APPLIED ROOTS

Data Quality & Verification @ Uber, Netflix & Amazon

APPLIED ROOTS

Active Learning

↳ Participation & Discussion

Let's solve it together...

APPLIED ROOTS

Sources:

1. https://eng.uber.com/monitoring-data-quality-at-scale/ (2020) ← Uber DQM
2. https://databricks.com/session_na20/an-approach-to-data-quality-for-netflix-personalization-systems (2020)
3. https://www.amazon.science/publications/automating-large-scale-data-quality-verification (2018)

APPLIED ROOTS

Scale @ all companies

~ 1 - 10PB

10K - 100K tables

100s of Millions of events

1000's of metrics

APPLIED ROOTS

(Q) what platform(s) & techniques would you use to process internet-scale data?

APPLIED ROOTS

Data Quality impacts all downstream tasks

↓

Data Analysis

Biz decision-making

ML & DL models

Infra-costs

APPLIED ROOTS

Let's map Data Quality to an ML-problem?

Any Suggestions?

APPLIED ROOTS

Data Quality

- Time-series of metrics (detect anamolies)
  - KPI
  - utilization
  - Table
    - # rows
    - column-level
      - missing values
      - min; max; unique-values

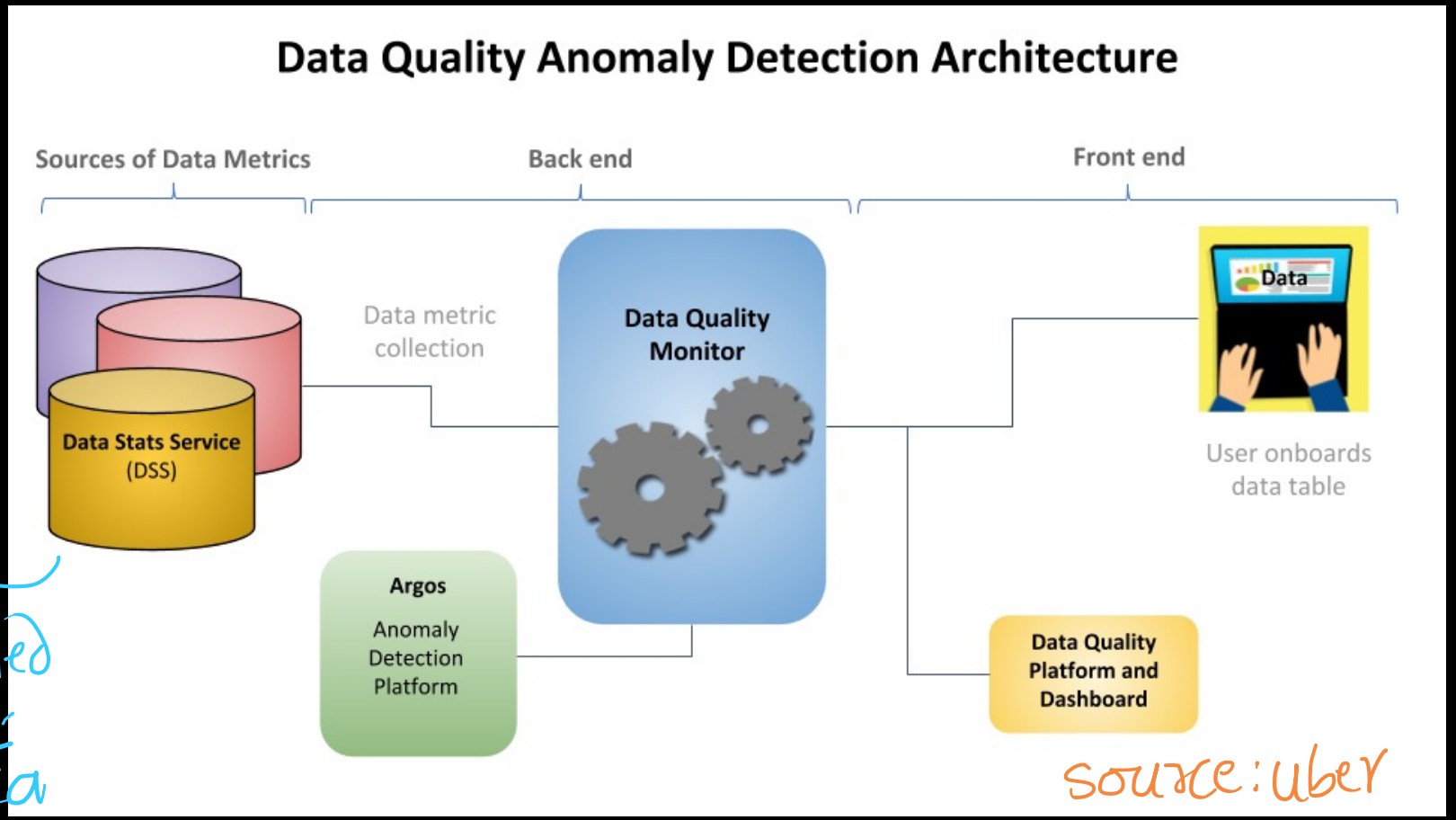
APPLIED ROOTS

— Data Anamolies $\mathcal{D} = \{x_i, y_i\}_{i=1}^{n}$

(Q) what techniques can we employ here ?

APPLIED ROOTS

Uber DQM

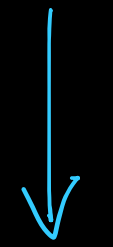


date partitioned

HIVE & Vertica

APPLIED ROOTS

DSS: multi-dimensional time series output



(DQM) → PCA & project to lower-dim

→ Time-series anamoly detection (using forecasting)

APPLIED ROOTS

(Q) why do you PCA is helpful here?

APPLIED ROOTS

many metrics are highly correlated

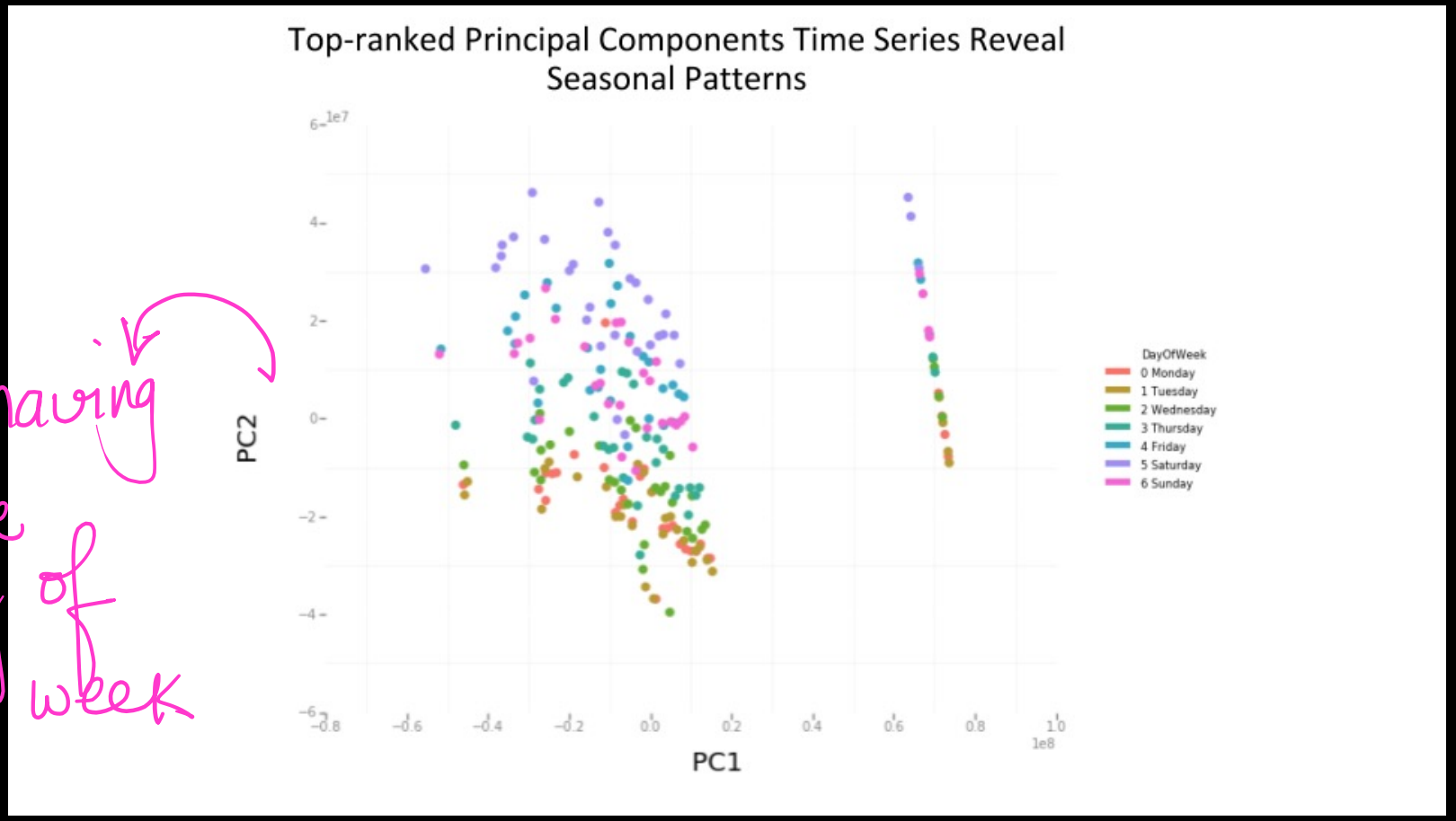
↓

eg: trip duration & trip distance

Uber: 90% of variability explained by the top-PC.

APPLIED ROOTS

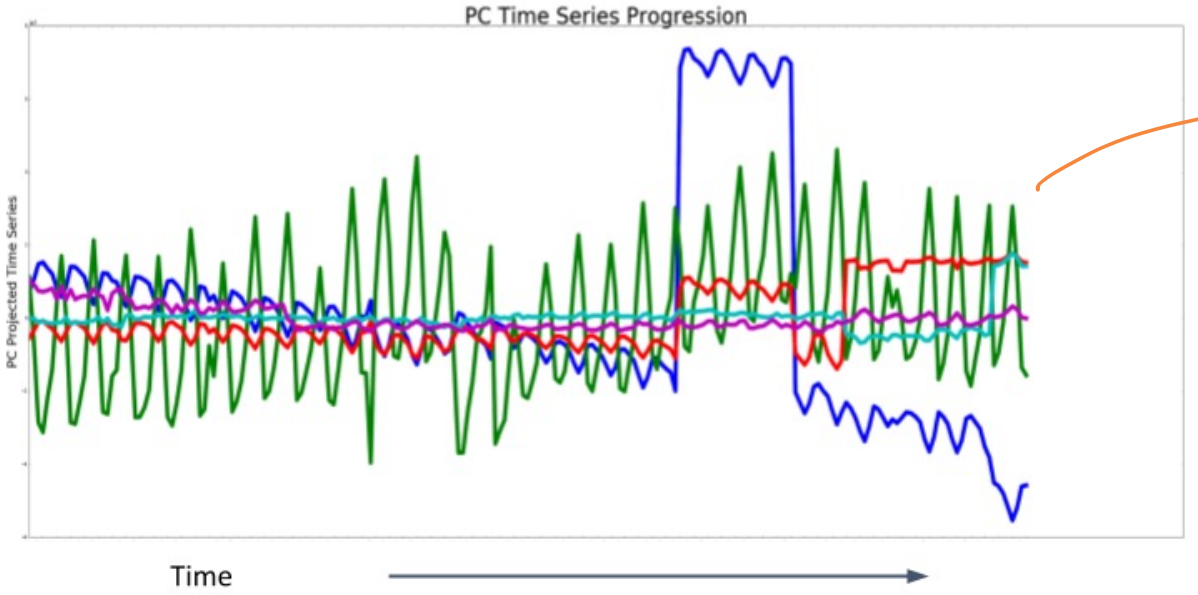
source: uber



behaving like day of week

APPLIED ROOTS

Principal Component Time Series Shows State Changes and Seasonalities



Top 5-PCs across time

APPLIED ROOTS

(Q) why not perform time-series anamoly detection on raw metrics ?

APPLIED ROOTS

Analyzing top-5 PCs

vs

Analyzing 100's of metrics

APPLIED ROOTS

Time-series anamoly detection

- via forecasting
- Holt-winters model
  - exponential moving average
  - Seasonality
  - Trends

https://otexts.com/fpp2/holt-winters.html

APPLIED ROOTS

(Q) why exponential smoothing ?

APPLIED ROOTS

recent data is given more weightage

↓

fast growing startup with sudden changes

APPLIED ROOTS

Seasonality → weekends; evening-hours; holidays ...

Trend → Typically to capture broader growth over time.

APPLIED ROOTS

Lesson: Use the 'simplest' model that works.

→ easy to interpret

↳ do NOT need DL everywhere

APPLIED ROOTS

Table-level scores

↳ Time-series of data in a table

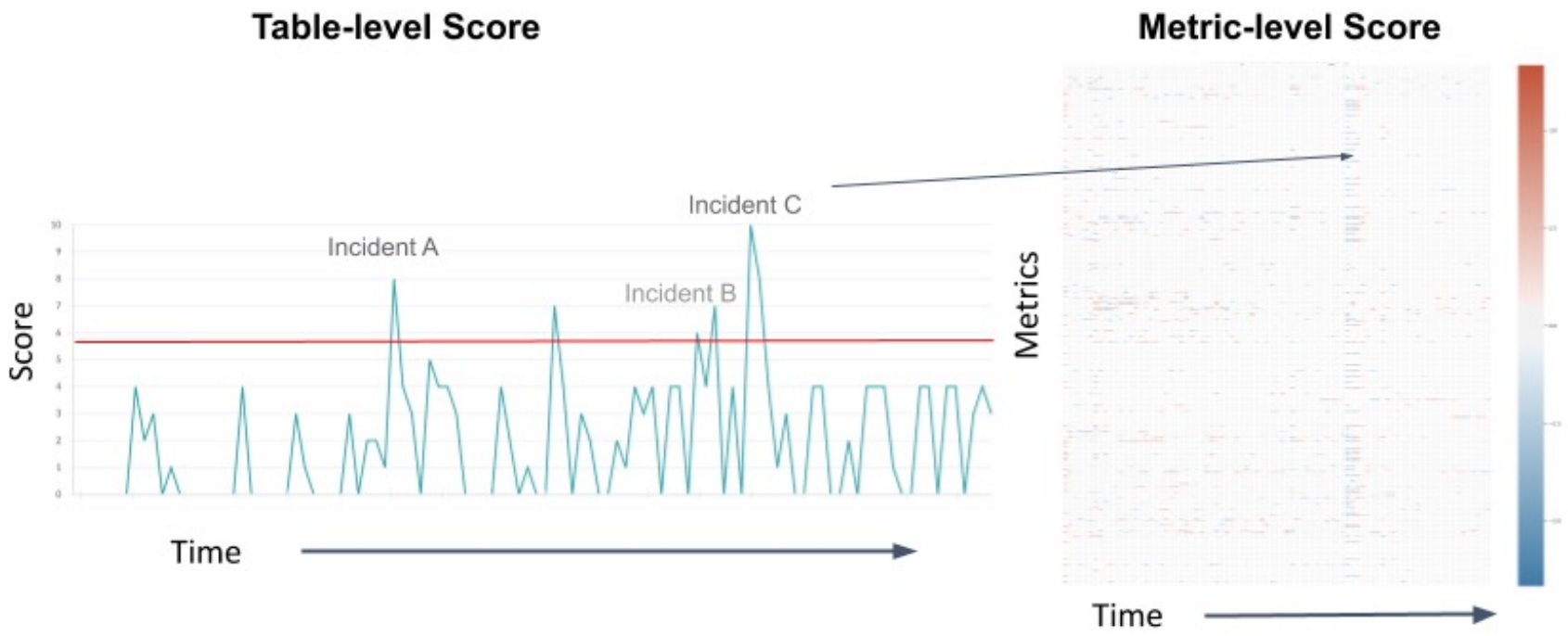
→ using top PC's with equal weightage

→ anomaly detection

APPLIED ROOTS

source: Uber

Anomaly Capture with Table and Metric-Level Scores



APPLIED ROOTS

DQM: Pyspark & Hive

↳ (what better platform than this?)

Vertica (columnar NoSQL database)

Figure 5. One of our next steps towards making alerts more intelligent is leveraging data table lineage information. In fact, we have observed strong correlation between data table quality and lineage as the clustering of table-level quality scores over time can reconstruct table ancestry. This is validated in practice as we see related tables have common root causes when they degrade in data quality.

Source : Uber

APPLIED ROOTS

Source: Netflix.com

Source ↓

https://databricks.com/session_na20/an-approach-to-data-quality-for-netflix-personalization-systems

(slides & video)

APPLIED ROOTS

Source: Netflix.com

lots of attributes

Netflix PI Historical Fact Store

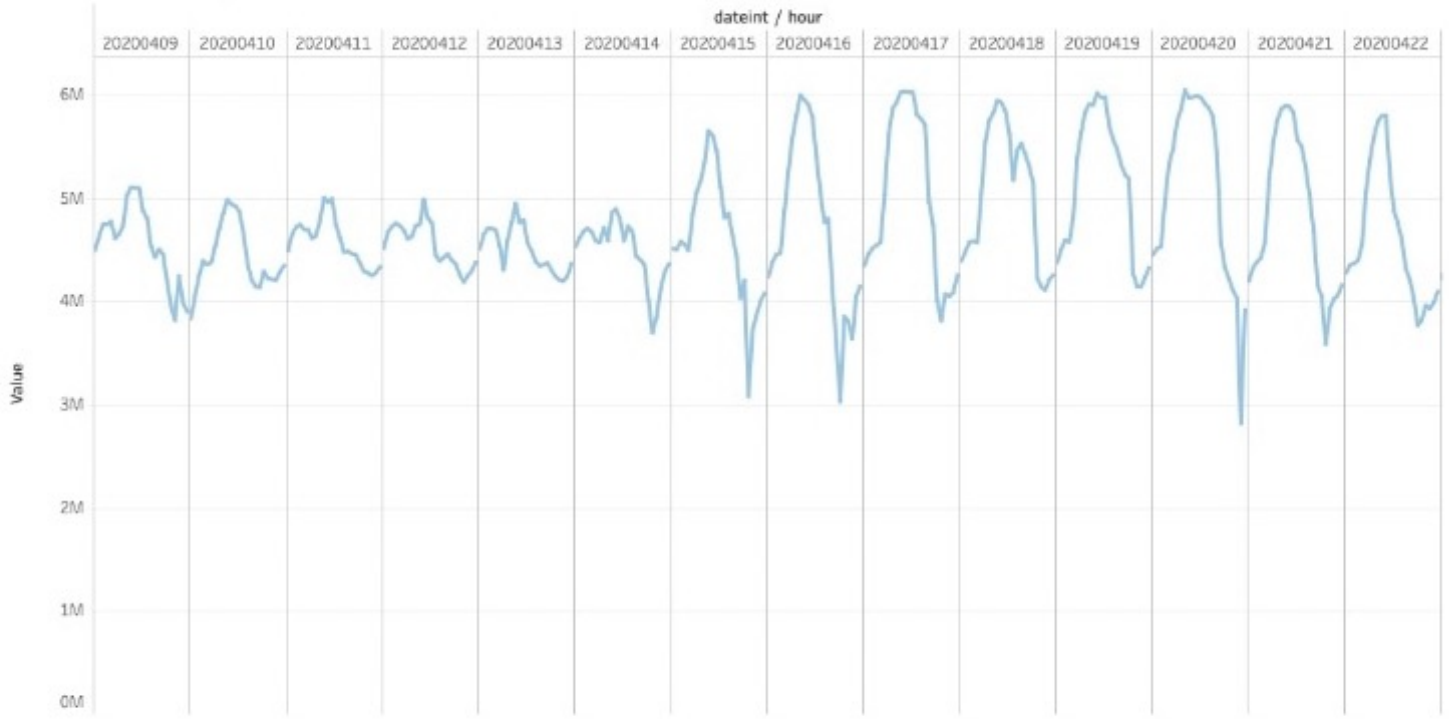
- Manages more than 10PB of data
- Manages hundreds of attributes
- More than 1 Billion rows flow through the ETL pipelines per day
- This data is used by several machine learned models and algorithms that enable Personalization

HIVE, Spark

APPLIED ROOTS

e.g: profile mix-up (kids vs adults) bug.

Bad Data Example 1: Drift



APPLIED ROOTS

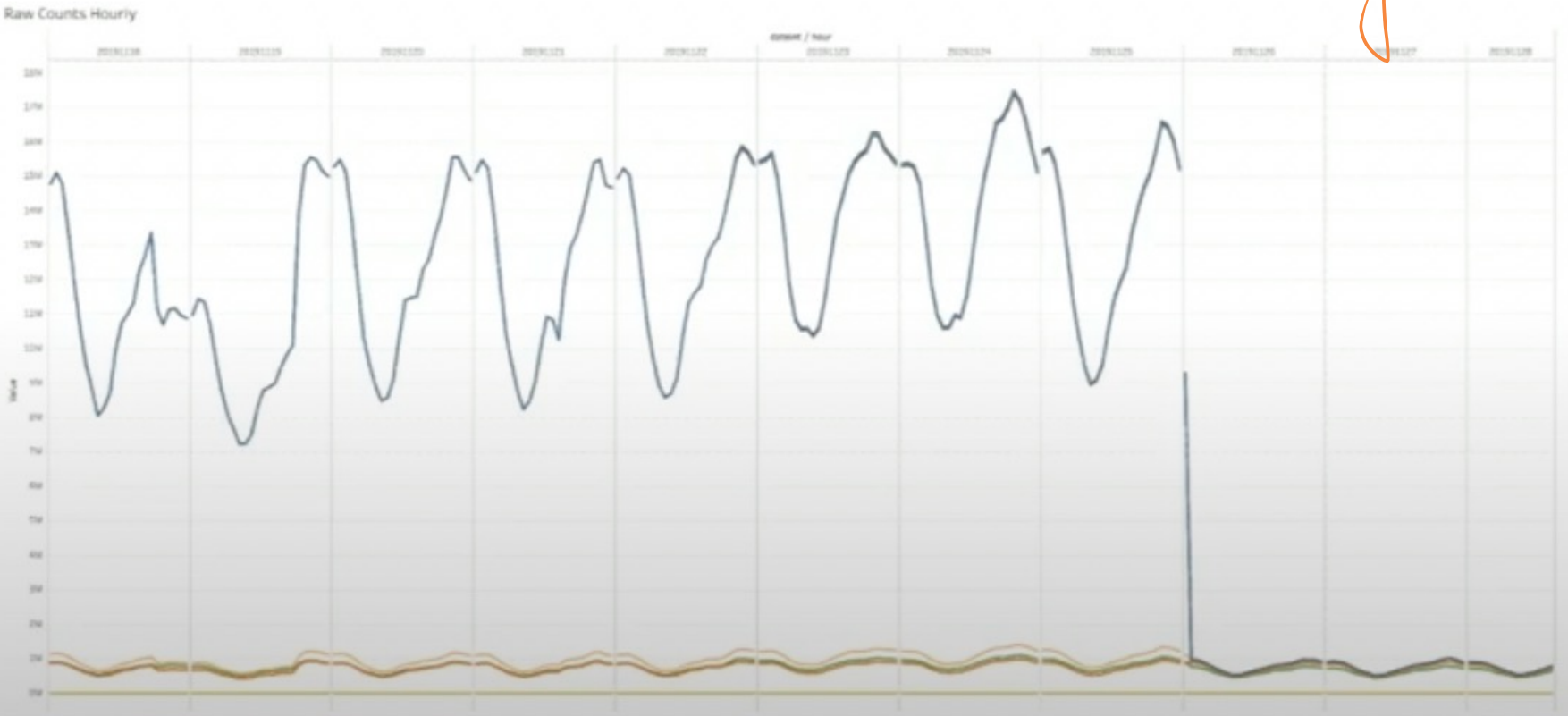
Source: Netflix.com

Bad Data Example 2: Drastic changes

APPLIED ROOTS

Very important but often ignored

Bad Data Example 3: Under Utilization



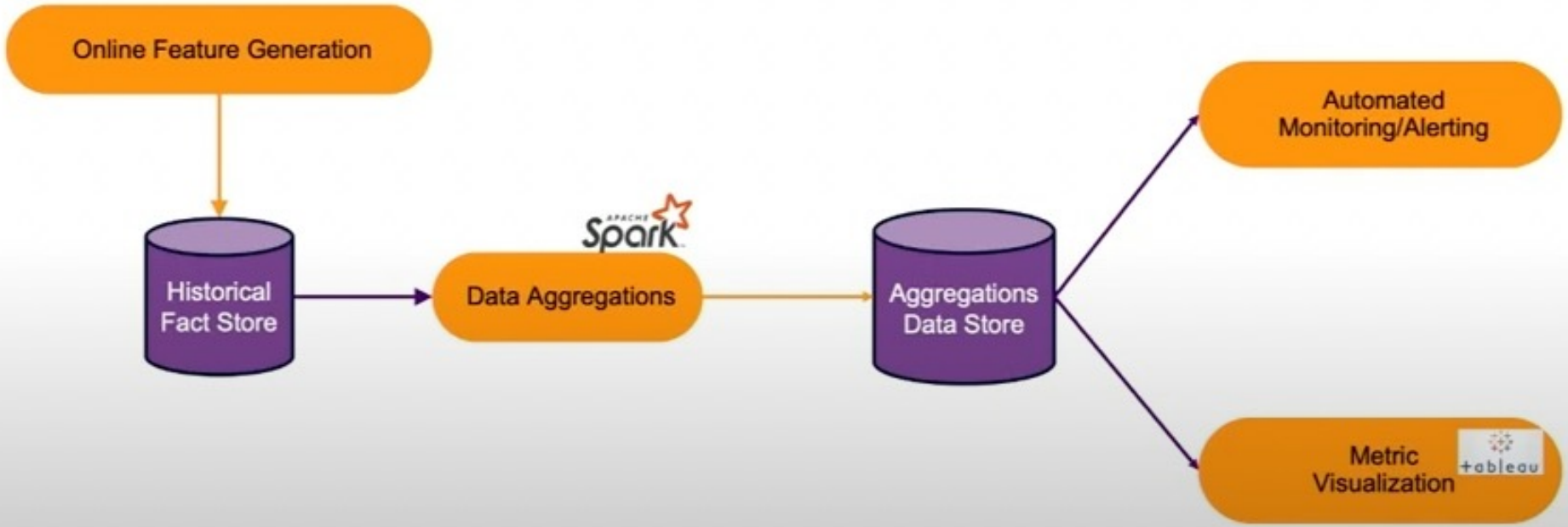
COST – SAVINGS

Source: Netflix.com

APPLIED ROOTS

Source: Netflix.com

Data Quality Architecture



APPLIED ROOTS

Source: Netflix.com

Example Data

ID	Member ID	Video ID	Timestamp (seconds)	Play Duration (seconds)	$\log_{10}$(Play Duration)
1	10001	101	1577000001	360	2.556
2	10002	102	1577000002	780	2.892
3	10002	101	1577000003	NULL	2.556
4	10003	102	1577000004	300	2.477
5	10004	103	1577000005	1000000000	9.0
6	10005	105	1577000006	100	2.0
7	10006	104	1577000007	NULL	2.556

Imputed

APPLIED ROOTS

Source: Netflix.com

Aggregations

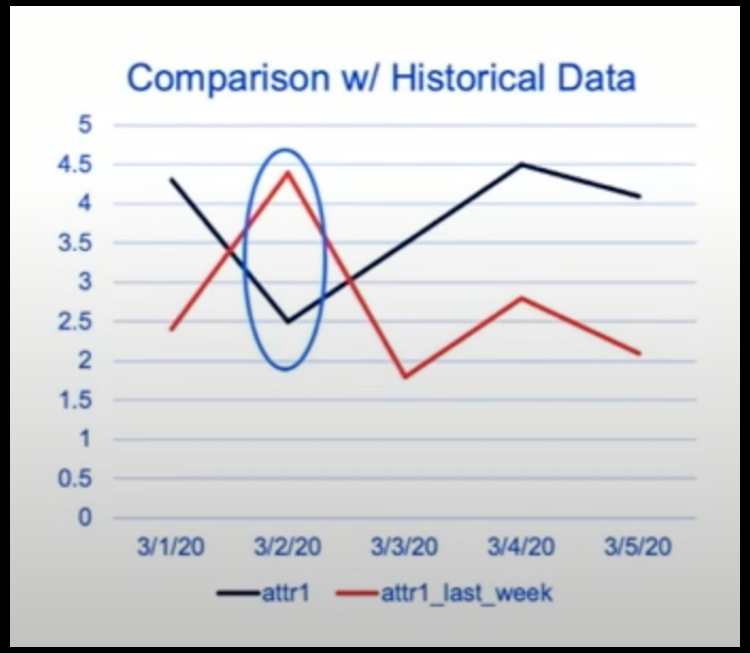
Date (DD/MM/YYYY)	Number Of Null Plays	Median Play Duration (seconds)	99th Percentile Play Duration (seconds)
1/1/2020	3	360	450
1/2/2020	20	270	780
1/3/2020	1	150	570

APPLIED ROOTS

Source: Netflix.com

Automated Monitoring:

↳ distribution - matching of attributes



APPLIED ROOTS

(Q) Any suggestions ?

APPLIED ROOTS

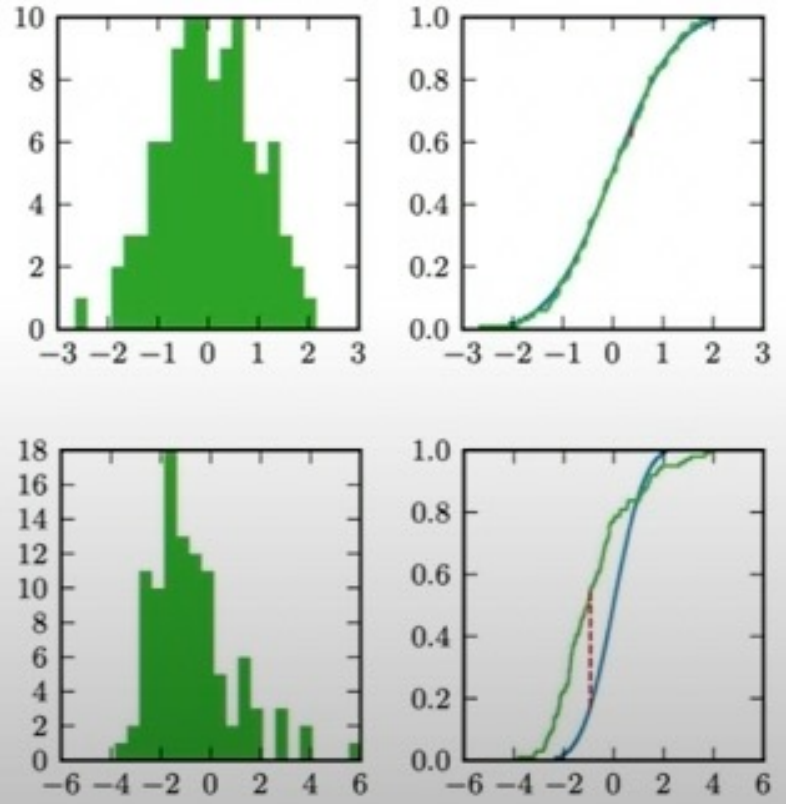
Source: Netflix.com

Distribution Checks – Statistical Test

Kolmogorov-Smirnov statistical test[1]

$$F_n(a) = \frac{1}{n} \sum_i \mathbb{I}(x_i < a)$$

$$D_n = \max_x |F_n^1(x) - F_n^2(x)|$$



[1]Source: MIT 6.S085

APPLIED ROOTS

Source: Netflix.com

Audits

```
from pyspark.sql import SQLContext
import AuditRunner, Audit

df = SQLContext.sql('SELECT * FROM ml.aggregations WHERE dateint = 20200303')

# Audit Pseudo code
auditRunner = AuditRunner().add(
    Audit (df,  'num_null_plays',      { 'upper_bound': 10 },              { 'alert': 'pager' } ),
    Audit (df,  'num_null_plays',      { 'normal_distribution': True },    { 'alert': 'dev_notification' } ),
    Audit (df,  'avg_play_duration',   { 'query_anomaly_detection': 'rad' }, { 'alert': 'dev_notification' } )
    # ...
)
```

Thresholds

distribution matching

Robust Anamoly Detection

APPLIED ROOTS

Netflix RAD:

https://netflixtechblog.com/rad-outlier-detection-on-big-data-d6b0494371cc (demo)

↳ Robust PCA (repeatedly calculate SVD and apply thresholds)

https://arxiv.org/pdf/0912.3599.pdf (original Research Paper 2009)

APPLIED ROOTS

(Q) If you were to design Robust-PCA using regular PCA, how would you do it ?

$$\min_{u} \; u^T S u$$

$$s.t \quad u^T u = 1$$

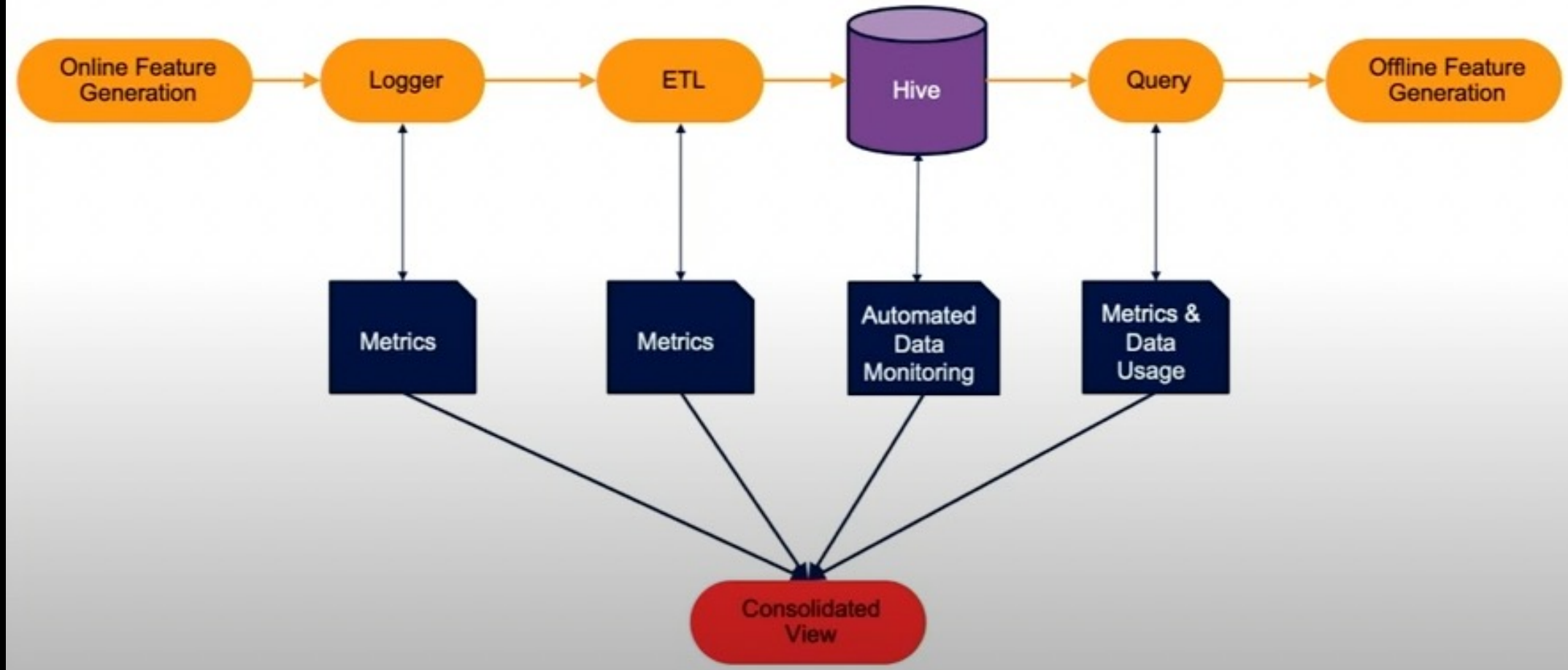
APPLIED ROOTS

Source: Netflix.com

Plain old visualizations

APPLIED ROOTS

Data Quality Checkpoints



APPLIED ROOTS

Automating Large-Scale Data Quality Verification

Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann
Amazon Research
{sseb,langed,phschmid,celikelm,biessmann}@amazon.com

Andreas Grafberger*
University of Augsburg
andreas.grafberger@student.uni-augsburg.de

https://www.amazon.science/publications/automating-large-scale-data-quality-verification

Spark-based

SOURCE

VLDB 2018

'Unit tests' for data

- user-defined constraints
- pre-defined tests (optimal implementation)

↳ examples below:

Table 1: Constraints available for composing user-defined data quality checks.

constraint	arguments	semantic
dimension *completeness*		
`isComplete`	column	check that there are no missing values in a column
`hasCompleteness`	column, udf	custom validation of the fraction of missing values in a column
dimension *consistency*		
`isUnique`	column	check that there are no duplicates in a column
`hasUniqueness`	column, udf	custom validation of the unique value ratio in a column
`hasDistinctness`	column, udf	custom validation of the unique row ratio in a column
`isInRange`	column, value range	validation of the fraction of values that are in a valid range
`hasConsistentType`	column	validation of the largest fraction of values that have the same type
`isNonNegative`	column	validation whether all values in a numeric column are non-negative
`isLessThan`	column pair	validation whether values in the 1s column are always less than in the 2nd column
`satisfies`	predicate	validation whether all rows match predicate
`satisfiesIf`	predicate pair	validation whether all rows matching 1st predicate also match 2nd predicate
`hasPredictability`	column, column(s), udf	user-defined validation of the predictability of a column
statistics (can be used to verify dimension *consistency*)		
`hasSize`	udf	custom validation of the number of records
`hasTypeConsistency`	column, udf	custom validation of the maximum fraction of values of the same data type
`hasCountDistinct`	column	custom validation of the number of distinct non-null values in a column
`hasApproxCountDistinct`	column, udf	custom validation of the approx. number of distinct non-null values
`hasMin`	column, udf	custom validation of a column's minimum value
`hasMax`	column, udf	custom validation of a column's maximum value
`hasMean`	column, udf	custom validation of a column's mean value
`hasStandardDeviation`	column, udf	custom validation of a column's standard deviation
`hasApproxQuantile`	column, quantile, udf	custom validation of a particular quantile of a column (approx.)
`hasEntropy`	column, udf	custom validation of a column's entropy
`hasMutualInformation`	column pair, udf	custom validation of a column pair's mutual information
`hasHistogramValues`	column, udf	custom validation of column histogram
`hasCorrelation`	column pair, udf	custom validation of a column pair's correlation
time		
`hasNoAnomalies`	metric, detector	validation of anomalies in time series of metric values

APPLIED ROOTS

e.g.:

```
1  val numTitles = callRestService(...)
2  val maxExpectedPhoneRatio = computeRatio(...)
3
4  var checks = Array()
5
6  checks += Check(Level.Error)
7   .isComplete("customerId", "title",
8    "impressionStart", "impressionEnd",
9    "deviceType", "priority")
10  .isUnique("customerId", "countryResidence",
11   "deviceType", "title")
12  .hasCountDistinct("title", _ <= numTitles)
13  .hasHistogramValues("deviceType",
14    _.ratio("phone") <= maxExpectedPhoneRatio)
15
16 checks += Check(Level.Error)
17  .isNonNegative("count")
18  .isLessThan("impressionStart", "impressionEnd")
19  .isInRange("priority", ("hi", "lo"))
20
21 checks += Check(Level.Warning, on="delta")
22  .hasNoAnomalies(Size, OnlineNormal(stdDevs=3))
23 checks += Check(Level.Error, on="delta")
24  .hasNoAnomalies(Size, OnlineNormal(stdDevs=4))
25
26 checks += Check(Level.Warning)
27  .hasPredictability("countryResidence",
28    ("zipCode", "cityResidence"), precision=0.99)
29
30 Verification.run(data, checks)
```

Listing 1: Example for declarative data quality constraint definitions using our API.

API for 'unit tests'

APPLIED ROOTS

Why use the API-based software design?

APPLIED ROOTS

Implementation of the API-functions:

- Incremental computation e.g: mean
- Approximate values: e.g: Quantiles, CountDistinct
- Predictability: build a simple model & use standard metrics

APPLIED ROOTS

Incremental computation

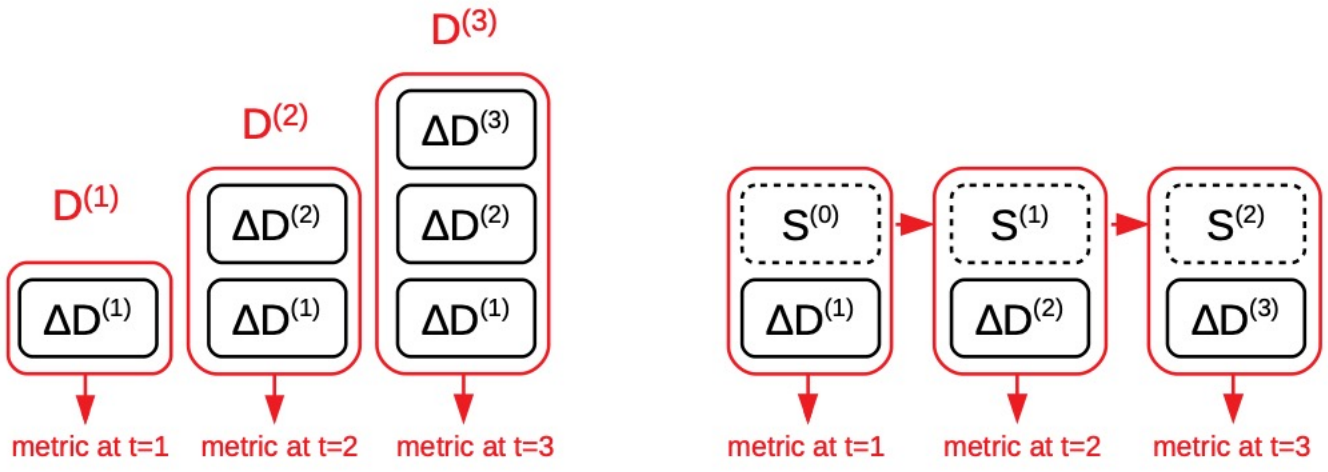


Figure 1: Instead of repeatedly running the batch computation on growing input data D, we support running an incremental computation that only needs to consume the latest dataset delta $\Delta D^{(t)}$ and a state S of the computation.

APPLIED ROOTS

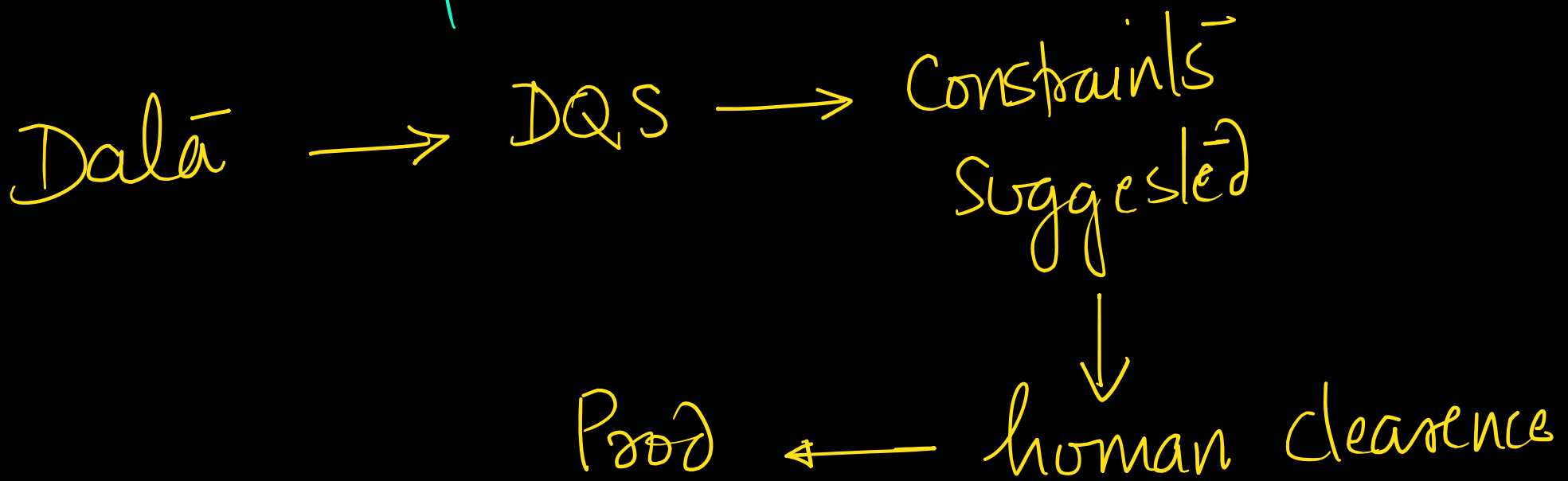
Constraint Suggestions

- heuristics-based
- single-column profiling
  - (1) → datatype; completeness; #distinct-values; size,
  - (2) → Summary stats for Numerical fields;
  - (3) → distribution-fitting

APPLIED ROOTS

Constraint Suggestion

- human in the loop
- increases adoption



APPLIED ROOTS

(Q) Can you suggest "Predictability" constraints also?

(not discussed in the research paper)

APPLIED ROOTS

Anamoly Detection :

- historical time-series
- BYOT : Bring your own technique

APPLIED ROOTS

Architecture :

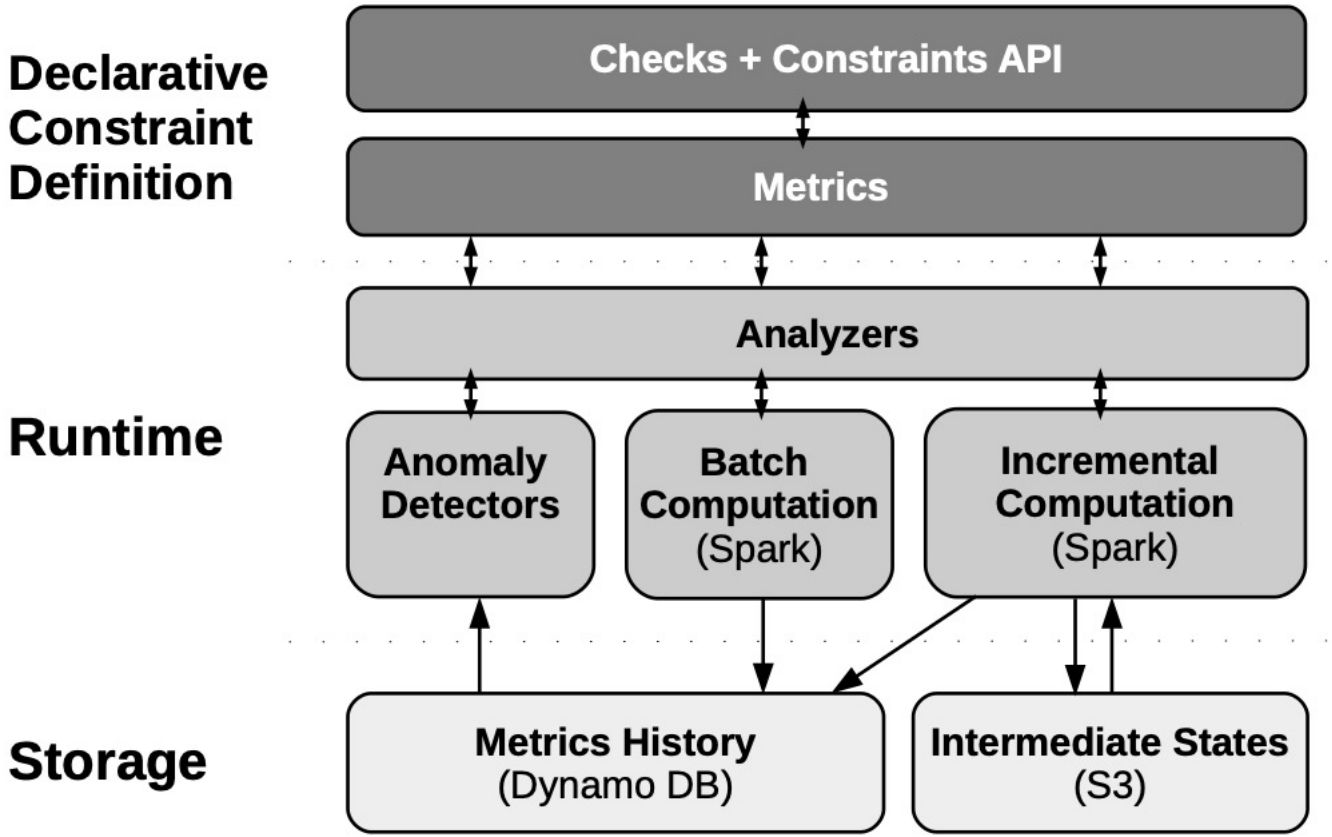


Figure 2: System architecture: users declaratively define checks and constraints to combine with their verification code. The system identifies the required metrics and efficiently computes them in the runtime layer. The history of metrics and intermediate states of incremental computations are maintained in AWS storage services.

APPLIED ROOTS

Section 5 in the research paper

↳ experiments on public datasets

APPLIED ROOTS

Key-take aways

- Spark & HIVE
- Time-Series anamolies
- Simple & Robust algorithms/techniques
- You can do this

APPLIED ROOTS

Idea for an open-source project

DQM based on Spark

(very similar to Amazon's system)