



# Elastic Machine Learning Workshop

---

**Rich Collier**  
Solutions Architect



# Challenges that Anomaly Detection Solves

- IT Operations
  - How do I know my systems are behaving normally?
  - Where to set thresholds for good alerting?
  - How to find the root cause of problems when I don't know what to look for?
- IT Security
  - Do I have systems that are compromised with malware?
  - Which users could be an insider threat?
- Internet of Things / SCADA / Other
  - Is my factory working normally?
  - What do I do with thousands of time-series data points?
  - Which traffic incidents are causing the most delay?

# Overview

- Anomaly Detection Concepts
- How to Learn “Normal”
- Lab 1: The Single Metric Job
- Splitting the Analysis
- Lab 2: Multi-Metric Jobs
- The Anomaly Explorer View
- Custom URLs
- Forecasting
- Lab 3: Forecasting
- Population Analysis
- Custom Rules
- Calendars
- Advanced Jobs
- Other Types of Analysis
  - Rarity
  - Categorization
- Lab 4: Choose Your Own Adventure
  - Root Cause Analysis
  - Rogue IPs in Weblogs
  - NYC Taxi Ridership

# Anomaly Detection Concepts

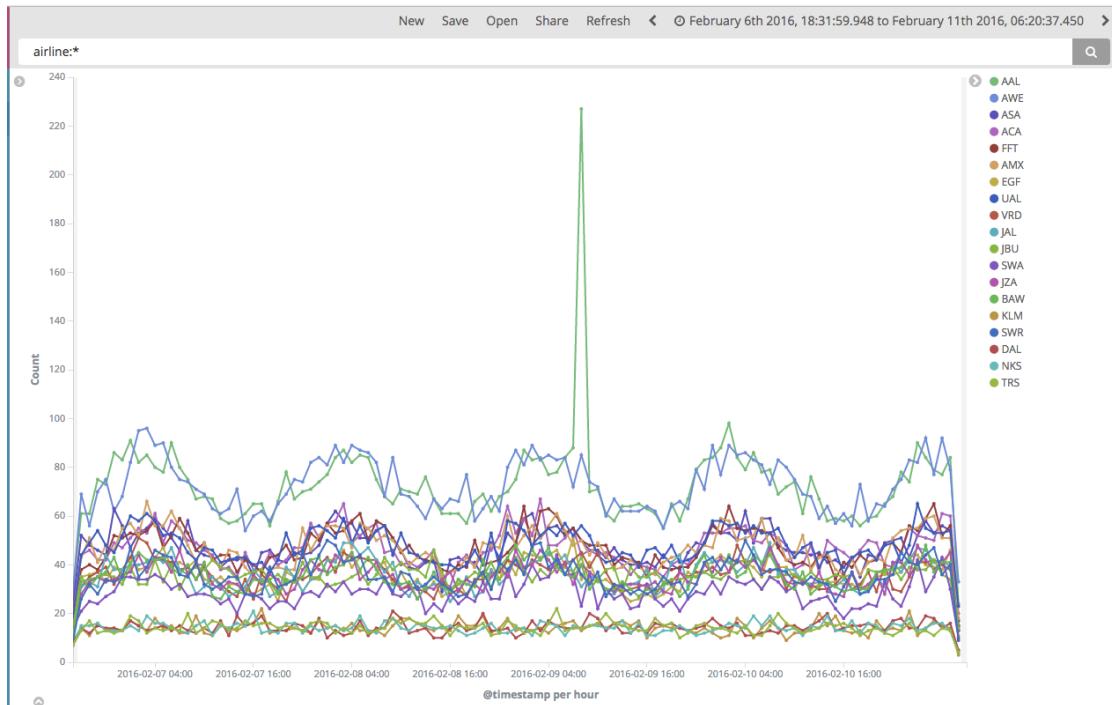
# Terminology

- **Machine Learning**
  - Broad term, but Elastic Machine Learning is automated anomaly detection and forecasting for time-series data (for now).
- **Anomaly Detection**
  - Discovery of what's "weird" or "different", not what's "bad"
- **Unsupervised Learning**
  - Learning without human-labeled examples (without being "taught"). Rely only on the data
- **Bayesian**
  - An approach based on probability in which prior results are used to calculate probabilities of certain present or future events

# What is “Abnormal”?

What's abnormal here?

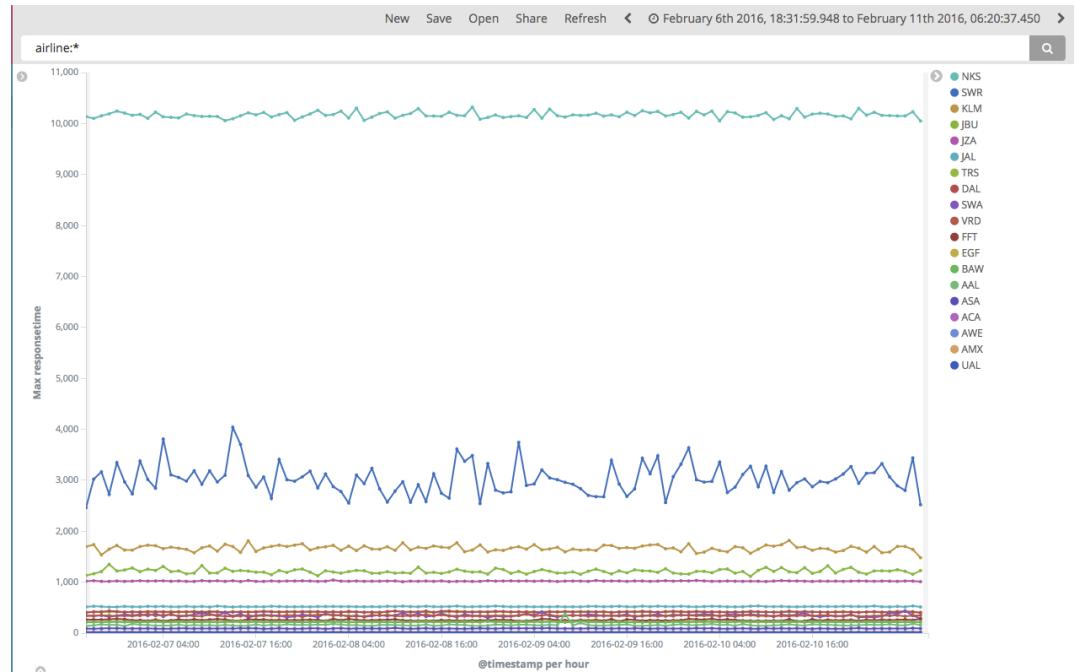
Why?



# What is “Abnormal”?

What's abnormal here?

Why?



# What is “Abnormal”?

What's abnormal here?

Why?

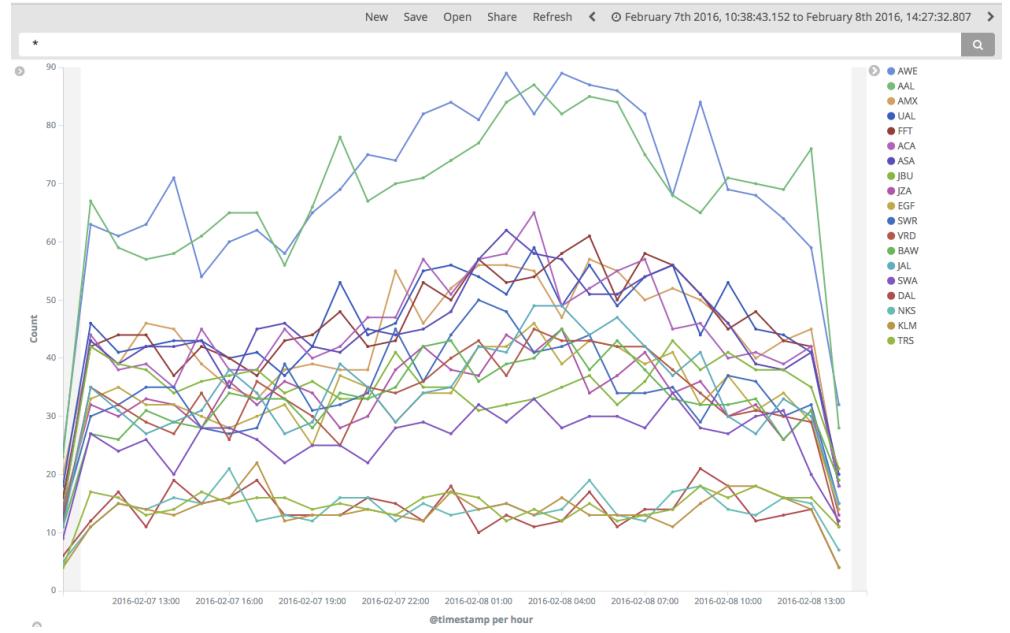


# What is “Normal”?

In general, this question can be answered in two ways:

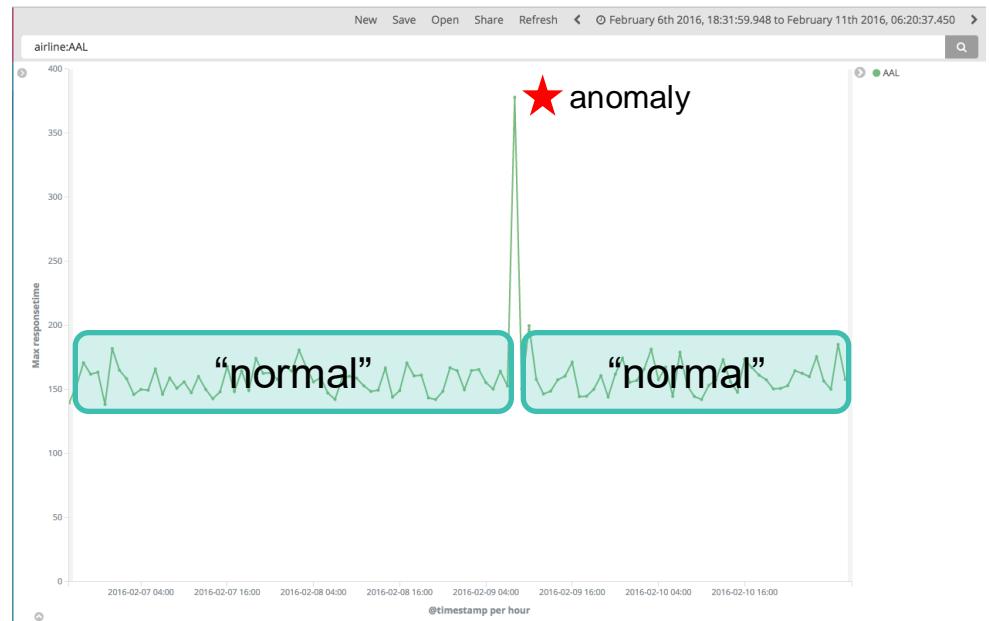
1) Something behaves in a consistent way with respect to itself, over time

2) Something behaves in a consistent way compared against similar entities



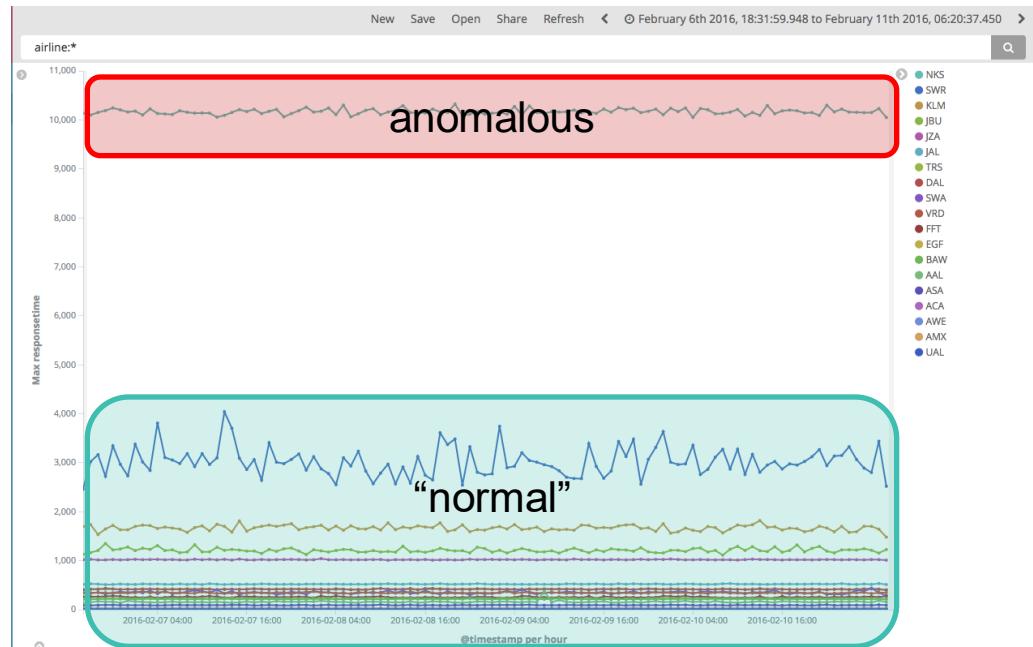
# What is “Abnormal”?

1) If something changes its behavior, compared to its own history – that change is *anomalous*.



# What is “Abnormal”?

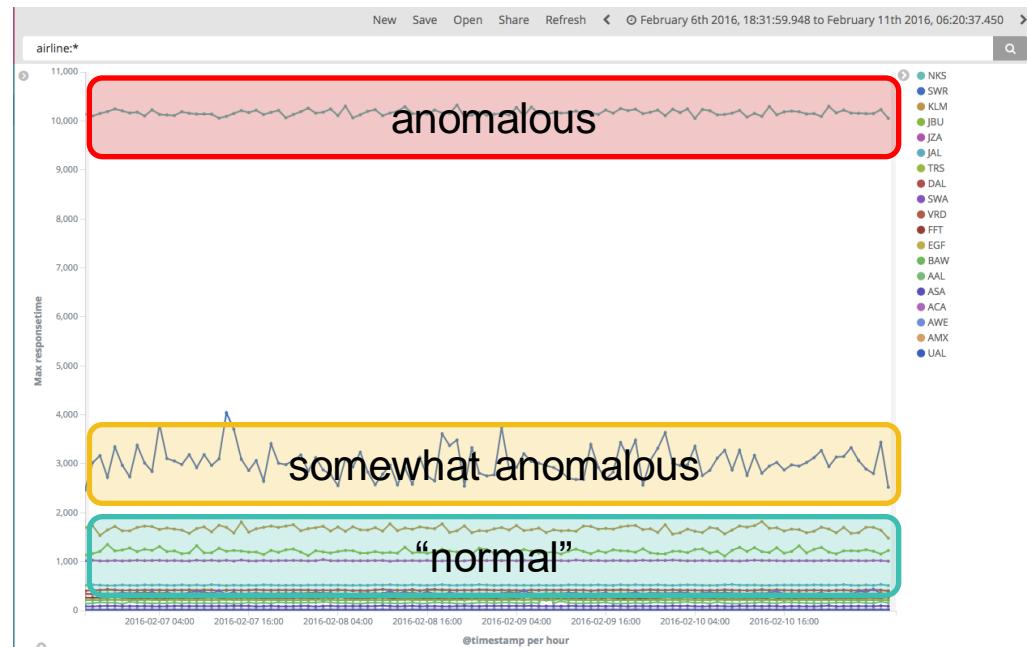
2) If something is drastically different than others within a population, then that entity is ***anomalous***.



# What is “Abnormal”?

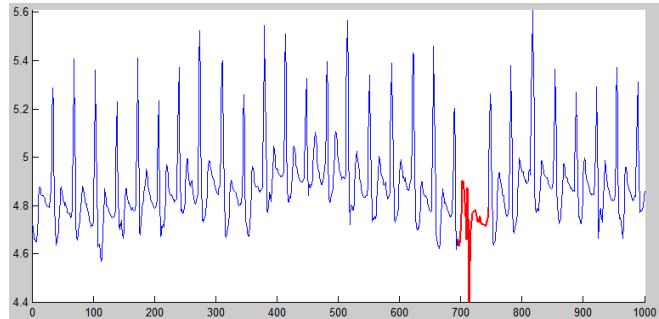
2) If something is drastically different than others within a population, then that entity is ***anomalous***.

There's also the concept of being “somewhat anomalous”



# In Summary, Anomaloussness is:

1) When an entities' ***behavior changes*** significantly and suddenly



2) When an entity is drastically ***different than others*** within a population



# How to Learn “Normal”

# An Analogy

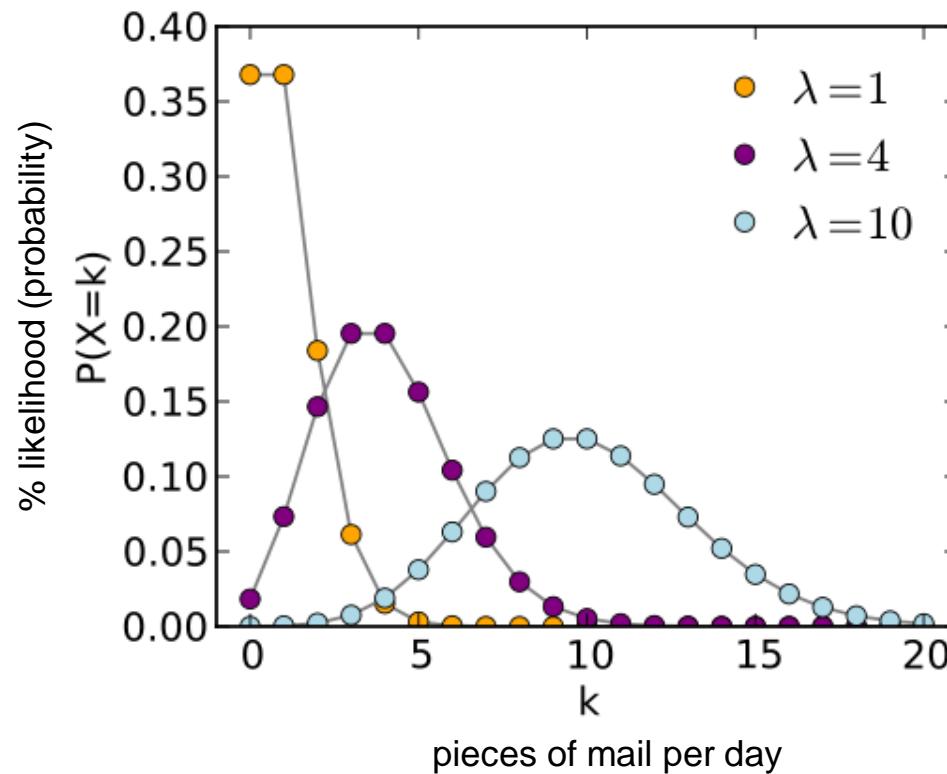
- How could I learn how much Postal-mail you get daily and how could I use that information to predict how much you might get tomorrow?



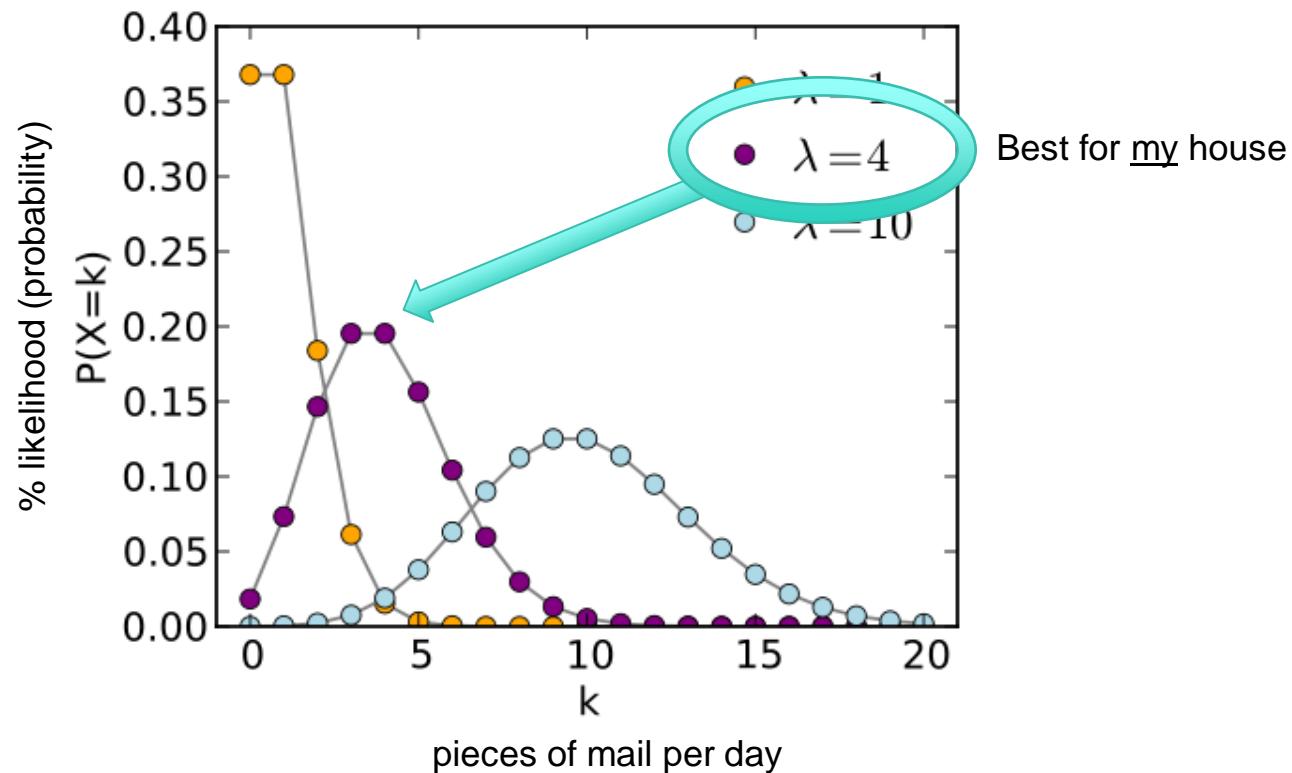
# I could stand on your front porch and...

- Watch your mail delivery volume for a while, but how long?
  - 1 day?
  - 1 week?
  - 1 month?
- Notice, that you intuitively feel like you'll gain accuracy in your predictions with more data that you see.
- Use those observations to create a model...

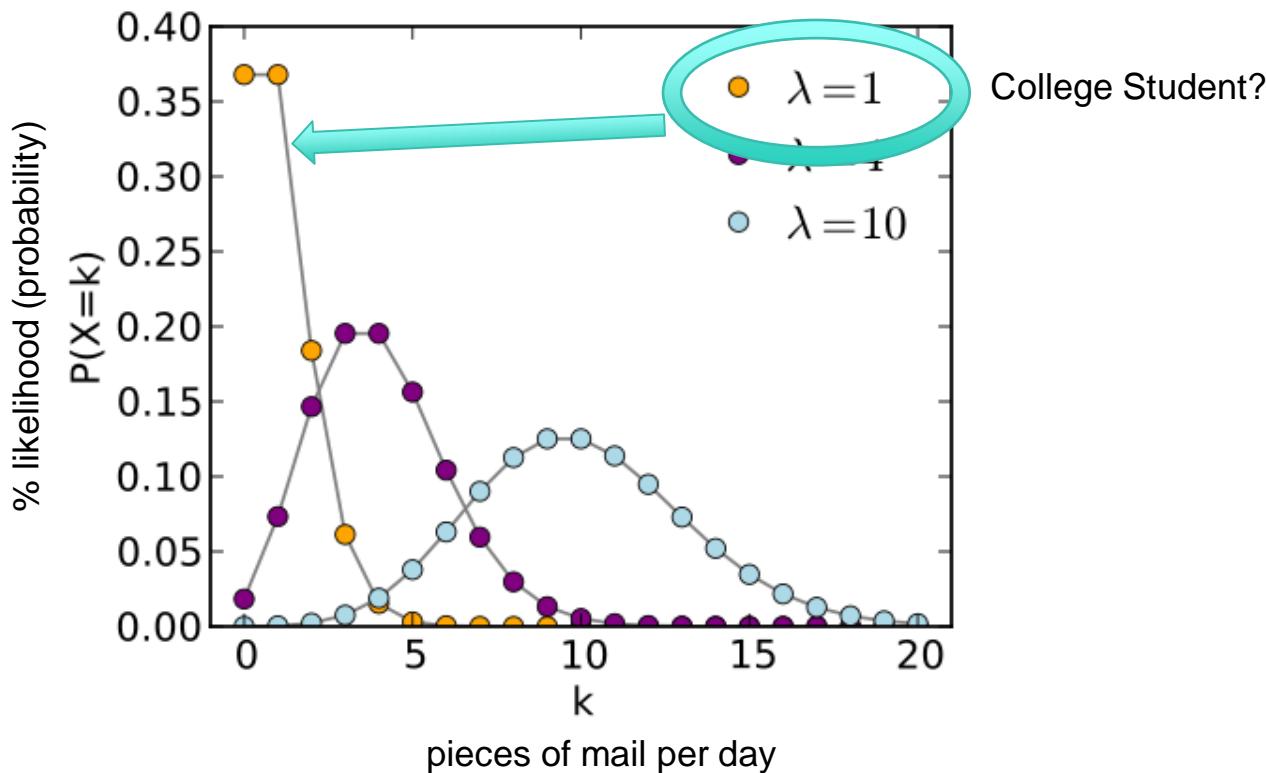
# Probability Distribution Function



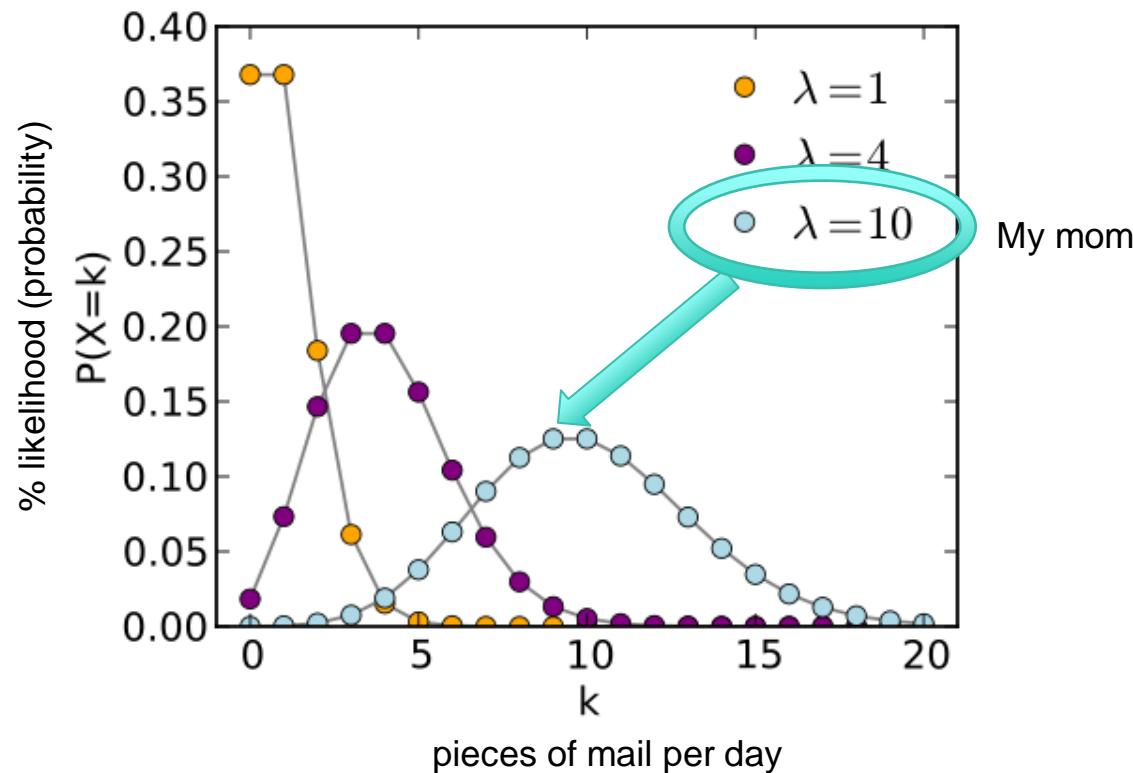
# Probability Distribution Function



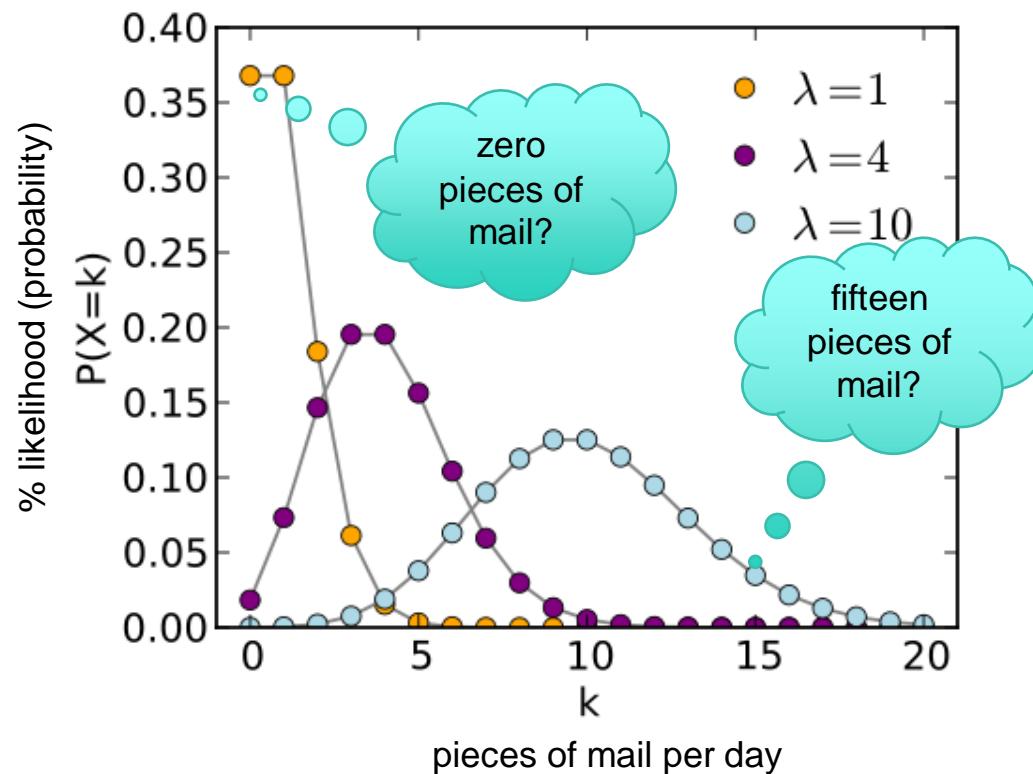
# Probability Distribution Function



# Probability Distribution Function



# Using the Model to Find What is Unexpected



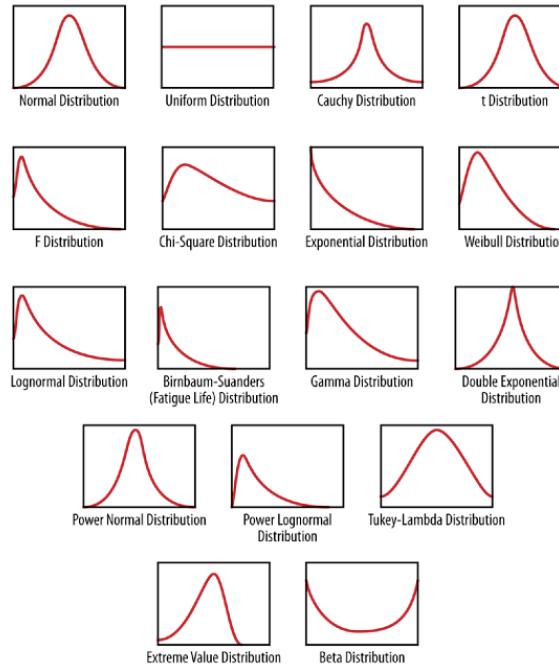
# Relate back to IT/Security data

- # Pieces of mail = # events of a certain type
  - number of failed logins
  - number of errors in a log
  - number of events with certain status codes
- Or, metrics
  - response time
  - utilization
  - orders per minute

=> Every kind of data will need its own unique “model” (probability distribution function)

# How does one pick a model?

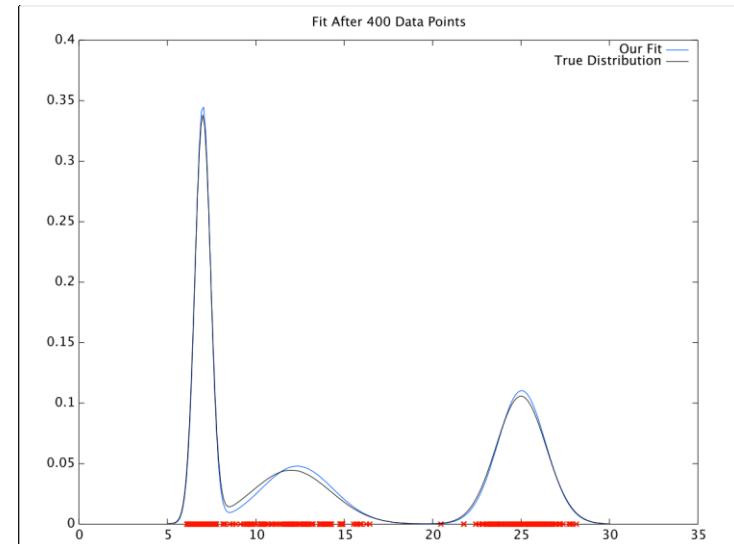
Which one best fits  
your data?



source: "Doing Data Science"  
O'Neil & Schutt

# Machine Learning picks it for you

- ML uses sophisticated machine-learning techniques to best-fit the right statistical model for your data.
- Better models = better outlier detection = less false alarms
- Anomalies occur when observation is in low probability area



# Also, model needs to account for any periodicity



After 2 full days,  
daily periodicity  
has been learned.

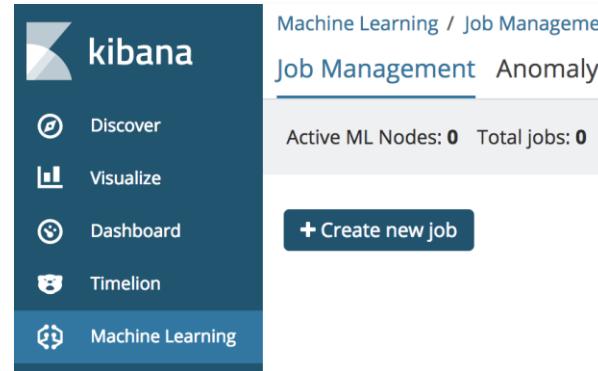
After 2 full weeks,  
weekly periodicity  
has been learned.

# Lab 1: The Single Metric Job

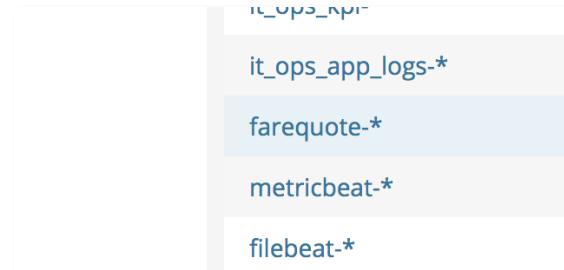
**Goal:** Detect anomalies in a single time series

# Steps to Complete

1) In Machine Learning,  
Create new job



2) Choose the “farequote-\*”  
index pattern



# Steps to Complete

3) pick the Single Metric Wizard

The screenshot shows the Kibana interface with a dark teal sidebar on the left containing icons and labels for various features: Discover, Visualize, Dashboard, Timelion, Canvas, Machine Learning (which is highlighted in blue), Infrastructure, Logs, and APM. To the right of the sidebar, the URL bar shows "Machine Learning / Job Management / Create New Job". The main content area has a light gray background with the title "Create a job from the i" partially visible. Below it, a section titled "Use a wizard" is shown with the sub-instruction "Use one of the wizards to create a machine learning j". There are two large white callout boxes. The first callout box contains a green circle with a plus sign and the text "Single metric Detect anomalies in a single time series.". The second callout box is partially visible on the right. At the bottom of the main content area, there is a link "Learn more about your data".

# Steps to Complete

4) Aggregation: count

5) Field: <leave blank>

6) Bucket span: 10m

7) Click the “use full farequote data” button

8) Name: “farequote”

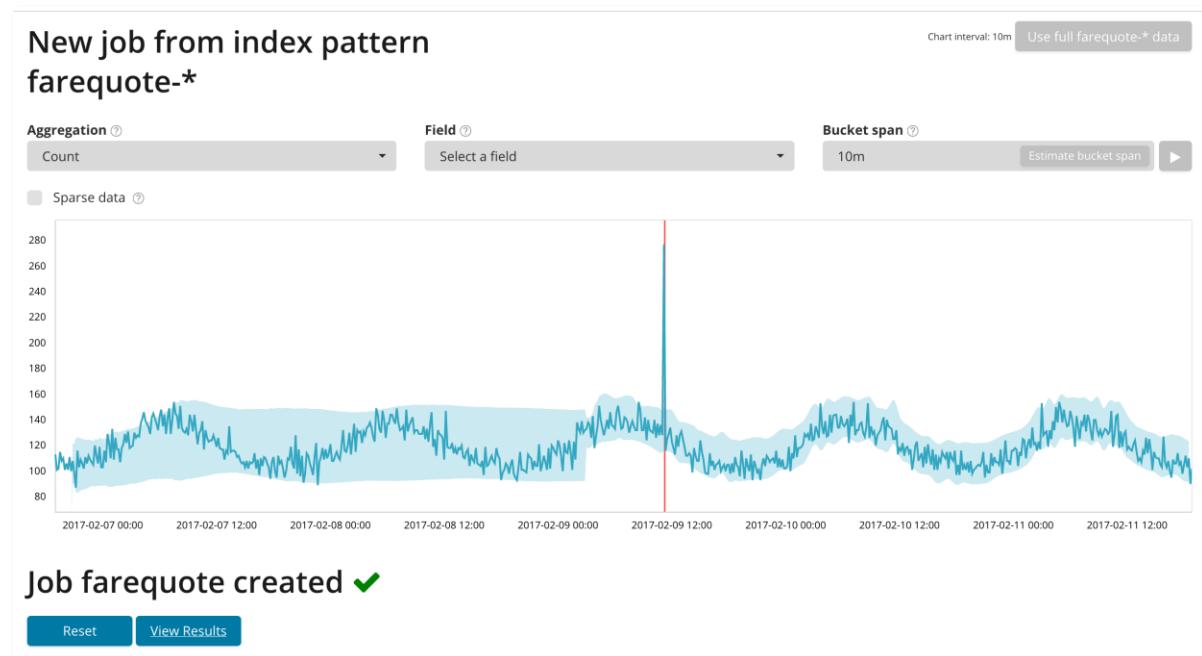
9) click “Create Job”



# Steps to Complete

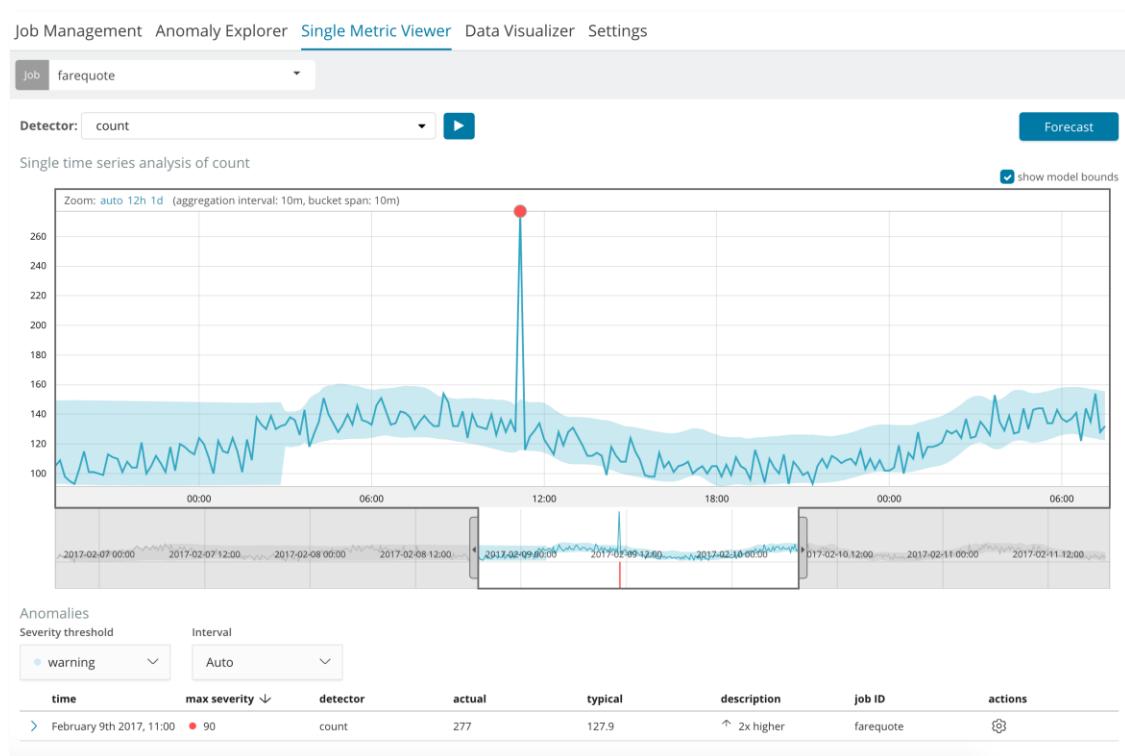
10) See animated learning

11) click “View Results”



# Steps to Complete

12) Zoom in on anomaly



# Splitting the Analysis

# Feature Selection and Selection

- Which attributes of this data could be used to judge its unusualness?

## raw logs

```
2016/02/08 06:20:43 INFO [http-8680]: FareQuoteImpl - FareQuoteImpl.getFare(AAL): exiting: 92.5638  
2016/02/08 06:20:44 INFO [http-8680]: FareQuoteImpl - FareQuoteImpl.getFare(JZA): exiting: 990.4628  
2016/02/08 06:20:46 INFO [http-8680]: FareQuoteImpl - FareQuoteImpl.getFare(JBU): exiting: 877.5927  
...
```

document

```
{  
  "_index": "farequote",  
  "_type": "response",  
  "_id": "AVNQ1__XRcuRIYtw-jH",  
  "_score": 3.290889,  
  "_source": {  
    "sourcetype": "farequote",  
    "airline": "AAL",  
    "responsetime": "92.5638",  
    "time": "2016-02-08T06:20:43+0000"  
  }  
}
```



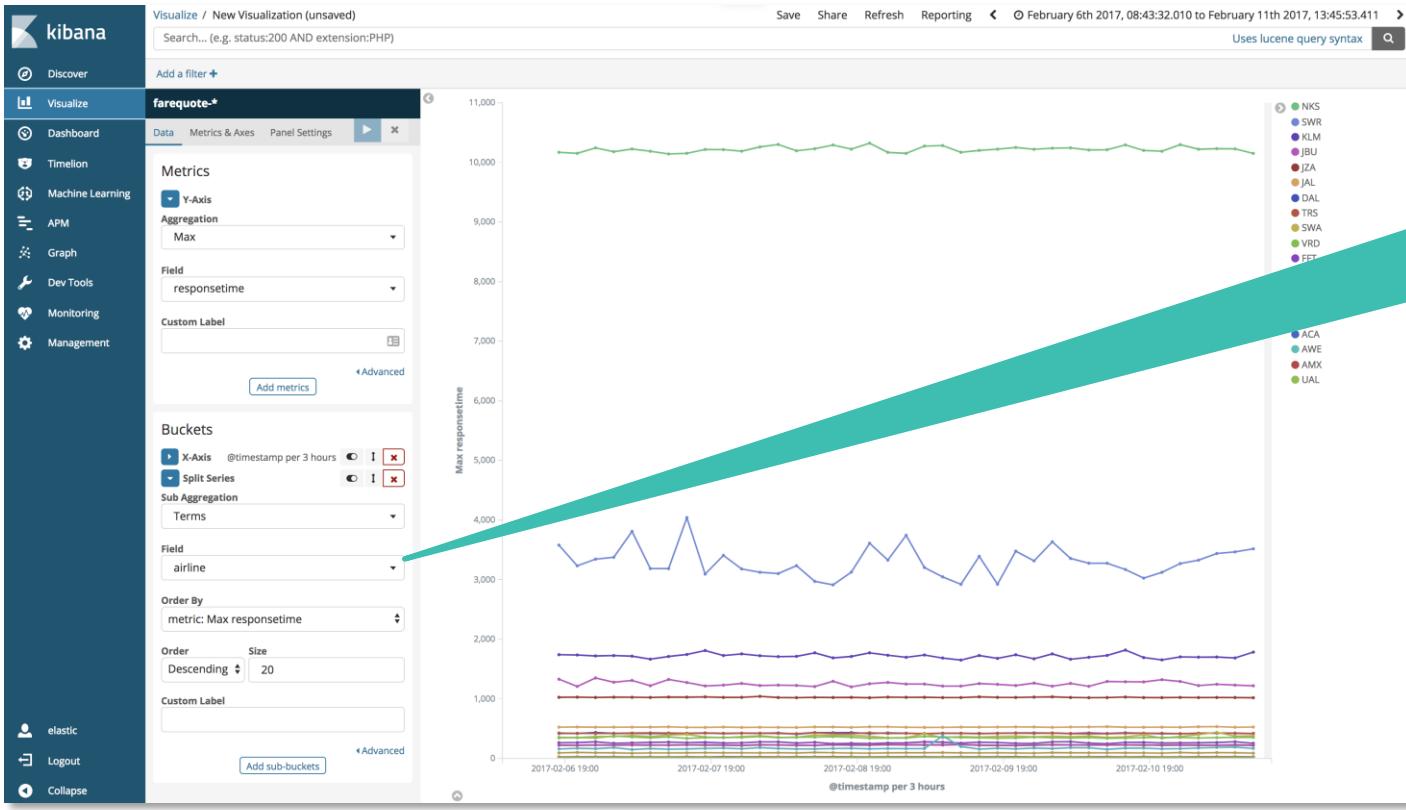
“airline”  
(categorical)

“responsetime”  
(metric)

# What Kinds of Questions Can be Answered?

QUESTION	ANSWERABLE?
Is there an unusual <b>amount</b> of requests per unit time (total)?	Yes
Is there any particular airlines with unusual <b>amounts</b> of requests per unit time?	Yes
Is the <b>total</b> response time of <b>all</b> API calls unusually long?	Yes
Is the response time of API calls <b>per airline</b> unusually long?	Yes
Are there any airlines with excessive take-off delays?	No

# In Kibana: How to Answer that Question



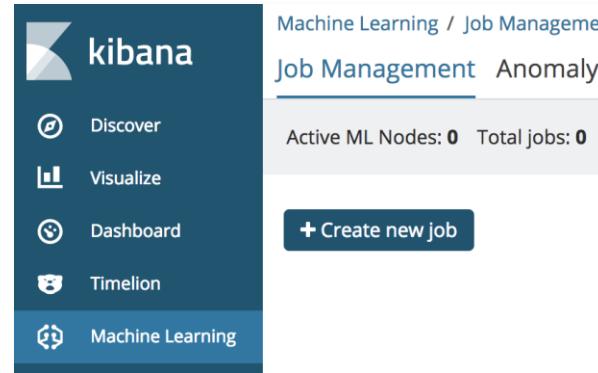
*ML has the same notion of “splitting” along a categorical field*

# Lab 2: Multi-Metric Jobs

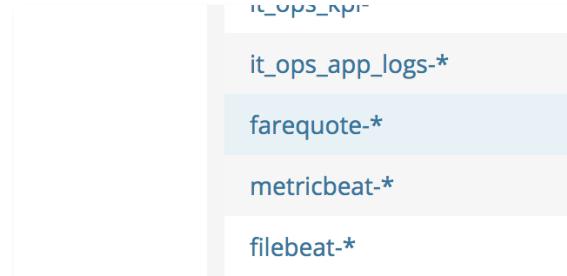
Goal: Detect anomalies in parallel time series

# Steps to Complete

1) In Machine Learning,  
Create new job



2) Choose the “farequote-\*”  
index pattern



# Steps to Complete

- 3) pick Multi metric Job Wizard



## Multi metric

Detect anomalies in multiple metrics by splitting a time series by a categorical field.

# Steps to Complete

4) choose

- event rate, count
- responsetime, max

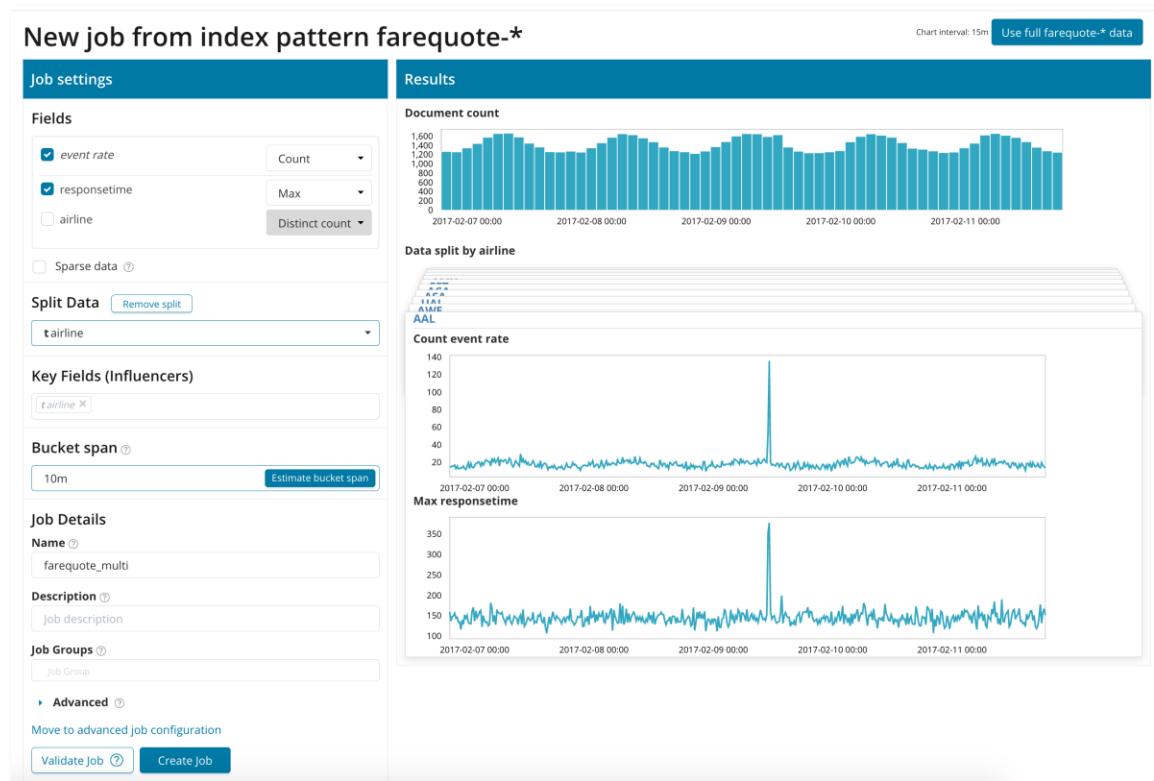
5) bucket span: 10m

6) Split Data: airline

7) click “use full farequote data”

8) Name: “farequote\_multi”

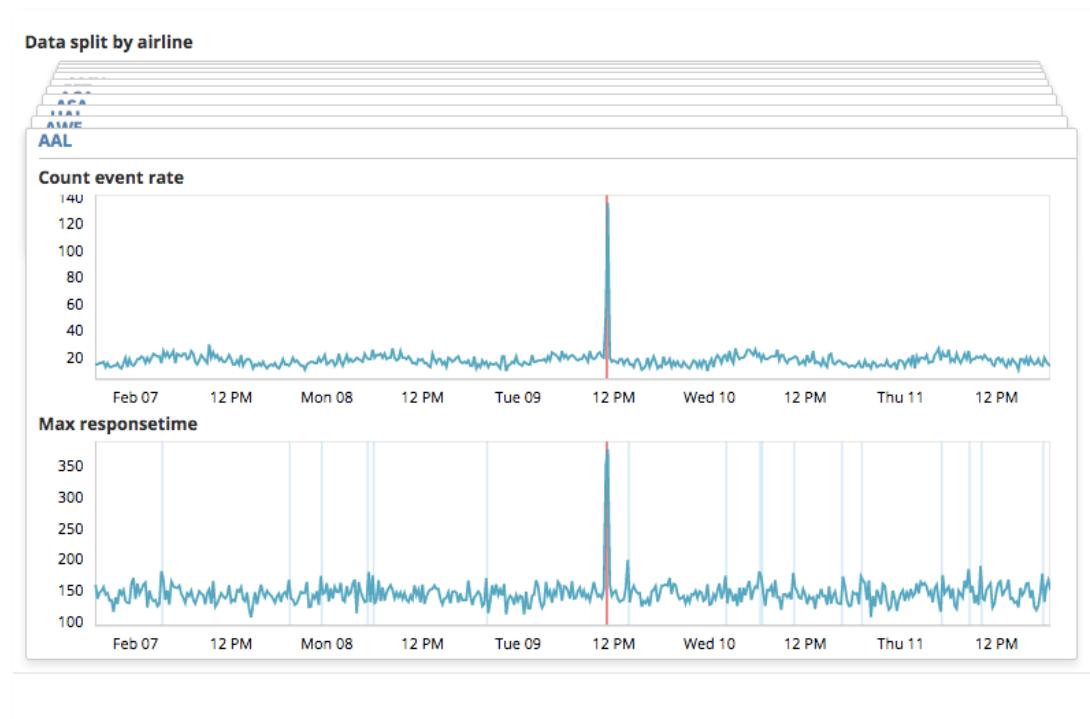
9) click “Create Job”



# Steps to Complete

10) See animated learning

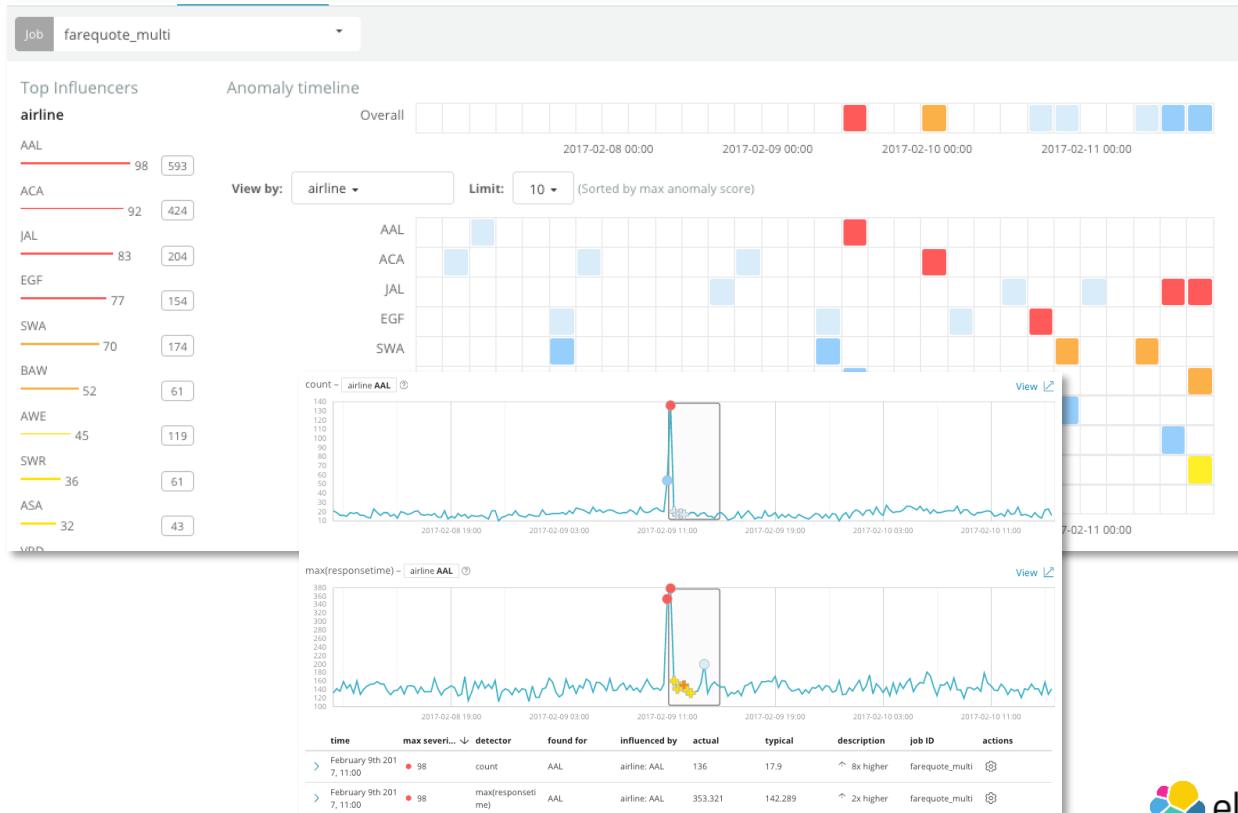
11) click “View Results”



# Steps to Complete

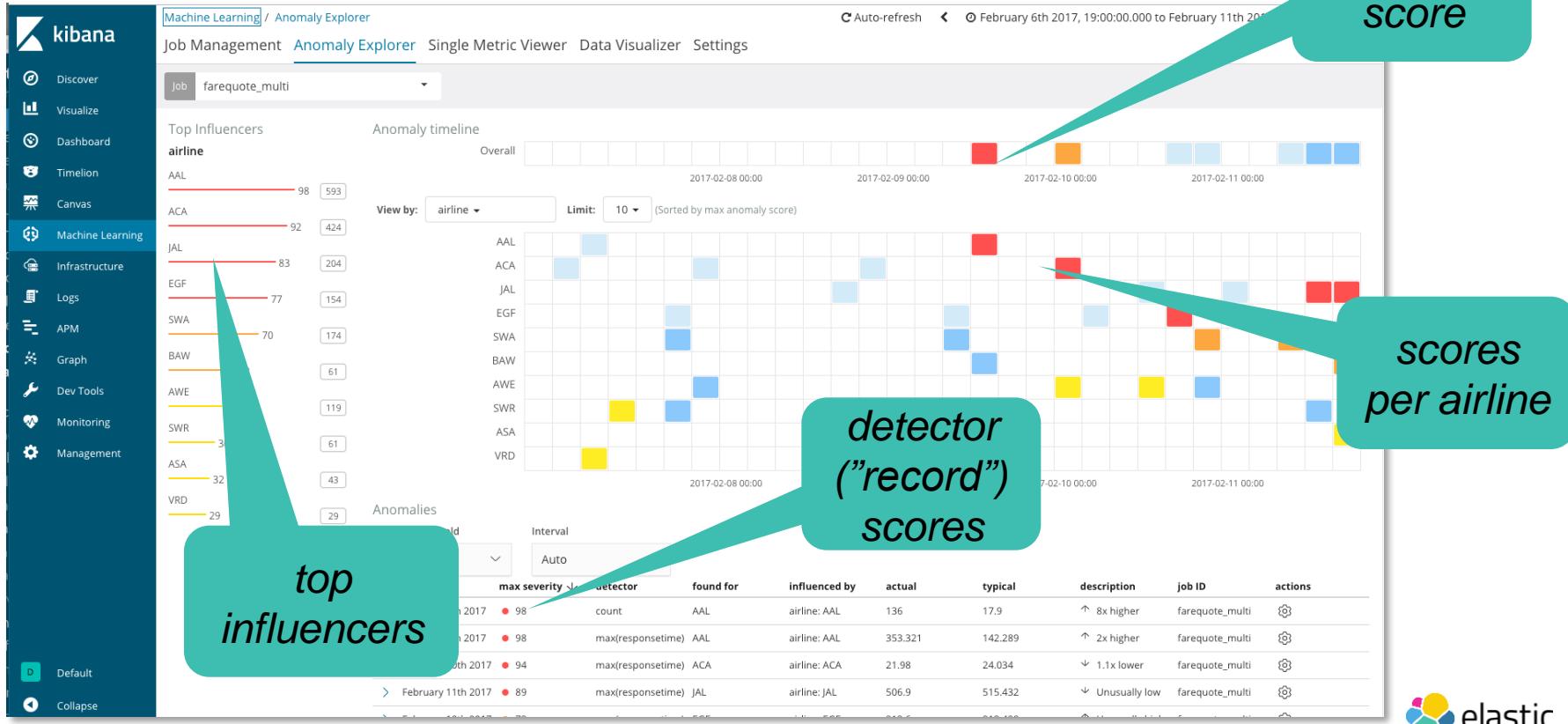
## 12) Result:

top unusual airline: AAL  
anomalies  
in both count  
and response time



# The Anomaly Explorer View

# Anomaly Explorer



# Concept: What is an Influencer?

- An Influencer is a field, selected at configuration time, that would be a logical entity "to blame" if an anomaly were to exist
- Doesn't have to be a field in the actual detector, but fields used to split the data are often good candidates
- Will get its own score based upon how influential that entity is on the anomaly

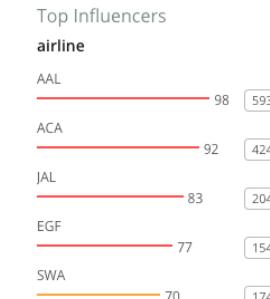
# Scoring

- Overall Job score is 90
  - How unusual is that bucket, given all airlines?
- Detector scores are 98
  - How unusual is the response time and count of airline=AAL?
- airline=AAL is the top influencer in this time range
  - 98 is the max score
  - 593 is the sum of all scores over the time range

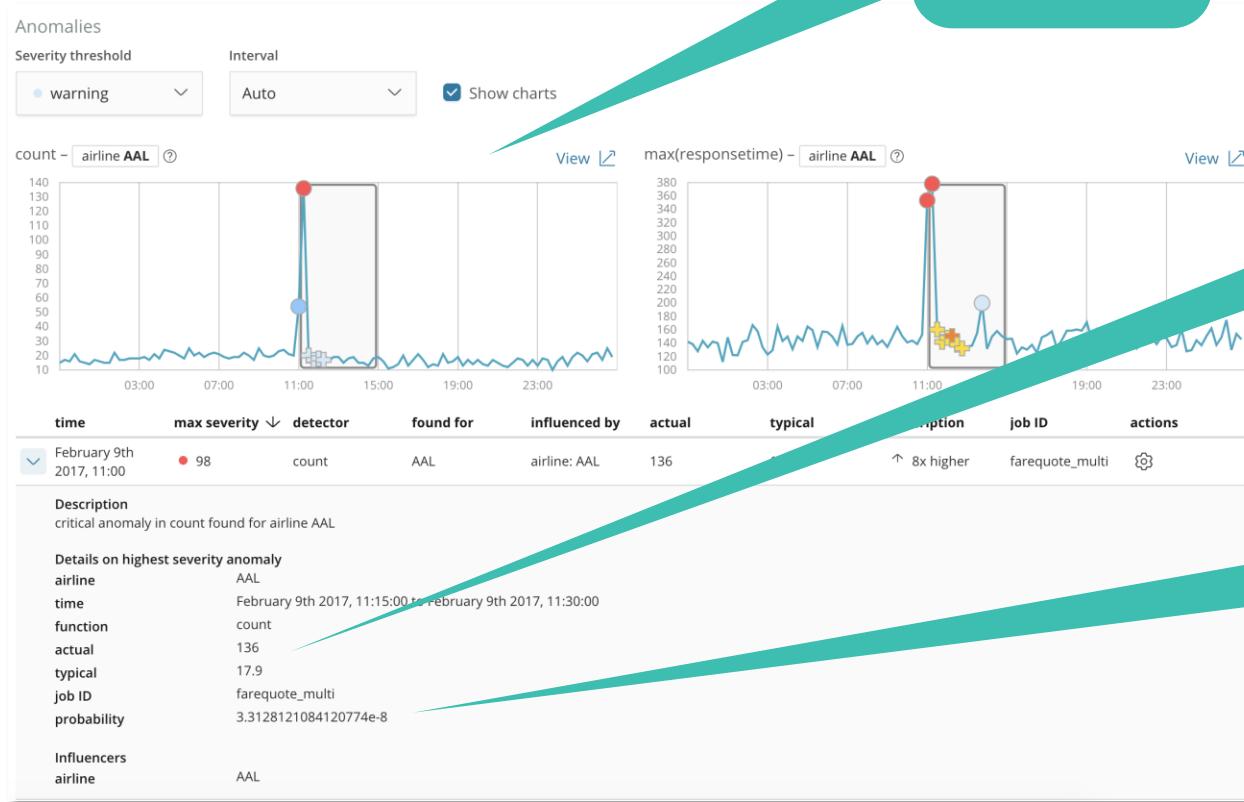
Anomaly timeline

Overall 2017-02-08 00:00 2017-02-09 00:00 February 9th, 2017, 11:00 Max anomaly score: 90 2017-02-11 00:00

Severity threshold	Interval	time	max severity	detector	found for	influenced by	actual	typical	description	job ID	actions
warning	Auto	> February 9th 2017	98	count	AAL	airline: AAL	136	17.9	↑ 8x higher	farequote_multi	
		> February 9th 2017	98	max(responseTime)	AAL	airline: AAL	353.321	142.289	↑ 2x higher	farequote_multi	



# Anomaly Details



view individual anomalies

actual vs. “typical”

raw probability

# Custom URLs: Linking to other places, in context

# Custom URLs in Job config

Job details Detectors Datafeed Custom URLs

Create new custom URL

Label: Raw data

Link to: Discover

Index pattern: farequote-\*

Query entities: airline

Time range: auto

Add Test ↗

The screenshot shows the 'Edit' interface of a job configuration. The 'Custom URLs' tab is active. A modal window titled 'Create new custom URL' is displayed, containing fields for labeling the URL ('Label: Raw data'), linking it to a specific type ('Link to: Discover'), specifying an index pattern ('Index pattern: farequote-\*'), filtering by query entities ('Query entities: airline'), and setting a time range ('Time range: auto'). Buttons for 'Add' and 'Test' are at the bottom of the modal.

Name the  
label

where to  
link

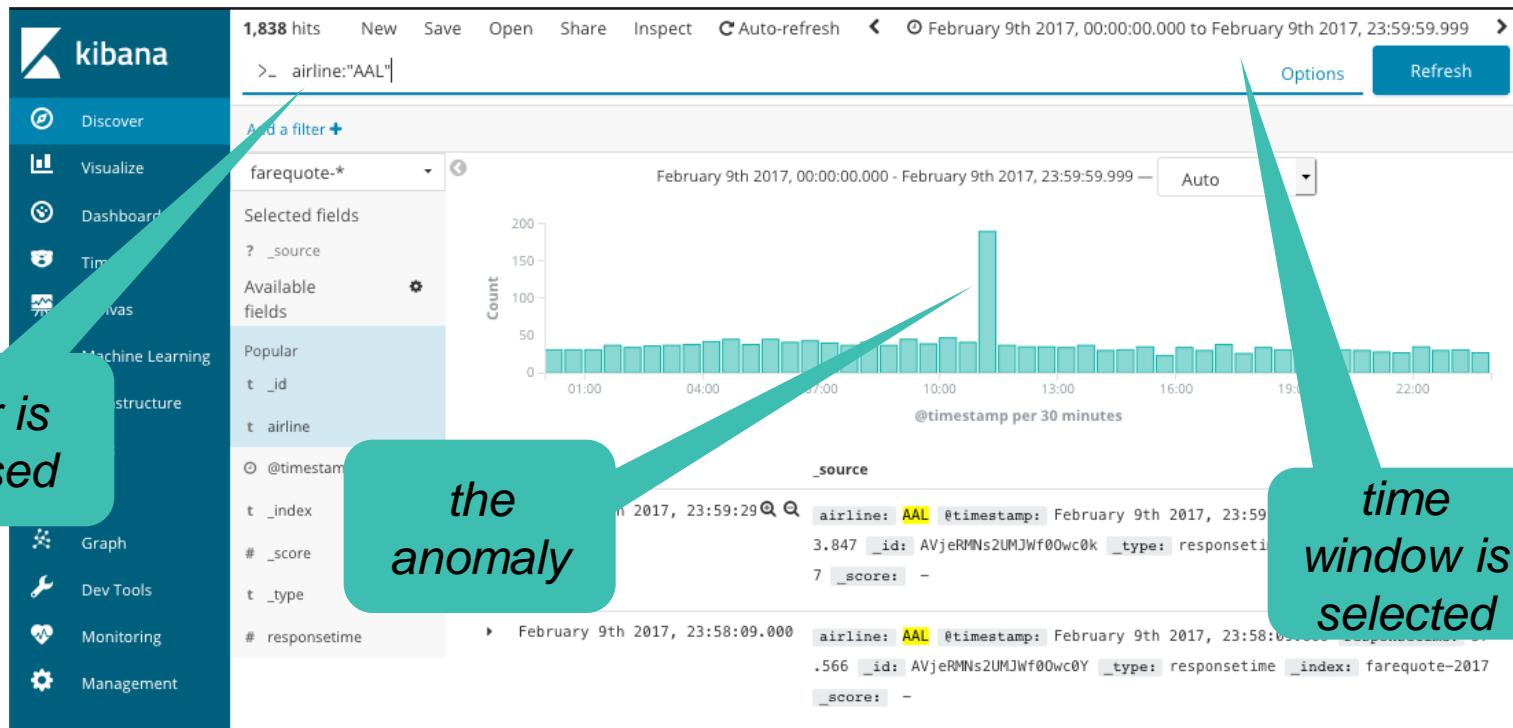
fields for  
filtering

# In Anomaly Explorer

Anomalies									
Severity threshold		Interval							
time	max severi... ↓	detector	found for	influenced by	actual	typical	description	job ID	actions
> February 9th 2017	● 98	count	AAL	airline: AAL	136	17.9	↑ 8x higher	farequote_multi	⚙️
> February 9th 2017	● 98	max(responsetime)	AAL	airline: AAL	353.321	142.289	↑ 2x higher	farequote	↗ Raw data
> February 10th 2017	● 94	max(responsetime)	ACA	airline: ACA	21.98	24.034	↓ 1.1x lower	farequote	↗ View series
> February 11th 2017	● 89	max(responsetime)	JAL	airline: JAL	506.9	515.432	↓ Usually low	farequote	⚙️ Configure rules

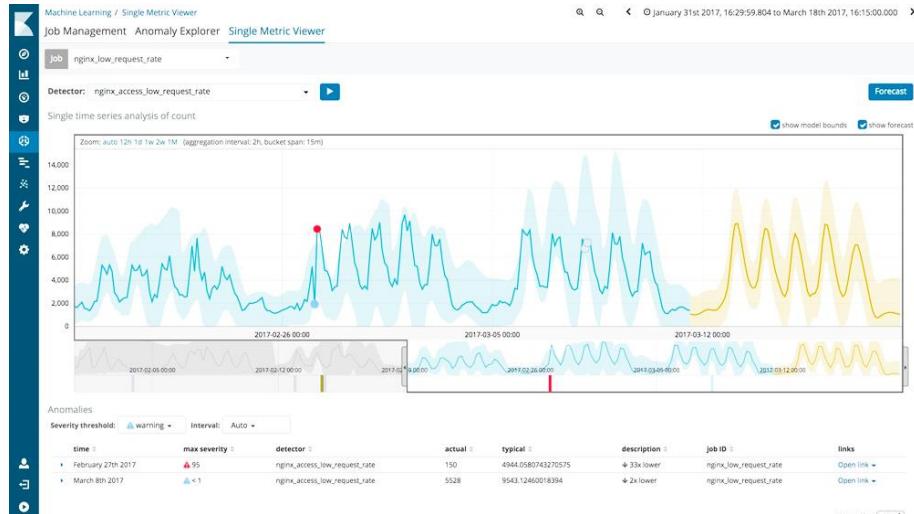
*Link now available*

# Clicking on Custom URL, passing context



# Forecasting

# Extrapolate into the future with forecasting



## Example use cases

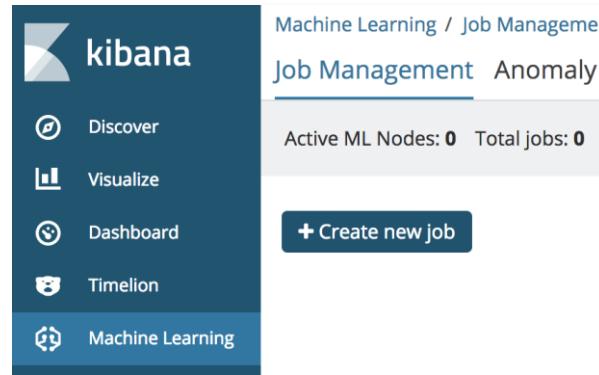
- I want to know when capacity will exceed 80%
- What is my expected visitor count next Saturday at 8am?

# Lab 3: Forecasting

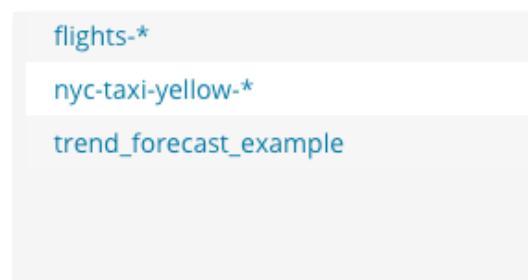
Goal: Extrapolate into the future

# Steps to Complete

1) In Machine Learning,  
Create new job



2) Choose the “trend\_forecast\_example”  
index pattern



# Steps to Complete

3) single metric job

4) sum of field “amount”

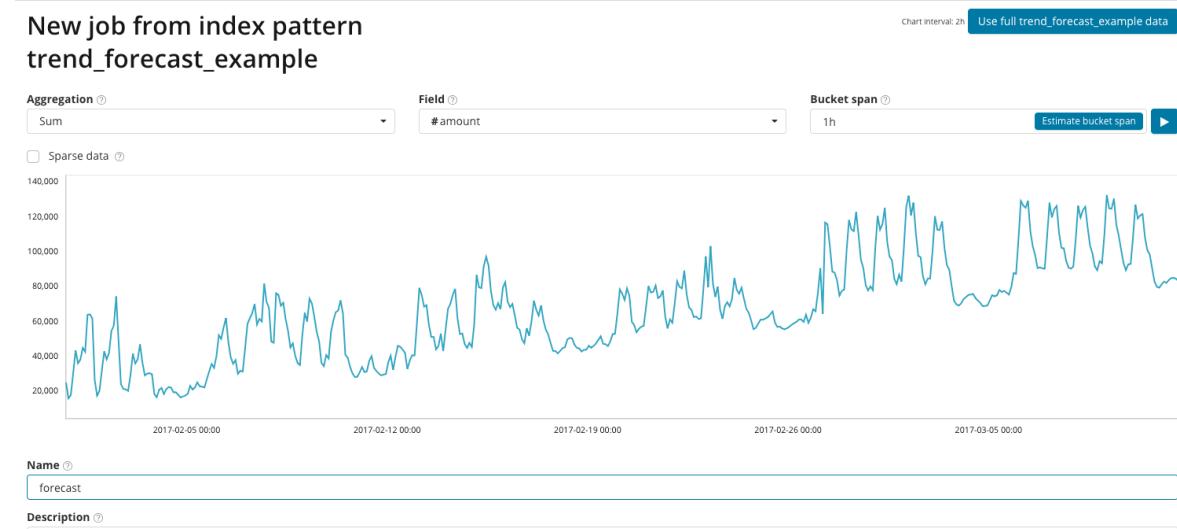
5) bucket span: 1h

6) click “use full data”

7) Name: “forecast”

8) click “Create Job”

9) click “View Results”



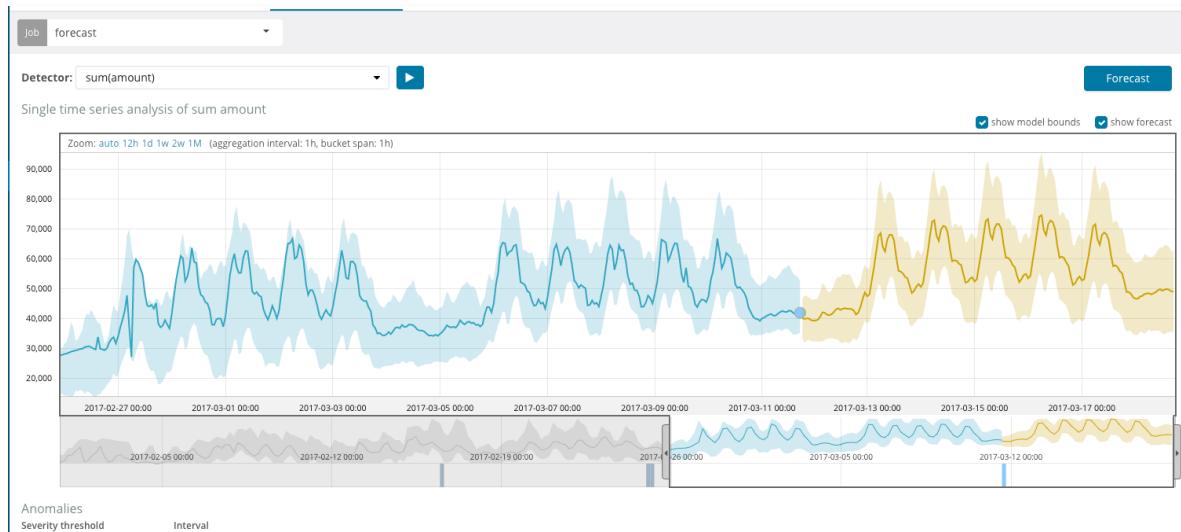
# Steps to Complete

10) click “Forecast” button

11) Enter “7d” for duration

12) click “Run” button

13) see forecast in UI  
(results also available via API)



# Population Analysis

# Population Analysis

- Use when:
  - Compare entities against peers
  - Not against own history
- Helpful when:
  - Entities have high-cardinality (i.e. external IP addresses)
  - Data for specific entities may be sparse in time (individual customers placing orders)
  - The behavior of the population as a whole is mostly homogeneous
- Not appropriate when:
  - Members of the population have vastly different behavior inherently.



## Population

Detect activity that is unusual compared to the behavior of the population.

# Population Analysis

*population field*

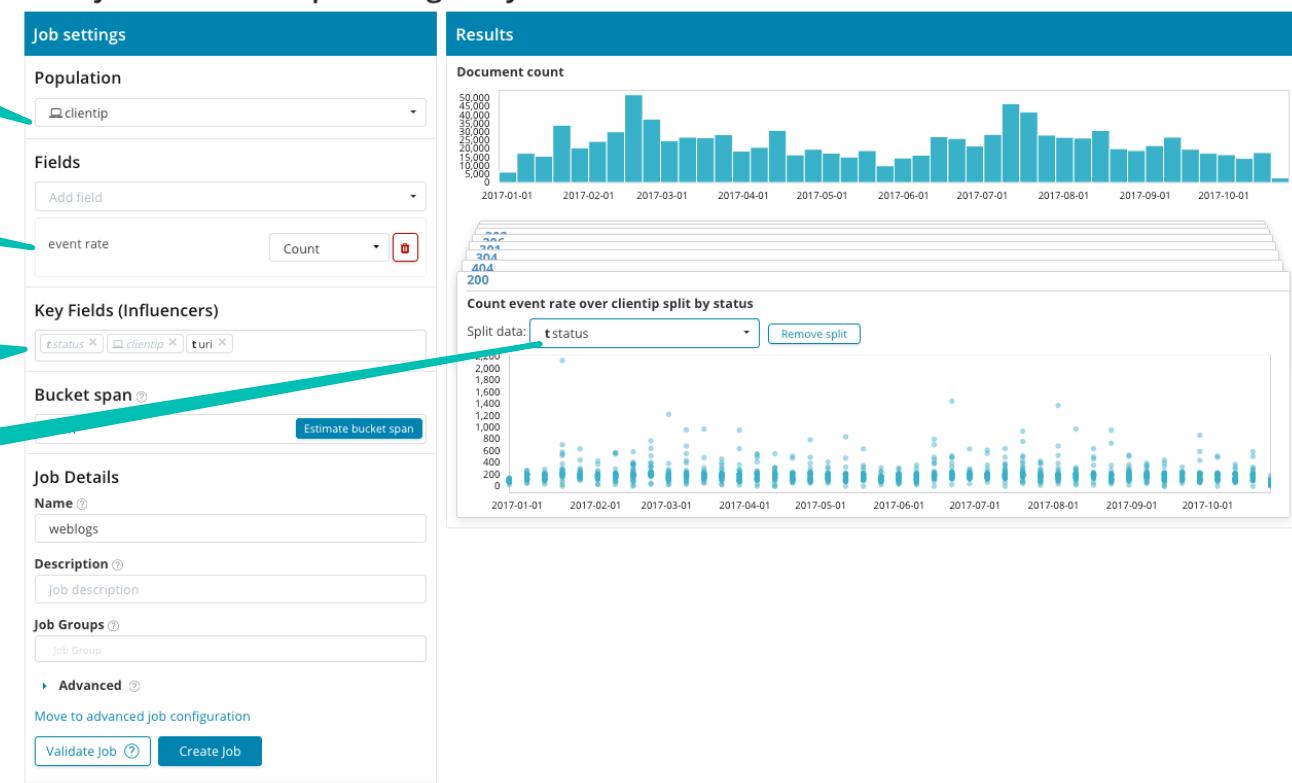
*the detector*

*influencers*

*optional split*

New job from index pattern gallery\*

Chart interval: 12h Use full gallery-\* data



# Population Analysis

clientip: 173.203.78.60  
status: 404  
uri: /wp-login.php



# Custom Rules

# Adding Custom Rules to ML (v6.4+)

Define rules that:

- Ignore the creation of anomalies
- Disqualify data from being modeled

Meet certain conditions:

- on “actual”, “typical” or difference

Optional scoping

- limit when rule is applied



## Create Rule

Job ID: farequote\_multi  
Detector: max(responsetime)  
Selected: actual 353.3, typical 142.3

Rules instruct anomaly detectors to change their behavior based on domain-specific knowledge that you provide. When you create a rule, you can specify conditions, scope, and actions. When the conditions of a rule are satisfied, its actions are triggered. [Learn more](#)

### Action

Choose the action(s) to take when the rule matches an anomaly.

- Skip result (recommended) ⓘ  
 Skip model update ⓘ

### Conditions

Add numeric conditions for when the rule applies. Multiple conditions are combined using AND.

WHEN actual IS LESS THAN 300 ⓘ

[Add new condition](#)

### Scope

Add a filter list to limit where the rule applies.

#### Rerun job

Changes to rules take effect for new results only.

To apply these changes to existing results you must clone and rerun the job. Note rerunning the job may take some time and should only be done once you have completed all your changes to the rules for this job.

[Close](#)

[Save](#)

# Calendars

# Ignoring Timeframes with Calendars

Define timeframes for ML to

- not create anomalies
- not allow model to see new data

Good for known, upcoming times

Calendars can apply to individual jobs or job groups

The screenshot shows a table with one row for the 'nyc\_holidays' calendar. The columns are 'ID', 'Jobs', and 'Events'. The 'ID' column contains 'nyc\_holidays', the 'Jobs' column contains 'taxi', and the 'Events' column contains '12 events'. There are 'Edit' and 'Delete' buttons at the bottom right of the table.

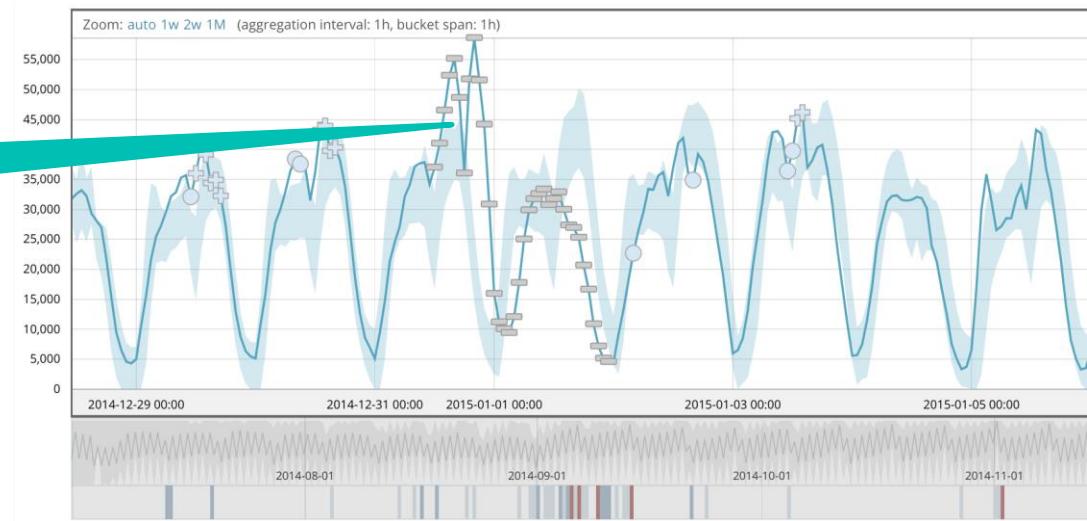
ID	Jobs	Events
nyc_holidays	taxi	12 events

The screenshot shows the 'Edit calendar nyc\_holidays' form. It includes fields for 'Calendar ID' (set to 'nyc\_holidays') and 'Description'. Under the 'Jobs' section, 'jobs' is listed. In the 'Groups' section, 'taxi' is listed. The 'Events' section lists 12 events with columns for 'Description', 'Start', and 'End'. Events include 'Christmas Day' (2014-12-25 to 2014-12-26), 'Christmas Eve' (2014-12-24 to 2014-12-25), 'Columbus Day' (2014-10-13 to 2014-10-14), 'Day after Christmas' (2014-12-26 to 2014-12-27), 'Day after Thanksgiving' (2014-11-28 to 2014-11-29), 'Independence Day' (2014-07-04 to 2014-07-05), 'Labor Day' (2014-09-01 to 2014-09-02), 'Martin Luther King Jr Day' (2015-01-19 to 2015-01-20), 'New Years Day' (2015-01-01 to 2015-01-02), 'New Years Eve' (2014-12-31 to 2015-01-01), 'Thanksgiving' (2014-11-27 to 2014-11-28), and 'Veterans Day' (2014-11-11 to 2014-11-12). Each event has a 'Delete' button. A note at the bottom says 'Times are displayed in the browser timezone'.

Description	Start	End
Christmas Day	2014-12-25 00:00:00	2014-12-26 00:00:00
Christmas Eve	2014-12-24 12:00:00	2014-12-25 00:00:00
Columbus Day	2014-10-13 00:00:00	2014-10-14 00:00:00
Day after Christmas	2014-12-26 00:00:00	2014-12-27 00:00:00
Day after Thanksgiving	2014-11-28 00:00:00	2014-11-29 00:00:00
Independence Day	2014-07-04 00:00:00	2014-07-05 00:00:00
Labor Day	2014-09-01 00:00:00	2014-09-02 00:00:00
Martin Luther King Jr Day	2015-01-19 00:00:00	2015-01-20 00:00:00
New Years Day	2015-01-01 00:00:00	2015-01-02 00:00:00
New Years Eve	2014-12-31 12:00:00	2015-01-01 00:00:00
Thanksgiving	2014-11-27 00:00:00	2014-11-28 00:00:00
Veterans Day	2014-11-11 00:00:00	2014-11-12 00:00:00

# Ignoring Timeframes with Calendars

*Ignored times marked  
with  
“—” symbol*



# Advanced Jobs

# Advanced Jobs

- Gives more flexibility
- Access to additional capabilities
- Requires more understanding of configuration parameters

Create a job from the index pattern gallery-\* 

**Use a wizard**  
Use one of the wizards to create a machine learning job to find anomalies in your data.

 **Single metric**  
Detect anomalies in a single time series.

 **Multi metric**  
Detect anomalies in multiple metrics by splitting a time series by a categorical field.

 **Population**  
Detect activity that is unusual compared to the behavior of the population.

 **Advanced**  
Use the full range of options to create a job for more advanced use cases.

**Learn more about your data**  
If you're not sure what type of job to create, first explore the fields and metrics in your data.

 **Data Visualizer**  
Learn more about the characteristics of your data and identify the fields for analysis with machine learning.

# Configuration of Advanced Job

*Tabs of different information*

### Create a new job

Job Details    Analysis Configuration    Datafeed    Edit JSON    Data Preview

**Name** ?  
farequote\_response

**Description** ?  
Job description

**Job Groups** ?  
Job Group

**Custom URLs** ?  
+ Add Custom URL

Use dedicated index ?

**Model memory limit** ?  
1gb

[Validate Job](#) ?    [Save](#)    [Cancel](#)

# Analysis Configuration

*Setting of  
bucket\_span*

### Create a new job

Job Details   Analysis Configuration   Datafeed   Edit JSON   Data Preview

**bucket\_span** ⓘ  
15m

**summary\_count\_field\_name** ⓘ  
Select...

**categorization\_field\_name** ⓘ  
Select...

**Detectors** ⓘ

+ Add Detector

**Influencers** ⓘ

- action
- clientip
- file
- method

*Creation of  
“detector”*

### Add new detector

**Description** ⓘ  
max(responsetime) partition\_field\_name=airline

<b>function</b> ⓘ	<b>field_name</b> ⓘ	<b>by_field_name</b> ⓘ
max	responsetime	Select...
<b>over_field_name</b> ⓘ	<b>partition_field_name</b> ⓘ	<b>exclude_frequent</b> ⓘ
Select...	airline	Select...

Help for max ⓘ

Add   Cancel

# Detector Details

- **function** – min, max, mean, etc.
- **field\_name** – field function “operates on”
- **by\_field\_name** – “dependent” split
- **partition\_field\_name** – “independent” split
- **over\_field\_name** – defines a population

Add new detector

Description ?

max(responsetime) partition\_field\_name=airline

function <small>?</small>	field_name <small>?</small>	by_field_name <small>?</small>
max	responsetime	Select...
over_field_name <small>?</small>	partition_field_name <small>?</small>	exclude_frequent <small>?</small>
Select...	airline	Select...

Help for max ↗

Add Cancel

# Datafeed Details

- Datafeed manages how ML queries Elasticsearch
- See ML docs for configuration options

### Create a new job

Job Details   Analysis Configuration   **Datafeed**   Edit JSON   Data Preview

Datafeed job ⓘ

**Query** ⓘ  
{"match\_all":{}}

**Query delay** ⓘ  
60s

**Frequency** ⓘ  
450s

**scroll\_size** ⓘ  
1000

**Index**  
gallery-\*

**Time-field name**  
@timestamp

**Buttons:** Validate Job ⓘ   Save   Cancel

# Other Types of Analysis

# Rarity Analysis

# Rare Analysis

- Finding items that rarely occur is also often useful
  - Rarely occurring log messages
  - Rare running process names
  - Rare connection destinations
- ML has a rare function, but it should be noted that:
  - It is relative, i.e. it takes into account the frequency of other field values

A,B,A,B,A,C,B,A,B,A,C,A,B,C,A,C,B,A,C,B,A,C,X <=X is rare

A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X <= X is not rare

- Therefore it works best when there are plenty of routine messages to contrast the rare ones

# Example of Rare Analysis

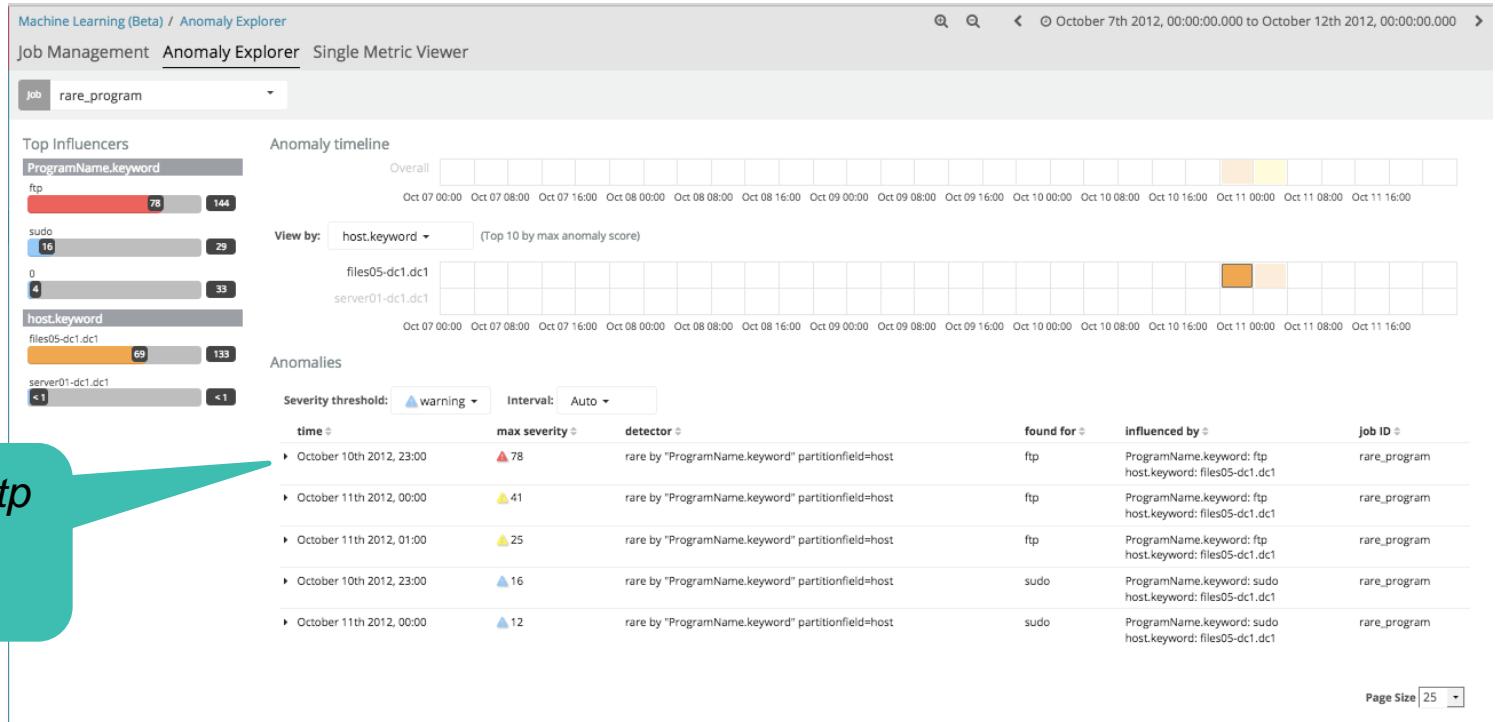
- Use Case: Security team @ services company
- Wanted to profile typical processes on each host using netstat

```
Active Internet connections (servers and established)
(index=netstat host="ids01-dc2" State=LISTEN (7/16/13 1:32:51AM))
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp      0      0  10.220.174.41:561        0.0.0.0:*
tcp      0      0  0.0.0.0:22             0.0.0.0:*
tcp      0      0  0.0.0.0:1241           0.0.0.0:*
tcp      0      0  0.0.0.0:8089           0.0.0.0:*
tcp      0      0  127.0.0.1:25            0.0.0.0:*
tcp      0      0  0.0.0.0:8000           0.0.0.0:*
tcp      0      0  0.0.0.0:8834           0.0.0.0:*
tcp      0      0  127.0.0.1:199          0.0.0.0:*
tcp      0      0  10.220.174.41:56002    10.220.174.40:3204  ESTABLISHED 4055/bashscriptd
```

- Goal was to identify rare processes that “start up and communicate” for each host, individually

# Example of Rare Analysis

detector: *rare by ProgramName partition\_field=host*



# Categorization

# Categorization

- What is it?
  - Automatically grouping similar log messages into the same “category”
- How?
  - Via an algorithm that looks at string similarity and clusters similar log lines
- Why?
  - Bring structure to semi-structured data so that it can be analyzed

Benefit: Find anomalies in logs without knowing what they contain

# Example

```
Oct 22 18:17:58 localhost sshd[8903]: Invalid user admin from 41.43.112.199 port 41805
Oct 22 18:17:58 localhost sshd[8903]: input_userauth_request: invalid user admin [preauth]
Oct 22 18:17:59 localhost sshd[8903]: Connection closed by 41.43.112.199 port 41805 [preauth]
Oct 22 20:58:03 localhost sshd[2074]: Received disconnect from 72.93.85.203 port 53552:11: disconnected by user
Oct 22 20:58:03 localhost sshd[2074]: Disconnected from 72.93.85.203 port 53552
Oct 22 20:58:03 localhost sshd[2072]: pam_unix(sshd:session): session closed for user ec2-user
Oct 22 21:32:54 localhost sshd[8944]: pam_unix(sshd:session): session opened for user ec2-user by (uid=0)
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session opened for user ec2-user by ec2-user(uid=0)
Oct 22 21:35:16 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
```

# Step 1 – remove mutable text

```
Oct 22 18:17:58 localhost sshd[8903]: Invalid user admin from 41.43.112.199 port 41805
Oct 22 18:17:58 localhost sshd[8903]: input_userauth_request: invalid user admin [preauth]
Oct 22 18:17:59 localhost sshd[8903]: Connection closed by 41.43.112.199 port 41805 [preauth]
Oct 22 20:58:03 localhost sshd[2074]: Received disconnect from 72.93.85.203 port 53552:11: disconnected by user
Oct 22 20:58:03 localhost sshd[2074]: Disconnected from 72.93.85.203 port 53552
Oct 22 20:58:03 localhost sshd[2072]: pam_unix(sshd:session): session closed for user ec2-user
Oct 22 21:32:54 localhost sshd[8944]: pam_unix(sshd:session): session opened for user ec2-user by (uid=0)
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
Oct 22 21:35:15 localhost runuser: pam_unix(runuser-l:session): session opened for user ec2-user by ec2-user(uid=0)
Oct 22 21:35:16 localhost runuser: pam_unix(runuser-l:session): session closed for user ec2-user
```

# Step 2 – cluster similar messages together

Oct 22 18:17:58 localhost sshd[8903]: Invalid user admin from 41.43.112.199 port 41805

Oct 22 18:17:58 localhost sshd[8903]: input\_userauth\_request: invalid user admin [preauth]

Oct 22 18:17:59 localhost sshd[8903]: Connection closed by 41.43.112.199 port 41805 [preauth]

Oct 22 20:58:03 localhost sshd[2074]: Received disconnect from 72.93.85.203 port 53552:11: disconnected by user

Oct 22 20:58:03 localhost sshd[2074]: Disconnected from 72.93.85.203 port 53552

Oct 22 20:58:03 localhost sshd[2072]: pam\_unix(sshd session): session closed for user ec2-user

Oct 22 21:32:54 localhost sshd[8944]: pam\_unix(sshd session): session opened for user ec2-user by (uid=0)

Oct 22 21:35:15 localhost runuser: pam\_unix(runuser-l session): session closed for user ec2-user

Oct 22 21:35:15 localhost runuser: pam\_unix(runuser-l session): session opened for user ec2-user by ec2-user(uid=0)

Oct 22 21:35:16 localhost runuser: pam\_unix(runuser-l session): session closed for user ec2-user

## Step 2 – cluster similar messages together

Oct 22 18:17:58 localhost sshd[8903]: Invalid user admin from 41.43.112.199 port 41805

mlcategory:1

Oct 22 18:17:58 localhost sshd[8903]: input\_userauth\_request: invalid user admin [preauth]

mlcategory:2

Oct 22 18:17:59 localhost sshd[8903]: Connection closed by 41.43.112.199 port 41805 [preauth]

mlcategory:3

Oct 22 20:58:03 localhost sshd[2074]: Received disconnect from 72.93.85.203 port 53552:11: disconnected by user

mlcategory:4

Oct 22 20:58:03 localhost sshd[2074]: Disconnected from 72.93.85.203 port 53552

mlcategory:5

mlcategory:6

Oct 22 20:58:03 localhost sshd[2072]: pam\_unix(sshd:session): session closed for user ec2-user

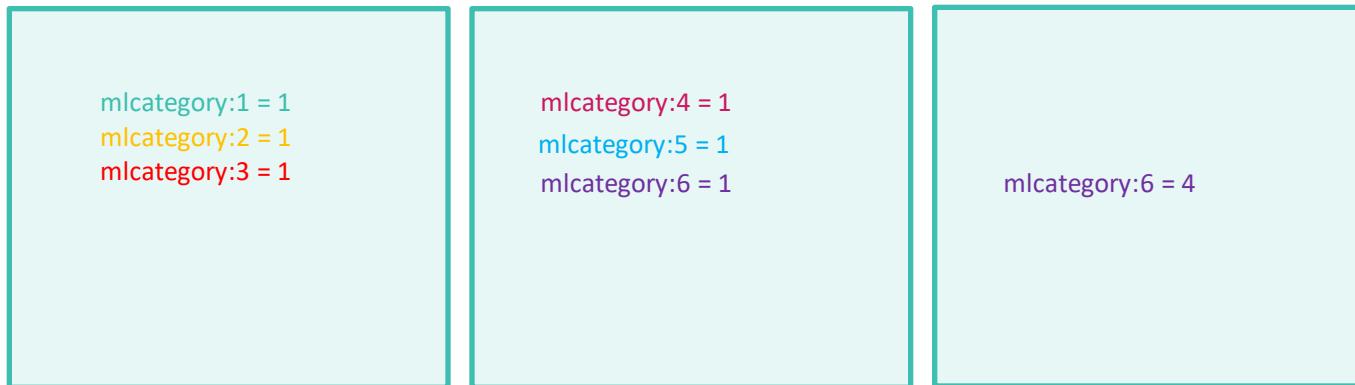
Oct 22 21:32:54 localhost sshd[8944]: pam\_unix(sshd:session): session opened for user ec2-user by (uid=0)

Oct 22 21:35:15 localhost runuser: pam\_unix(runuser-l:session): session closed for user ec2-user

Oct 22 21:35:15 localhost runuser: pam\_unix(runuser-l:session): session opened for user ec2-user by ec2-user(uid=0)

Oct 22 21:35:16 localhost runuser: pam\_unix(runuser-l:session): session closed for user ec2-user

## Step 3 – count per time bucket



time

# Categorization Example

- Configure an ML job to use:
  - “message” as the `categorization_field_name`
  - there will be a new, “magic” field called “`mlcategory`” that is dynamically created by ML to group similar messages together

The screenshot shows the 'Categorization' section of the Elasticsearch Machine Learning Settings. It includes fields for 'bucket\_span' (set to 10m), 'summary\_count\_field\_name' (empty), 'categorization\_field\_name' (set to 'message'), and a 'Categorization Filters' section with a '+ Add Categorization Filter' button. Below this is a 'Detectors' section containing a single entry: 'Unusual message counts count by mlcategory' with edit and delete icons, and a '+ Add Detector' button.

bucket\_span ⓘ  
10m

summary\_count\_field\_name ⓘ

categorization\_field\_name ⓘ  
message

Categorization Filters ⓘ  
+ Add Categorization Filter

Detectors ⓘ

Unusual message counts  
count by `mlcategory`

+ Add Detector

# Categorization Example – count by mlcategory

Anomaly timeline



View by: job ID (Top 10 by max anomaly score)



Anomalies

Severity threshold:	warning	Interval:	Auto	time	max severity	detector	found for	actual	typical	description	job ID	links	category examples
►	February 8th 2016, 10:00	▲ 66	count by mlcategory	mlcategory 11	49	0.0820658	↑ More than 100x higher	logs		Open link ↴ Fall To Connect Database ReActivate Application / Client Connection !			
►	February 8th 2016, 10:00	▲ 66	count by mlcategory	mlcategory 10	49	0.0820658	↑ More than 100x higher	logs		Open link ↴ DBMS ERROR : db=10.16.1.63!svc_prod#uid=dbadmin1;pwd=##### ! DBMS ERROR : db=svc_prod Err=-17 [Microsoft][ODBC SQL Server Driv			
►	February 8th 2016, 10:00	▲ 43	count by mlcategory	mlcategory 9	1	0.00336345	↑ More than 100x higher	logs		Open link ↴ DB Not Updated [Master] Table;dbhost=dbserver.acme.com;physical			
►	February 8th 2016, 05:00	▲ 16	count by mlcategory	mlcategory 6	1	0.00502013	↑ More than 100x higher	logs		Open link ↴ Transaction Match In DB / Duplicate Transaction;dbhost=dbserver.ac			
►	February 8th 2016, 09:00	▲ 8	count by mlcategory	mlcategory 6	1	0.00657718	↑ More than 100x higher	logs		Open link ↴ Transaction Match In DB / Duplicate Transaction;dbhost=dbserver.ac			
►	February 8th 2016, 10:00	▲ 4	count by mlcategory	mlcategory 2	49	0.081315	↑ More than 100x higher	logs		Open link ↴ REC Not INSERTED [DB TRAN] Table;dbhost=dbserver.acme.com;physi			
►	February 8th 2016, 10:00	▲ 2	count by mlcategory	mlcategory 5	7	0.0600863	↑ More than 100x higher	logs		Open link ↴ Opening Database = DRIVER={SQL Server};SERVER=10.16.1.63!netwo			
►	February 8th 2016, 10:00	▲ 2	count by mlcategory	mlcategory 3	1	0.0128763	↑ 78x higher	logs		Opening Database = DRIVER={SQL Server};SERVER=127.0.0.1;network			
►	February 8th 2016, 06:00	▲ 2	count by mlcategory	mlcategory 7	1	0.013673	↑ 73x higher	logs		Opening Database = DRIVER={SQL Server};SERVER=sssvcdbj1.acme.com!svc_prod#uid=dbadmin1;pwd=#####;db			
►	February 8th 2016, 10:00	▲ 1	count by mlcategory	mlcategory 4	2	0.0202842	↑ 99x higher	logs		Using: 10.16.1.63!svc_prod#uid=dbadmin1;pwd=#####;dbhost=dbserver.acme.com!svc_prod#uid=dbadmin1;pwd=#####;db			
►	February 8th 2016, 10:00	▲ 1	count by mlcategory	mlcategory 1	1	0.00010000	↑ 100x higher	logs		Actual Transaction Not Found In DB To VOID;dbhost=dbserver.acme.com!svc_prod#uid=dbadmin1;pwd=#####;db			
►	February 8th 2016, 10:00	▲ 1	count by mlcategory	mlcategory 1	1	0.00010000	↑ 100x higher	logs		012 Head Office Link Active 127.0.0.1;dbhost=dbserver.acme.com!ph;			

category name

example matching log messages

# Lab 4: Choose your own Adventure

# Options

- Track 1: Root Cause Analysis (the 4 `it_ops-*` indices)
  - Analyze logs and metrics together to solve a real-world problem
- Track 2: Analyze Web logs (`gallery-*` index)
  - Look for a brute-force authentication attack
- Track 3: Analyze NYC Taxi data (`nyc-taxi-yellow-*` index)
  - Find anomalies in taxicab ridership



# Lab 4, Track 1: Root Cause Analysis

# Lab 4: Track 1 - Problem

- Transaction processing app records # transactions per minute in an index called “it\_ops\_kpi-\*”. Find the anomaly in it.
- Also, SQL Database and Network performance metrics were gathered for the application environment (it\_ops\_network-\* and it\_ops\_sql-\*). Use multi-metric jobs to see what might be going on in there.
- Lastly, there’s an index (it\_ops\_app\_logs-\*) that has the application’s log lines in them. See if there’s anything anomalous in there as well.
- View all jobs correlated together in the Anomaly Explorer

# Lab 4, Track 2: Analyze Web Logs

## Lab 4: Track 2 - Problem

- There's been suspicion that automated bots are slamming the web site from time to time.
- Use a population analysis job to find any rogue IPs and see what URLs were being requested
- Data is in gallery-\* index

# Lab 4, Track 3: Analyze NYC Taxis

## Lab 4: Track 3 - Problem

- Taxi ridership for NYC Yellow cabs is in the nyc-taxi-yellow-\* index
- Determine time periods in which an unusual number of people take cab rides
- If those times are to be expected, use the Calendar to exclude them

# Appendix

# Alerting

# Alerting from the ML User Interface

- After job configuration, or starting a datafeed, see option for creating a “watch” on the live data:

The screenshot shows a line chart with data points fluctuating between 80 and 120 over time (Feb 07 to Mon 08). Below the chart, a message says "Job live\_farequote created". There are two buttons: "Reset" and "View Results". Underneath, there are two checked checkboxes: "Continue job in real-time" and "Create watch for real-time job". A table below has columns "Interval" and "Severity threshold". The "Interval" is set to "10m" and the "Severity threshold" is set to "critical". A blue arrow points from a callout bubble to the "Severity threshold" dropdown.

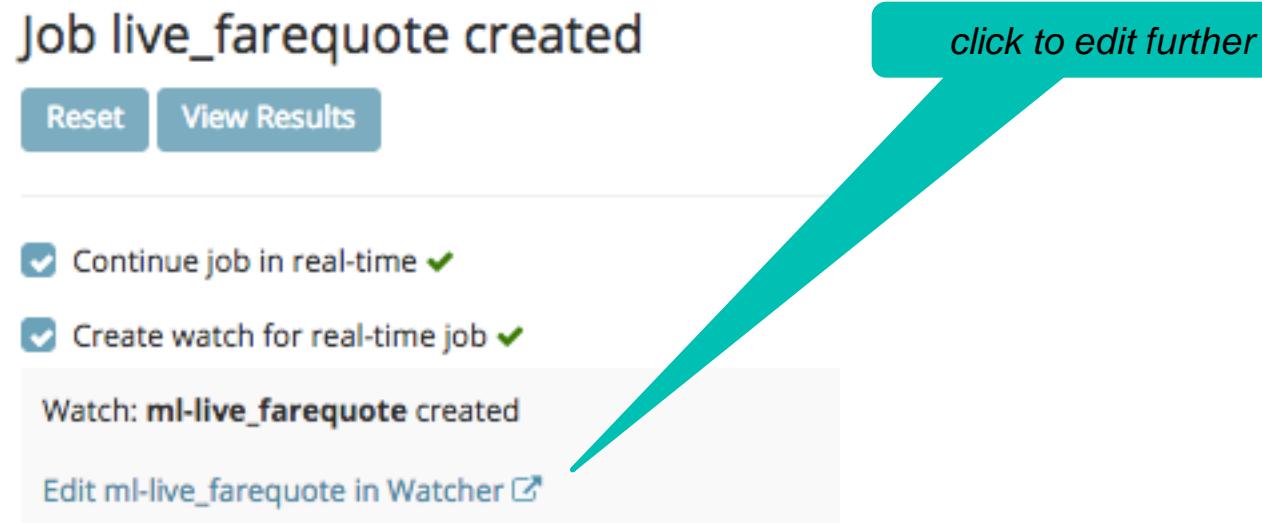
Interval	Severity threshold
10m	critical

Apply

*The minimum severity to alert upon*

# Alerting from the ML User Interface

- Clicking “Apply”:



# Alerting from the ML User Interface

- Now can edit Watch JSON to further customize things like action, etc.



Save Cancel Delete

Edit	Simulate	Simulation Results
------	----------	--------------------

ID  
ml-live\_farequote

Name

Watch JSON ([Syntax](#))

```
1  {
2    "trigger": {
3      "schedule": {
4        "interval": "92s"
5      }
6    },
7    "input": {
8      "search": {
9        "request": {
10          "search_type": "query_then_fetch",
11          "indices": [
12            ".ml-anomalies-*"
13          ],
14          "types": [],
15          "body": {
16            "size": 0,
17            "query": {
18              "bool": {
19                "filter": [
20                  {
21                    "term": {
22                      "job_id": "live_farequote"
23                    }
24                  },
25                  {
26                    "term": {
27                      "job_id": "live_farequote"
28                    }
29                  }
30                ]
31              }
32            }
33          }
34        }
35      }
36    },
37    "output": {
38      "index": "ml-live_farequote"
39    }
40  }
```

# Under the Hood: Connecting ML to Alerting

- For all jobs - all anomalies are stored in `.ml-anomalies-*` indices
- Every job also creates an index alias called `.ml-anomalies-<jobname>`
- You can configure Watches to query this index in 3 different ways:
  - “bucket” level
    - Answers: *How unusual was the job in a particular bucket of time?*
    - Essentially an aggregated anomaly score – useful for rate-limited alerts
  - “record” level
    - Answers: *What individual anomalies are present in a range of time?*
    - All the detailed anomaly information, but records can be numerous in big data
  - “influencers”
    - Answers: *What are the most unusual entities in a range of time?*
- See [https://github.com/elastic/examples/tree/master/Alerting/ml\\_examples](https://github.com/elastic/examples/tree/master/Alerting/ml_examples)

# Example – Querying for bucket results

- If we execute the query:

```
#farequote results bucket search
GET .ml-anomalies-*/_search
{
  "query": {
    "bool": {
      "filter": [
        { "range" : { "timestamp" : { "gte": "now-2y" } } },
        { "term" : { "result_type" : "bucket" } },
        { "term" : { "job_id" : "farequote_response" } },
        { "range" : { "anomaly_score" : {"gte" : "75"} }}]
    }
  }
}
```

# Example – An anomaly bucket

- We get the following:
  - anomaly\_score: anomaly score of a bucket at time = timestamp
  - at this point, can also query “records” or “influencers” at this time for more detail
    - querying for records or influencers right away could lead to a lot of results for jobs with many detectors or entities in by/partition/over fields

```
1 {  
2   "took": 48,  
3   "timed_out": false,  
4   "_shards": {  
5     "total": 25,  
6     "successful": 25,  
7     "failed": 0  
8   },  
9   "hits": {  
10    "total": 1,  
11    "max_score": 0,  
12    "hits": [  
13      {  
14        "_index": ".ml-anomalies-shared",  
15        "_type": "result",  
16        "_id": "farequote_response_1455034500000_300",  
17        "_score": 0,  
18        "_source": {  
19          "job_id": "farequote_response",  
20          "timestamp": 1455034500000,  
21          "anomaly_score": 90.7,  
22          "bucket_span": 300,  
23          "initial_anomaly_score": 85.08,  
24          "record_count": 2,  
25          "event_count": 179,  
26          "is_interim": false,  
27          "bucket_influencers": [  
28            {  
29              "job_id": "farequote_response",  
30              "result_type": "bucket_influencer",  
31              "influencer_field_name": "airline",  
32              "initial_anomaly_score": 85.08,  
33              "anomaly_score": 90.7,  
34              "raw_anomaly_score": 43.1559,  
35              "probability": 2.661e-44,  
36              "timestamp": 1455034500000,  
37              "bucket_span": 300,  
38              "sequence_num": 5,  
39              "is_interim": false  
40            },  
41            {  
42              "job_id": "farequote_response",  
43              "result_type": "bucket_influencer",  
44              "influencer_field_name": "bucket_time",  
45            }  
46          ]  
47        }  
48      }  
49    }  
50  }  
51 }  
52 }
```

# A Simple Watch

- Trigger at a rate equal to or faster than bucket\_span
- Define the input section as “search”
  - obviously make the “range” more “real-time” by using “now-1m”. It is longer in example because of date range of demo data.
- For the “condition” section, check to see if you get any payload hits
- For actions, can obviously do things more sophisticated than log (such as email, etc.)

```
# basic bucket watch
POST _xpack/watcher/watch/_execute
{
  "watch": {
    "trigger" : {
      "schedule" : { "interval" : "5m" }
    },
    "input" : {
      "search" : {
        "request" : {
          "indices" : [ ".ml-anomalies-*" ],
          "body" : {
            "query": {
              "bool": {
                "filter": [
                  { "range" : { "timestamp" : { "gte": "now-2y" } } },
                  { "term" : { "result_type" : "bucket" } },
                  { "term" : { "job_id" : "forequote_response" } },
                  { "range" : {"anomaly_score" : { "gte" : "75" }}}]
                ]
              }
            }
          }
        }
      }
    }
  },
  "condition" : {
    "compare" : { "ctx.payload.hits.total" : { "gt" : 0 } }
  },
  "actions" : {
    "log" : {
      "logging" : {
        "text" : "Anomalies:\n{{#ctx.payload.hits.hits}}score={{_source.anomaly_score}}\n{{/_source.timestamp}}\n{{/ctx.payload.hits.hits}}"
      }
    }
  }
}
```

# Watch Output

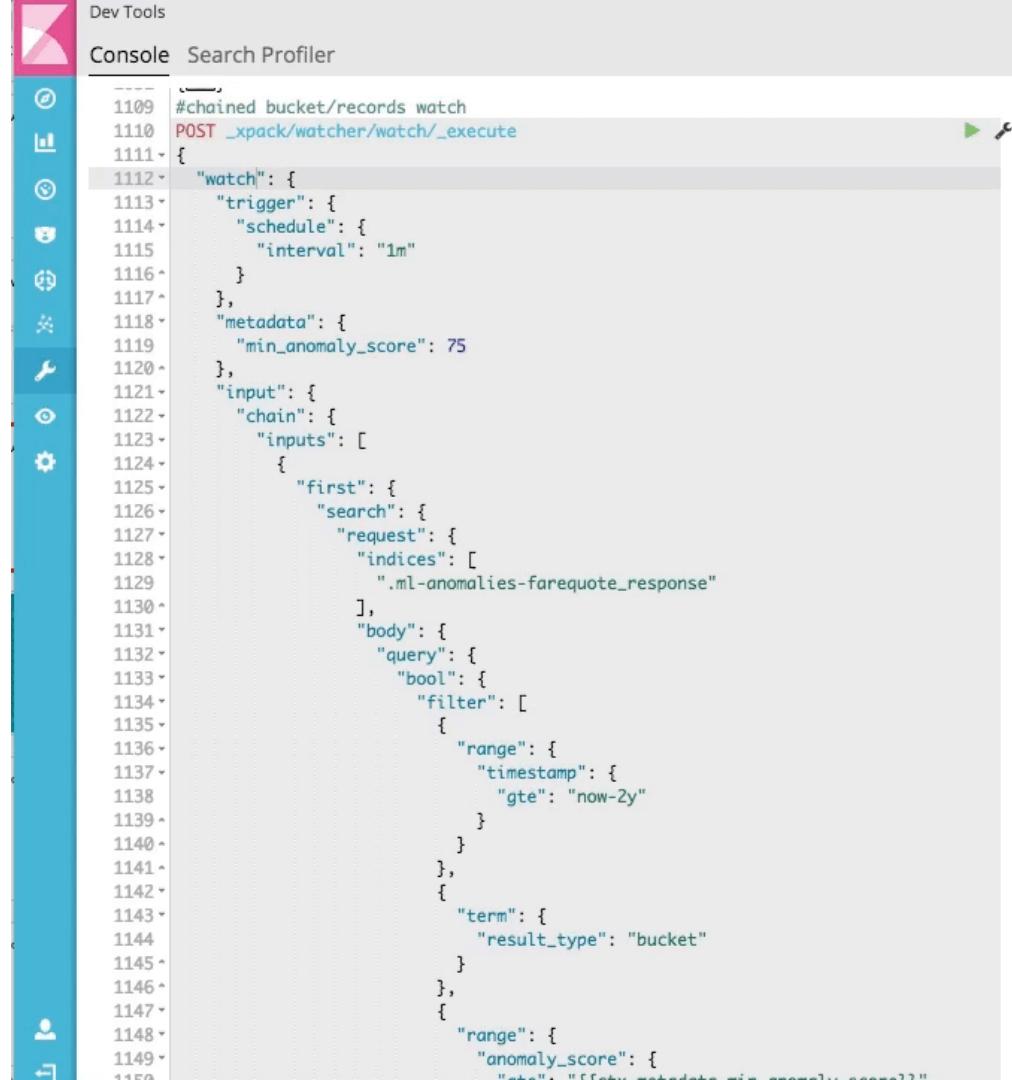
- The following is the resolved output of the watch, as seen in elasticsearch.log:

```
[2017-06-06T09:37:56,604][INFO ][o.e.x.w.a.l.ExecutableLoggingAction] [GJgDody] Anomalies:score=90.7 at time=1455034500000
```

# A Chained Watch

- Same idea, but use “chained” inputs to make 2 queries:
  - First to “bucket” level to test aggregated anomaly score
  - Second to “records” for that bucket time to return the records, if the bucket score was worthy
  - Output:

Anomaly of score=90.7 at 2016-02-09 11:15:00 Influenced by:  
airline=AAL: score=95.4659, responsetime=296.19ms (typical=99.2962ms)  
airline=AWE: score=0.00871093, responsetime=19.1644ms (typical=19.9918ms)



The screenshot shows the Kibana Dev Tools interface with the "Console" tab selected. The console window displays a JSON configuration for a "chained bucket/records watch". The configuration includes a "watch" object with a "trigger" (interval of 1m), "metadata" (min\_anomaly\_score of 75), and an "input" object. The "input" object uses a "chain" to execute two requests: a "search" request to find indices ".ml-anomalies-farequote\_response" and a "body" request to filter documents based on a timestamp range from now-2y to now. Both requests have a "bool" query with a single "filter" clause. The "term" filter has a "result\_type" of "bucket". Finally, there is a "range" filter on the "anomaly\_score" field with a minimum value of 75.

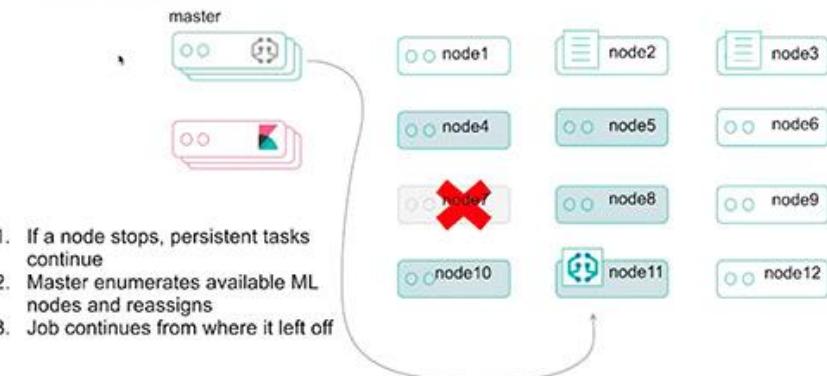
```
#chained bucket/records watch
POST _xpack/watcher/watch/_execute
{
  "watch": {
    "trigger": {
      "schedule": {
        "interval": "1m"
      }
    },
    "metadata": {
      "min_anomaly_score": 75
    },
    "input": {
      "chain": {
        "inputs": [
          {
            "first": {
              "search": {
                "request": {
                  "indices": [
                    ".ml-anomalies-farequote_response"
                  ],
                  "body": {
                    "query": {
                      "bool": {
                        "filter": [
                          {
                            "range": {
                              "timestamp": {
                                "gte": "now-2y"
                              }
                            }
                          }
                        ]
                      }
                    }
                  }
                }
              }
            }
          }
        ]
      }
    }
  }
}
```

# Architecture

# Architecture

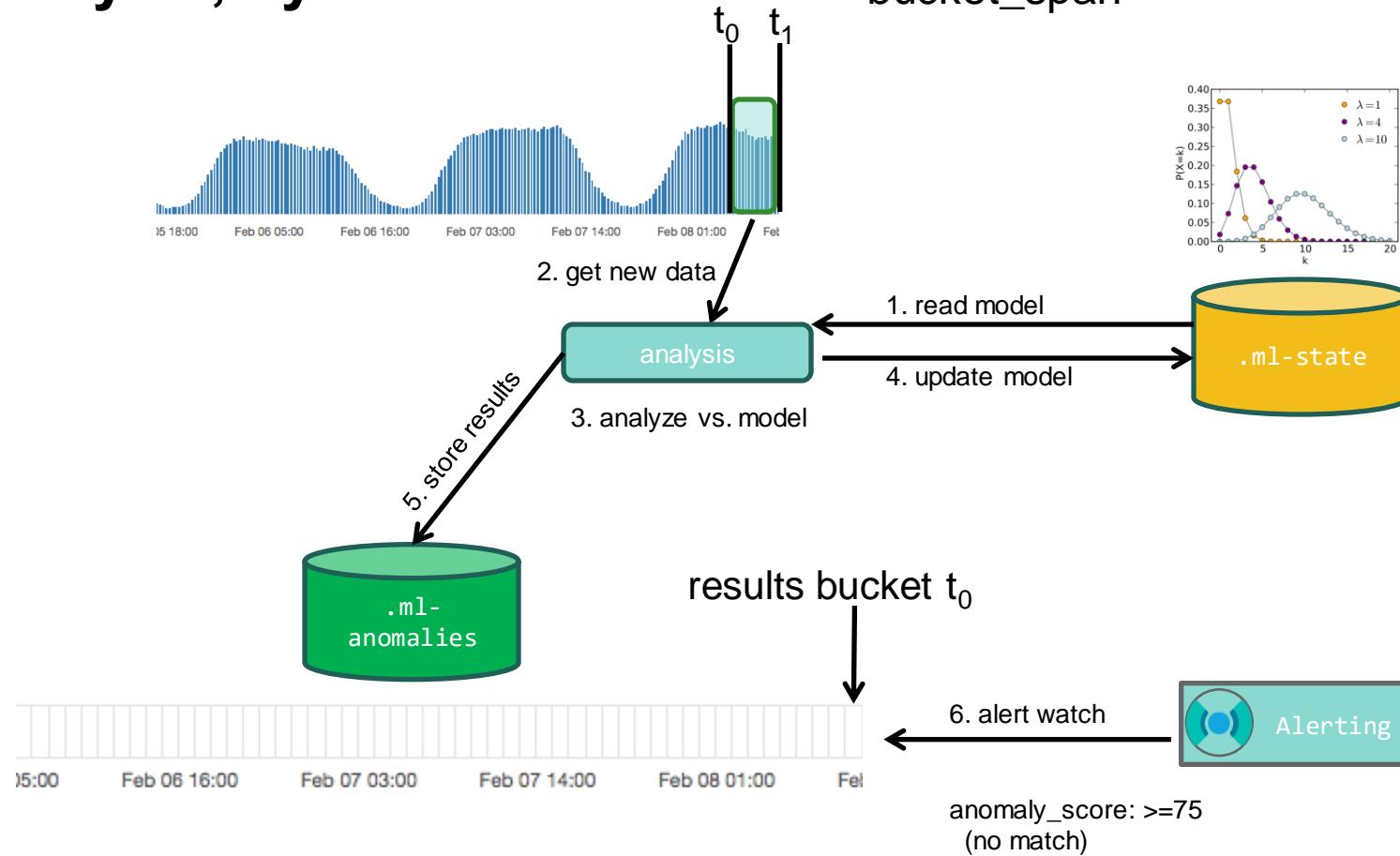
- ML jobs run on any ML-eligible node (node.ml:true)
- Runs ***outside*** of JVM (because it is C++)
- Best practice: Dedicated ML Nodes with ample CPU/RAM
- Jobs are tied to a specific node, but can failover to other ML-eligible nodes
- Job state (model) is stored in .ml-state index on data nodes of cluster
- Job results (anomalies) stored in .ml-anomalies index on data nodes of cluster

## Job resilience

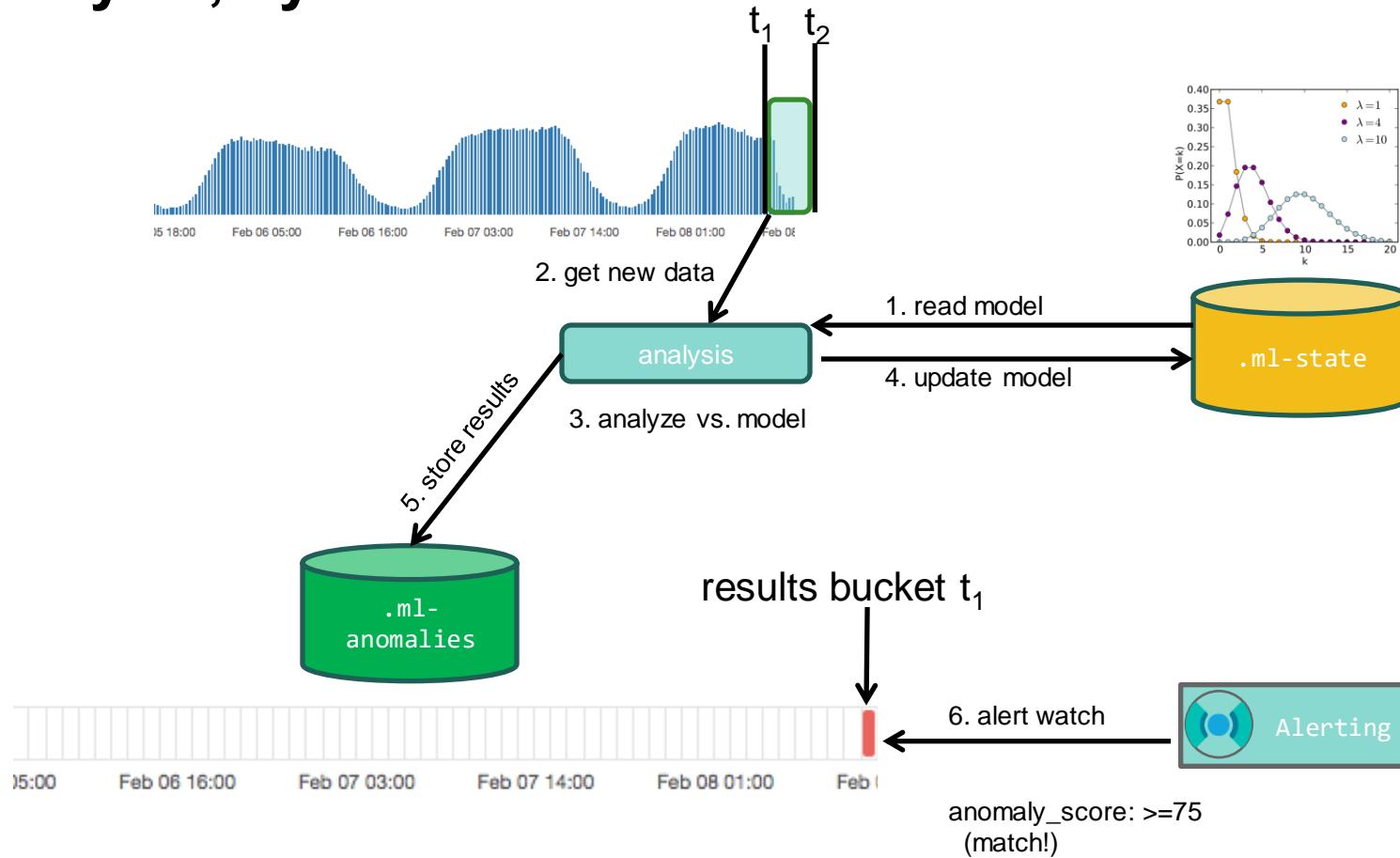


# Analysis, by “bucket”

$t_n - t_{n-1}$  is called the  
“bucket\_span”



# Analysis, by “bucket”



# API Control

# Controlling ML via API

- Full documentation of API available at

<https://www.elastic.co/guide/en/x-pack/current/ml-api-quickref.html>

## Docs

### API Quick Reference

All machine learning endpoints have the following base:

`/_xpack/ml/`

The main machine learning resources can be accessed with a variety of endpoints:

- [/anomaly\\_detectors/](#): Create and manage machine learning jobs
- [/datafeeds/](#): Select data from Elasticsearch to be analyzed
- [/results/](#): Access the results of a machine learning job
- [/model\\_snapshots/](#): Manage model snapshots
- [/validate/](#): Validate subsections of job configurations

### /anomaly\_detectors/

- [POST /anomaly\\_detectors](#): Create a job
- [POST /anomaly\\_detectors/<job\\_id>/\\_open](#): Open a job
- [POST /anomaly\\_detectors/<job\\_id>/\\_data](#): Send data to a job
- [GET /anomaly\\_detectors](#): List jobs
- [GET /anomaly\\_detectors/<job\\_id>](#): Get job details
- [GET /anomaly\\_detectors/<job\\_id>/\\_stats](#): Get job statistics
- [POST /anomaly\\_detectors/<job\\_id>/\\_update](#): Update certain properties of the job configuration
- [POST /anomaly\\_detectors/<job\\_id>/\\_flush](#): Force a job to analyze buffered data
- [POST /anomaly\\_detectors/<job\\_id>/\\_close](#): Close a job
- [DELETE /anomaly\\_detectors/<job\\_id>](#): Delete a job

### /datafeeds/

- [PUT /datafeeds/<datafeed\\_id>](#): Create a datafeed
- [POST /datafeeds/<datafeed\\_id>/\\_start](#): Start a datafeed
- [GET /datafeeds](#): List datafeeds
- [GET /datafeeds/<datafeed\\_id>](#): Get datafeed details
- [GET /datafeeds/<datafeed\\_id>/\\_stats](#): Get statistical information for datafeeds
- [GET /datafeeds/<datafeed\\_id>/\\_preview](#): Get a preview of a datafeed

## On this page

[/anomaly\\_detectors/](#)

[/datafeeds/](#)

[/results/](#)

[/model\\_snapshots/](#)

[/validate/](#)

\* X-Pack Reference: 5.4 (current) 

[Introduction](#)

[Installing X-Pack](#)

[Migrating to X-Pack](#)

[Securing Elasticsearch and Kibana](#)

[Monitoring the Elastic Stack](#)

[Alerting on Cluster and Index Events](#)

[Reporting from Kibana](#)

[Graphing Connections in Your Data](#)

[Profiling your Queries and Aggregations](#)

[Machine Learning in the Elastic Stack](#)

[Overview](#)

[Getting Started](#)

[Configuring Machine Learning](#)

[API Quick Reference](#)

\* X-Pack Settings

\* X-Pack APIs

\* Troubleshooting

\* Limitations

\* License Management

\* Release Notes

# Example API Control

- All major operations are available via API
  - Create/Delete jobs and datafeeds
  - Job control (start/stop)
- Plus actions that are ONLY available via API
  - Model snapshot/restore

```
printf "\n\n== Creating job... \n"
curl -u elastic:changeme -s -X PUT -H 'Content-Type: application/json' ${JOBS}/${JOB_ID}?pretty -d '{
  "description" : "Unusual responsetimes by airlines",
  "analysis_config" : {
    "bucket_span": "5m",
    "detectors" : [{"function":"max", "field_name":"responsetime", "by_field_name":"airline"}],
    "influencers" : [ "airline" ]
  },
  "data_description" : {
    "time_field": "@timestamp"
  }
}'

printf "\n\n== Creating datafeed... \n"
curl -u elastic:changeme -s -X PUT -H 'Content-Type: application/json' ${DATAFEEDS}/datafeed-${JOB_ID}?pretty -d '{
  "job_id" : "${JOB_ID}",
  "indexes" : [
    "farequote"
  ],
  "types" : [
    "responsetime"
  ],
  "scroll_size" : 1000
}'

printf "\n\n== Opening job for ${JOB_ID}... "
curl -u elastic:changeme -X POST ${JOBS}/${JOB_ID}/_open

printf "\n\n== Starting datafeed-${JOB_ID}... "
curl -u elastic:changeme -X POST "${DATAFEEDS}/datafeed-${JOB_ID}/_start?start=1970-01-02T10:00:00Z&end=2017-01-01T00:00:00Z"

printf "\n\n== Finished ==\n\n"
```



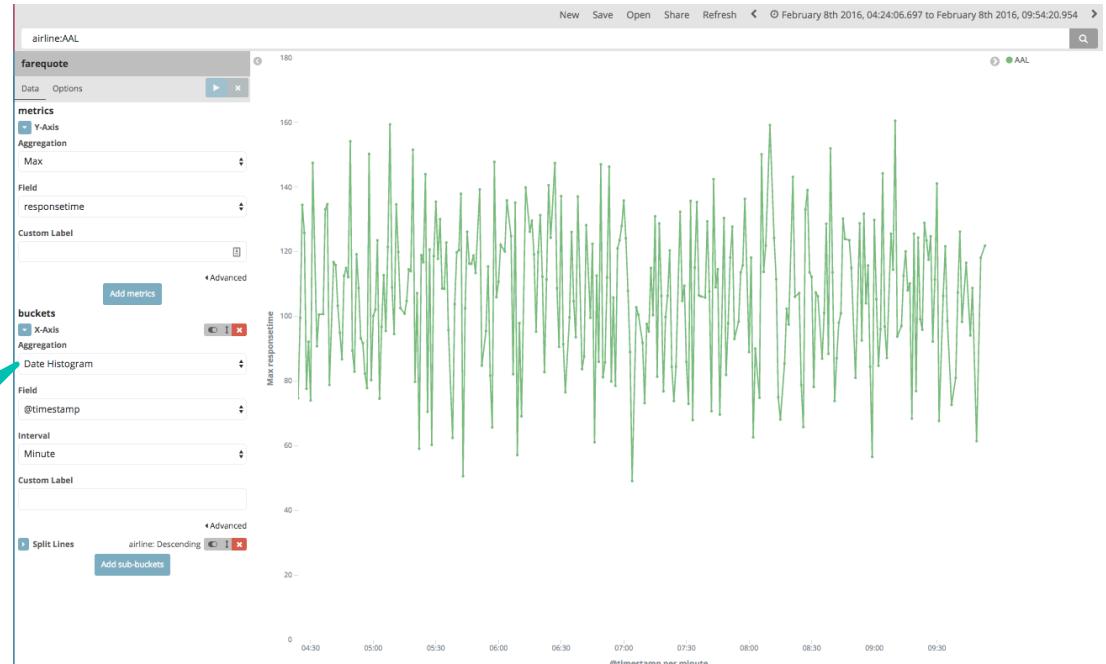
# The End

# Extra Slides

# bucket\_span

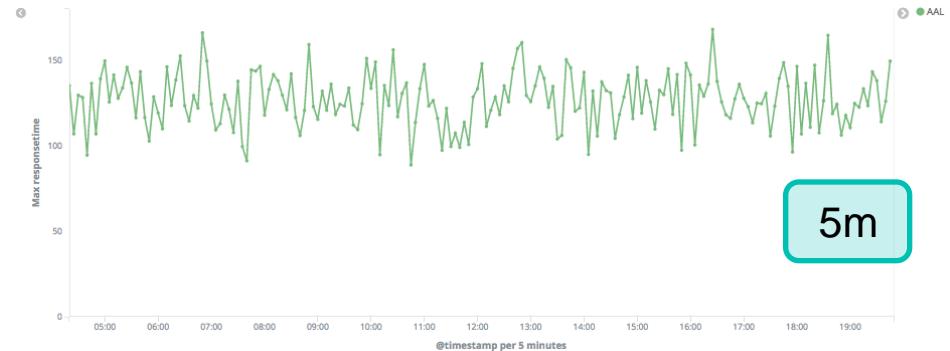
- Similar to a “Date Histogram” aggregation for an x-axis visualization in Kibana
- Defines the resolution (in time) that the data should be aggregated to

*Date Histogram*



# What will bucket\_span control?

- Granularity of data being analyzed
- Ultimately defines the volume of data analyzed per bucket
- How often the data is queried, when running in “real-time”



# **bucket\_span: Rules of Thumb**

- If you have “sampled” data (i.e. monitoring data every X minutes) then you could make bucket\_span=X
- If you have arbitrary-rate data (i.e. log events), then pick a bucket\_span that balances
  - timeliness (time to analyze/alert)
  - enough time to count, sum, etc. those events up to make the number of samples per bucket meaningful, given the velocity of data
- In other words, pick a bucket\_span that's approximately equivalent to the duration of an anomaly that you would care about.

# By vs. Partition

- Both methods split data to establish separate baselines.
- Can be applied together in one detector.
- **By**
  - Creates attributes in existing model
  - Easier to scale
  - Best used to find anomalies in system as a whole
  - Scoring considers history of other by-fields
  - Required if using “rare” function
- **Partition**
  - Establishes different models
  - Requires more memory to store models
  - Best used to find anomalies in individual behavior
  - Scoring is more independent

# Complex Detector Example

- high\_count by ErrorCode over Host partition\_field=App
  - Partition: App
    - Split the data into separate data sets for each App. Create a model (and population) for each.
  - Over: Host
    - For each App, create a population model out of all the Hosts and detect when a Host stands out.
  - By: ErrorCode
    - When modeling and evaluating Hosts, baseline each ErrorCode separately.
  - Function: High\_Count
    - Measure the number of times an ErrorCode is observed in the time bucket.