU4EE7DP207893</a>	3330
<a href="#">2014-09-09-4: SYFBURHE0EP056678</a>	-3324
<a href="#">2014-09-09-4: SYFBURHE9EP013005</a>	-3324
<a href="#">2014-09-08-4: 2T1BU4EESBC741286</a>	3281
<a href="#">2014-09-07-4: SYFBU4EE9DP157546</a>	3226
<a href="#">2014-09-07-4: SYFBU4EE3DP156506</a>	3221
<a href="#">2014-09-10-4: 2T1BURHE7EC208102</a>	3211
<a href="#">2014-09-07-4: 2T1BU4EE4BC604300</a>	-3162
<a href="#">2014-09-08-4: 2T1BURHE3EC150103</a>	3126

## Links to upstream data for outliers

### Allows manual investigation

- Is the model to blame for the outlier?
- Is the data garbage?

# Indirect Evaluation

## Recommendations on details page

- Can't measure relevance or quality
- Conversion rate is important to business

new door added to the front bedroom. House is adorable and ready for your congress, dining, shops and Stacy Pool.

**Appliances:** Cook Top Gas, Disposal, Dishwasher, Water Heater Gas, Microwave Oven, Refrigerator

**Laundry:** Dryer Connection - Electric, Dryer Connections, Utility/Laundry Room

**Finishing Description:** Kitchen

**Stability Features:** 0

**Interior Features:** Deck, Private Backyard

**Construction:** Frame

**Foundation:** Pier & Beam

**Roofing:** Composition Shingle

**Utilities:** Electricity on Property, Natural Gas on Property, Phone Available

**Location:** City at Street

**Address:** City

**Parking:** Off Street

**Parking Spaces:** 2

**Guest Accommodations:** 0

**Pool Y/N:** 0

**Fence:** Partial, Privacy, Wood

**Sprinkler System:** 0

**View:** No View

**Waterfront Y/N:** 0

**Taxes:** 11642.00, 8433.00

Community Pool, Deck, Dishwasher, Electricity Available, Family Room, Fire Alarm, Gas Available, Gas Cooktop, Gas Water Heater, Home Office, Internet, Pergola, Pier And Beam Foundation, Public Sewer, Refrigerator, Smoke Detector, Stone Available, Tile Floors, Walk-In Closet, Wood Floors, Yard

**You may also be interested in these homes:**

**3003 Birdwood Circle, Austin, TX 78704**  
\$385,000  
MLS # 2165734  
BEDS: 2, BATHS: 1, SQ. FT.: 1,196  
Listing Office: Sherry Fields Properties

**4013 Clawson Road, Austin, TX 78704**  
\$550,000  
MLS # 2910980  
BEDS: 2, BATHS: 1, SQ. FT.: 896  
Listing Office: Sherry Fields Properties

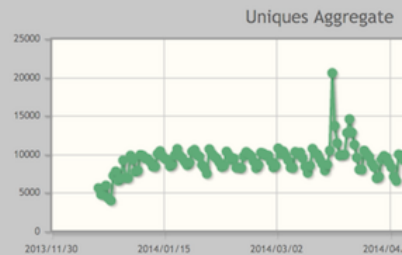
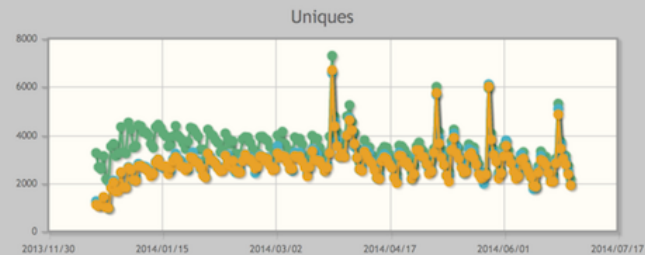
**1501 Barton Springs Road Unit 105, Austin, TX 78704**  
\$419,000  
MLS # 6877365  
BEDS: 2, BATHS: 1, SQ. FT.: 1,000  
Listing Office: Sherry Fields Properties

**2113 Thornton Road, Austin, TX 78704**  
\$545,000  
MLS # 2031177  
BEDS: 2, BATHS: 1, SQ. FT.: 1,680  
Listing Office: Sherry Fields Properties

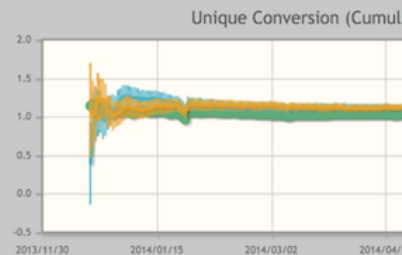
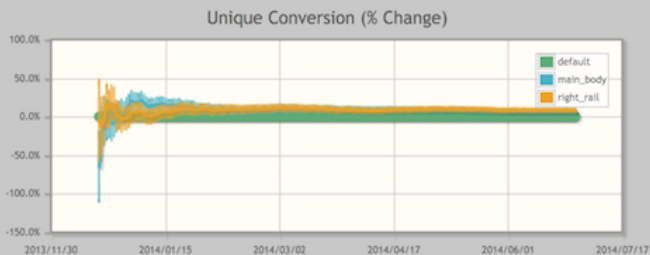
Test Summary	
Test:	recom_homes
Publisher:	
Start Date:	2013-12-20
Total Uniques:	1746503
% Users:	0%



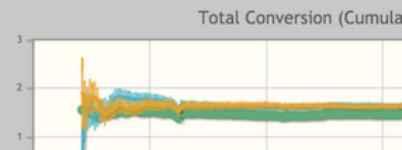
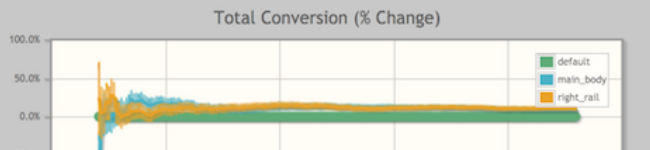
Uniques		
Variant	% Alloc	Uniques
default	0%	664093
main_body	0%	547809
right_rail	0%	534601



Unique Conversion			
Variant	Value	%-Change	P-Value
default	1.01%	-	-
main_body	1.08%	6.69%	.0001
right_rail	1.09%	7.99%	.0000



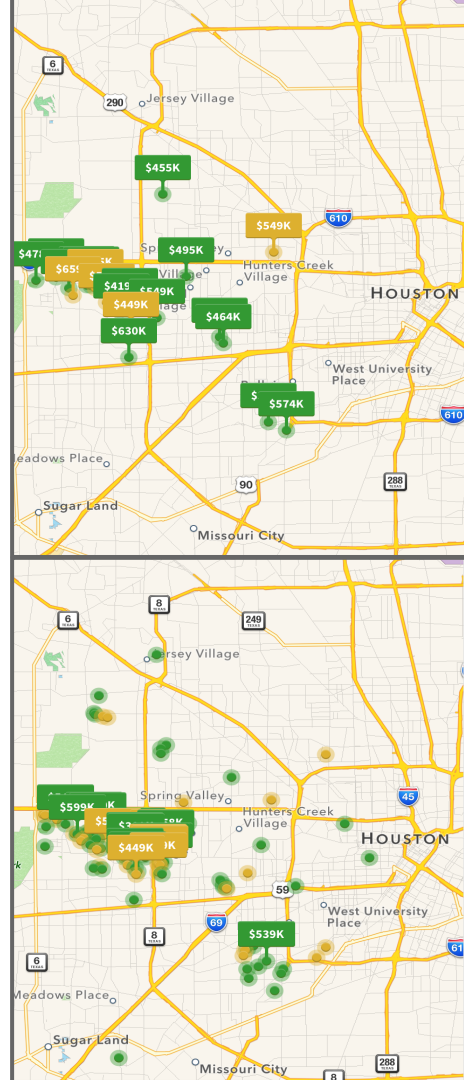
Total Conversion			
Variant	Value	%-Change	P-Value
default	1.44%	-	-



# Human Judgment

Sort in early versions of HomeStory

- Can't measure "relevance"
- Cold start: Not enough traffic to optimize conversion rate yet
- Best we can do is "not embarrassing"
- Be good enough to attract traffic, set up future experiments







# Evaluation Recommendations

Do as much as you can

- Direct evaluation against truth
- Indirect evaluation against business metrics
- Human judgment

Goals

- Data driven decisions
- As much automation as possible



# Plugs

Back end engineering team is hiring

Looking for Strong Junior / Senior Java/Scala  
person

Resumes to Olivier: [omodica@vast.com](mailto:omodica@vast.com)