Forward-Looking Statements

This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's current expectations, estimates and assumptions based on the information currently available to us. These forward-looking statements are not guarantees of future performance and involve significant risks, uncertainties and other factors that may cause our actual results, performance or achievements to be materially different from results, performance or achievements expressed or implied by the forward-looking statements contained in this presentation.

For additional information about factors that could cause actual results to differ materially from those described in the forward-looking statements made in this presentation, please refer to our periodic reports and other filings with the SEC, including the risk factors identified in our most recent quarterly reports on Form 10-Q and annual reports on Form 10-K, copies of which may be obtained by visiting the Splunk Investor Relations website at www.investors.splunk.com or the SEC's website at www.sec.gov. The forward-looking statements made in this presentation are made as of the time and date of this presentation. If reviewed after the initial presentation, even if made available by us, on our website or otherwise, it may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statement based on new information, future events or otherwise, except as required by applicable law.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. We undertake no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.

Anomaly Detection, Sealed with a KISS

PLA1553B

Matthew Khan
Data Engineer | IG Group

Rupert Truman
Solutions Engineer | Splunk

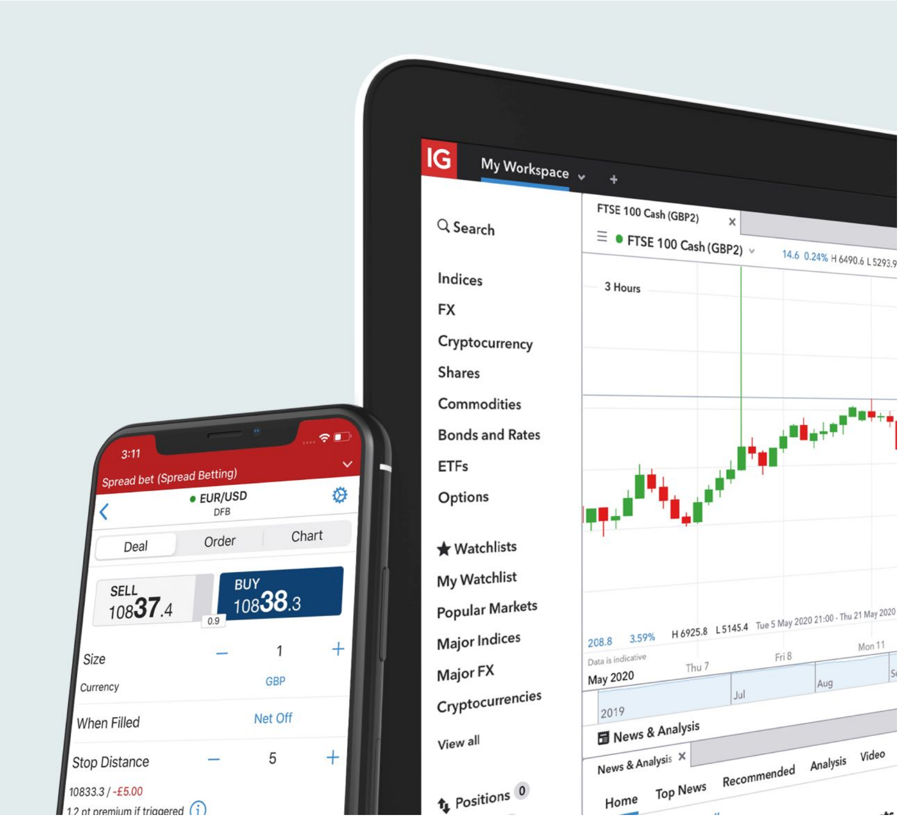
splunk> .conf21

Matthew Khan
Data Engineer | IG Group

Rupert Truman
Solutions Engineer | Splunk

IG

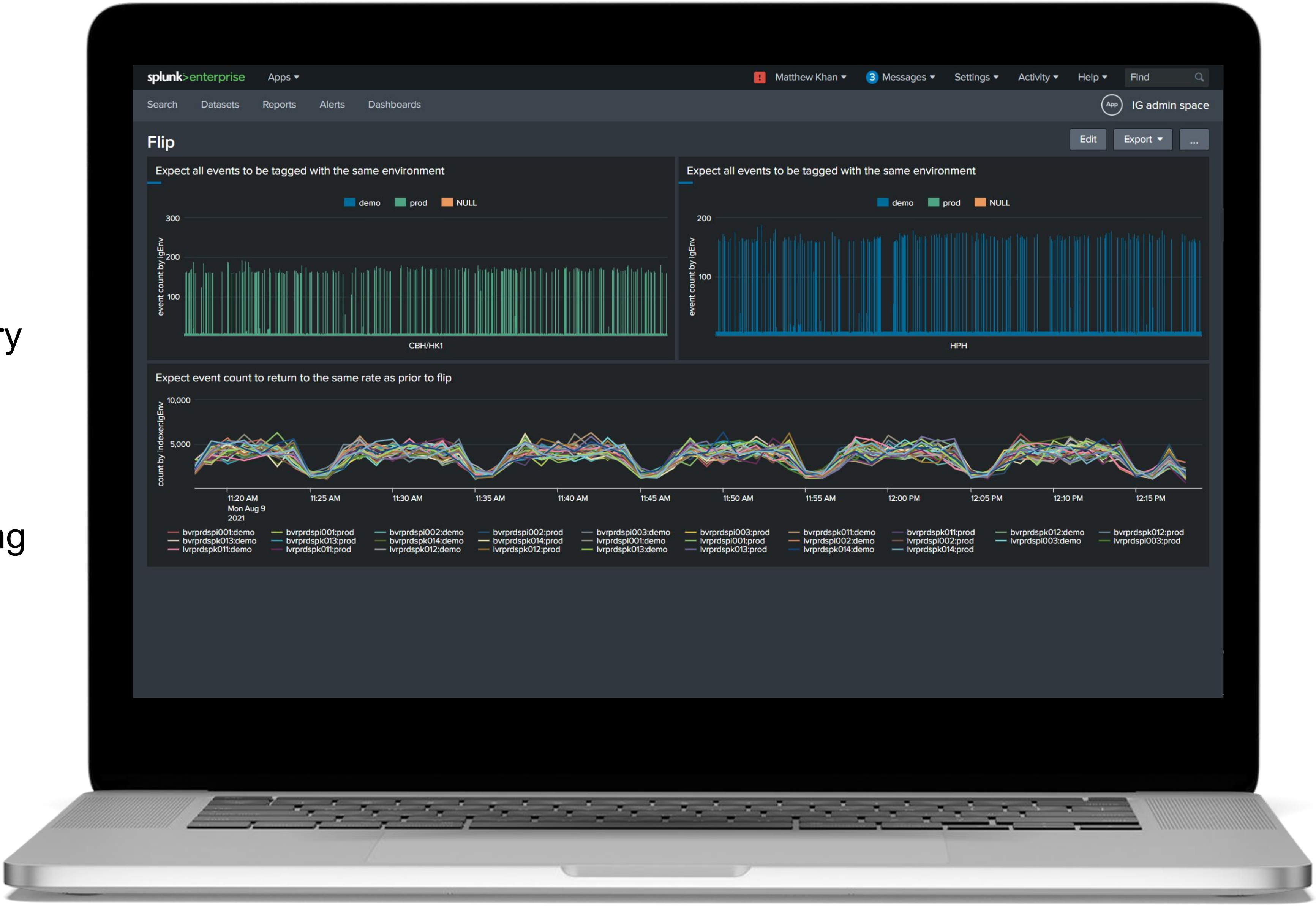
- Founded in 1974 as IG (Investors Gold) Index for retail spread betting on gold prices
- World leader in online trading*
- Access to 17,000+ markets
- FTSE 250 company, publicly tradable on the London Stock Exchange
- 230,000 active clients



* For CFDs and spread betting, based on revenue excluding FX (published financial statements, June 2020). Best overall personal wealth provider as awarded at the Online Personal Wealth Awards, 2020. Authorised by ASIC, JFSA, MAS, FINMA, FCA, & CFTC.

Splunk at IG

Grown to 10TB since 2009



- Monitoring, alerting, regulatory archiving of application, network device, OS and security event logs
- Business Intelligence reporting
- Transaction tracing
- Incident analysis
- Change tracking
- Maintenance window trigger

The Challenge of Service Outages

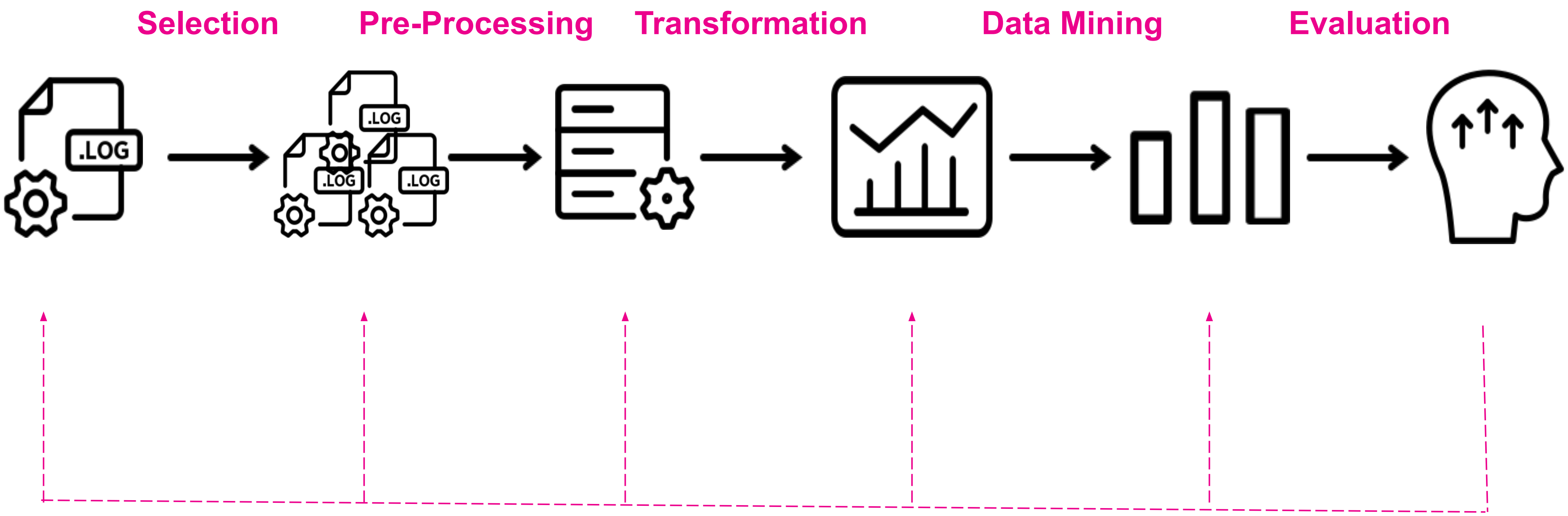
- Service outage is damaging - both financially and reputationally
- Anomalous app behaviour may herald service degradation or outage but impossible to manually detect over 1500+ applications
- Static/global thresholds are too simplistic;
  - **too high** - anomalous behaviour potentially undetected
  - **too low** - noise, alert fatigue

Proposed Solution:

Use machine learning to implement adaptive anomaly detection across the IG platform

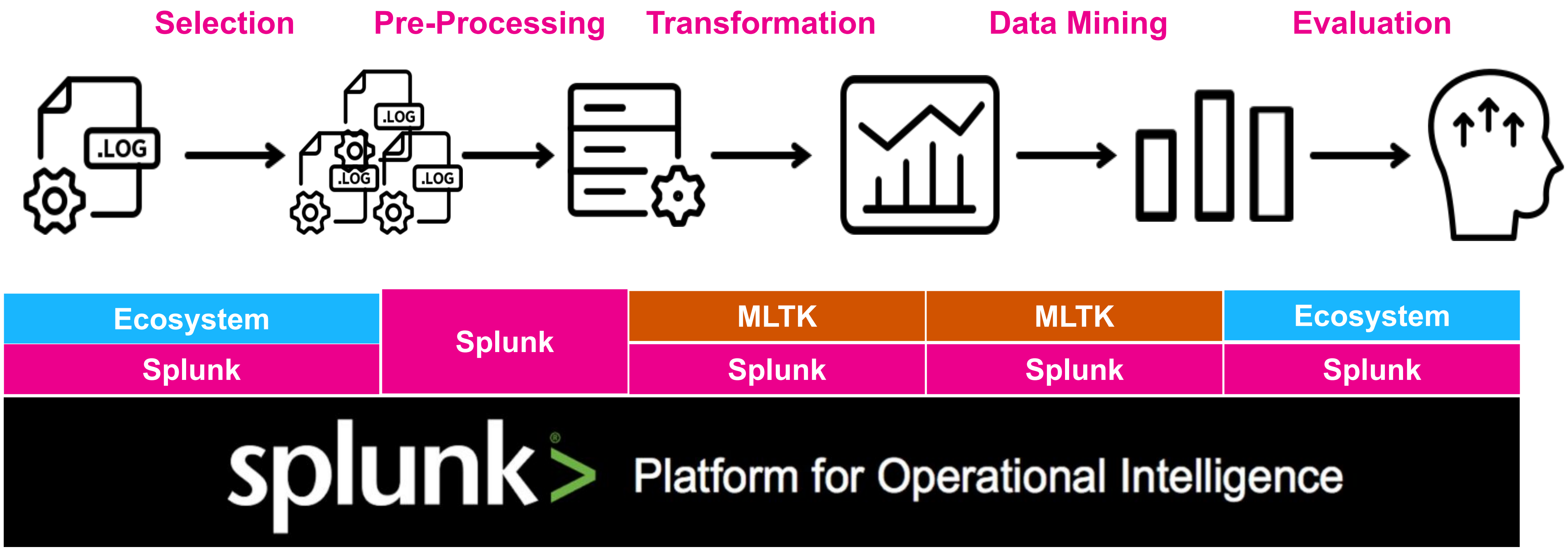
Data Science Methodology

Knowledge Discovery in Databases (Fayaad, 1996)



Data Science Methodology

Splunk as an end-to-end data pipeline



Which Data Covers Service Performance?

Apache Tomcat access log:

Processing Time

```
***.***.***.*** [26/Jul/2021:14:01:21.999 +0100] 0.001 20 {***.***.***.***} GET
/login-service/api/session 403 [iId=****************] [gId=****************]
```

App URL

HTTP Status

Can compute RED Metrics:

- **Rate** - total count of app requests
- **Error rate -** total count of app requests with an error status
- **Duration** - average processing time of request

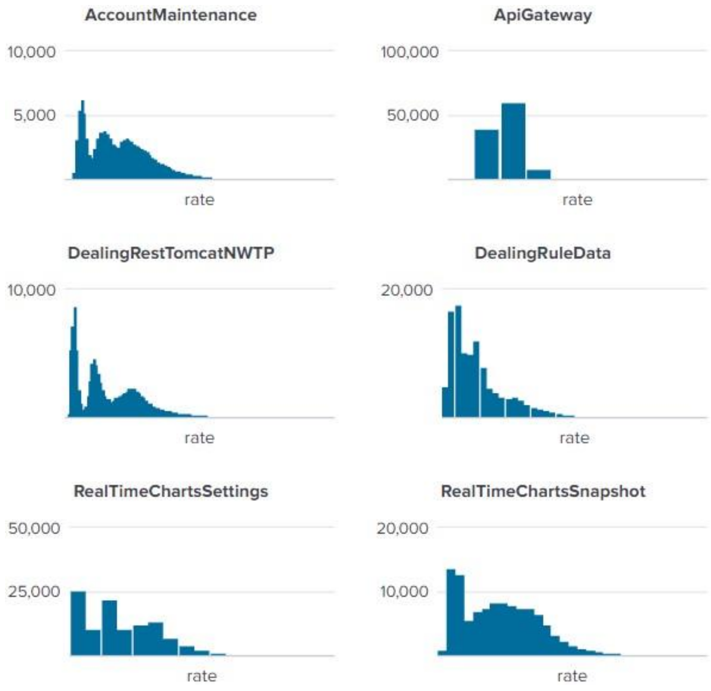
Extraction:

- Can extract at ingestion for performance at scale with **tstats**
- usage of **lookups** and **loadjob** for rapid data analysis

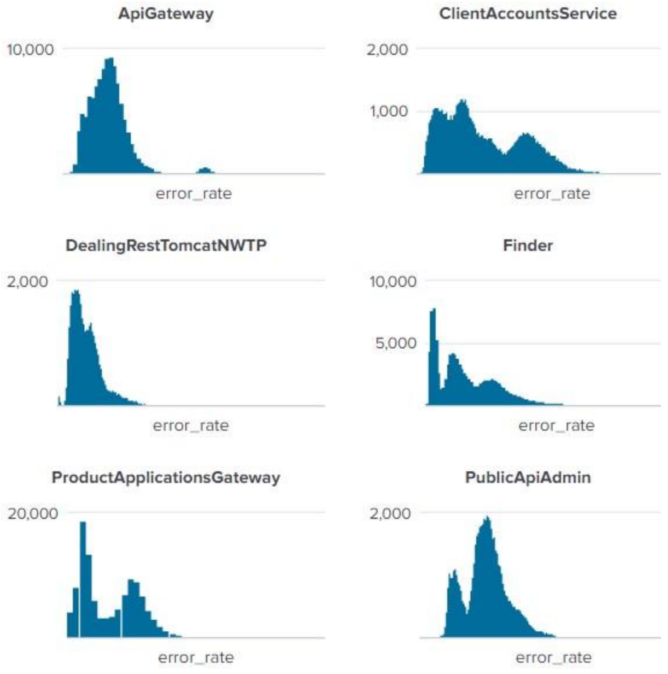
Data Analysis: Rate, Error & Duration

Scaling differences, but similarities in distribution...

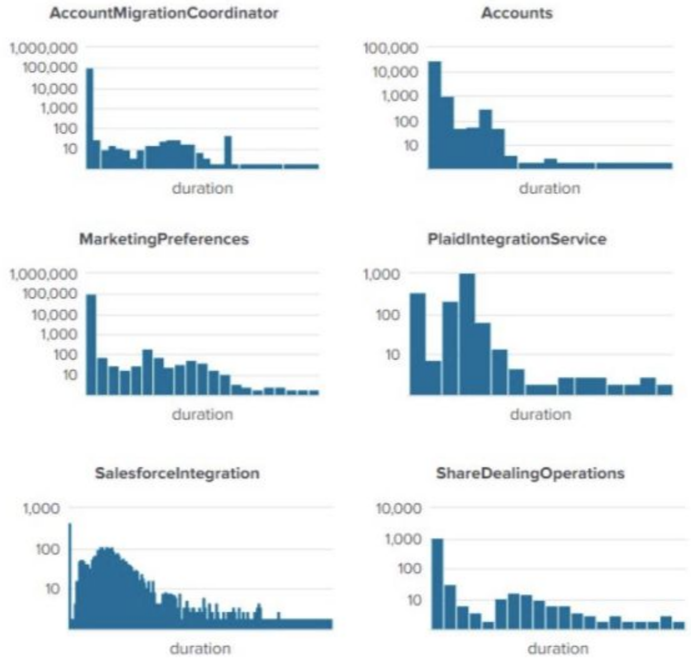
Rate



Error Rate



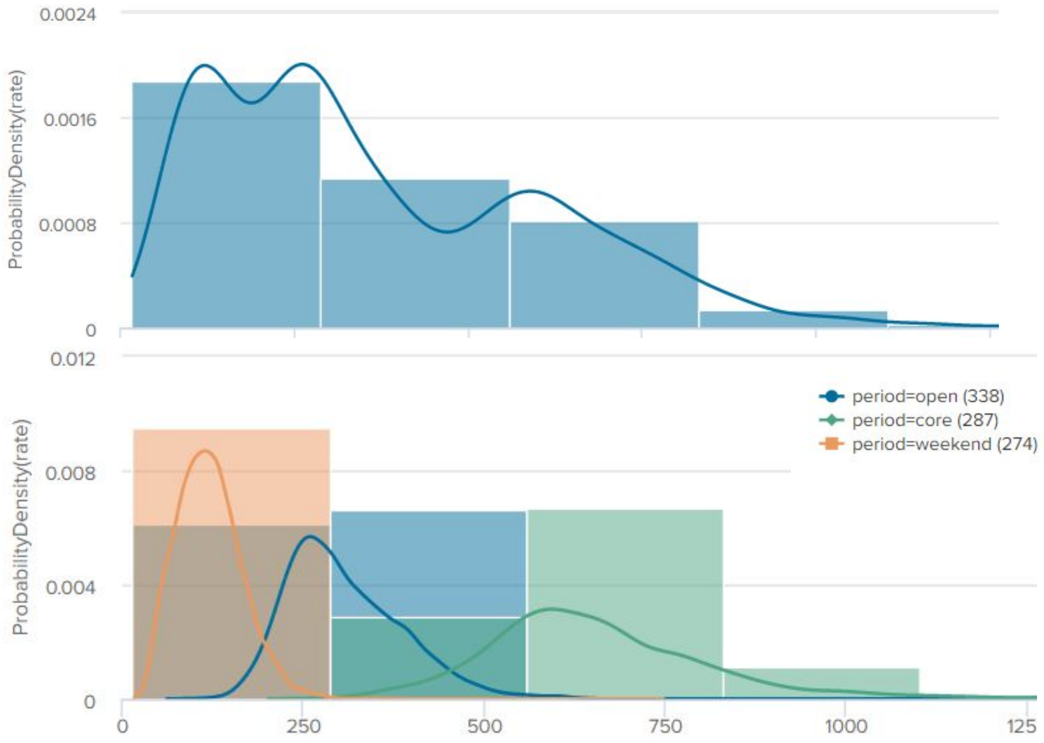
Duration



Data Analysis: Density Distribution

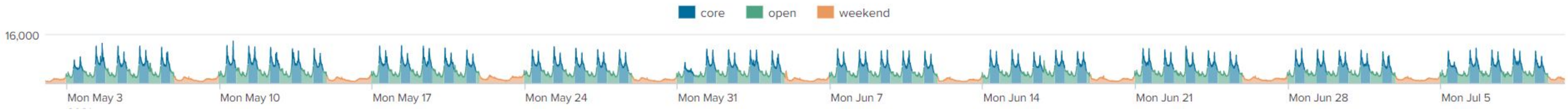
Investigating with **DensityFunction** (Don't worry we'll cover that shortly)

- Using the example of the LoginService Rate metric we can see three humped distribution
- Breaking the requests down into three time periods (market open, core banking hours and weekend) produces distributions which are closer to normal

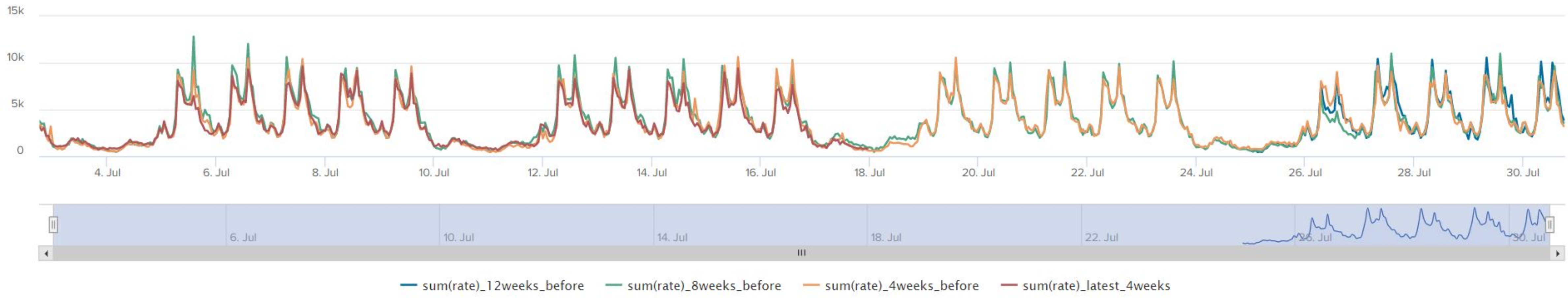


Data Analysis: How Does a Service Change?

1) Using the example of Rate for the Login Service we can see seasonality



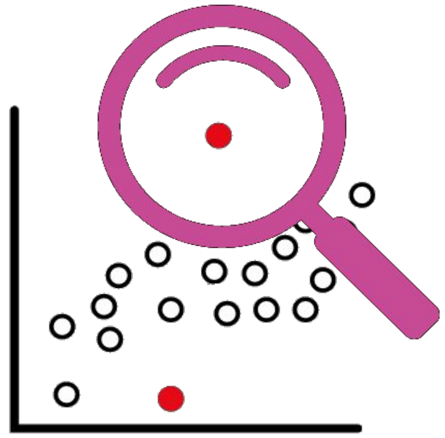
2) App RED metrics broadly align to previous week on week values



Anomaly Detection

If there's something strange in your neighborhood, how can you tell?

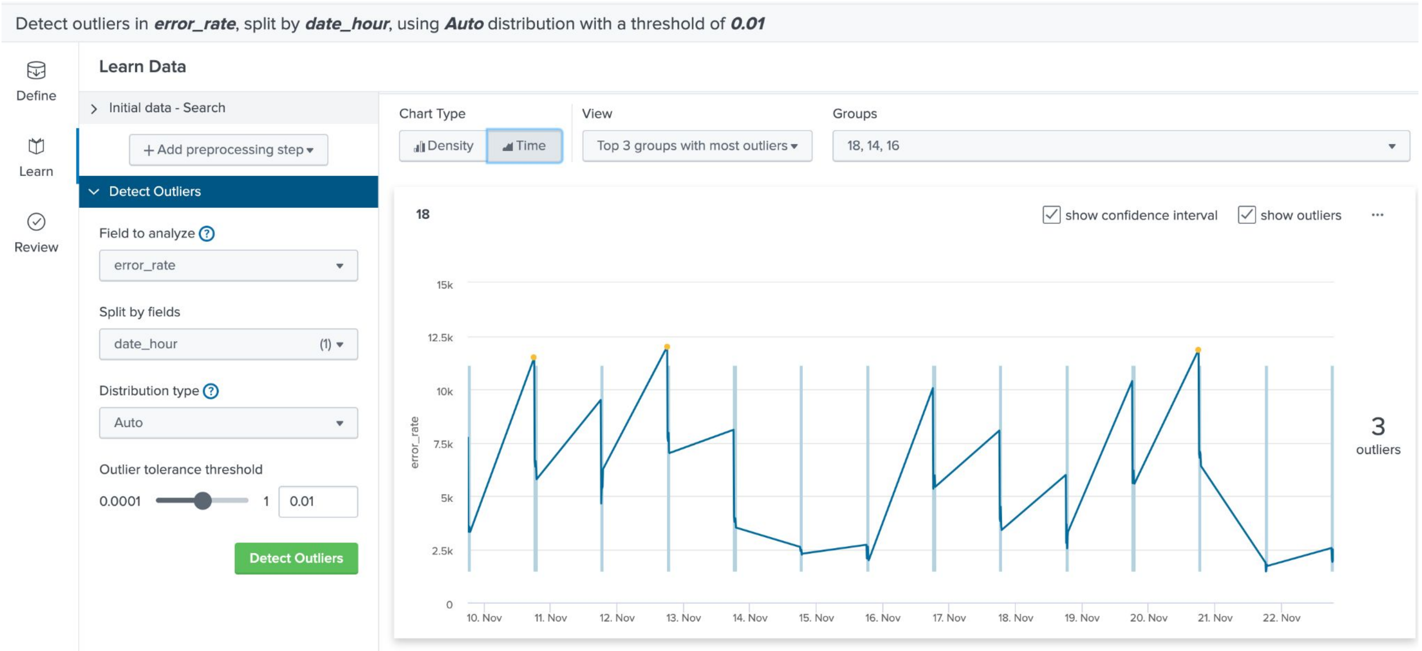
- Deviation from expected behaviour, be it based on a change from historic activity or discrepancy with current behavior of peers
- Anomalies often consist of observed outliers - unusual values
- Anomaly detection is valuable everywhere:
  – **IT Ops -** Unusually high CPU utilization %
  – **Security -** Inconsistency of login patterns
  – **Fraud -** Unexpected size or frequency of transactions
  – **IoT -** Discrepancy in temperatures detected by factory sensors



Finding our Gain Threshold

This sounds like a job for the MLTK!

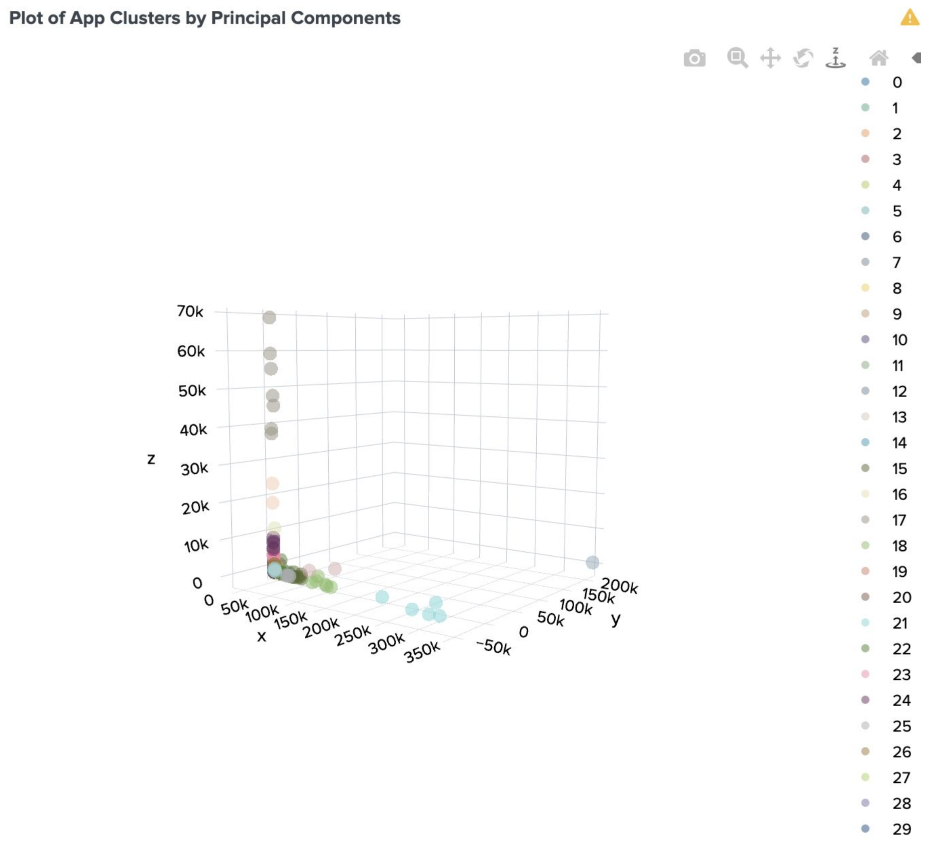
- The **DensityFunction** workflow produces a model of anomalies through the density distribution of the values supplied to it
- However, this approach would require a distinct model for **every application**
- Impractical to train and manage, but what if we could model **groups of apps**...



Cluster(ing) Headaches

Sometimes the answer isn't to add another algorithm...and another….

- Identifying which apps behave similarly by grouping their data points with algorithms like **GMeans**
- Depending on the the algorithm this still produces a number of groups…
- So what if alerted on when an app was clustered into a group it shouldn't be?
- Can use a classification algorithm like **RandomForest** to classify cluster placement, but...



It Doesn’t Reduce Alert Noise...

Confusion Matrix of Cluster Outputs (Columns 1-11 of 38)

#	Predicted actual	Predicted 0	Predicted 1	Predicted 10	Predicted 11	Predicted 12	Predicted 13	Predicted 14	Predicted 15	Predicted 16	Predicted 17
1	0	2493	20	0	0	0	220	0	0	0	0
2	1	2386	796	0	0	0	103	0	0	0	0
3	10	64	418	122	0	0	6	0	0	0	1
4	11	45	2	0	887	147	2	144	50	209	0
5	12	4	0	1	4	742	74	229	0	29	0
6	13	818	47	0	0	0	625	0	0	0	0
7	14	0	0	1	5	492	12	515	0	242	0
8	15	0	0	0	244	97	0	145	616	115	0
9	16	3	0	2	17	333	4	584	0	567	0
10	17	12	2	19	0	96	1	150	0	60	243

Back to the Drawing Board

What do we know about the IG platform?

- Individual app metrics follow hourly and daily patterns
- Anomalies are the deviations from these patterns
- Separate machine learning models won’t scale
- Grouping apps for modelling produces too much noise

So what can we do?

Keep **I**t **S**imple **S**tupid

Defining Normal With | stats

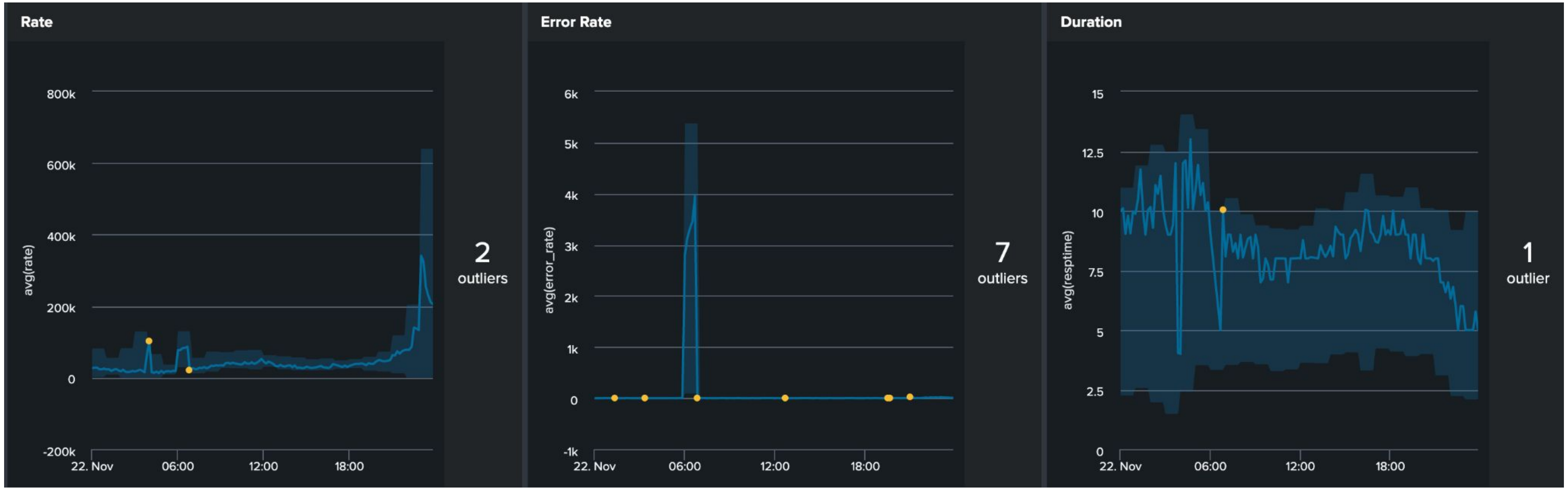
I'm a stats man

- Simple statistical approach
- RED values broken into hour and weekday/end buckets
- **avg** and **stdev** used to calculate adjustable upper and lower bounds
- Output saved as lookup
- Operationalised through daily scheduling of search

```
index=ig
| bin _time span=1h
| eval HourOfDay=strftime(_time, "%H")
| eval DayOfWeek=strftime(_time, "%A")
| eval weekday=if(in(DayOfWeek,"Saturday","Sunday"),"No","Yes")
| stats avg(rate) as avg_r stdev(rate) as stdev_r avg(error_rate) as
avg_e stdev(error_rate) as stdev_e avg(resptime) as avg_d
stdev(resptime) as stdev_d by HourOfDay,weekday,app
| eval r_lowerBound=(avg_r-stdev_r*exact(2.25)),
r_upperBound=(avg_r+stdev_r*exact(2.25))
| eval e_lowerBound=(avg_e-stdev_e*exact(2.25)),
e_upperBound=(avg_e+stdev_e*exact(2.25))
| eval d_lowerBound=(avg_d-stdev_d*exact(2.25)),
d_upperBound=(avg_d+stdev_d*exact(2.25))
| fields app, HourOfDay,weekday,
r_lowerBound,r_upperBound,e_lowerBound,e_upperBound,d_lowerBound,d_upp
erBound
| outputlookup app_metric_bounds.csv
```

…Allows us to Build Adaptive Thresholds

Scalable anomaly detection on an app by app basis



And Finally, a Working Solution!

Key Takeaways

or how to not get sucked into a science project

1) Define a clear problem statement

2) Know your data

3) Be iterative

4) **Keep it simple stupid!**

References

- The RED method for microservice monitoring: https://www.weave.works/blog/the-red-method-key-metrics-for-microservices-architecture/
- The essential “Cyclical Statistical Forecasts and Anomalies” series by Manish Sainani and Greg Ainslie-Malik: https://www.splunk.com/en_us/blog/platform/cyclical-statistical-forecasts-and-anomalies-part-1.html
- TSTATS and Prefix by Richard Morgan: https://conf.splunk.com/files/2020/slides/PLA1089C.pdf
- INGEST_EVAL and CLONE_SOURCETYPE by Richard Morgan and Vladimir Skoryk: https://conf.splunk.com/files/2020/slides/PLA1154C.pdf

Thank You

Please provide feedback via the

SESSION SURVEY