
SPARK+AI
SUMMIT 2019
Build.
Unify.
Scale.
WIFI SSID:SparkAISummit | Password: UnifiedAnalytics
ORGANIZED BY
databricks

Cloud Storage Spring Cleaning: A Treasure Hunt

Zach Musgrave and Steven Moy

Yelp Inc

#UnifiedAnalytics #SparkAISummit

Agenda today

- What is our treasure?
- Where can we find some clues?
- How do we uncover the treasure map?
- What can we find with it?

Motivation

- Insight into cloud storage patterns
- Justification to delete or transition data
- Classification of data via real-world accesses

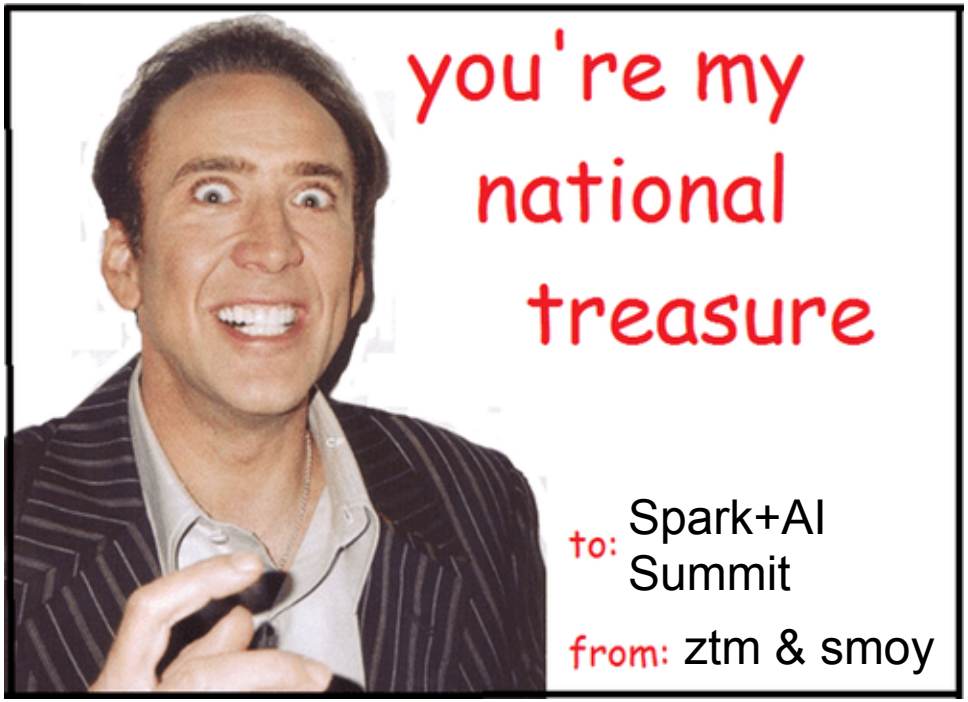
Motivation

- In the beginning, there was infinite storage.
- Yelp said, "Yes, please!"
- Now, we don't have a problem, *per se...*

Motivation

- Now, we don't have a problem, *per se...*
  - We have dozens of PBs worth of problems :/

What is our treasure?



What is our treasure?

We keep many things in S3, and we want to lifecycle data into the right cost tier. How do we make the right decision?

For example, should we archive after 365 days? 400 days?

Motivation

- Cloud object storage: It's everywhere!
  - S3 continues to eat the world's storage workloads
  - No more filing tickets to add hard drives
- As long as you pay the monthly bill, it’s easy...
  - Put(key, object)
  - Get(key)

Eventually, leadership says...

__Why__ are we spending so much money?
Do we __really__ need tens of petabytes __stored in the cloud?__

Eventually, leadership says...

***Why** are we spending so much money?*
*Do we **really** need tens of petabytes **for immediate access at all times?***

Eventually, leadership says...

***Why** are we spending so much money?*
*Do we **really** access* **all data equally all the time?**

And then we say...

*If only we knew something - **anything!** - about engineers' access patterns...*

We need a map...

What is the treasure?

- Large storage savings, measured in $$$
  - You may end up in your org's quarterly slide deck
- Security implications
  - Principle of least privilege made ~~easy~~ easier
- Access trails
  - Make decisions without rolling a 12-sided dice

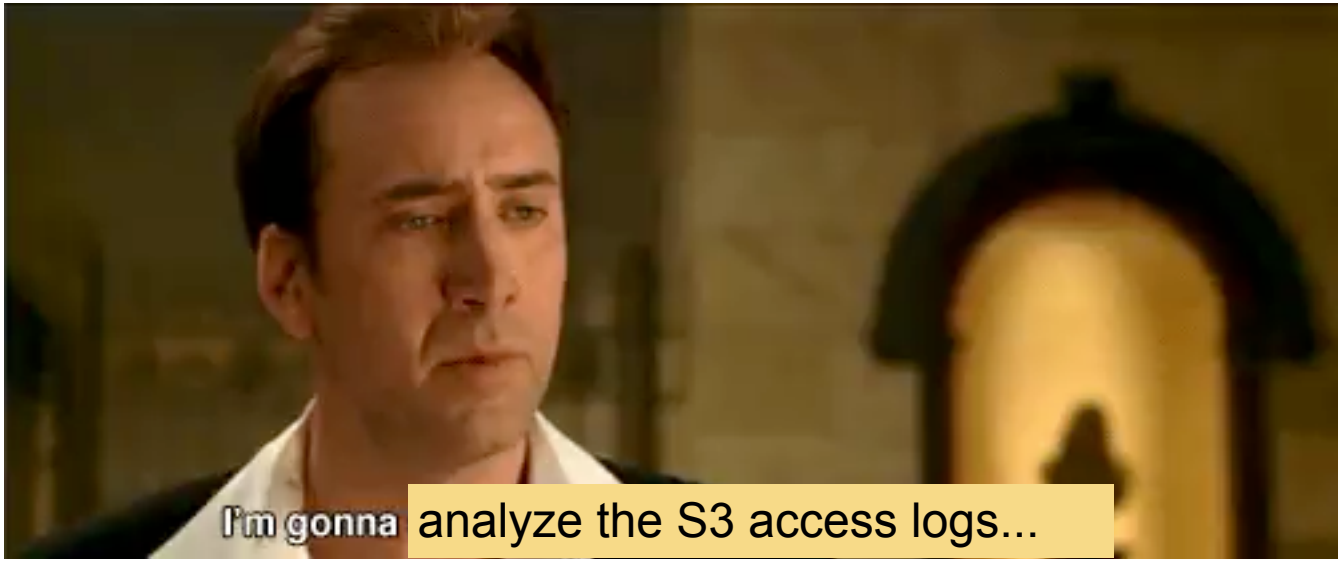
Where are the clues?

- We use AWS S3 as our example
  - Your storage provider likely has similar features
- S3 Server Side Logging
  - Apache-like access logs for every operation

Where are the clues?

- Use S3?
  - Get access logs.
- Where do those go?
  - ... Inside S3 ...
  - If you enable them.

Where are the clues?



How do we do that?

Let's look at the contents of these logs.

S3 Access Logs

- Non-compressed
- Row-based format
- Files can be very small
  - Range from KB to MB
  - Dozens - to many thousands - of rows
- Require a complex regex to parse

S3 Access Logs: Example

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d521
8e7cd47ef2be awsexamplebucket [06/Feb/2019:00:00:38
+0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d521
8e7cd47ef2be 3E57427F3EXAMPLE REST.GET.VERSIONING -
"GET /awsexamplebucket?versioning HTTP/1.1" 200 -
113 - 7 - "-" "S3Console/0.4" -
s9lzHYrFp76ZVxRcpX9+5cjAnEH2ROuNkd2BHfIa6UkFVdtjf5mK
R3/eTPFvsiP/XV/VLi31234= SigV2 ECDHE-RSA-AES128-GCM-
SHA256 AuthHeader awsexamplebucket.s3.amazonaws.com
```

Persistence and structure

- Conversion is a bit expensive
  - No need to repeat conversion over 10+ years of data
  - May as well store a queryable result
  - Determinism: Easy to replicate prior results.

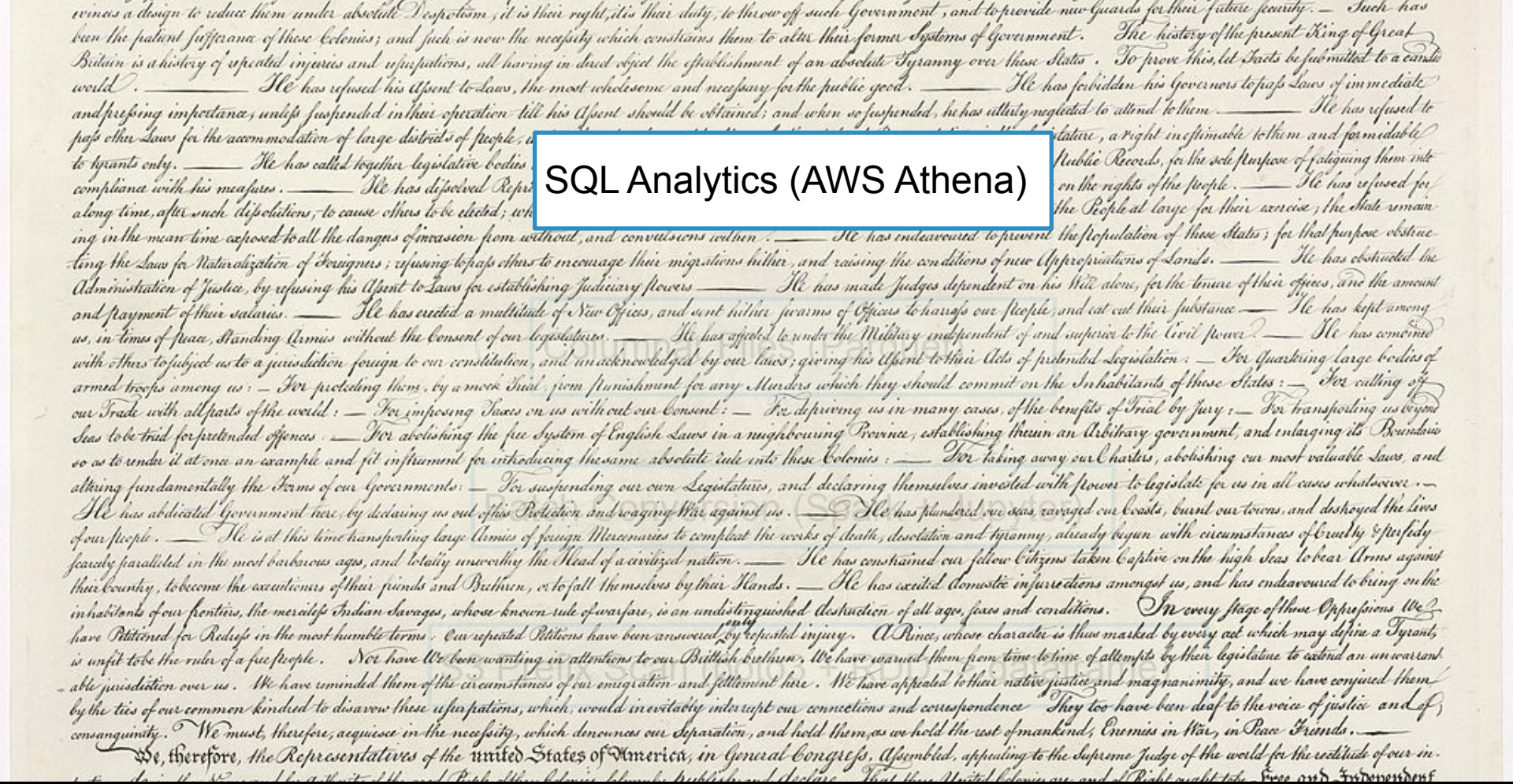
Before we go further

All queries, code fragments, snippets, etc. we show you
Are included in our Github repository!

github.com/Yelp/
aws_logs_to_parquet_converter

So, no need to photograph each slide :)

Where are the clues?


SQL Analytics (AWS Athena)

Clue #1 - SQL declarative analysis

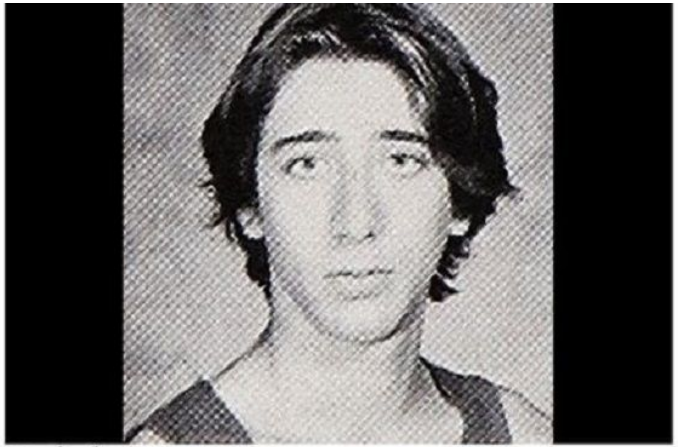
- Our developers and analysts speak SQL
- We can execute SQL against flat files
  - Lots of different ways to do this
  - AWS Athena is easiest for us

Clue #1(a) Nomenclature

- Use your naming conventions
  - `s3://bucketname/logs/2019/03/04/any_file_name_here.tgz`
- Extract object's **creation date**:

```
date_parse(
    array_join(
        regexp_extract_all(key, '/(\d+)', 1), '-'
    ),
    '%Y-%m-%d'
) AS dt_written     ==> date(2019, 03, 04)
```

Clue #1(a) Nomenclature



Nicholas Cage

Hobbies: Acting

Future goals: "I'm going to steal the Declaration of Independence."

Clue #1(b) Days Apart

- We don't care **when** accesses occur
- We do care **how far in the past** the access is

```
date_diff(
    'day',
    date_parse(array_join(regexp_extract_all(..... ,
    date_trunc('day', request_time)
) AS days_apart,
```

Clue #1(c) ARN Bingo

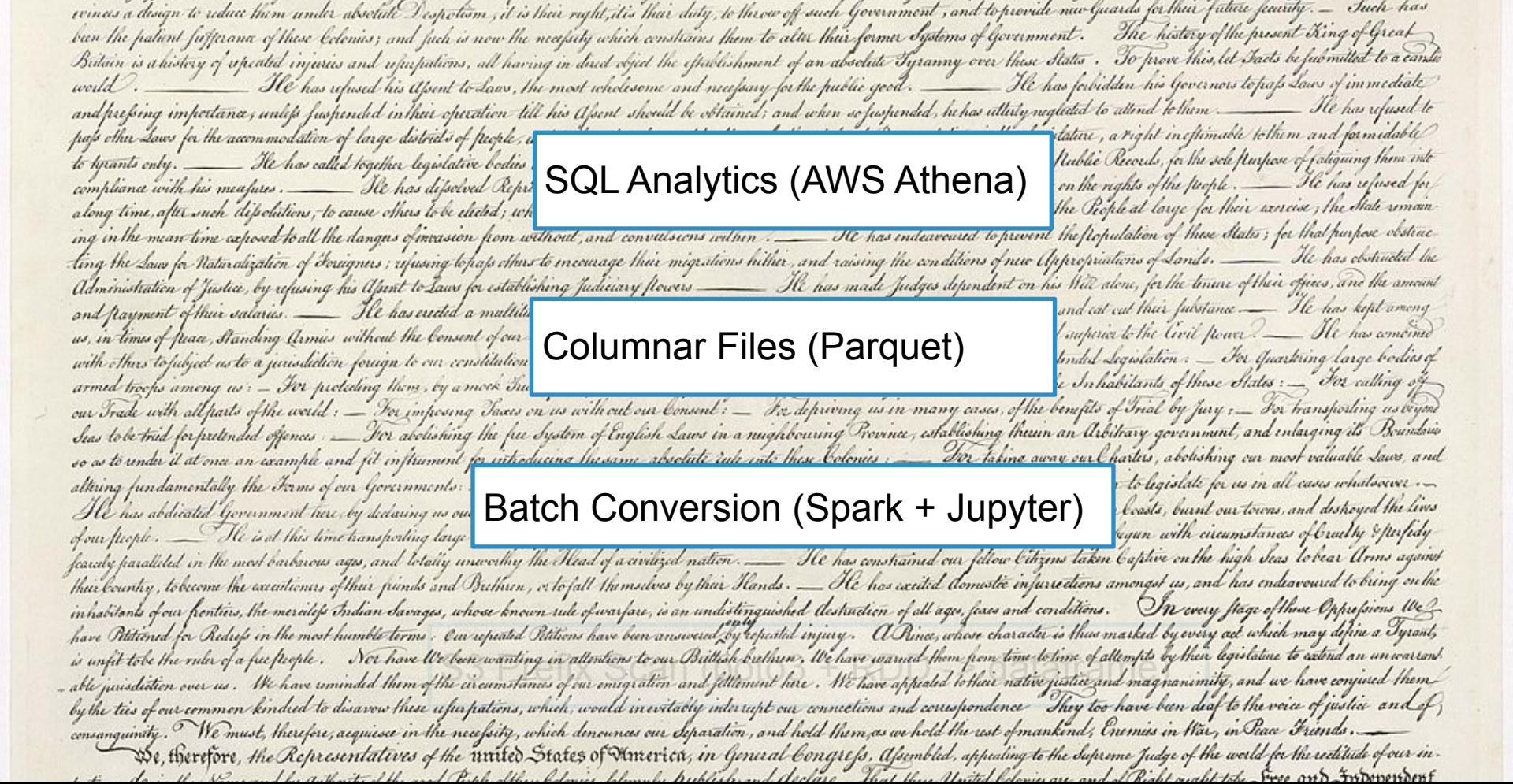
- Carefully break cardinality? Meaningful results!

	requester
1	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-00aa2e7cc1cd03b20
2	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-0f26f91605a6408c5
3	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-0af17382a31413147
4	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-05ea56d0ca67476e8
5	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-0aaa490443938b9c4
6	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-0859888b9f7dc8a57
7	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-0525d799d4cf98a48
8	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-0a9cac4bbe4ecbf60
9	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-0052def119a2e6c5b
10	arn:aws:sts::[redacted]:assumed-role/mrjob-ec2/i-01d49e1b77775244f

Clue #1(d) Group to Understand

- Any large org has **billions** of accesses per year
  - Need to aggregate accesses carefully

```
SELECT
    requester,
    log_name,
    count(*) AS access_count,
    sum(bytes_sent) AS total_bytes
FROM tmp_workspace WHERE
    days_apart > 365
GROUP BY 1, 2
ORDER BY access_count DESC
```

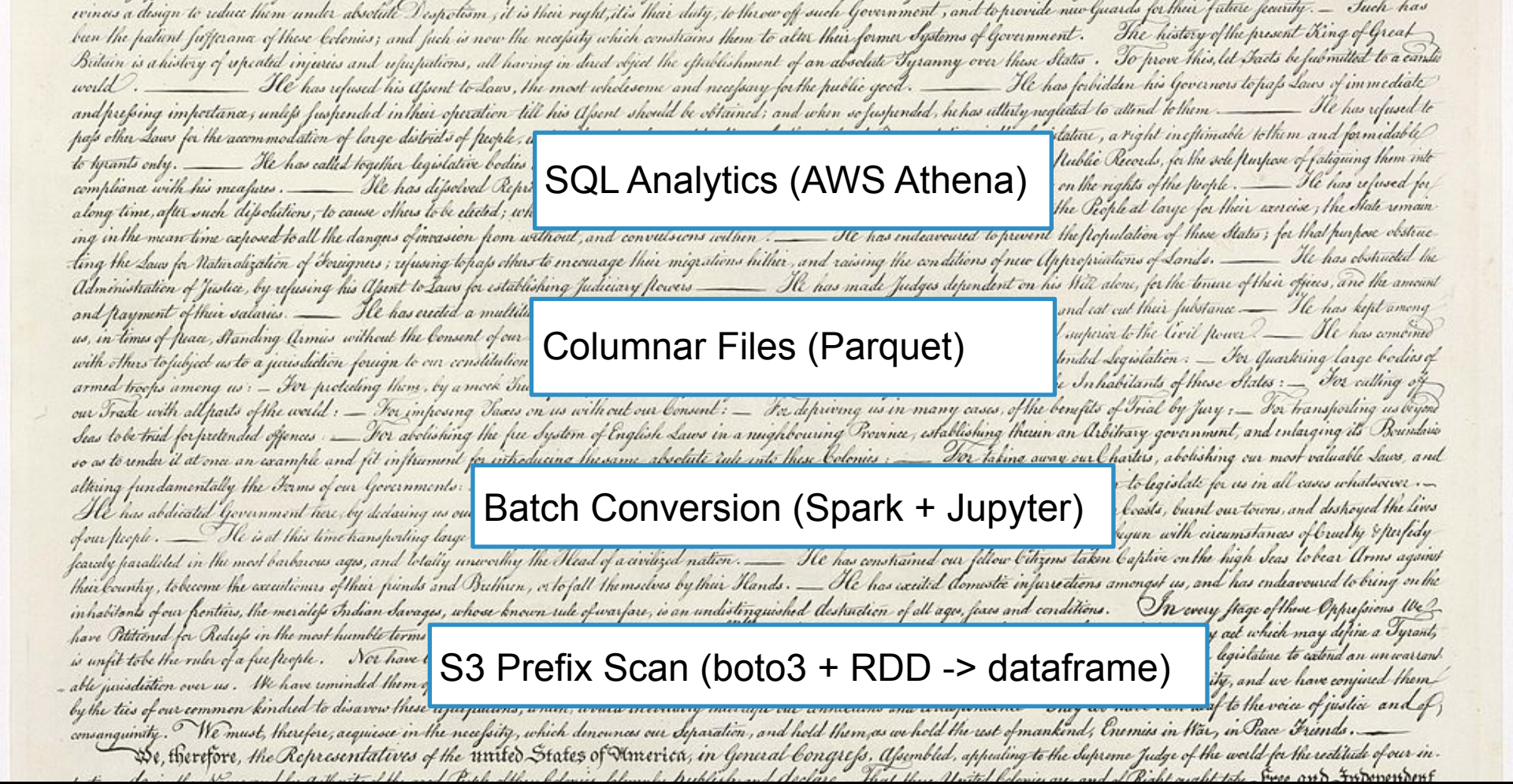

SQL Analytics (AWS Athena)
Columnar Files (Parquet)
Batch Conversion (Spark + Jupyter)

Clue #2 Columnar File Format

- Many small files? List them all? Impossible.
  - Our access logs have **over a million** keys per prefix!
- Queries don't use all possible columns:
  - Amazon's schema has 19 columns
  - We usually reference 7 of those

Clue #2 Columnar File Format

- Convert row-based access logs
  - Make them snappy-compressed Parquet files!
- Write a converter in Spark:
  - Good scalability
  - Great support for Parquet file format


SQL Analytics (AWS Athena)
Columnar Files (Parquet)
Batch Conversion (Spark + Jupyter)
S3 Prefix Scan (boto3 + RDD -> dataframe)

Clue #3 S3 is not a filesystem

- Spark's S3 integration is *sort of* like HDFS
  - Many entries at a *directory?* Can't list them all.
  - `OutOfMemoryException` :'(
- Access log keys
  - Prefixed with date information
  - Use a prefix-based S3 query
  - Paginate? Paginate.

Clue #3 S3 is not a filesystem

```
127 def list_bucket_with_prefix(s3_client, bucket, prefix):
128     token = None
129     more_keys = True
130     keys = []
131     while more_keys:
132         kwargs = {
133             'Bucket': bucket,
134             'Prefix': prefix,
135         }
136         if token is not None:
137             kwargs['ContinuationToken'] = token
138         response = s3_client.list_objects_v2(
139             **kwargs
140         )
141         token = response.get('NextContinuationToken', None)
142         more_keys = (token is not None)
143         keys.extend([content['Key'] for content in response['Contents']])
144     return keys
```

Clue #3 S3 is not a filesystem

- List all log files with a date-based prefix
  - Limit the number of results from S3
  - Constrain your memory utilization
- Generate RDD from the previous results
  - Convert the RDD to a dataframe?
  - Better ecosystem integrations

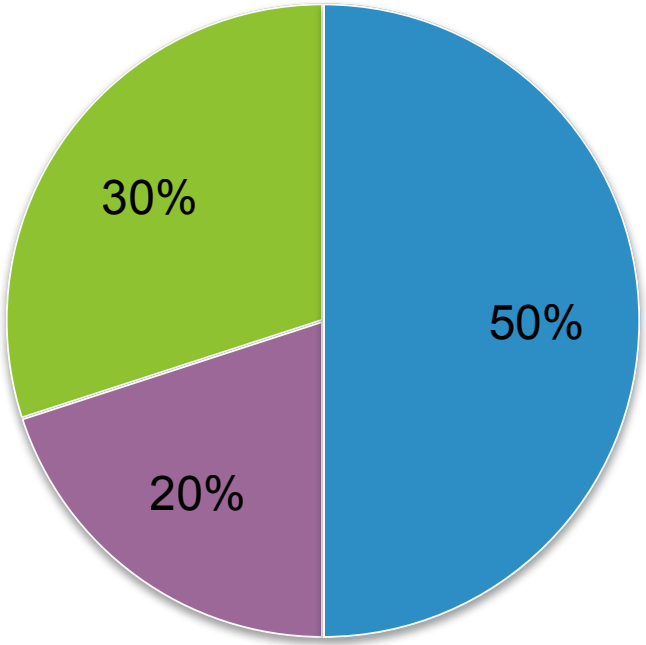
Two-step analysis?

- Self-contained Spark job
  - Process S3 access logs
  - Parquet-ify, then write back to S3
- Query Parquet files with AWS Athena
  - We can all write SQL against them
  - Accessible in the org-wide Data Lake

Org-wide Considerations

- First time at Yelp all data isn't instantly available
  - Glacier bulk restores require ~12 hours
  - Faster (15 minute) restores possible for small files
- Resulting developer impact
  - Very minor! Under ten restore operations so far.
  - Processing time for large computations **already** measured in days.

What did we find?



What did we find?

- ~20% data - 90 to 400 days old - rarely accessed
  - S3 Infrequent Access
    - Pay per download
    - Still immediately available
    - Developer impact: None!

What did we find?

- ~50% data - over 400 days old - archivable
  - AWS S3 Glacier!
    - Pay to archive (once)
    - Pay to retrieve (rarely, usually never)
    - Need to wait ~12 hours between request and usage
    - Developer impact: Need to plan ahead.

What did we find?

- Total savings to the org
  - **About 25%** of our monthly S3 bill. Gone for good!
- Total developer impact
  - **Very little.** Estimated at ~20 hours per year.
- Total pushback when implementing
  - **None.** Because we had the treasure map!
  - *Please sign off on this... unless you dislike money.*

What did we do?

We applied Spark, Parquet, Jupyter, and Athena to S3 access logs.

We wrote SQL to compute basic statistics.

We decreased Yelp's S3 bucket cost by ~25% going forward.

Did we steal the treasure?

Thanks so much!

All queries, code fragments, snippets, etc.
Are included in our Github repository!

github.com/Yelp/
aws_logs_to_parquet_converter

SPARK+AI SUMMIT 2019

DON'T FORGET TO RATE AND REVIEW THE SESSIONS

SEARCH SPARK + AI SUMMIT

SPARK+AI