mLogica IIT Presentation Summary

May 8th, 2020

mLogica IIT Presentation Agenda

Presentation Agenda

- Introduction of team members and guests
- Outline of original project and tasks
- Explanation and detail of tasks accomplished
- Summary and future work

Team

- mLogica
  - Neil Okhandiar - System Architect
  - Apurba Saha - DBA & Developer
- IIT/Information Technology and Management - Graduate Students
  - Balaji Jagannathan
  - Pablo Fernandez Diaz
  - Rutvik Bhamwari
  - Somendra Chaudhary
  - Mridul Khullar
  - Aniruddh Purohit
  - Professor Jeremy Hajek

Outline of Project

- Outline from mLogica of the background the initial contact in November of 2019
  - This is purely multi-phase evolutionary Data Analytics project for IIT offered from mLogica Inc.
- Both of the CEO and System Architect Neil Okhandiar, along with other upper management of the company, took great interest in this joint project
  - Giving Apurba a roadmap of what to accomplish with IIT
  - Apurba specified the guidelines and documented the work in each phase to be completed for the students
  - Each student carried out all the work individually in each phase
  - Apurba attended weekly, nearly daily short meetings to check progress, comment on work, and make adjustments to results

Phases of the Project

- We worked in three phases:
  - Learned how to build queries joining many relations and trying to achieve high granularity of knowledge while working in a simple domain, using AWS S3 and Athena.
  - Building queries over classifications, statistical mean, RMS, co-relations between data & answer some knowledge based questions for senior management, using AWS S3 and QuickSight.
  - Involved various aspects of data analytics like aggregates, text processing, multi-variate running totals (like pivot tables) and also visual presentations, machine learning algorithms, and the IT infrastructure needed to deploy this technology, using AWS S3 and SageMaker
- AWS credentials & accesses for this project were provided by mLogica

March 2006 - Rutvik

- What happened on March 14th, 2006?
  - Microsoft – Just released Windows Vista
  - iPhone didn't exist
  - Android didn't exist
- March 14th, 2006 - AWS S3 (Simple Storage Service)

Simple Storage Service

- Object storage over HTTP
  - Cost for storage dropped
  - Immutable object storage
  - Highly available
  - Data accessible via HTTP (can be used through any platform/OS)
  - Decoupled storage and compute

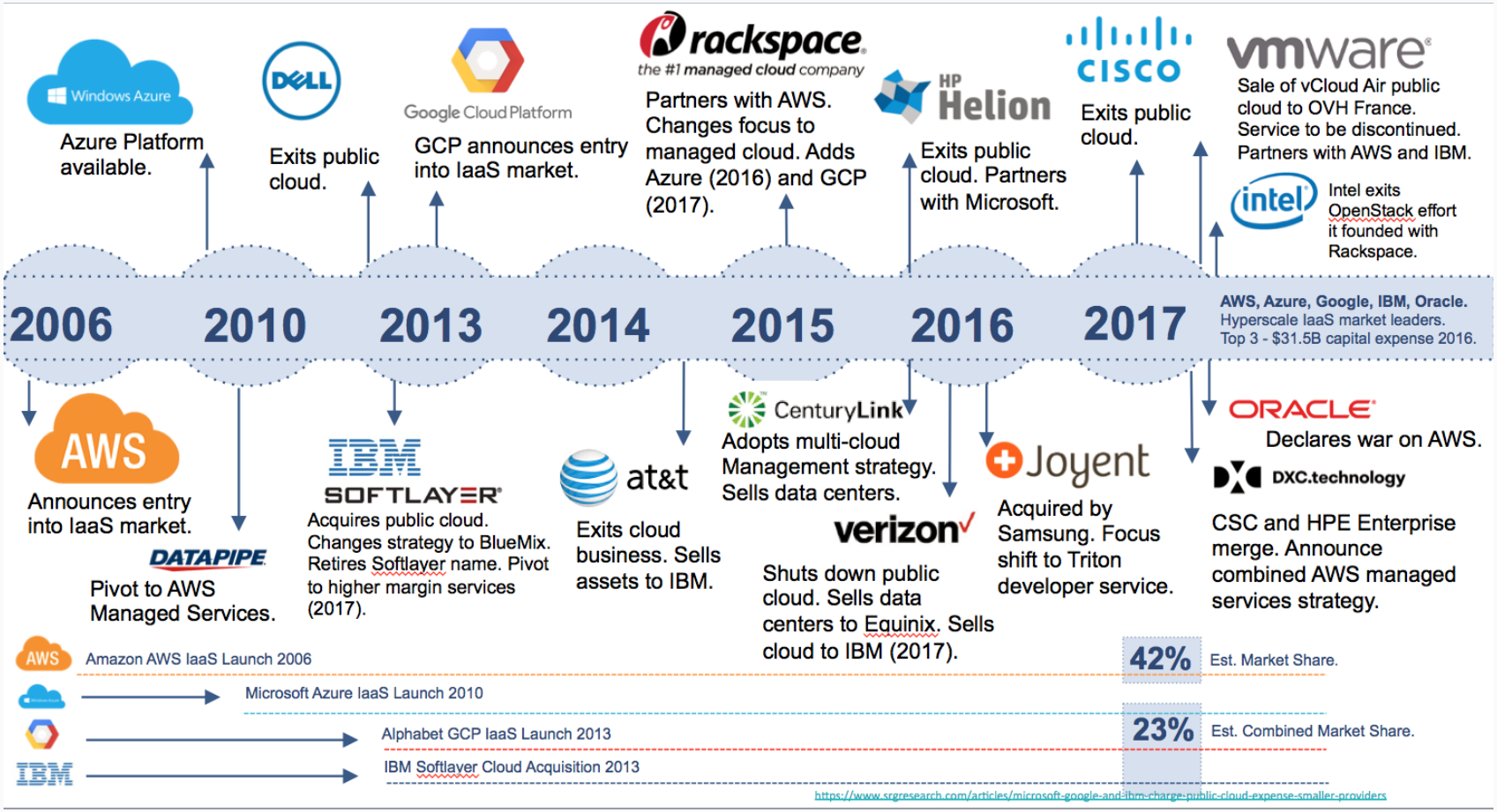


Figure 1: *Cloud Timeline*

When S3 grew up

- 2006 - December 2016
- Increasing data creates latency issues
- More time in transfer than doing analyses on that data

Solution to the problem

- December 2016
  - AWS Athena was born
  - Untraditional way
  - Service that works directly on the data where it sits (in S3)

Introduction to Athena - Rutvik

- Serverless
- Interactive query platform
- Uses standard SQL for querying data
- Uses Presto (a distributed SQL query engine for Big Data)
  - AWS Console
  - Athena API
  - Athena CLI
  - JDBC connection

Continuation

- Only pay for querying for data
- Integrates with ETL tools and AWS QuickSight for data visualization

Images from Athena - Rutvik

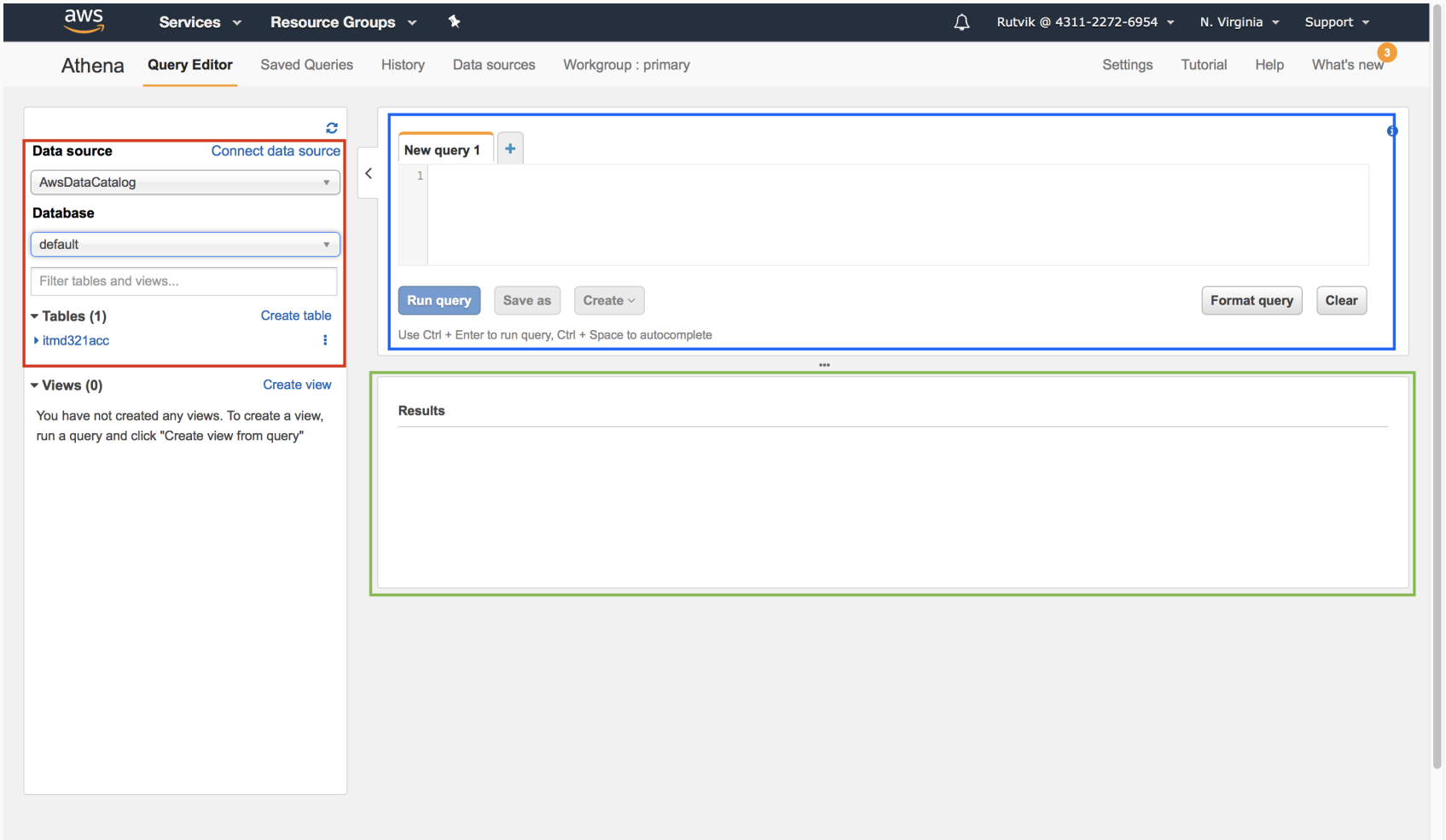


Figure 2: *AWS Athena dashboard*

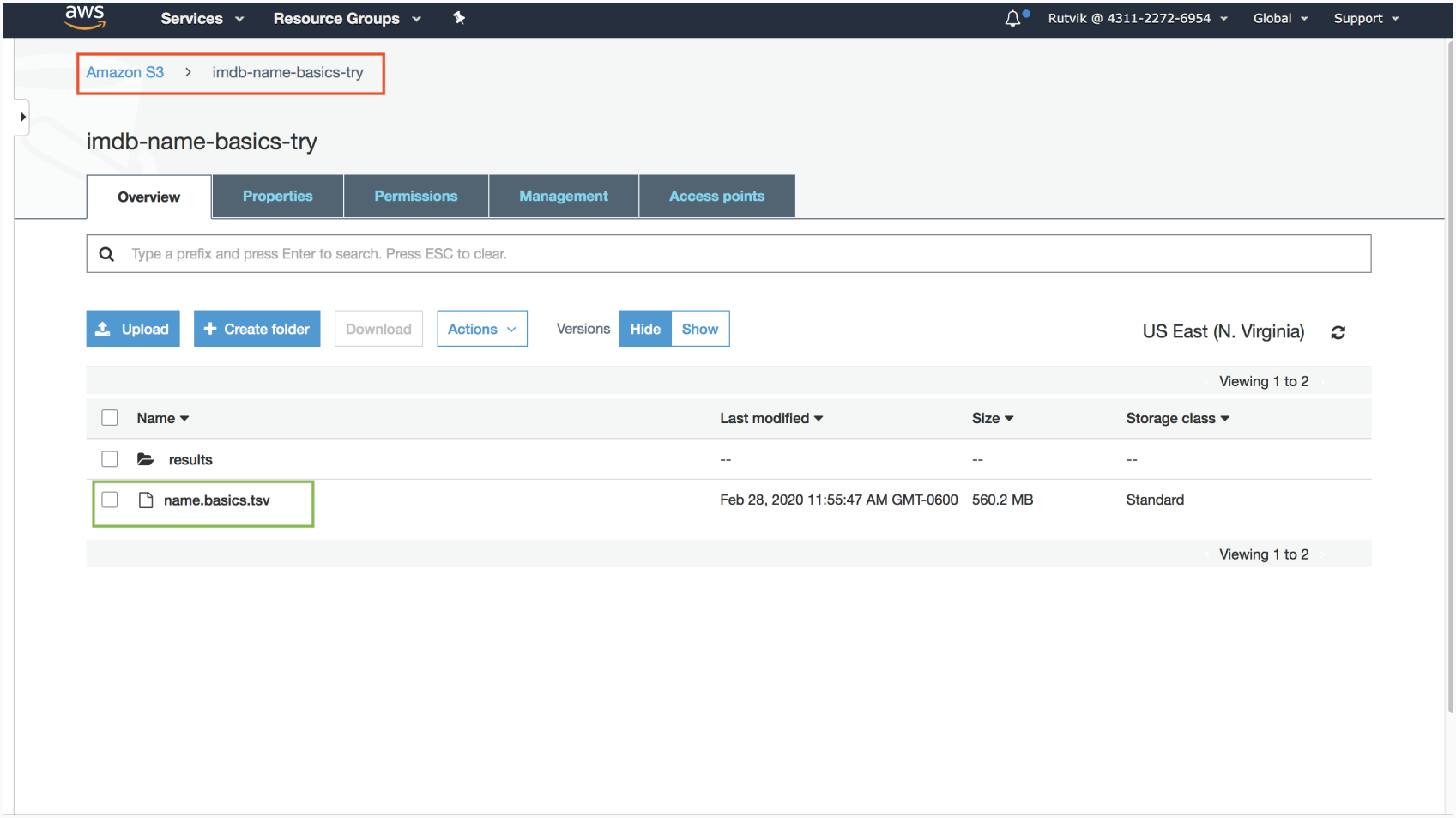


Figure 3: *S3 Bucket*

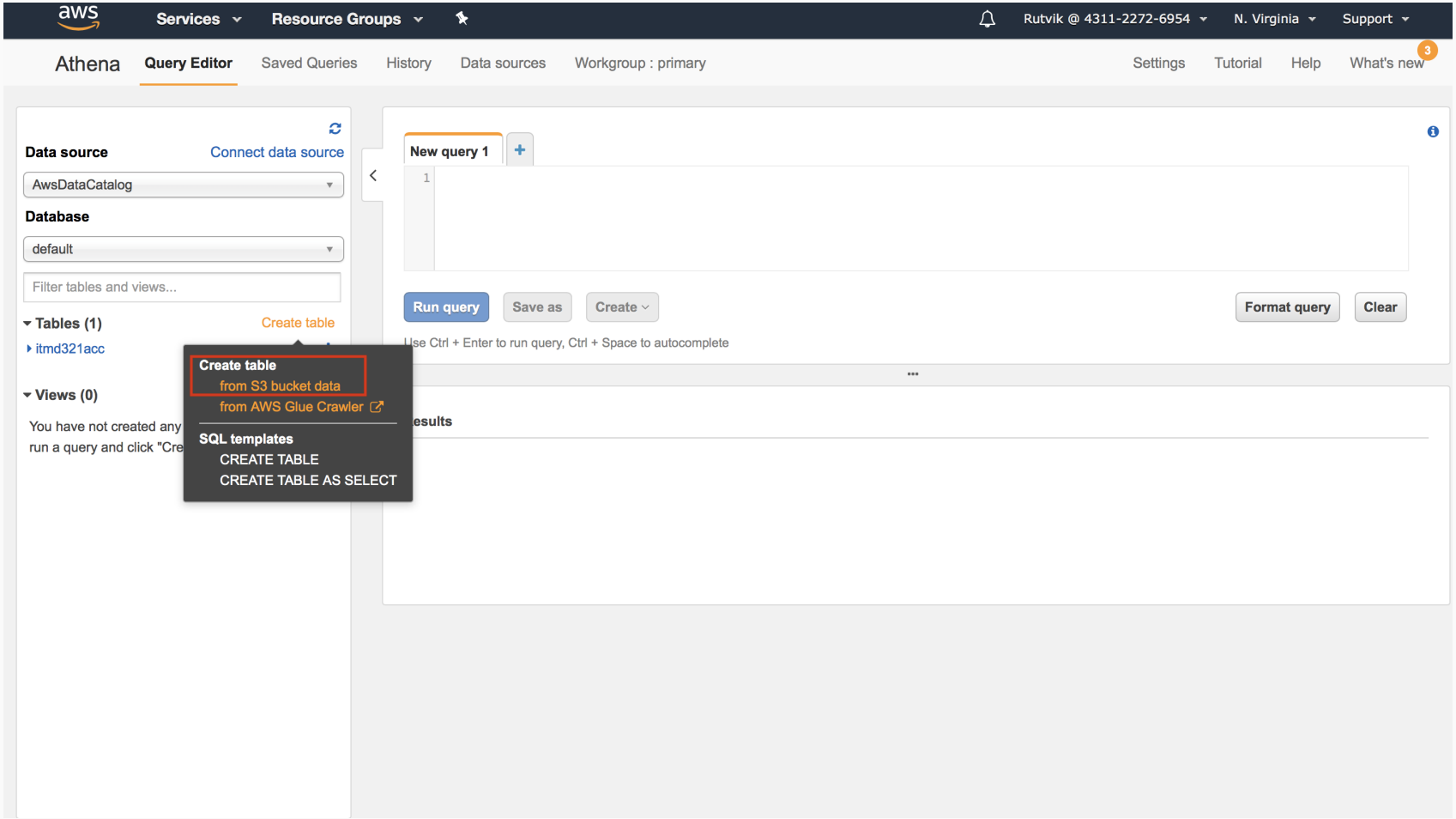


Figure 4: *Create Table*

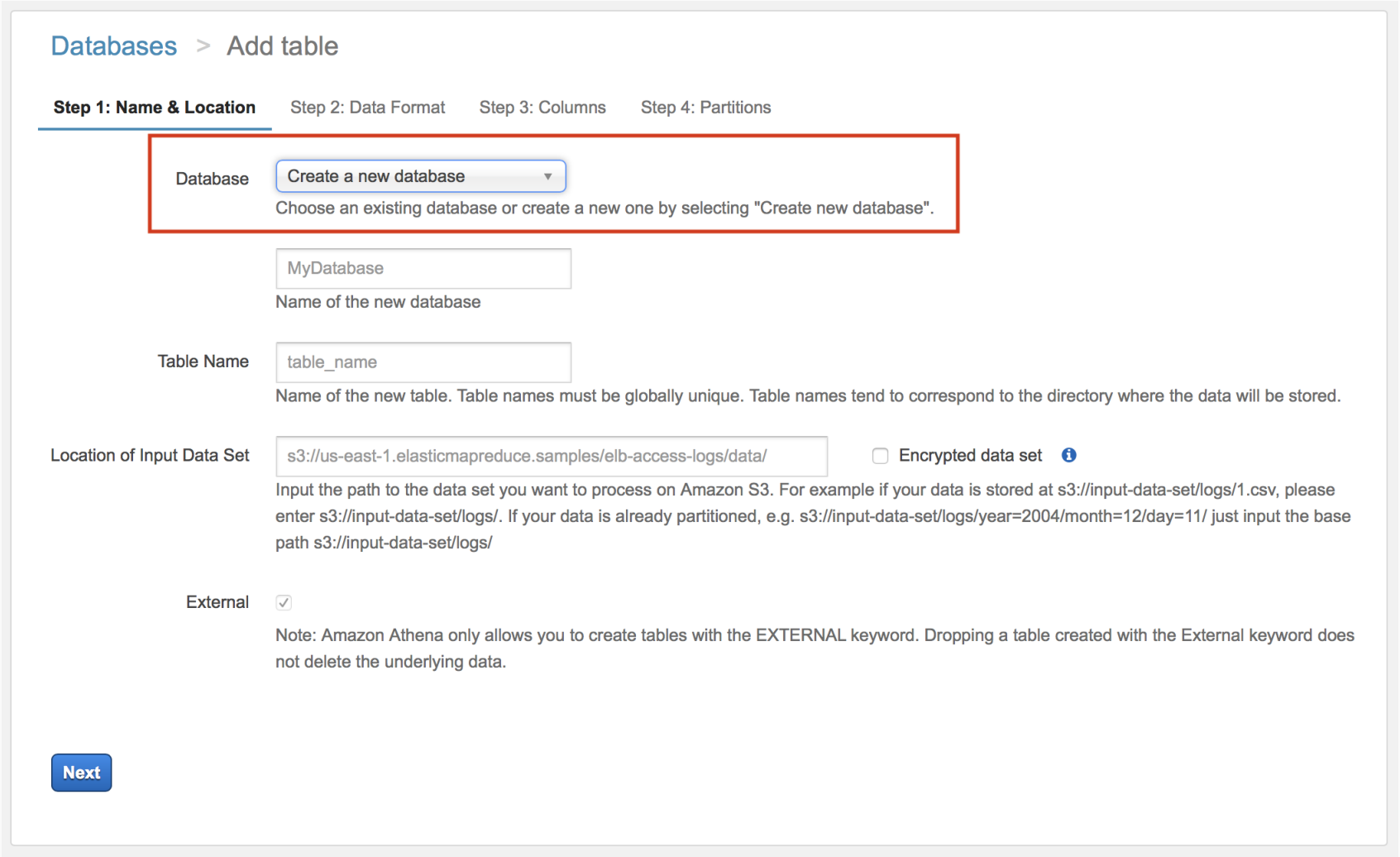


Figure 5: *Create Table2*

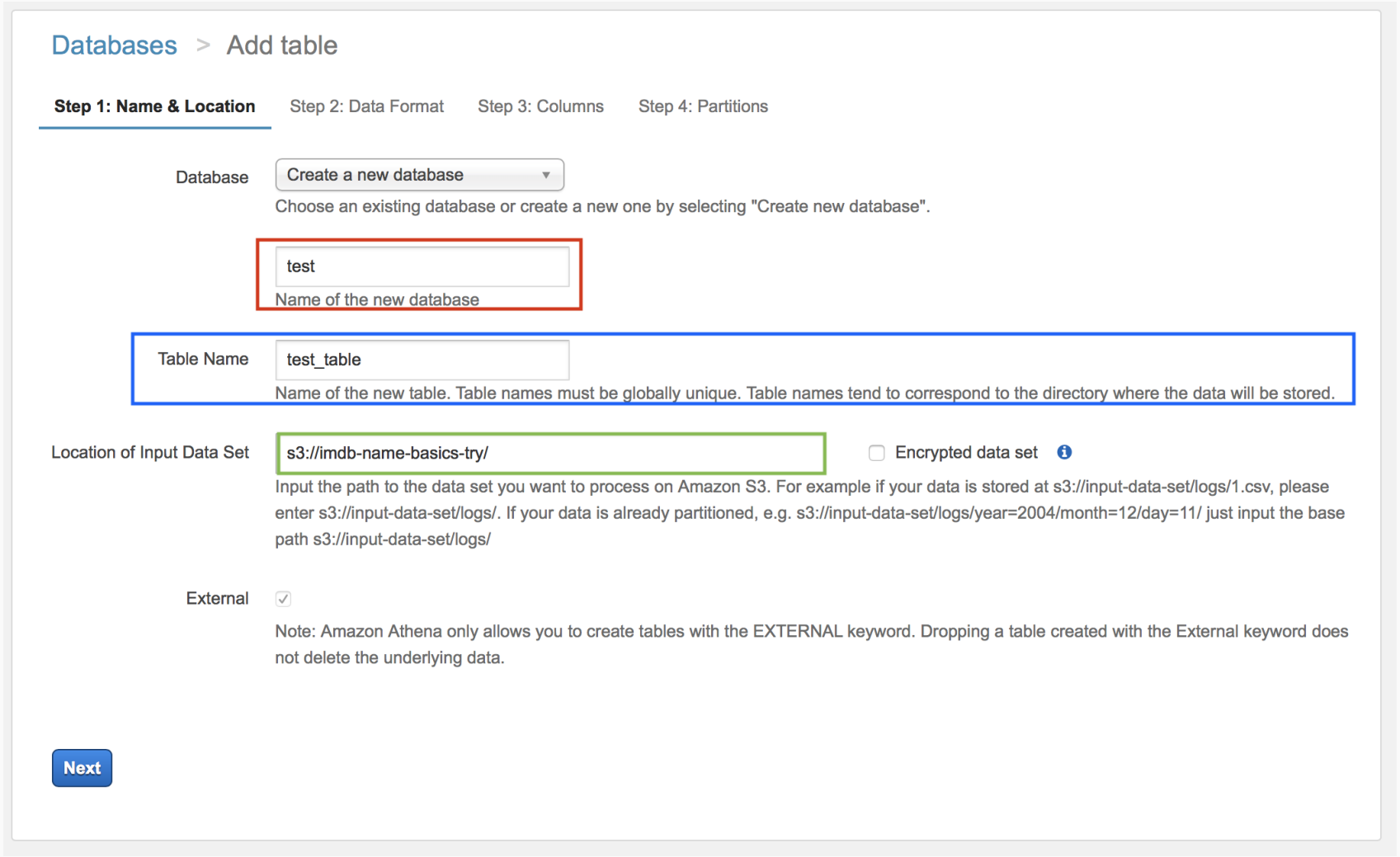


Figure 6: *Create Table3*

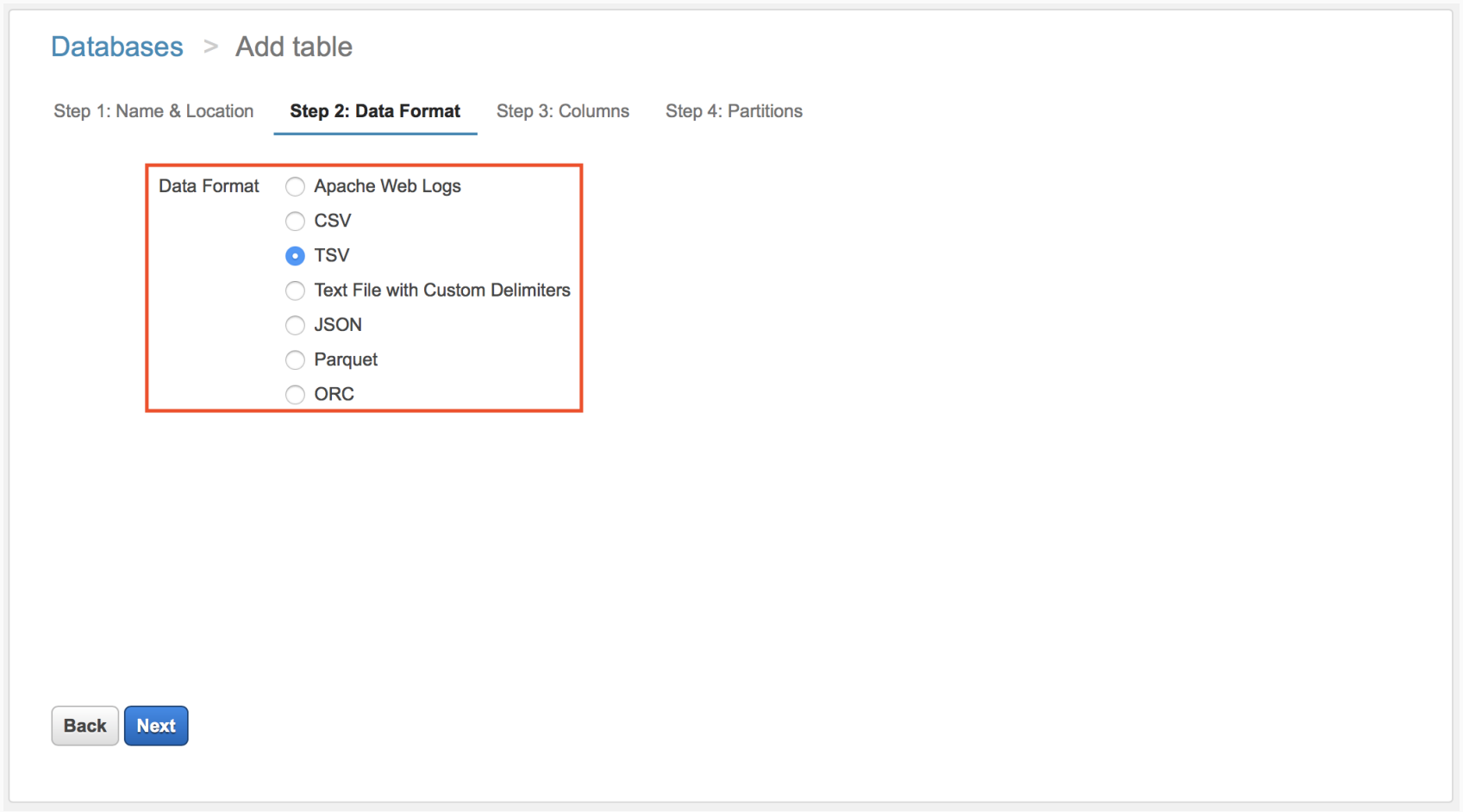


Figure 7: *Create Table4*

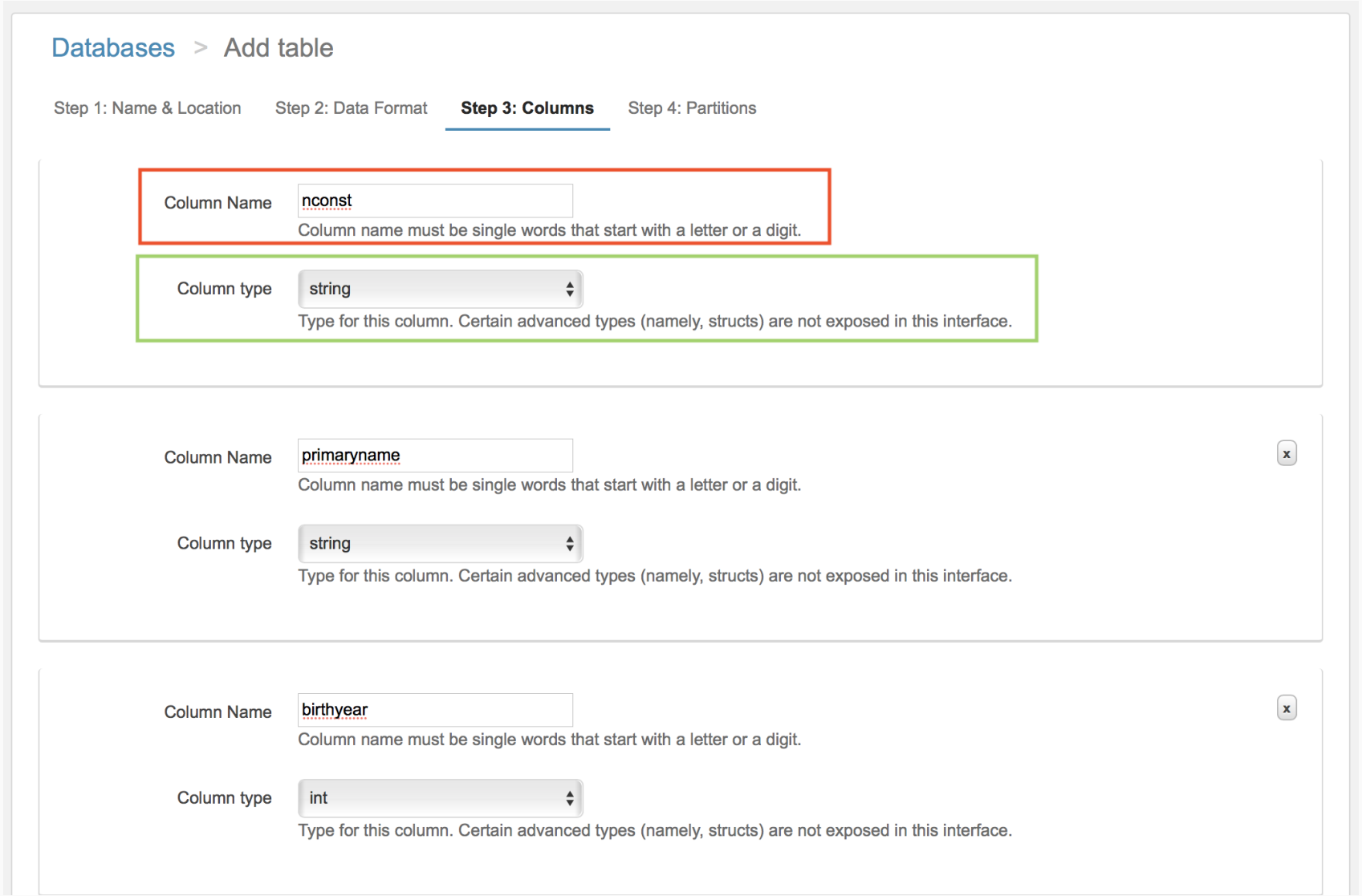


Figure 8: *Create Table5*

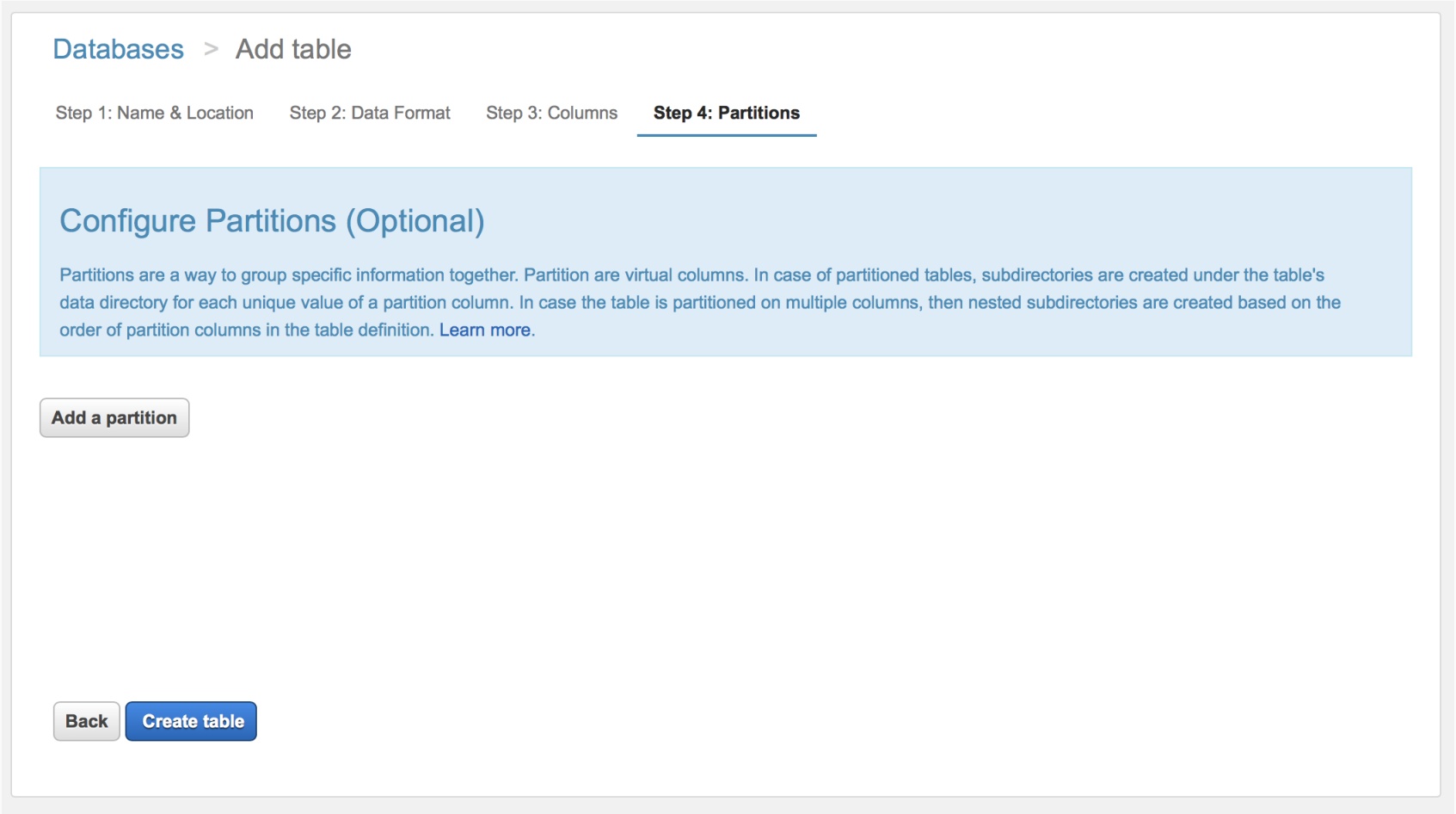


Figure 9: *Create Table6*

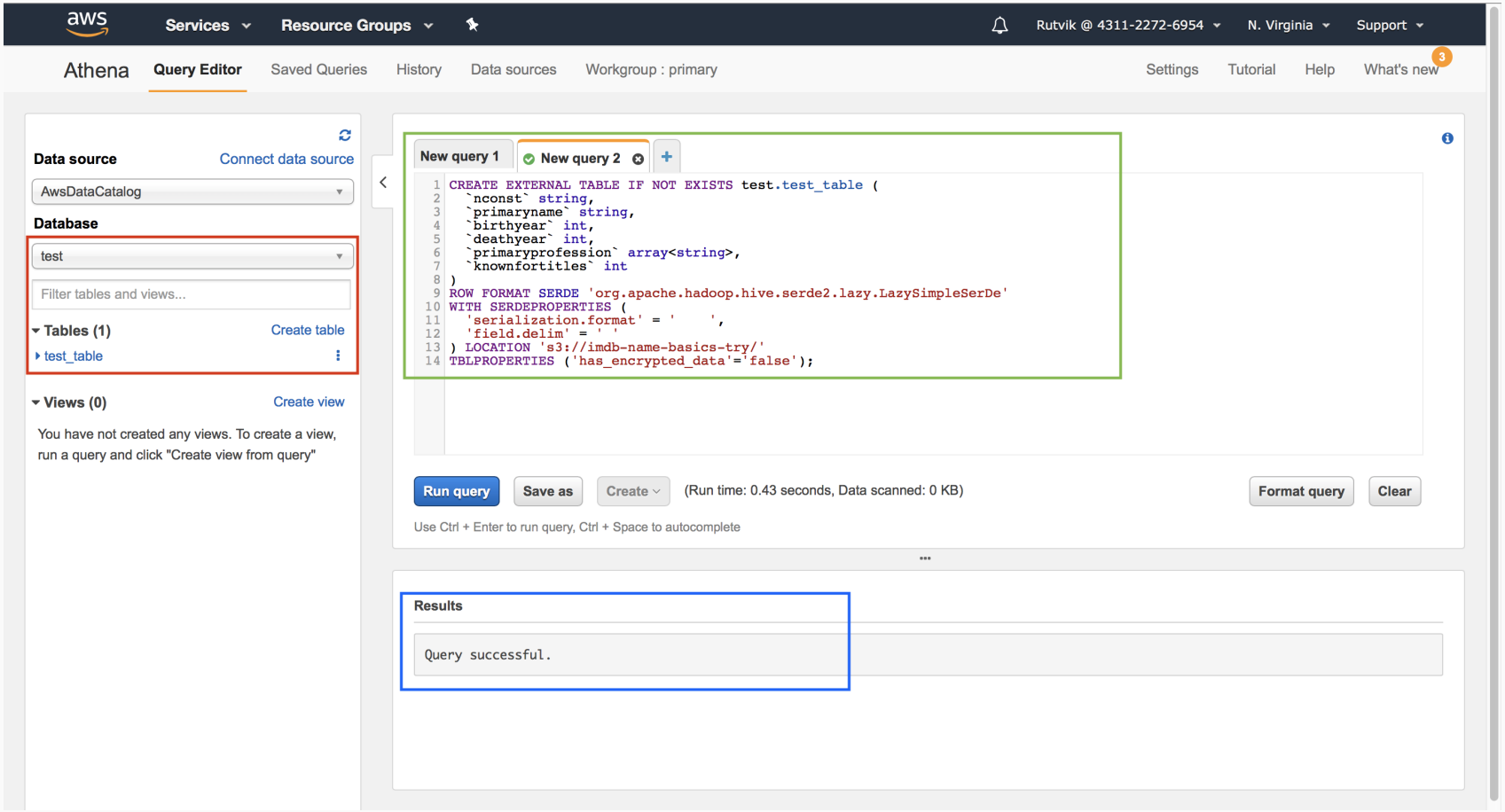


Figure 10: *Create Table7*

Internet Movie Database (IMDb) – Data Insights

- IMDB Dataset
- 7 datasets contained in a gzipped, tab-separated-values (TSV) formatted file
- Size of the datasets: 500 MB to 1 GB
- Contains information on movie titles, movie attributes, cast and crew associated with movies, tv episodes, ratings and votes information, actor names and details
- Loading the IMDB datasets:
  - Uploaded the datasets in S3 buckets
  - In Athena, created database and tables, defined the columns, pointed to the files in S3 bucket
  - Defined a Query result location in S3

IMDB Dataset Description

- Most of the column datatypes are string, integer or float, few of them are arrays
- Important features of dataset:
  - genres: includes up to three genres associated with the title
  - directors: director(s) of the given title
  - startyear, endyear: used to calculate age of movies, TV Series, etc.
  - runtime minutes: used to calculate min, max runtimes of movies, TV Series, etc.
  - density of movies with respect to average rating and number of votes

Issue Faced

- Problem while defining datatype for multi-valued columns:
  - Defined array datatype for multi-valued columns like genres and directors
  - Athena was unable to treat the values as different elements of array
  - Treating all the values as one single element of array

Solution Provided

- To deal with that, we used `split()` function to convert them into arrays, so the values can be read as different elements:

```
CREATE TABLE IF NOT EXISTS title_basics_new AS \
(SELECT tconst, titletype, primarytitle, \
originaltitle, isadult, startyear, endyear, \
runtimeminutes, split(element_at(genres,1),',') \
AS genre FROM title_basics)
```

- Values are now read as different elements of array

IMDB Queries Implementation

- Implemented 11 interactive queries in Athena to understand:
  - Athena's ability to handle complex requests
  - Time taken to perform different analysis
- Used inner joins, cross joins, case statements, nested queries, common table expressions, window functions, arrays
- Documentation on running SQL queries using Athena:
  - Athena User Guide

IMDB Queries With Some Results - Query 1

- Density of movies within last 5 years against number of votes
  -

New query 1 | New query 2 | +

```
1 with cte1 as (select case when numvotes >=1 and numvotes <=100000 then 100000 when numvotes >100000 and numvotes<=200000 then 200000 when
  numvotes >200000 and numvotes<=300000 then 300000 when numvotes >300000 and numvotes<=400000 then 400000 when numvotes >400000 and
  numvotes<=500000 then 500000 when numvotes >500000 and numvotes<=600000 then 600000 when numvotes >600000 and numvotes<=700000 then 700000
  when numvotes >700000 and numvotes<=800000 then 800000 when numvotes >800000 and numvotes<=900000 then 900000 when numvotes >900000 and
  numvotes<=1000000 then 1000000 when numvotes >1000000 and numvotes<=1100000 then 1100000 when numvotes >1100000 and numvotes<=1200000 then
  1200000 when numvotes >1200000 and numvotes<=1300000 then 1300000 when numvotes >1300000 and numvotes<=1400000 then 1400000 when numvotes
  >1400000 and numvotes<=1500000 then 1500000 when numvotes >1500000 and numvotes<=1600000 then 1600000 when numvotes >1600000 and
  numvotes<=1700000 then 1700000 when numvotes >1700000 and numvotes<=1800000 then 1800000 when numvotes >1800000 and numvotes<=1900000 then
  1900000 when numvotes >1900000 and numvotes<=2000000 then 2000000 when numvotes >2000000 and numvotes<=2100000 then 2100000 when numvotes
  >2100000 and numvotes<=2200000 then 2200000 else NULL end as num, t2.primarytitle, count(t2.primarytitle) over () as cnt from title_ratings
  t1 inner join title_basics_new t2 on t1.tconst = t2.tconst where t2.startyear >=2015 and t2.startyear <= 2019 and t2.titletype = 'movie') ,
  grouped_num_1 as (
2 select distinct t1.num as range, count(t1.primarytitle) over (partition by t1.num) as num_count, cast(t1.cnt as double) as tot from cte1 t1
  )
3 select t.range, (t.num_count/t.tot) as density from grouped_num_1 t order by t.range
4
```

Run query | Save as | Create | (Run time: 4.8 seconds, Data scanned: 98.21 MB) | Format query | Clear

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

	range	density
1	100000	0.9938290798631874
2	200000	0.0035638259704848236
3	300000	0.0012198330503001746
4	400000	5.740390824941998E-4
5	500000	2.3918295103924993E-4
6	600000	3.3485613145494987E-4
7	700000	1.1959147551962497E-4
8	800000	4.783659020784999E-5
9	900000	7.175488531177497E-5

IMDB Queries With Some Results - Query 2

- A report of minimum & maximum runtime for different types of movies such as ‘Documentary’, ‘Short’, ‘Animation’

New query 1 +

```
1 select startyear, genre_1, min(runtimeminutes) as min_runtime, max(runtimeminutes) as max_runtime
2 from title_basics_new
3 CROSS JOIN UNNEST(genre) AS t(genre_1)
4 where startyear >= 1960
5 and titletype = 'movie'
6 group by startyear, genre_1
7 order by startyear asc;
```

Run query | Save as | Create | (Run time: 4 seconds, Data scanned: 144.17 MB)

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Results

	startyear	genre_1	min_runtime	max_runtime
1	1960	Musical	65	204
2	1960	Action	54	208
3	1960	Comedy	51	180
4	1960	Sport	76	177
5	1960	Mystery	58	150
6	1960	War	47	197
7	1960	Fantasy	51	180
8	1960	Sci-Fi	58	113
9	1960	Adventure	56	197
10	1960	Documentary	5	135

IMDB Queries With Some Results - Query 3

- Movies with 10 largest sets of co-directors starting from the top largest and going down by size of co-director array, with average rating above 5
  -

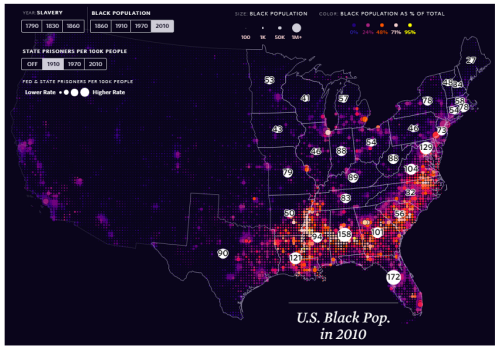
```
select row_number() over (order by cardinality(directors) desc) Rank, Movie_title_id, Movie_title, directors from (select a.tconst as Movie_title_id, b.primarytitle as Movie_title, a.directors from title_crew_new a inner join title_basics_new b on a.tconst = b.tconst inner join title_ratings c on a.tconst = c.tconst where c.averagerating > 5.0 and b.titletype = 'movie')
```

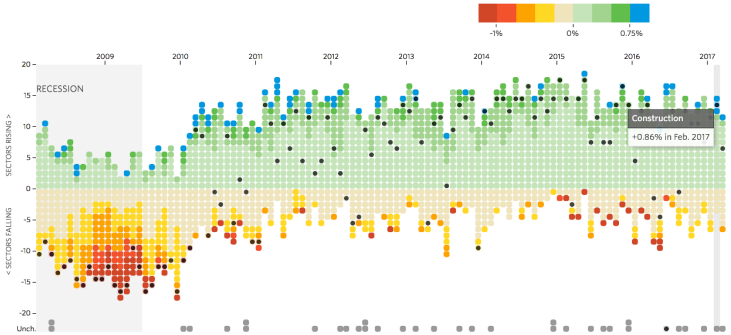
Results

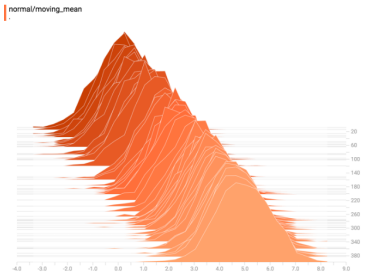
	Rank	Movie_title_id	Movie_title	directors
1	1	tt4050462	World of Death	[nm6775503, nm4718923, nm1121352, nm4673342, nm3877467, nm2371321, nm561
2	2	tt3064438	Venice 70: Future Reloaded	[nm0875354, nm0735134, nm0194156, nm0000934, nm0541391, nm0833708, nm010
3	3	tt2234076	Milano 55,1. Cronaca di una settimana di passioni	[nm5130417, nm0647240, nm4939013, nm4191373, nm5158574, nm2857485, nm515
4	4	tt4942694	60 Seconds to Die	[nm0019144, nm3859710, nm2597539, nm7684836, nm1346075, nm5693439, nm110
5	5	tt3528906	Our RoboCop Remake	[nm2655289, nm1926903, nm1890588, nm1697560, nm2466457, nm1739457, nm835
6	6	tt1436308	140	[nm3603655, nm0153559, nm1923131, nm4694283, nm3806871, nm1446204, nm329
7	7	tt3091166	Our Footloose Remake	[nm1341018, nm3374111, nm4100656, nm2940487, nm4949867, nm2214843, nm111
8	8	tt2243597	The King of Ads	[nm9565029, nm0399853, nm0000217, nm0487237, nm0001874, nm0001031, nm076
9	9	tt2249786	50 Kisses	[nm0427732, nm4951230, nm5877926, nm5405101, nm2853432, nm3863962, nm540
10	10	tt5141260	60 Seconds 2 Die: 60 Seconds to Die 2	[nm2563700, nm6871077, nm7606015, nm6935209, nm5226857, nm5787075, nm795
11	11	tt0113718	Lumière and Company	[nm0770234, nm0412465, nm0000490, nm0464846, nm0000958, nm0761762, nm014
12	12	tt0173616	L'addio a Enrico Berlinguer	[nm0011997, nm0013193, nm0024836, nm0029544, nm0035208, nm0070880, nm000

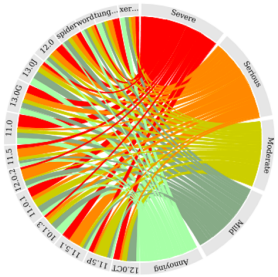
QuickSight - Somendra and Pablo

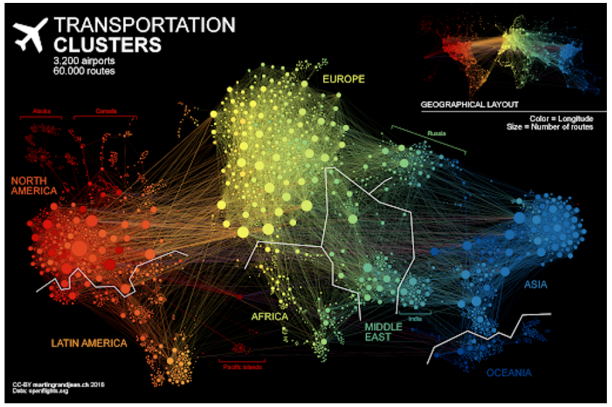
- Introduction to Data Visualization

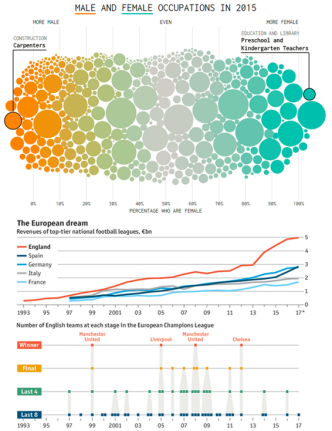


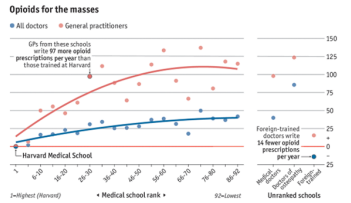












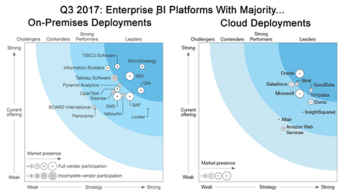


Figure 11: *introCharts*

Data Visualization Platforms - Pablo

- New technological advances are used by Data Visualization Platfroms

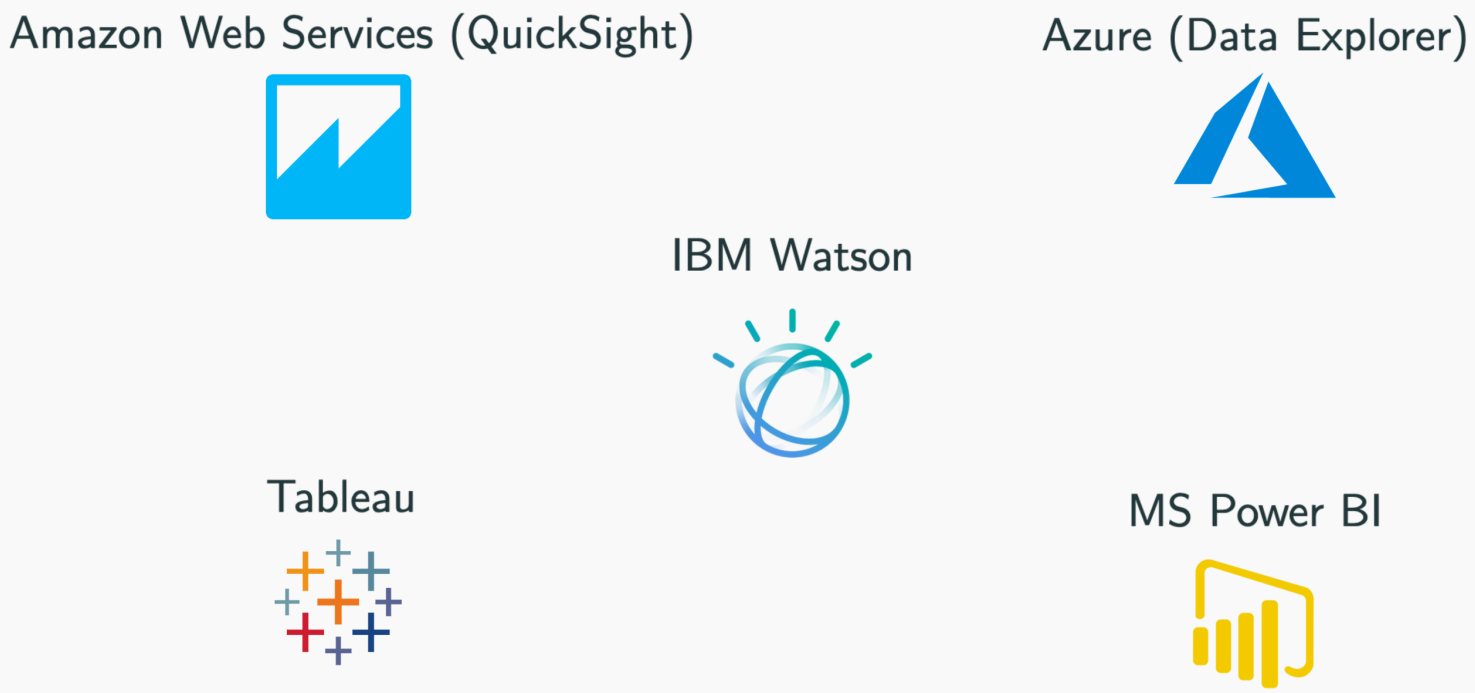


Figure 12: *logos*

The Importance of Data Visualization based on QuickSight

- Easy interactive Dashboards and Reports
- Not a graph picture, data still accessible

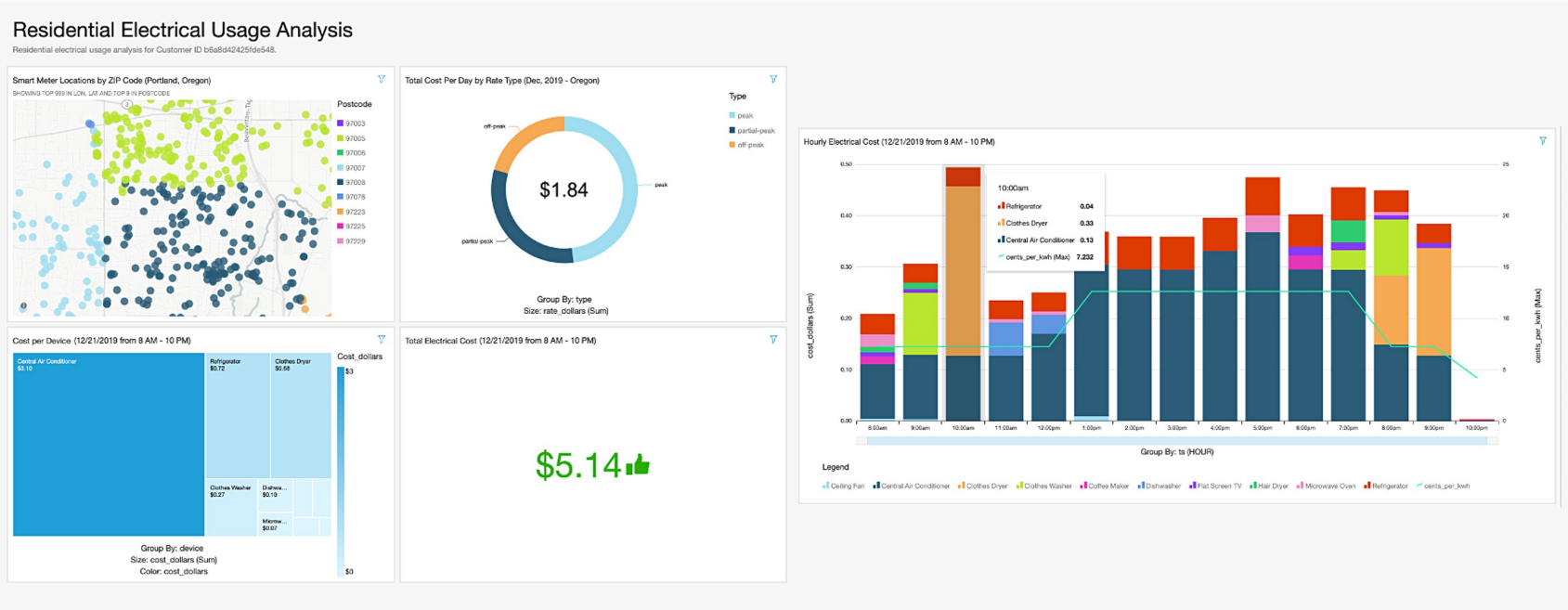


Figure 13: *example*

Main Features of AWS QuickSight & Competitors - Pablo

- Main features:
  - Cloud-based Platform
  - Pay-Per-Use Model
  - Creates Visualizations / Performs Analysis / Gets Business Insights
  - The AWS Business Intelligence and Analysis tool
- AWS QuickSight vs Tableau vs Power BI
  - Three very powerful BI and Analysis tools
  - Some main differences:
    - Pricing model
    - Learning Curve
    - Data Sources and Compatibility

Data Sources QuickSight Compatibility - Pablo

- Easy to connect with other Services and Data Sources
- Data Sources supported by AWS QuickSight:

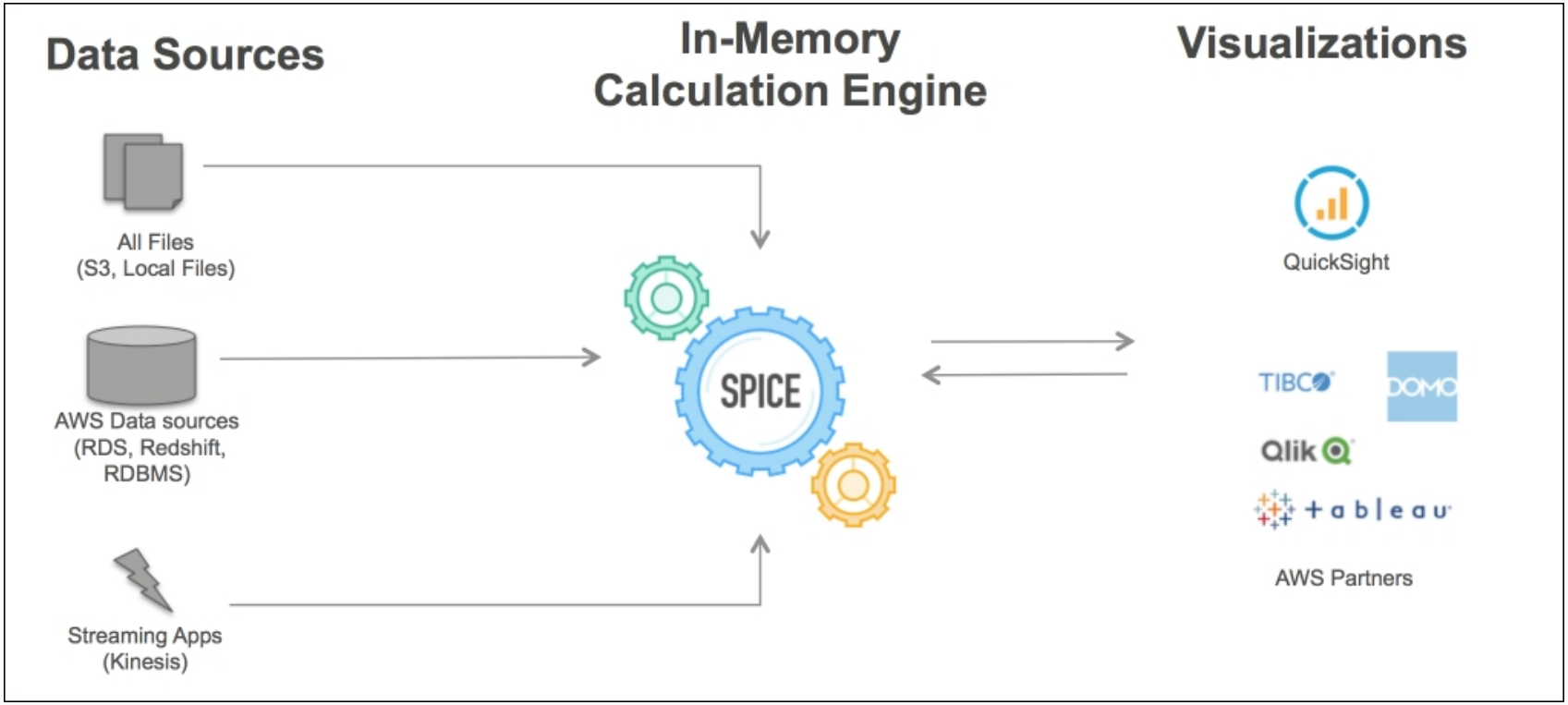


Figure 14: *SPICE_flow*

AWS QuickSight SPICE - Pablo

- SPICE (Super-fast, Parallel, In-memory, Calculation, Engine)
- Performance improve
  - In-memory Processing
  - Not using Direct Query to the Database

US Accidents dataset

- Downloaded from Kaggle.
- US Accidents (3.0 million records) dataset.
- Complex + big dataset
- Fields of interest:
  - Level of severity: 1 (least severe) to 4 (most severe).
  - Start time + End time.
  - Distance (in mi): The length of the road extent affected by the accident.
  - Description: Natural language description of the accident.
  - State.
  - Zipcode.
  - Weather condition.
  - Natural and infrastructure parameters.

Some interesting problems

- Number of accidents in each state for different levels of severity.
- Number of accidents by time range.
- Percentage of level of severity for accidents in good versus bad weather conditions.
- Number of accidents by month year and various parameter range for different levels of severity in good versus bad weather conditions.
- Precious words in description column.
- Average precipitation for 5 select states over time.

Interesting Tools to query data from Athena into SPICE

- Aggregate functions, Window functions, Regexp functions.
- Hierarchical and recursive queries.

Let's see some visuals created using QuickSight

No. of accidents by severity for different states

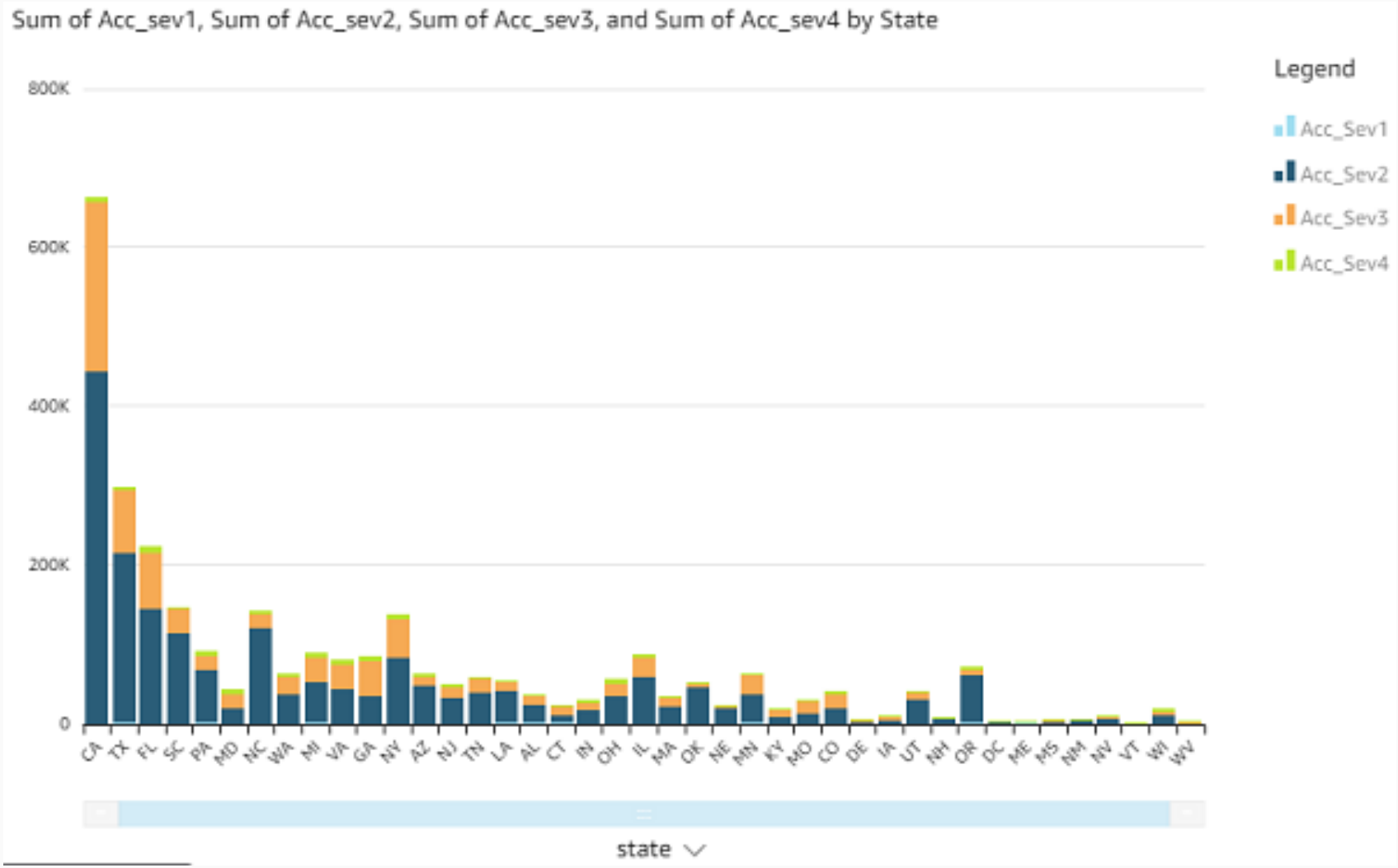


Figure 15: *CA, TX and FL have the highest number and highest severity level of accidents*

Duration of accidents

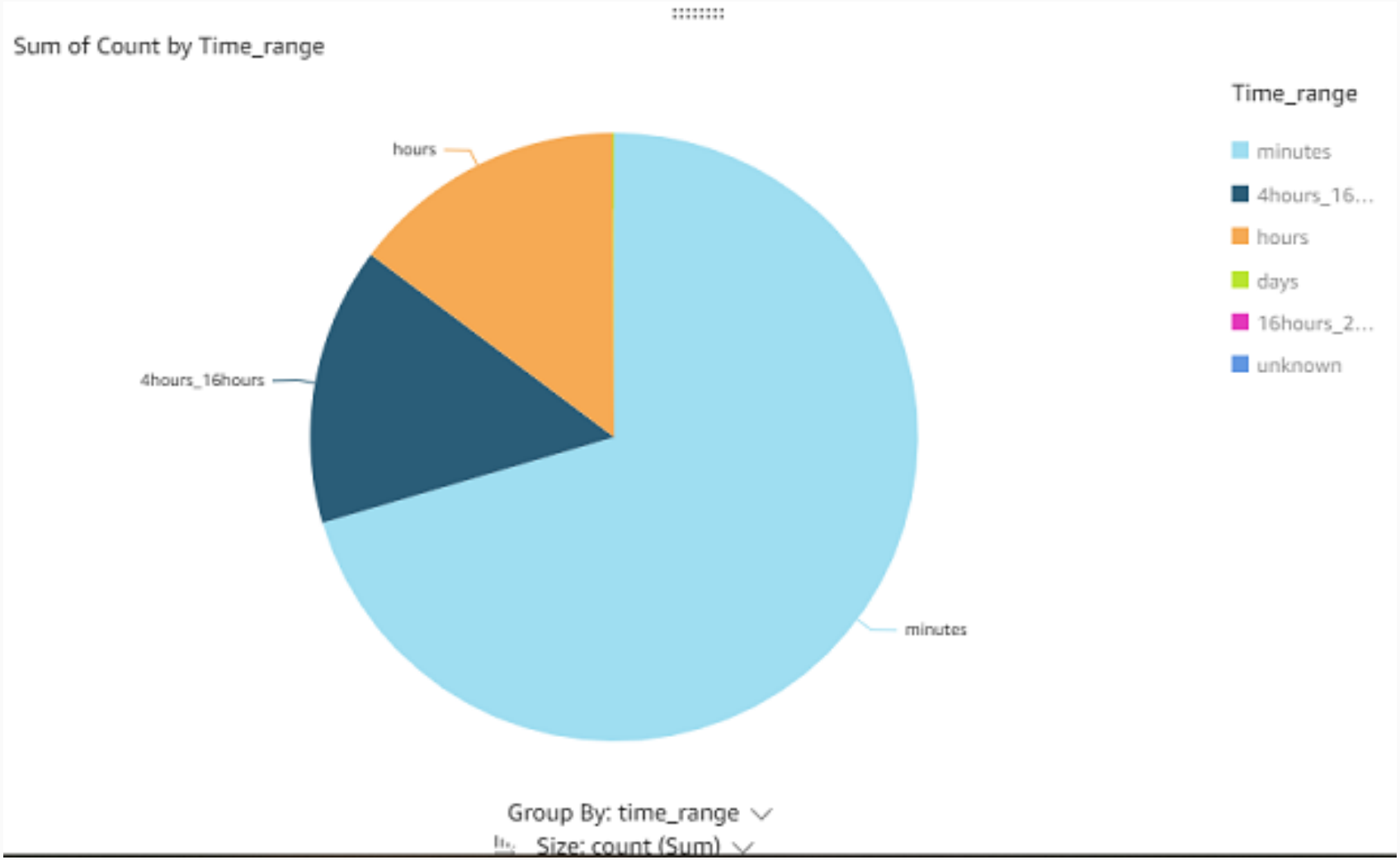


Figure 16: *Most accidents last only for a few minutes.*

Effect of good weather on % of each severity level covering all accidents

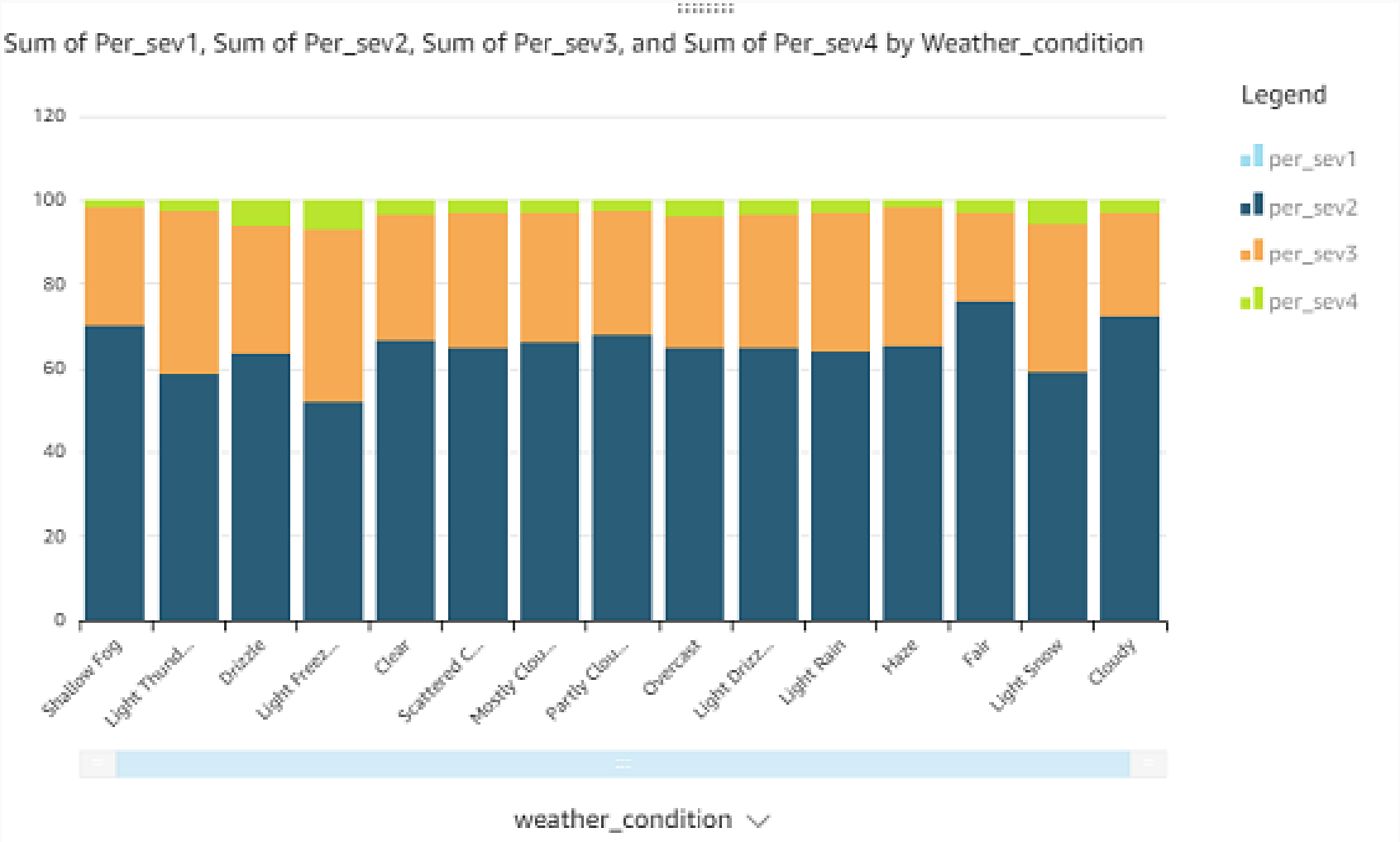


Figure 17: *Do we see any patterns?*

Effect of bad weather on % of each severity level covering all accidents

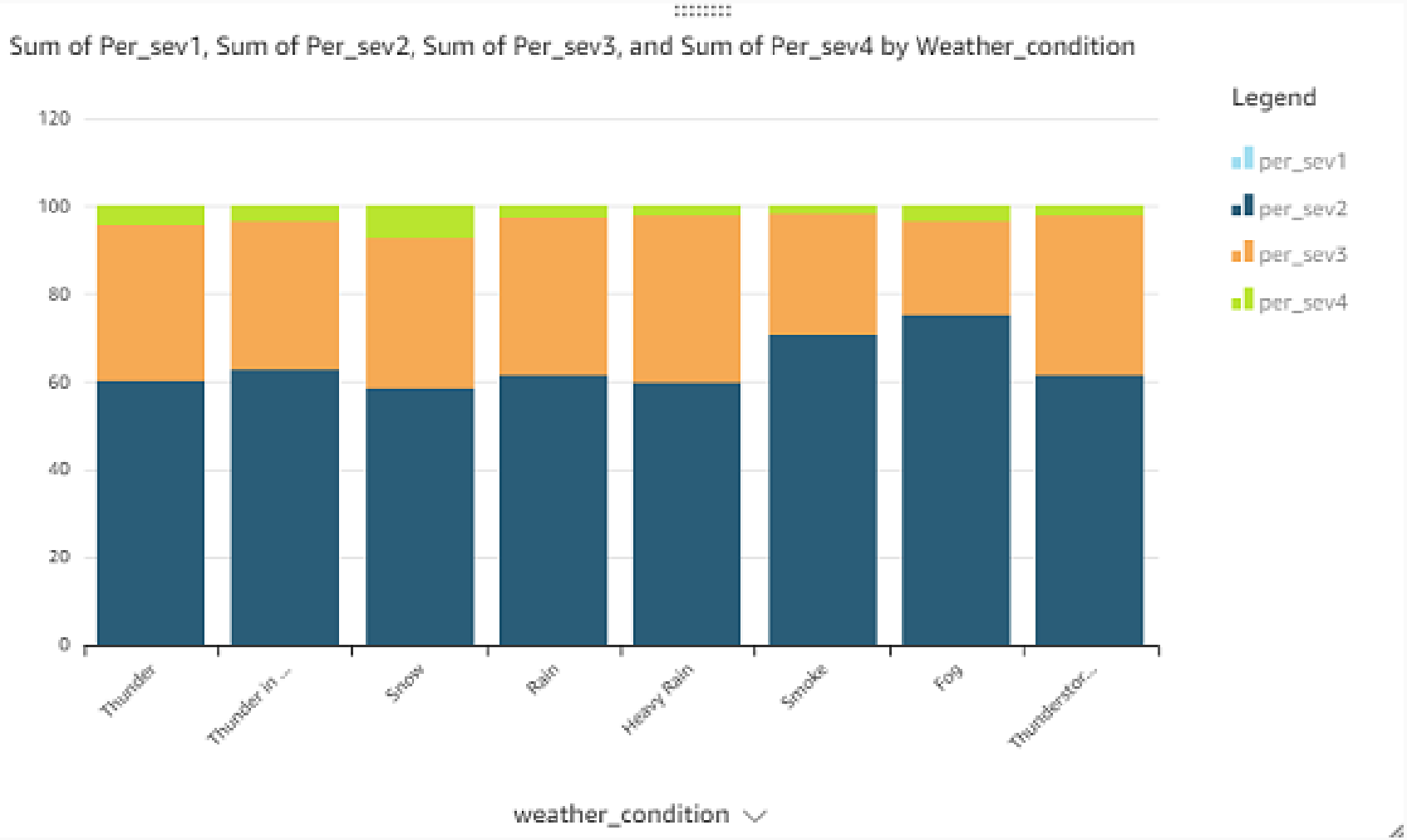


Figure 18: *Weather conditions don't affect the percentage of each level*

Combined effect of good weather and select parameters on accidents

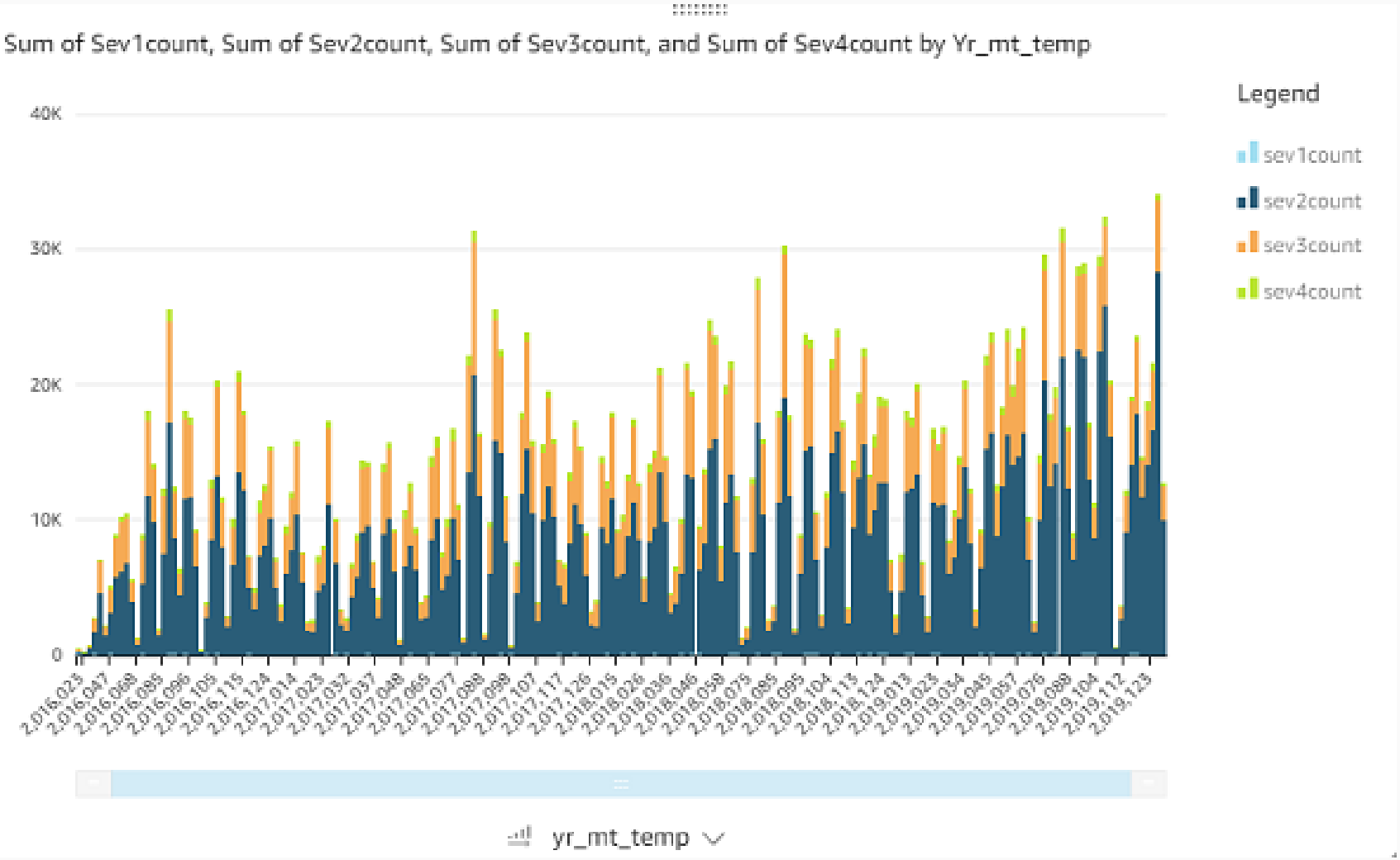


Figure 19: *Do we see any patterns?*

Combined effect of weather conditions and select parameters on accidents

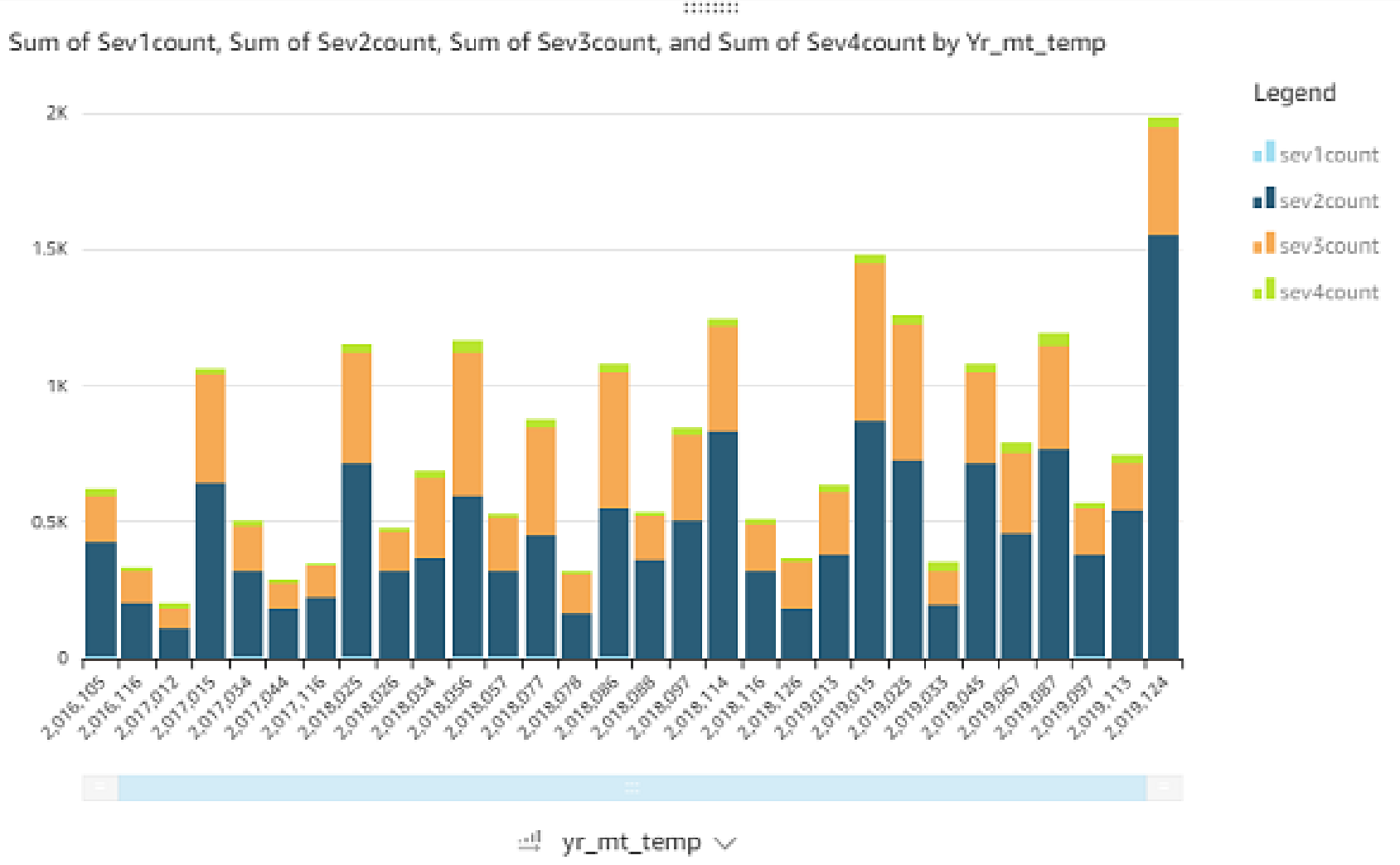


Figure 20: *Accidents are more likely to happen when the temperature is*

Word cloud: finding major theme

Figure 21: *'Local street' seems to be a major theme for accidents.*

Total average preciptation in select states

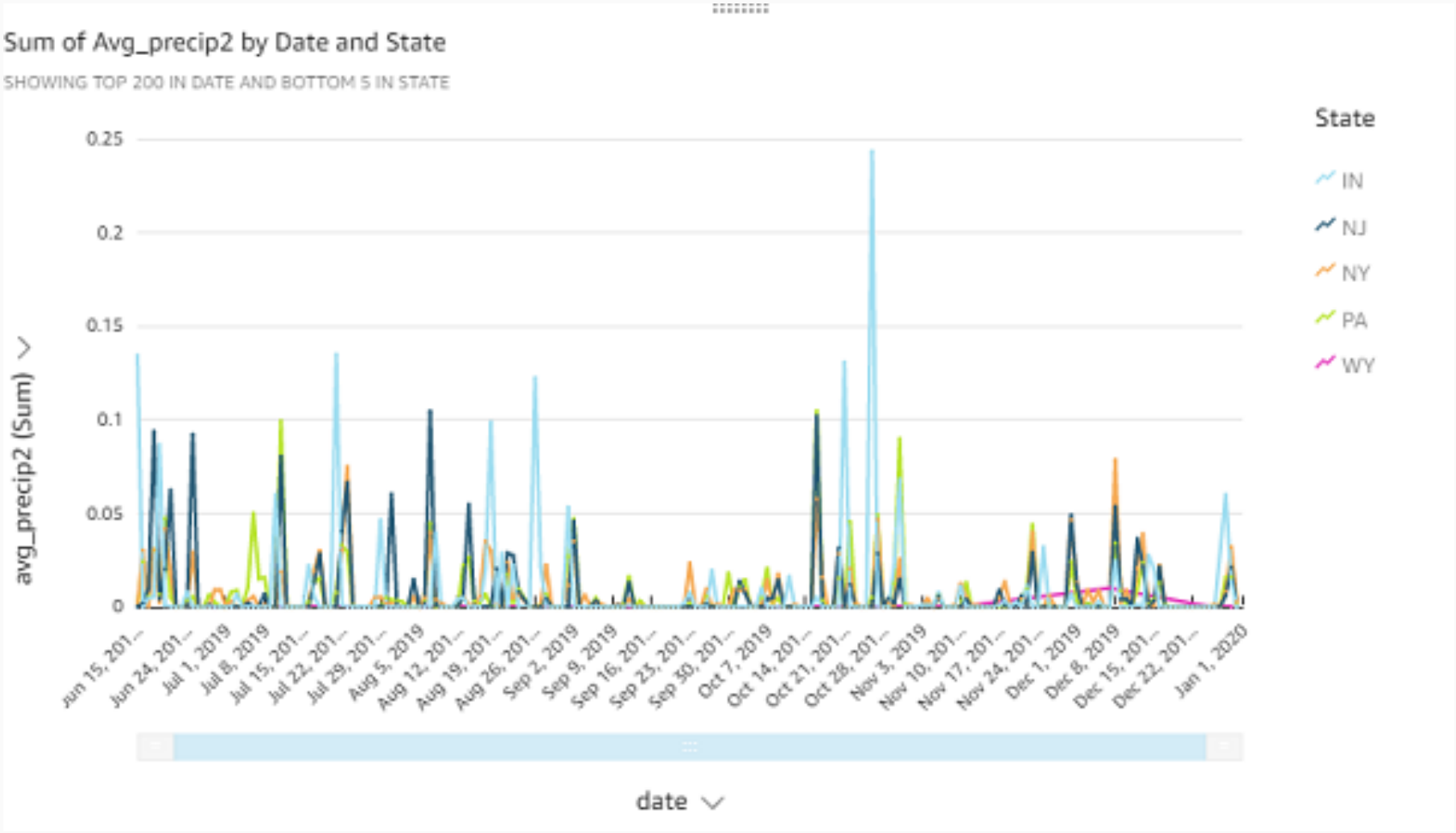


Figure 22: *Precipitation seems to follow a common theme in selected 5 states.*

Further investigation:

- How does severity affect the duration of accident?
- How would the graphs of number of accidents for different weather conditions compare if duration of weather conditions was normalized?
- Why are unusual temperatures causing more accidents in bad weather?

AWS SageMaker Introduction - Balaji

- Its a fully managed machine learning service.
- Easy to build, train and deploy models into a production-ready hosted environment.
- Integrated with SageMaker Studio and Jupyter notebook instance for easy access to your data sources for exploration and analysis
- SageMaker is flexible and provides secure and scalable environment within few clicks from SageMaker Studio.
- Training and hosting are billed by minutes of usage, with no minimum fees and no upfront commitments.

Features Of Sagemaker - Balaji

- Amazon SageMaker Studio Notebooks and Notebook Instances
  - Can build, train, deploy, and analyze your models all in the same application.
  - Sagemaker studio consists of File Manager, Terminal, Git, Metrics and Graphs sections.
  - Comes with different frameworks like Python, R, PySpark, Mxnet, TensorFlow, Pytorch, Chainer.
- Amazon SageMaker Autopilot
  - It provides automatic machine learning that can quickly build classification and regression models.
  - Users only need to provide a tabular dataset and select the target column to predict.

Features Of SageMaker Cont1- Balaji

- Amazon SageMaker Ground Truth
  - Ground Truth helps you build high-quality training datasets for your machine learning models.
  - It helps to label the dataset automatically or manually. E.g. Image, Text classification.
- Amazon SageMaker Debugger
  - Enabling the inspection of all the training parameters and data throughout the training process.
  - Use of smdebug Python library that implements the core debugging functionality.

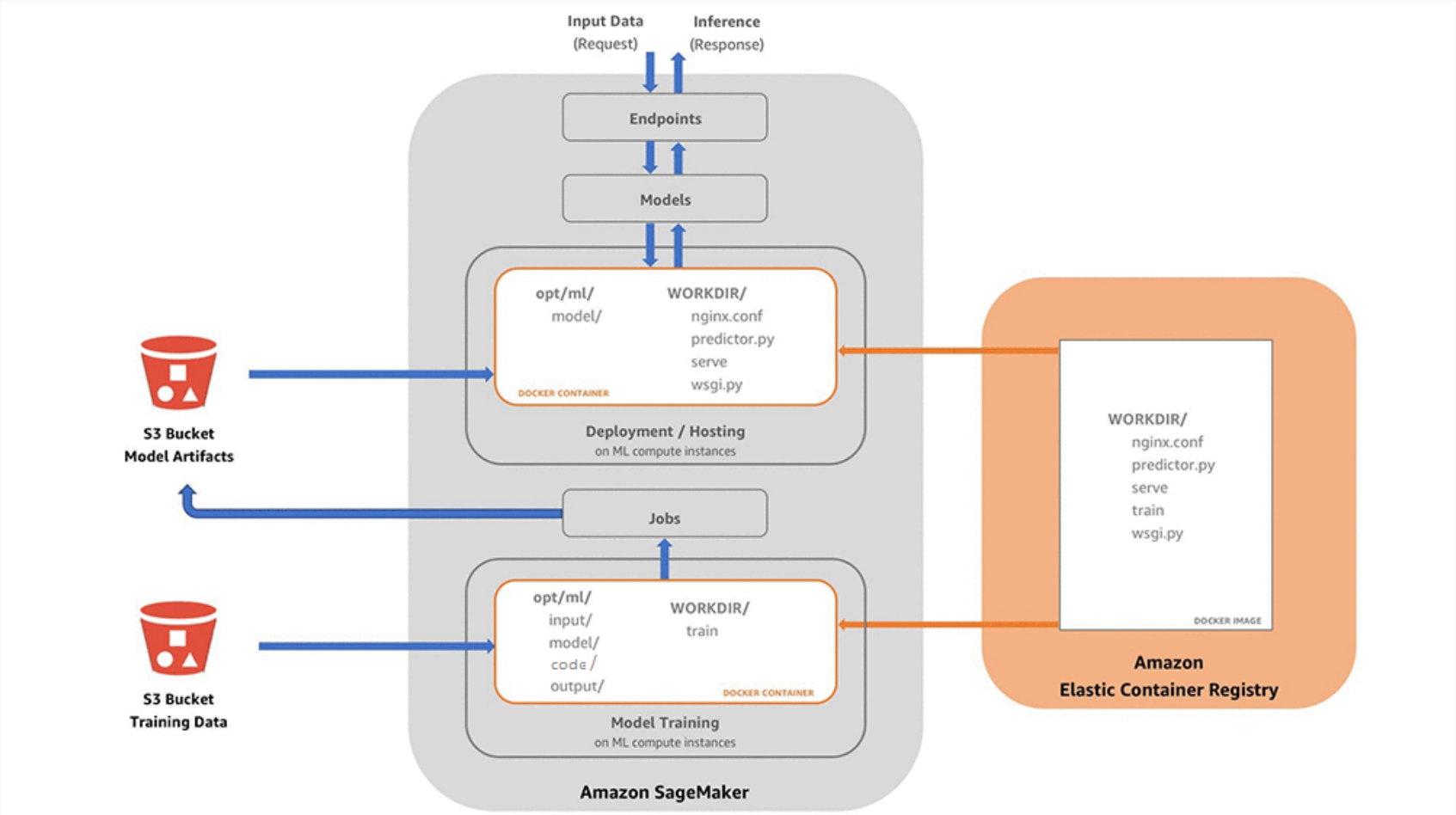
Features Of SageMaker Cont2- Balaji

- Amazon SageMaker Elastic Inference
  - Speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models.
  - It allows you to add inference acceleration to a hosted endpoint for a fraction of the cost of using a full GPU instance.
- Amazon SageMaker Model Monitor:
  - Amazon SageMaker Model Monitor is a tool for the monitoring and analysis of models in production
  - Model Monitor continuously monitors and analyzes the prediction requests. Model Monitor can store this data and use built-in statistical rules to detect common issues such as outliers in data and data drift.

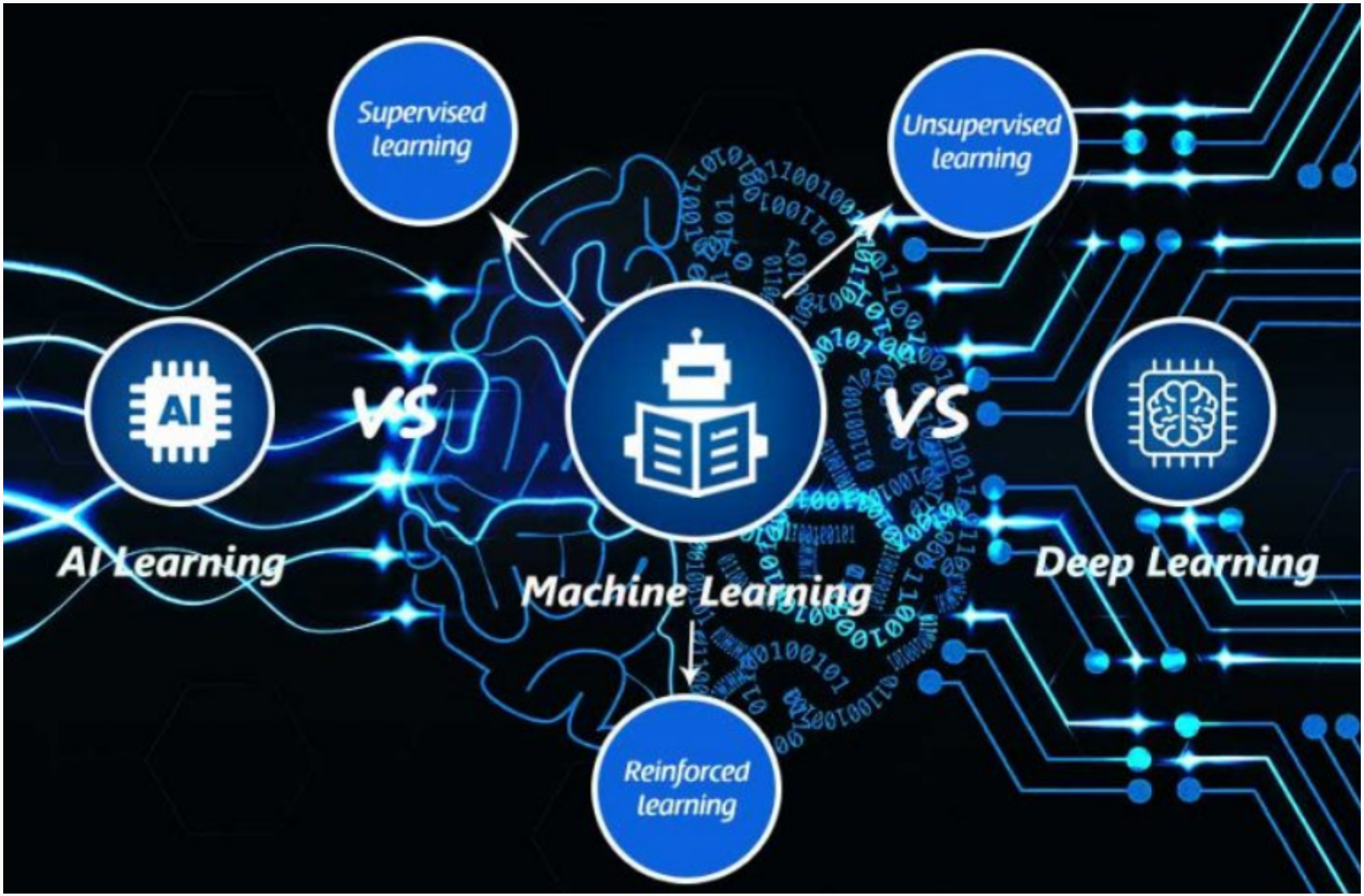
Architecture of SageMaker - Balaji

- Docker Container (Elastic Container Registry): Hosting our model for training and testing purpose.
- Model Training: Model trained once by writing code to train.py file
- Deployment Hosting: Testing and Evaluating model by using predictor.py file
- S3 (Simple Storage Service): Storing the model artifacts and dataset.
- Endpoint: Interface for data input and forwarding the data to trained model.
- Models: Trained model attached to the endpoint.

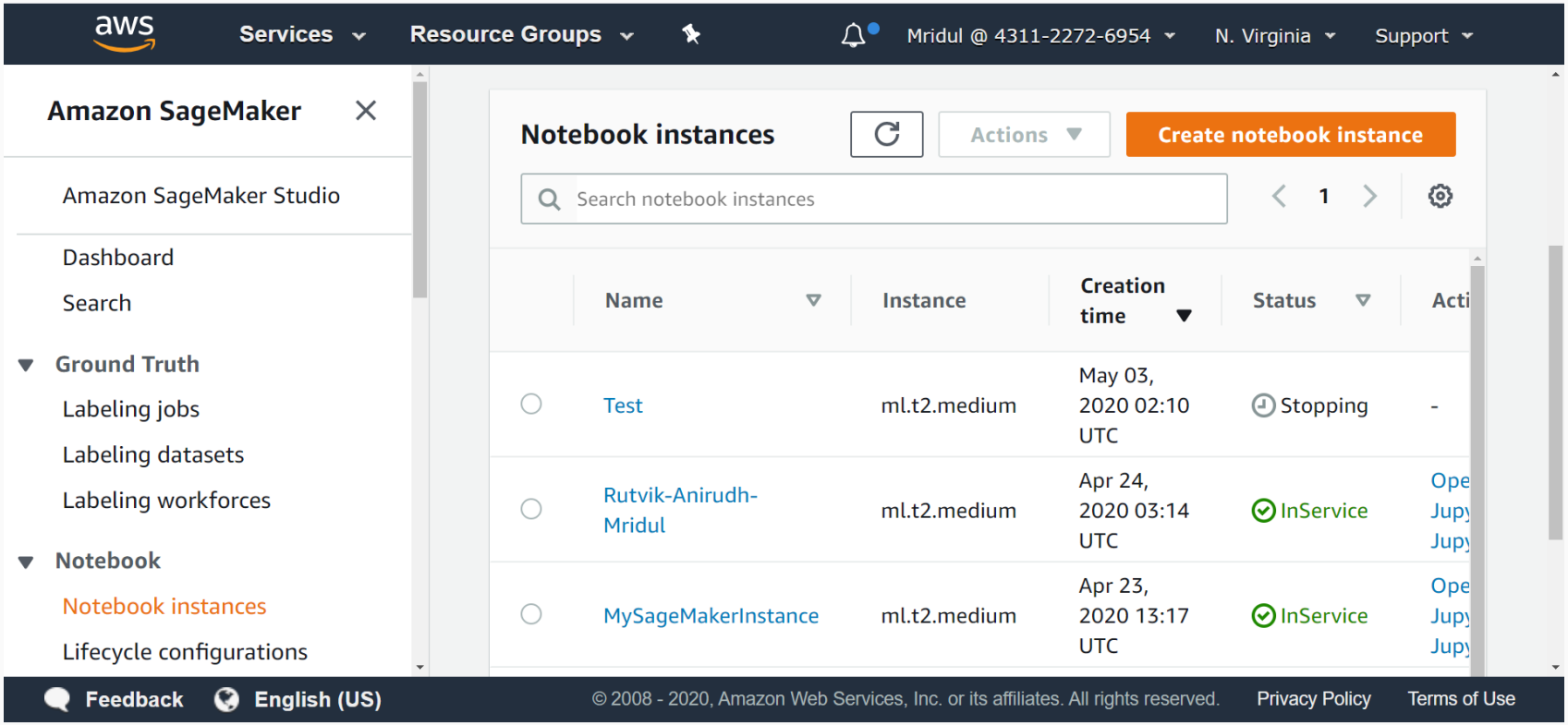
Architecture of SageMaker Contd - Balaji



Why Machine Learning



Notebook Instance - Mridul



- We can specify the notebook instance type for optimizing performance as desired

Dataset - Mridul

	date	symbol	open	close	low	high	volume
0	2016-01-05 00:00:00	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06 00:00:00	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07 00:00:00	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08 00:00:00	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11 00:00:00	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0
5	2016-01-12 00:00:00	WLTW	115.510002	115.550003	114.500000	116.059998	1098000.0

- Dataset Source: NYSE Data set from Kaggle.
- 7 columns. Our interest lies in predicting future stock price
- How we developed the predictors and response variable

Developing the Predictors

- Shift Function: dataframe.shift()

```
a1 = df['close'].shift(1) # creating column for yesterday's close price
a2 = df['close'].shift(2) # creating column for day before yesterday's close price
df = df.join(a1, rsuffix='_yesterday')
df = df.join(a2, rsuffix='_dayb4_yesterday')
#df = df[:-forecast_out]
df = df[2:] # to remove na values in new columns occured due to shift down

label = df['close'].shift(-5);#creating new column called label with the last 5 rows as nan

df1 = df.join(label, rsuffix='_frcst')


X = df1[['close', 'close_yesterday', 'close_dayb4_yesterday']]; #creating the feature array
```

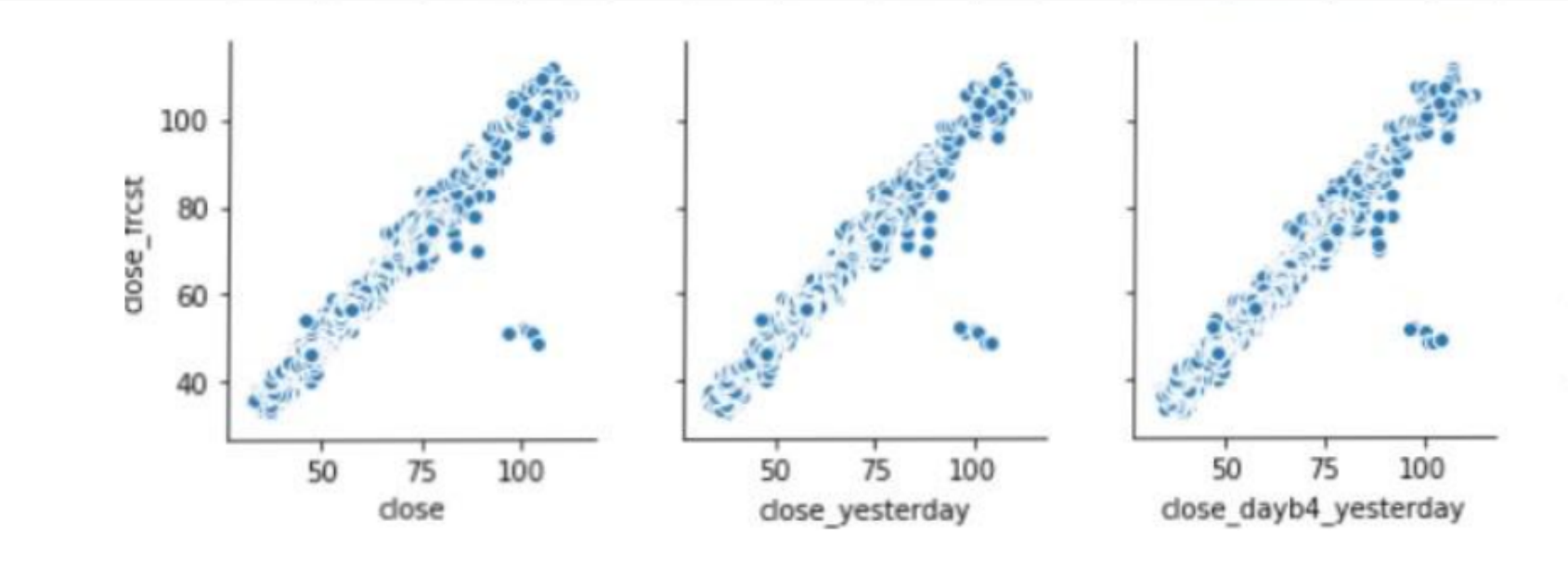
- X variables include: Today's Stock price, Yesterday's Stock Price, Day before yesterday's Stock Price
- Y variable: 5 days down Stock price

Final Dataset for Machine Learning

	date	close	close_yesterday	close_dayb4_yesterday	close_frcst
1292	2010-01-06	24.420000	24.580000	24.690001	24.639999
1760	2010-01-07	24.530001	24.420000	24.580000	24.950001
2228	2010-01-08	24.660000	24.530001	24.420000	24.400000
2696	2010-01-11	24.590000	24.660000	24.530001	24.850000
3164	2010-01-12	24.200001	24.590000	24.660000	24.410000
3632	2010-01-13	24.639999	24.200001	24.590000	23.980000
4100	2010-01-14	24.950001	24.639999	24.200001	22.969999
4568	2010-01-15	24.400000	24.950001	24.639999	22.990000
5036	2010-01-19	24.850000	24.400000	24.950001	22.900000
5504	2010-01-20	24.410000	24.850000	24.400000	23.150000

- Feature Explanation
- What are we predicting

Scatter plot for X variables with respect to Y variable



Machine Learning Models - Linear Regression

Linear Regression Model

n [14]:
```
learner = linear_model.LinearRegression(); #initializing linear regression m

learner.fit(X_train,Y_train); #training the linear regression model
y_predict =  learner.predict(X_test)

score=learner.score(X_test,Y_test);#testing the linear regression model
```

n [15]:
```
#Score
print('Score: ',score)

#MSE
print('MSE: ',mean_squared_error(Y_test, y_predict))
```

```
Score:  0.9859633934602169
MSE:  0.6417789507166864
```

Machine Learning Models - Extreme Gradient Boosting Regression (XGBR)

Extreme Gradient Boosting Regression (XGBR)

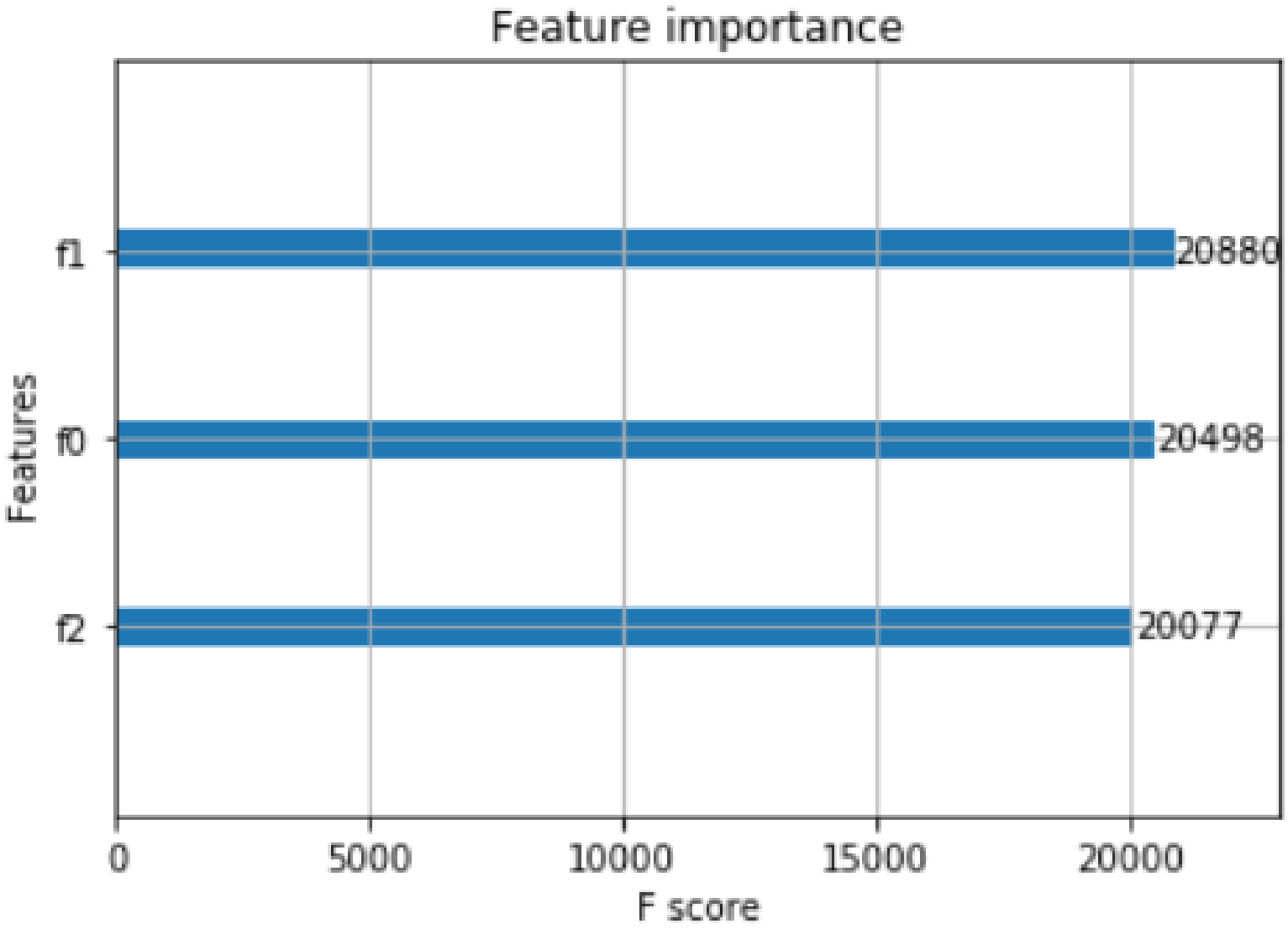
In [32]:

```
import xgboost
#import xgboost as xgb
xgbr = xgboost.XGBRegressor(colsample_bytree=0.4,
                 gamma=0,
                 learning_rate=0.07,
                 max_depth=3,
                 min_child_weight=1.5,
                 n_estimators=10000,
                 reg_alpha=0.75,
                 reg_lambda=0.45,
                 subsample=0.6,
                 seed=42)

#xgb_clf = XGBClassifier()

xgbr.fit(X_train, Y_train)

score = xgbr.score(X_test, Y_test)
print('Score: ',score)
mse1 = mean_squared_error(Y_test, xgbr.predict(X_test))
print('MSE: ',mse1)
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/distributed/config.py:63: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default Loader is unsafe. Please read https://msg.pyyaml.org/load for full details.
  config.update(yaml.load(text) or {})
```

```
Score:  0.9787715356470047
MSE:  1.0510065196673972
```

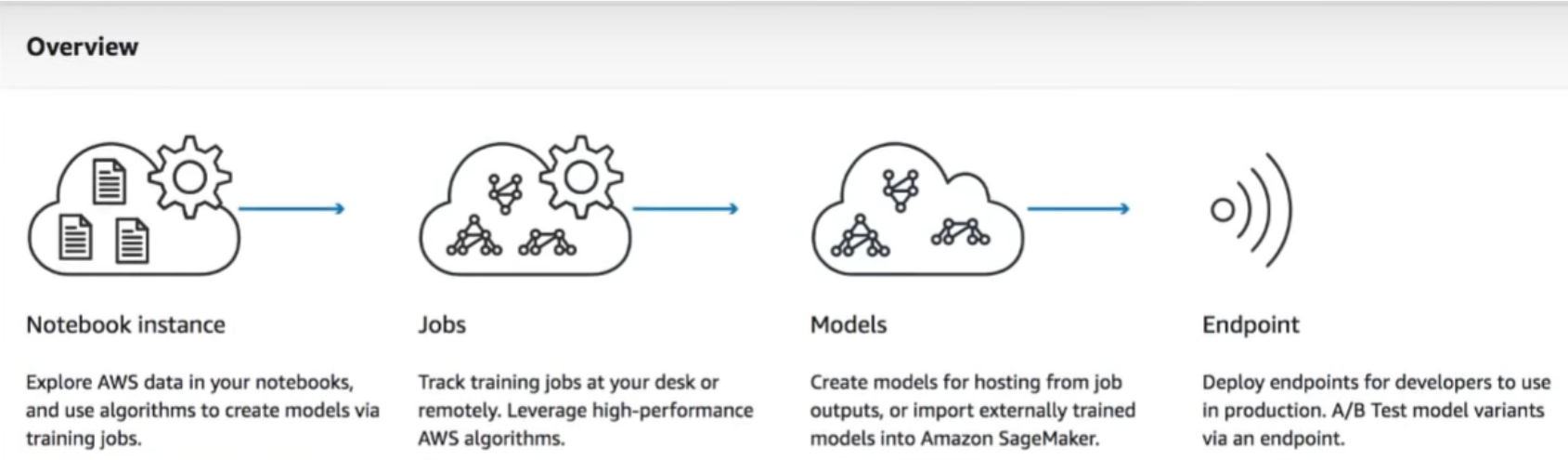
Feature Importance graph with XGBR



Importance of using SageMaker for Training Model

- This was demo for training a model using Jupyter Notebook.
- The advantage is the Optimized performance and Deployment of model from anywhere
- The main advantange lies in hosting any number of models to amazon SageMaker behind endpoint

Hosting a model in AWS SageMaker

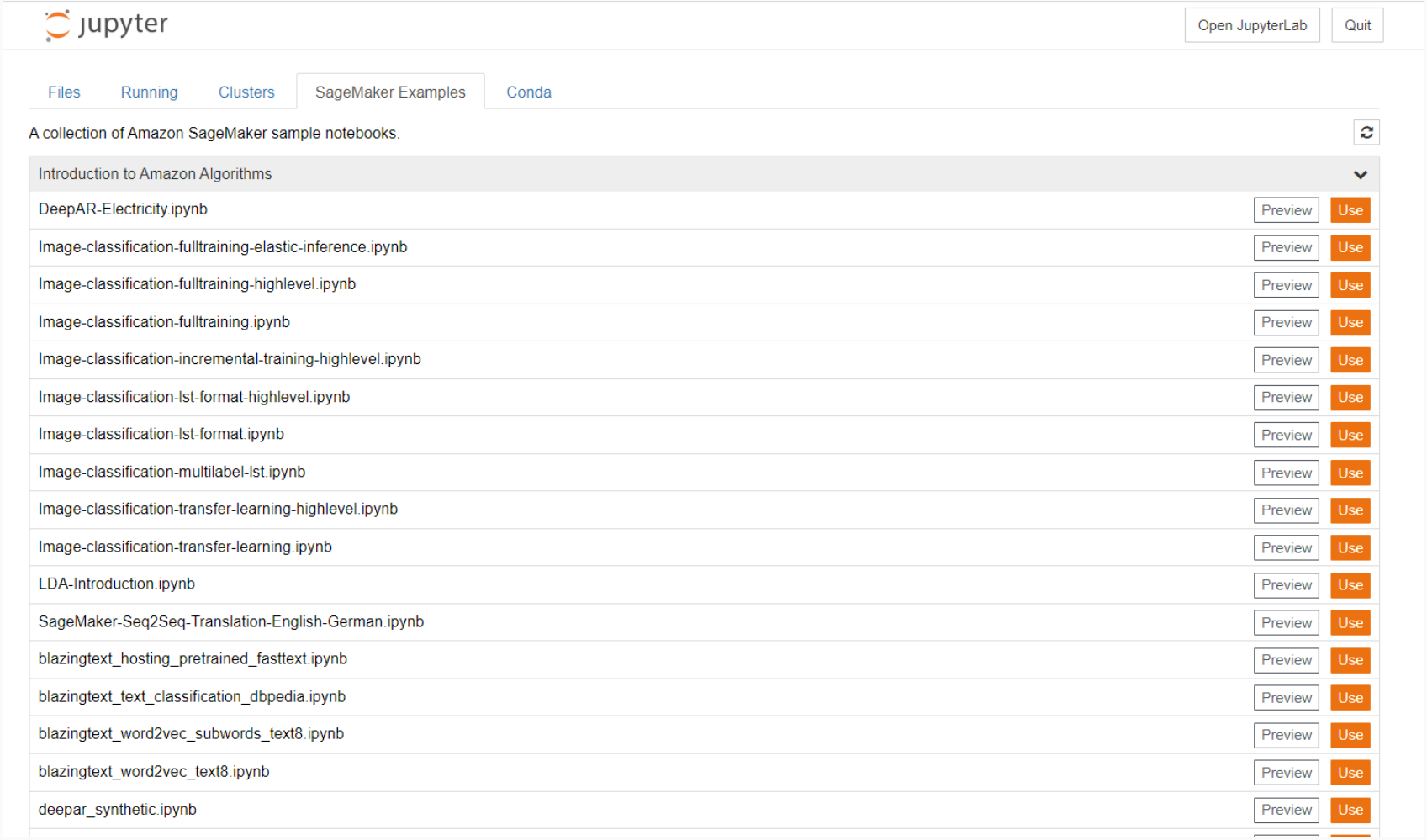


- Model can be trained using notebook instance and stored as a training job.
- This trained job/model can be imported to the container for hosting the model to SageMaker
- Configure the Endpoints that is mentioning instance type, no of models etc.
- Deploy the model behind the endpoint
- All the code for training jobs, hosting, endpoint creation is already present in AWS as Example

Advantages

- Less time consuming and higher performance.
- Very efficient for performing A/B Testing
- Takes 10 to 20 minutes for hosting multiple models which normally would take hours of coding

SageMaker inbuilt Examples for hosting Models



Example Sample

Regression with Amazon SageMaker XGBoost algorithm

Single machine training for regression with Amazon SageMaker XGBoost algorithm

---

Contents

Introduction

This notebook demonstrates the use of Amazon SageMaker's implementation of the XGBoost algorithm to train and host a regression model. We use the Abalone data originally from the UCI data repository. More details about the original dataset can be found here. In the libsvm converted version, the nominal feature (Male/Female/Infant) has been converted into a real valued feature. Age of abalone is to be predicted from eight physical measurements.

Setup

This notebook was created and tested on an ml.m4.4xlarge notebook instance.

Code Sample

Setup

This notebook was created and tested on an ml.m4.4xlarge notebook instance.

Let's start by specifying:

1. The S3 bucket and prefix that you want to use for training and model data. This should be within the same region as the Notebook Instance, training, and hosting.
2. The IAM role arn used to give training and hosting access to your data. See the documentation for how to create these. Note, if more than one role is required for notebook instances, training, and/or hosting, please replace the boto regexp with a the appropriate full IAM role arn string(s).

In [ ]:

```
%%time

import os
import boto3
import re
import sagemaker

role = sagemaker.get_execution_role()
region = boto3.Session().region_name

# S3 bucket for saving code and model artifacts.
# Feel free to specify a different bucket and prefix
bucket = sagemaker.Session().default_bucket()
prefix = 'sagemaker/DEMO-xgboost-abalone-default'
# customize to your bucket where you have stored the data
bucket_path = 'https://s3-{}.amazonaws.com/{}'.format(region, bucket)
```

Code Sample

In [ ]:
```
from sagemaker.amazon.amazon_estimator import get_image_uri
container = get_image_uri(region, 'xgboost', '0.90-1')
```

In [ ]:
```
%%time
import boto3
from time import gmtime, strftime

job_name = 'DEMO-xgboost-regression-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Training job", job_name)

#Ensure that the training and validation data folders generated above are reflected in the "InputDataConfig" parameter below.

create_training_params = \
{
    "AlgorithmSpecification": {
        "TrainingImage": container,
        "TrainingInputMode": "File"
    },
    "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": bucket_path + "/" + prefix + "/single-xgboost"
    },
    "ResourceConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.m5.2xlarge",
        "VolumeSizeInGB": 5
    },
    "TrainingJobName": job_name,
    "HyperParameters": {
        "max_depth":"5",
        "eta":"0.2",
        "gamma":"4",
        "min_child_weight":"6",
        "subsample":"0.7",
        "silent":"0",
        "objective":"reg:linear",
        "num_round":"50"
    },
```

Code Sample

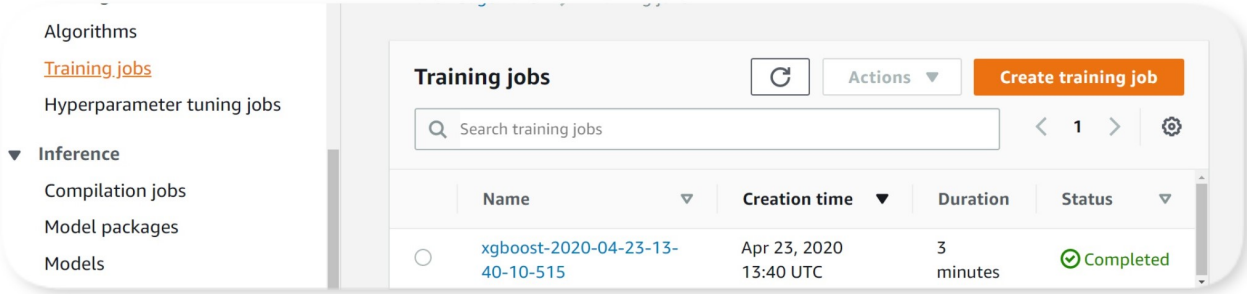
Create endpoint

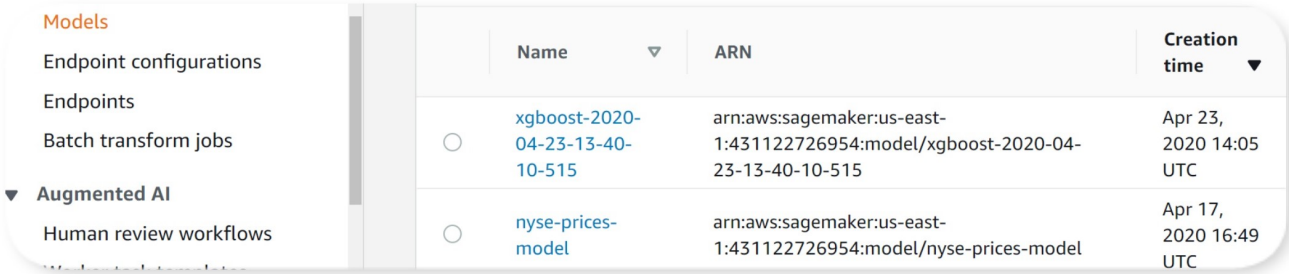
Lastly, the customer creates the endpoint that serves up the model, through specifying the name and configuration defined above. The end result is an endpoint that can be validated and incorporated into production applications. This takes 9-11 minutes to complete.

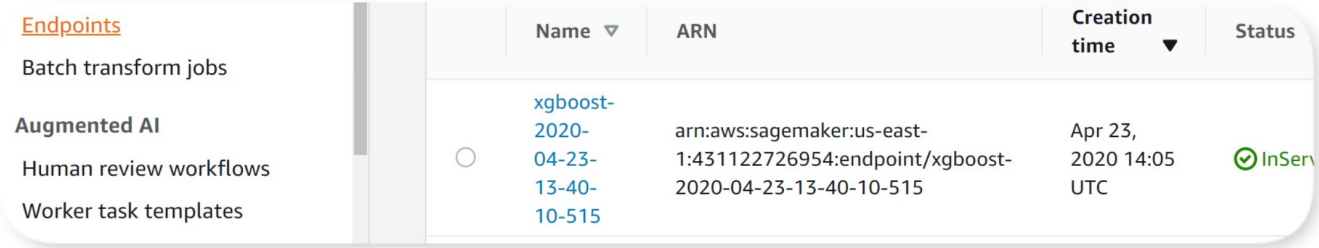
In [ ]:

```
%%time
import time

endpoint_name = 'DEMO-XGBoostEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_name)
create_endpoint_response = client.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
print(create_endpoint_response['EndpointArn'])

resp = client.describe_endpoint(EndpointName=endpoint_name)
status = resp['EndpointStatus']
while status=='Creating':
    print("Status: " + status)
    time.sleep(60)
    resp = client.describe_endpoint(EndpointName=endpoint_name)
    status = resp['EndpointStatus']

print("Arn: " + resp['EndpointArn'])
print("Status: " + status)
```

Dashboard


Algorithms
Training jobs
Hyperparameter tuning jobs
Inference
Compilation jobs
Model packages
Models
Training jobs
Actions
Create training job
Search training jobs
1
Name
Creation time
Duration
Status
xgboost-2020-04-23-13-40-10-515
Apr 23, 2020 13:40 UTC
3 minutes
Completed


Models
Endpoint configurations
Endpoints
Batch transform jobs
Augmented AI
Human review workflows
Name
ARN
Creation time
xgboost-2020-04-23-13-40-10-515
arn:aws:sagemaker:us-east-1:431122726954:model/xgboost-2020-04-23-13-40-10-515
Apr 23, 2020 14:05 UTC
nyse-prices-model
arn:aws:sagemaker:us-east-1:431122726954:model/nyse-prices-model
Apr 17, 2020 16:49 UTC


Endpoints
Batch transform jobs
Augmented AI
Human review workflows
Worker task templates
Name
ARN
Creation time
Status
xgboost-2020-04-23-13-40-10-515
arn:aws:sagemaker:us-east-1:431122726954:endpoint/xgboost-2020-04-23-13-40-10-515
Apr 23, 2020 14:05 UTC
InServ

Conclusion

- Summary of how the mLogica and IIT teams working together
- Questions?