

ABD403

# Best Practices for Distributed Machine Learning & Predictive Analytics on AWS

Keith Steward, Ph.D.  
Specialist SA, AWS

AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This session, we focus on common use cases and design patterns for predictive analytics using Amazon EMR. We address accessing data from a data lake, extraction and preprocessing with Apache Spark, analytics and machine learning code development with notebooks (Jupyter, Zeppelin), and data visualization using Amazon QuickSight. We cover other operational topics, such as deployment patterns for ad hoc exploration and batch workloads using Spot and multi-user notebooks. The intended audience for this session includes technical users who are building statistical and data analytics models for the business using tools, such as Python, R, Spark, Presto, Amazon EMR, Notebooks

## What we'll cover:

1. **Predictive Analytics: What is it? Why is it important?**
2. **Predictive Analytics: How can you do it *at scale* and *with agility* on AWS?**
  - How to do a *Data Lake* with Amazon S3 and AWS Glue?
  - How to Train a *Predictive Machine Learning Model* in a *distributed* fashion, using Amazon EMR with Apache Spark ML + Apache Zeppelin?
  - How to perform *distributed SQL queries* on big data using Amazon Athena?
  - How to *visualize* data using Amazon QuickSight?
3. **Demo: *Use predictive analytics to identify target customers.***

AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

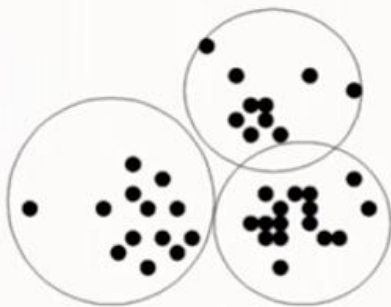


# 1. Predictive Analytics:

What is it?

Why is it important?

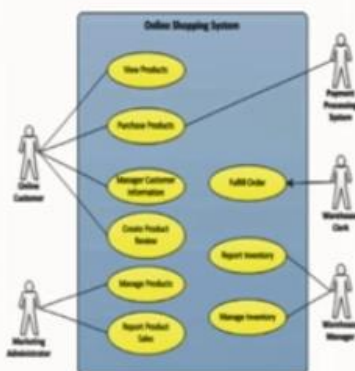
## Wikipedia Definitions:



**Analytics:** the discovery, interpretation, and communication of meaningful patterns in data.

**Predictive Analytics:** encompasses a variety of statistical techniques from predictive modeling, machine learning, and data mining that analyze current and historical facts to make predictions about future or otherwise unknown events.

## Predictive Analytics Use Cases:



### Customer predictions

- "Which customers are likely to be the most profitable?"
- "How much revenue should I expect this customer to generate?"
- "Which customers are likely to churn?"
- "Among all of our customers, which are likely to respond to a given offer?"
- "What's the probability that a given customer X will respond to a given offer Y?"

### Product or Service predictions

- "What products should we offer or develop?"
- "What items are likely to be purchased together?" (basket analysis)

### Business Operations predictions

- "Are the metrics for service X nominal or anomalous?"
- "Is equipment X likely to fail within time Y?"



## Why is Predictive Analytics Important?

- Companies have been accumulating "Big Data" about customers, products/services, and operations for many years now.
- Big Data technologies have provided proven solutions to address this *data management* need.
- But... Increasing pressure to turn data into insights about trends, classifications, detect anomalies, and provide feedback loops to improve their business



## Why is Predictive Analytics Important?

- Desire to evolve from backwards-looking monthly or quarterly reports to real-time alerts, and now to predicting the future

Business'  
Evolving  
Needs for  
Data



**AWS re:Invent**

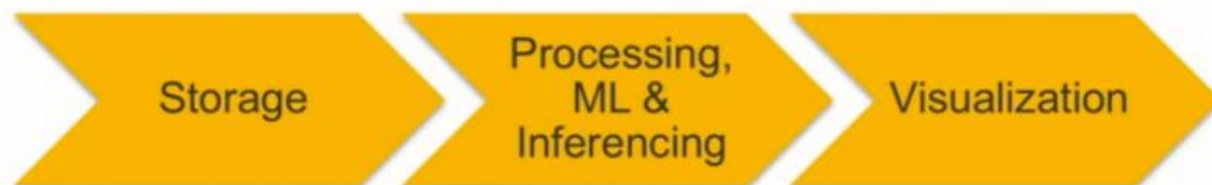
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



2. Predictive Analytics:  
How can you do it at *scale*  
and with *agility* on AWS?

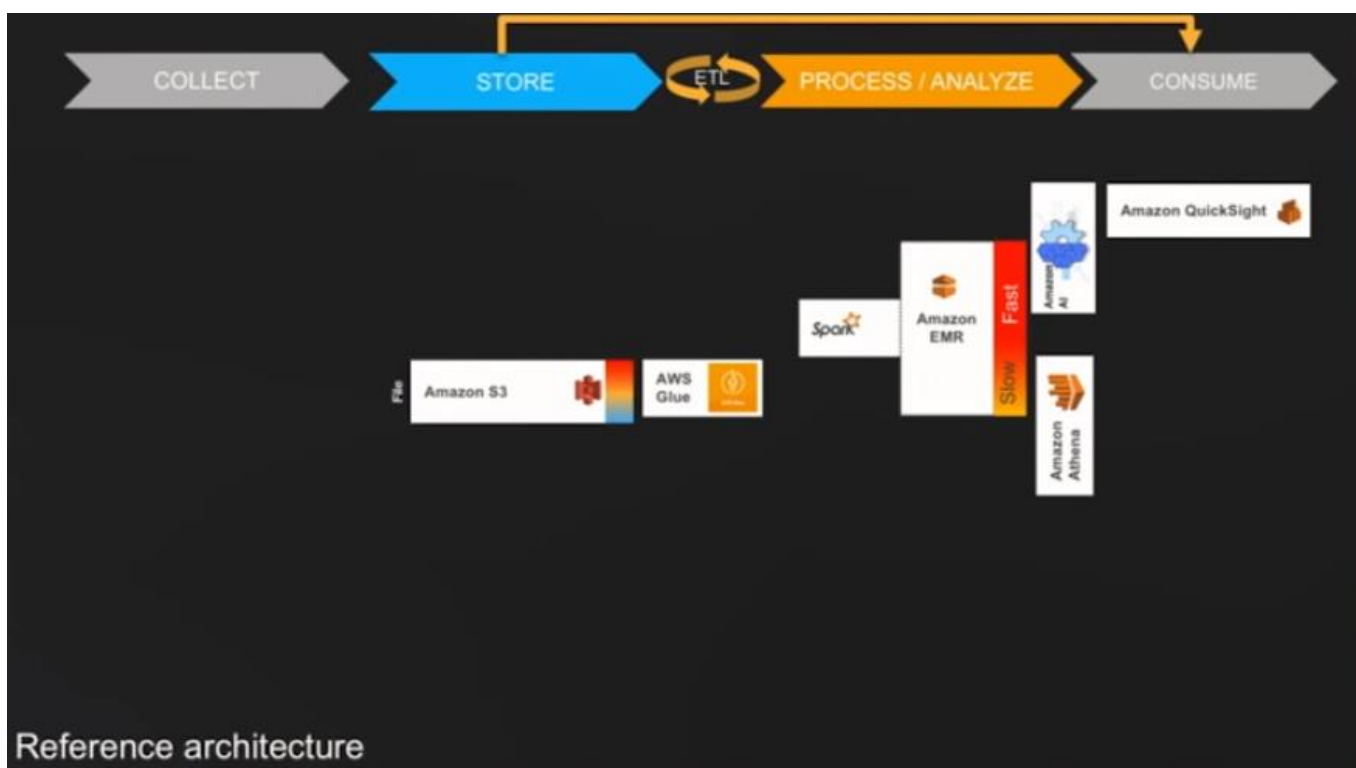
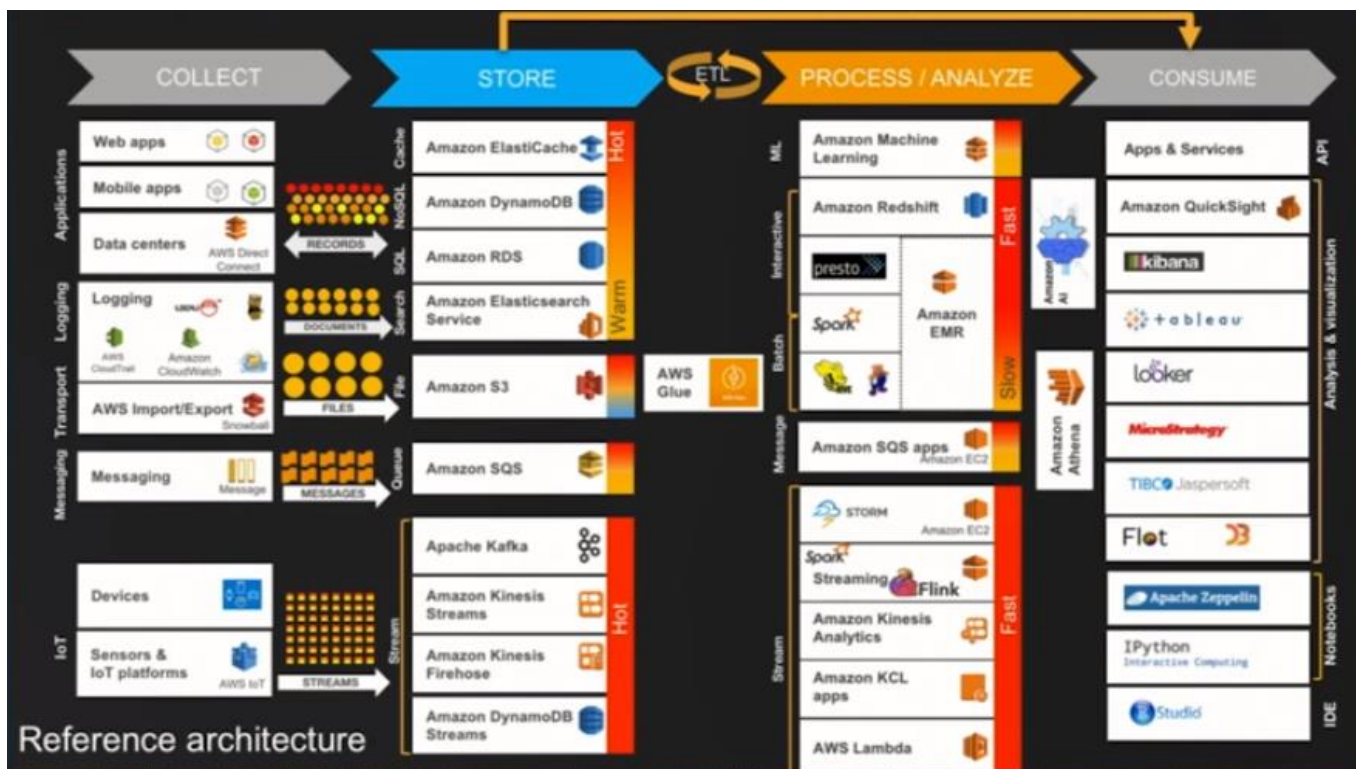


# Predictive Analytics Architecture



## Plethora of Tools







**Store anything (object storage)**

**Scalable / Elastic**

**99.999999999% durability**

**Effectively infinite inbound bandwidth**

**Extremely low cost: \$0.023/GB-Mo; \$23/TB-Mo**

**Data layer for virtually all AWS services**

## Why Amazon S3 for data lake?



### **Durable**

Designed for 11 9s  
of durability



### **Available**

Designed for  
**99.99%** availability



### **High performance**

- Multiple upload
- Range GET



### **Easy to use**

- Simple REST API
- AWS SDKs
- Read-after-create consistency
- Event notification
- Lifecycle policies



### **Scalable**

- Store as much as you need
- Scale storage and compute independently
- No minimum usage commitments



### **Integrated**

- Amazon EMR
- Amazon Redshift
- Amazon DynamoDB



AWS Glue

**Data Catalog**

**Managed ETL Engine**

**Job Scheduler**

**Built on Apache Spark**

**Integrated with S3, RDS, Redshift**

**Integrates with any JDBC data store**

QuickSight



**Glue data catalog**

Discover and organize your data sets

## Glue data catalog

Manage **table metadata** through a **Hive metastore API** or **Hive SQL**. Supported by tools such as Hive, Presto, Spark, etc.

We added a few extensions:

- **Search** metadata for data discovery
- **Connection info** – JDBC URLs, credentials
- **Classification** for identifying and parsing files
- **Versioning** of table metadata as schemas evolve and other metadata are updated

Populate using Hive DDL, bulk import, or automatically through **crawlers**.

Name	Database	Location	Classification
csv2	incomes3_db	s3://incomes-m/csv2/	csv
flightscsv	flights-db	s3://crawler-public-us-east-1/flight/...	csv
income_new_csv	incomes_db2	s3://incomes-m/csv/income_new.csv	csv
incomes_data	incomes_db2	s3://incomes-m/csv/incomes.data	csv
airflight_table2par	airflights-db1	s3://airdelays-klb/par/	parquet
par	flightdelays-db1	s3://airdelays-klb/par/	parquet
parquet	flight-delays3-db	s3://flightdelays-klb/parquet/	parquet
baltimore	incidents	s3://datasearch-blog2-jupyterpark-...	Unknown
baltimore_dict	incidents	s3://datasearch-blog2-jupyterpark-...	Unknown
detroit	incidents	s3://datasearch-blog-jupyterpark-b...	Unknown
detroit_dict	incidents	s3://datasearch-blog-jupyterpark-b...	Unknown

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

## Crawlers: auto-populate data catalog

Automatic schema inference:

- Built-in **classifiers** detect file type and **extract schema**: record structure and data types.
- Add your own or share with others in the Glue community - It's all Grok and Python.

Auto-detects Hive-style partitions, grouping similar files into one table.

Run crawlers on schedule to **discover** new data and **schema changes**.

Serverless – only pay when crawls run.

Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
flight-data...		Ready	Logs	27 secs	27 secs	0	1
flightdelays		Ready	Logs	26 secs	26 secs	0	1
flights-ors...		Ready	Logs	24 secs	24 secs	0	1
flights-test...		Ready	Logs	19 secs	19 secs	0	1
incomes2		Ready	Logs	25 secs	25 secs	0	1
incomes_...		Ready	Logs	15 secs	15 secs	0	1
incomes_...		Ready	Logs	20 secs	20 secs	0	0

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.





Scalable Hadoop clusters as a service

Hadoop, Hive, Spark, Presto, Hbase, etc.

Easy to use; fully managed

On demand, reserved, spot pricing

HDFS, S3, and Amazon EBS filesystems

Integrates with AWS Glue

End to end security



## EMR: Just some of the organizations using



# Amazon EMR / Spark ML offers these Models:

- **Classification**
  - Logistic regression
  - Decision tree classifier
  - Random forest classifier
  - Gradient-boosted tree classifier
  - Multilayer perceptron classifier
  - One-vs-Rest classifier (a.k.a. One-vs-All)
  - Naive Bayes
- **Regression**
  - Linear regression
  - Generalized linear regression
  - Decision tree regression
  - Random forest regression
  - Gradient-boosted tree regression
  - Survival regression
  - Isotonic regression

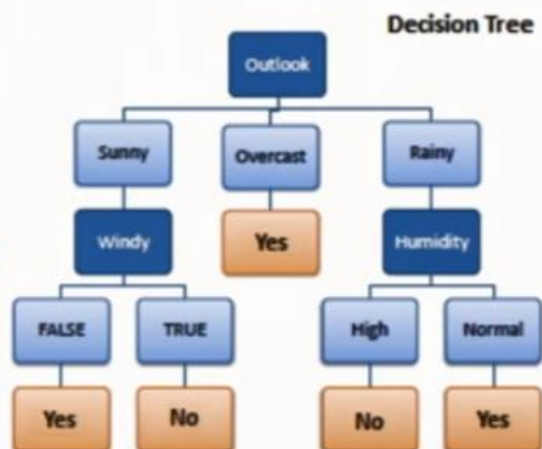


**SparkML** allows us to implement a pipeline using data frame transformations, we can persist the pipeline and reuse models that were earlier trained

## What are Decision Trees?

Weather predictors for Golf

Predictors				Target
Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



It builds a decision tree in-memory for the prediction problem

# Decision Trees: Training Data

Attributes				Target attribute
Name	Balance	Age	Employed	Write-off
Mike	\$200,000	42	no	yes
Mary	\$35,000	33	yes	no
Claudio	\$115,000	40	no	no
Robert	\$29,000	23	yes	yes
Dora	\$72,000	31	no	no

This is one row (example).  
Feature vector is: **<Claudio,115000,40,no>**  
Class label (value of Target attribute) is **no**

Figure 3-1. Data mining terminology for a supervised classification problem. The problem is supervised because it has a target attribute and some "training" data where we know the value for the target attribute. It is a classification (rather than regression) problem because the target is a category (yes or no) rather than a number.

Bank loan  
write-off  
predictions



## Fast Business Analytics / Data Visualization Service running in AWS

Access multiple AWS data sources (S3, Athena, Aurora, Redshift, etc) or data you upload

Perform ad hoc analyses to gain insights from your data

Supports 100's of thousands of users

Smart visualizations dynamically optimized for your data

Generate fast, interactive visualizations on large datasets



Amazon QuickSight is a Business Analytics Service that lets business users quickly and easily visualize, explore, and share insights from their data.



## QuickSight Overview

### Integrated with AWS

Redshift, RDS, Athena, S3, IAM, Roles, etc...

### Cloud Native

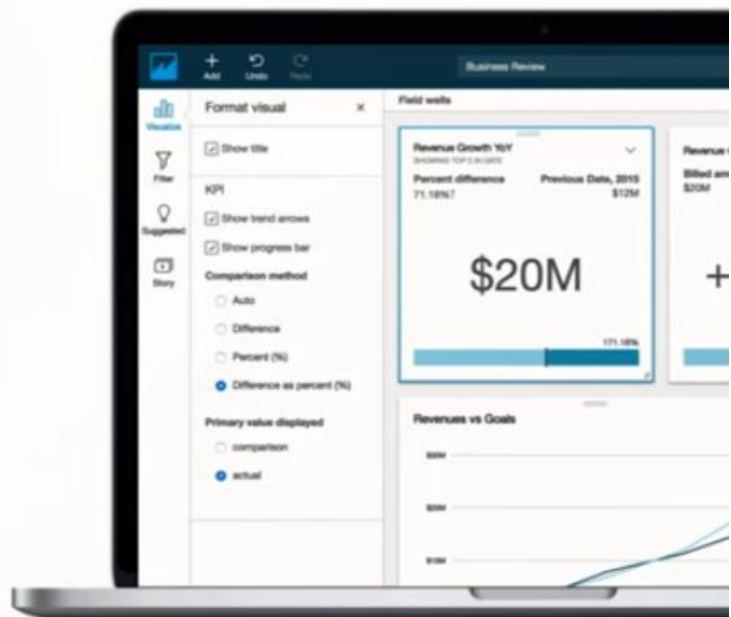
Fully managed, serverless analytics at scale

### Super Fast and Easy to Use

Backed by SPICE and a beautiful UI

### Cost Effective

Starts at \$9 per user per month





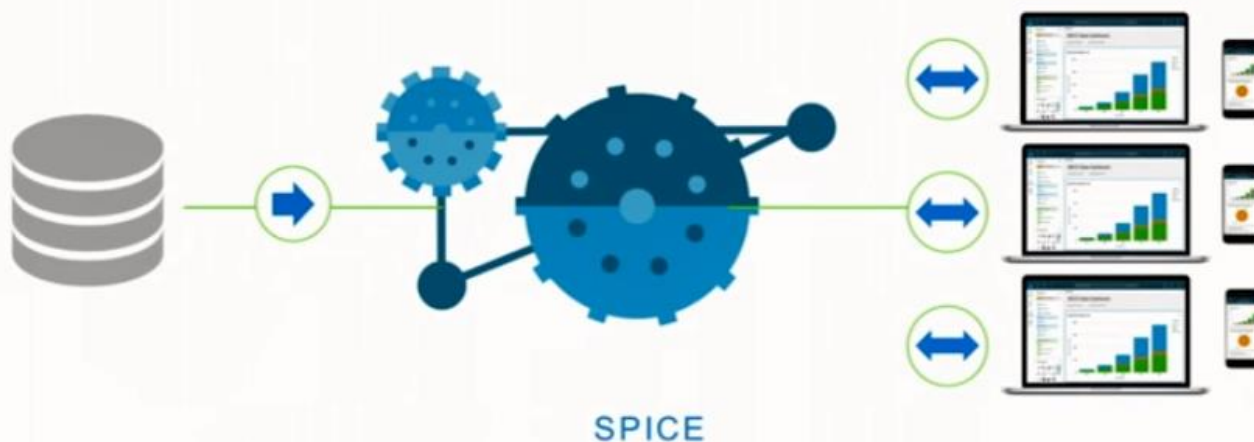
## Deep Integration with AWS Data Sources

QuickSight is **deeply integrated** with AWS data sources like Redshift, RDS, S3, Athena and others, as well as third-party sources like Excel, Salesforce, as well as on-premises databases.



## SPICE

QuickSight is powered by **SPICE**, a super-fast calculation engine that delivers unprecedented performance and scale, delivering insights at the speed of thought.



### 3. Demo:

*Use predictive analytics to identify target customers.*

## Our Scenario: We are LuxCars, Inc.

### The Challenge:

- ☐ We're bringing an expensive new luxury car (the "*Luxmobile*") to market.
- ☐ Want to focus marketing on customers likely to be *wealthy* enough to afford our luxury automobile.
- ☐ Have demographic data for our target geography, but we don't have income data.
- ☐ Need to *predict* salary from the demographic data.

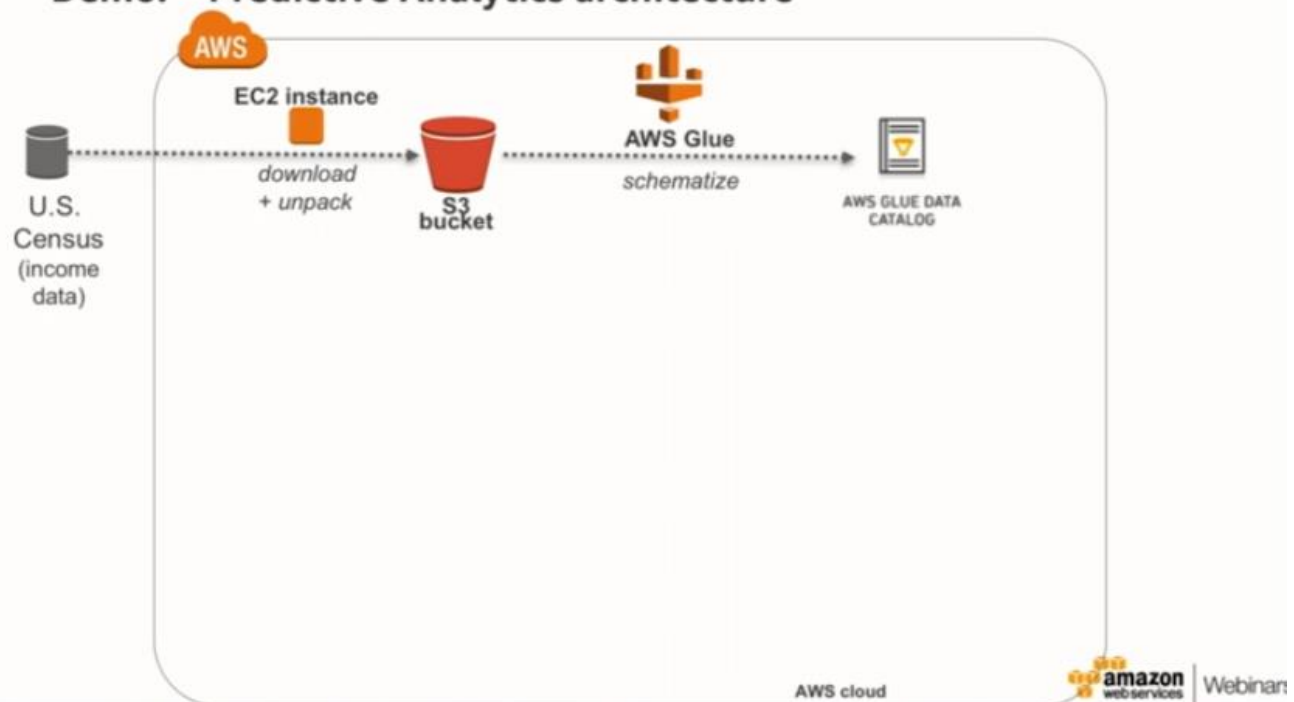
# Our Scenario: We are LuxCars Inc.

## Strategy:

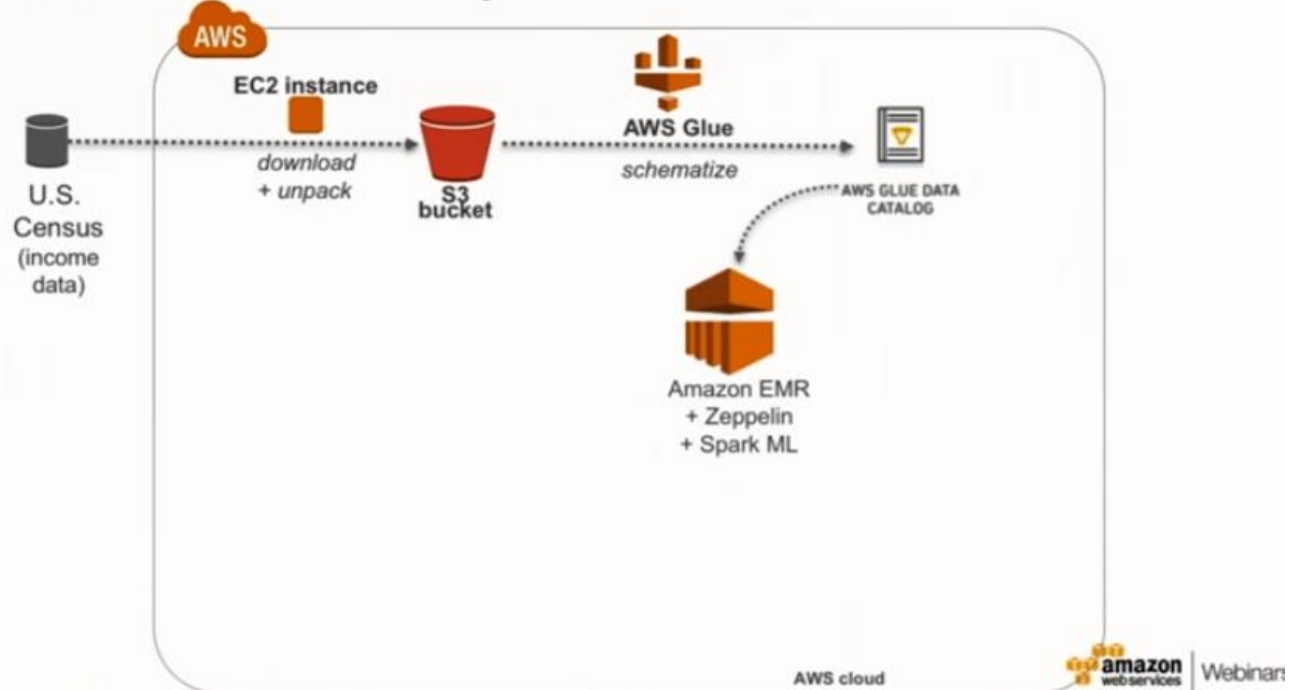
1. Collect U.S. Census Income data & put into the Cloud.
2. *Schematize* the data.
3. Train a Spark ML model on the data.
4. Do batch predictions of income for people in the target geography.
5. Compute mean incomes per sub-geography (“income density”)
6. Visualize those predictions.
7. Share results with our Marketing Director.

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

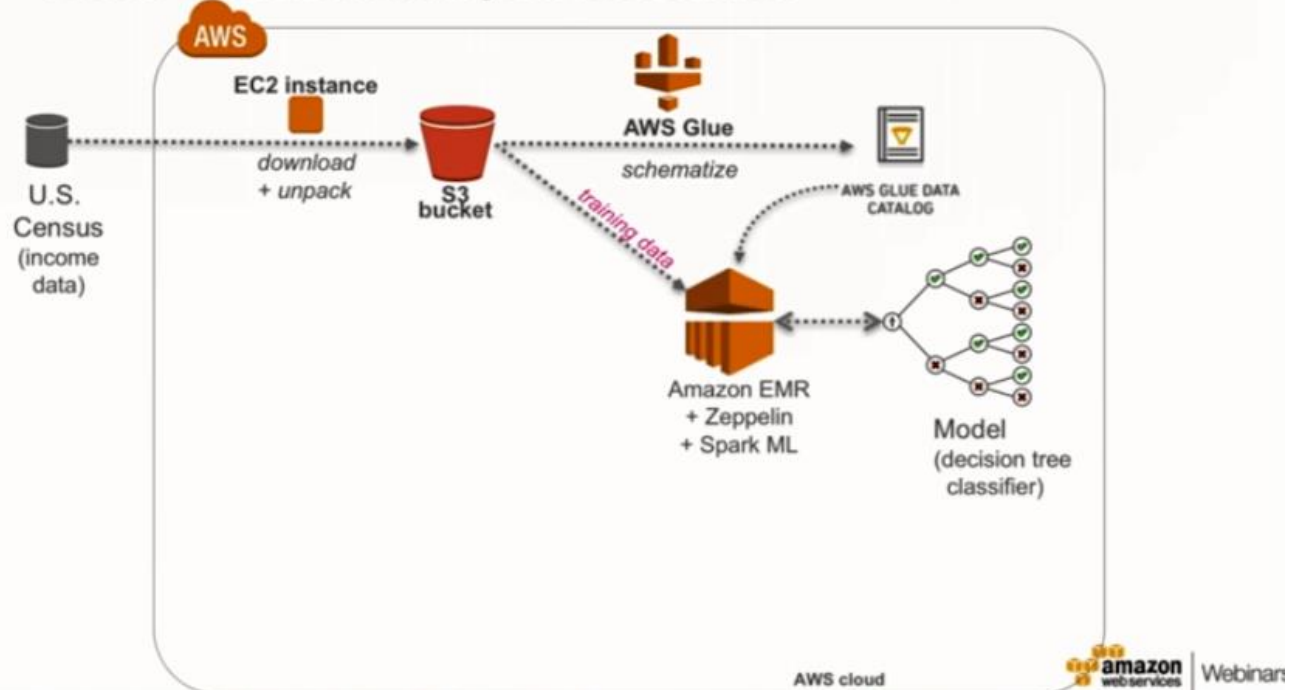
## Demo: Predictive Analytics architecture



## Demo: Predictive Analytics architecture

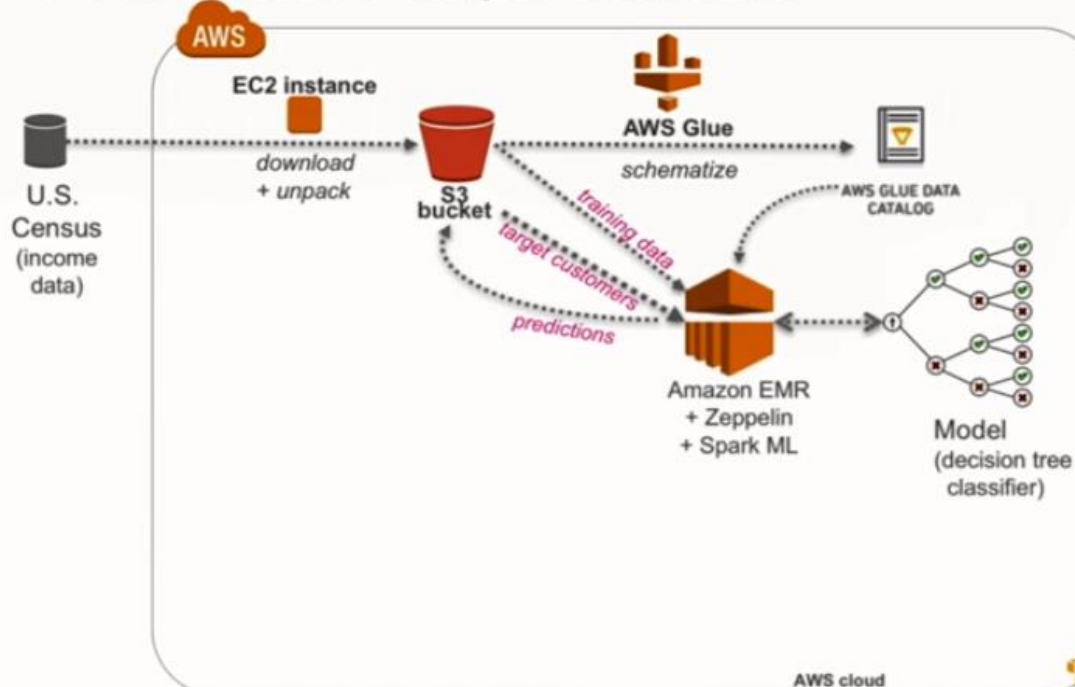


## Demo: Predictive Analytics architecture



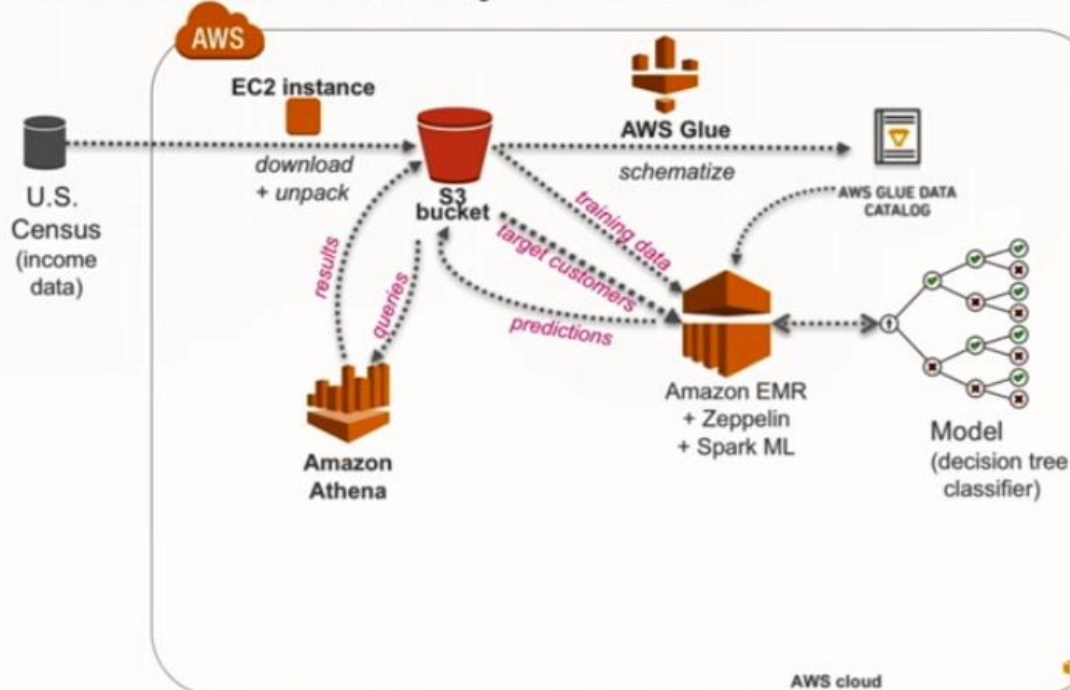


## Demo: Predictive Analytics architecture



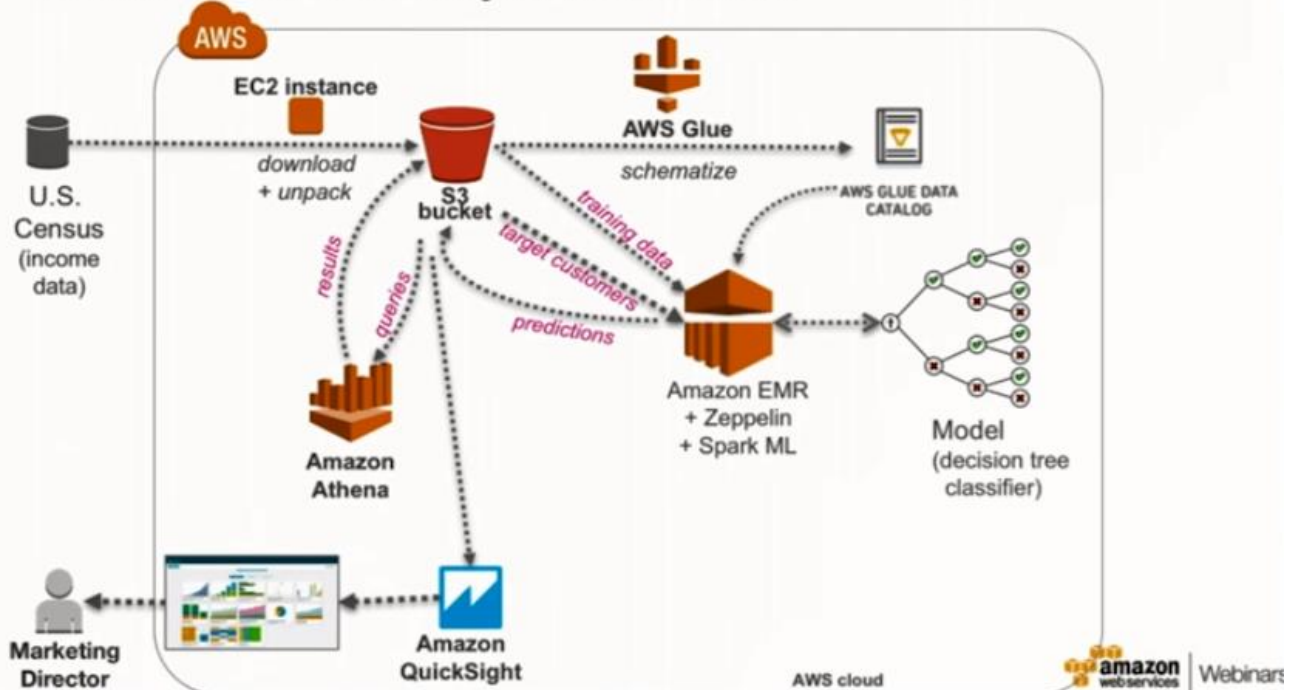
We then train a decision tree using **SparkML**, then we will take customers that we don't know their incomes and use that dataset to make predictions and store the results

## Demo: Predictive Analytics architecture



We then query the predictions data with Athena and store the query results

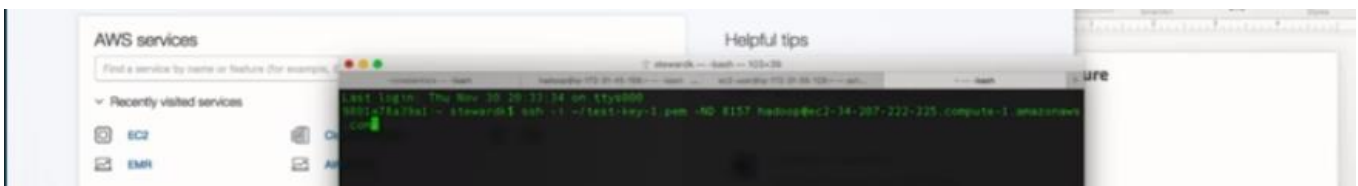
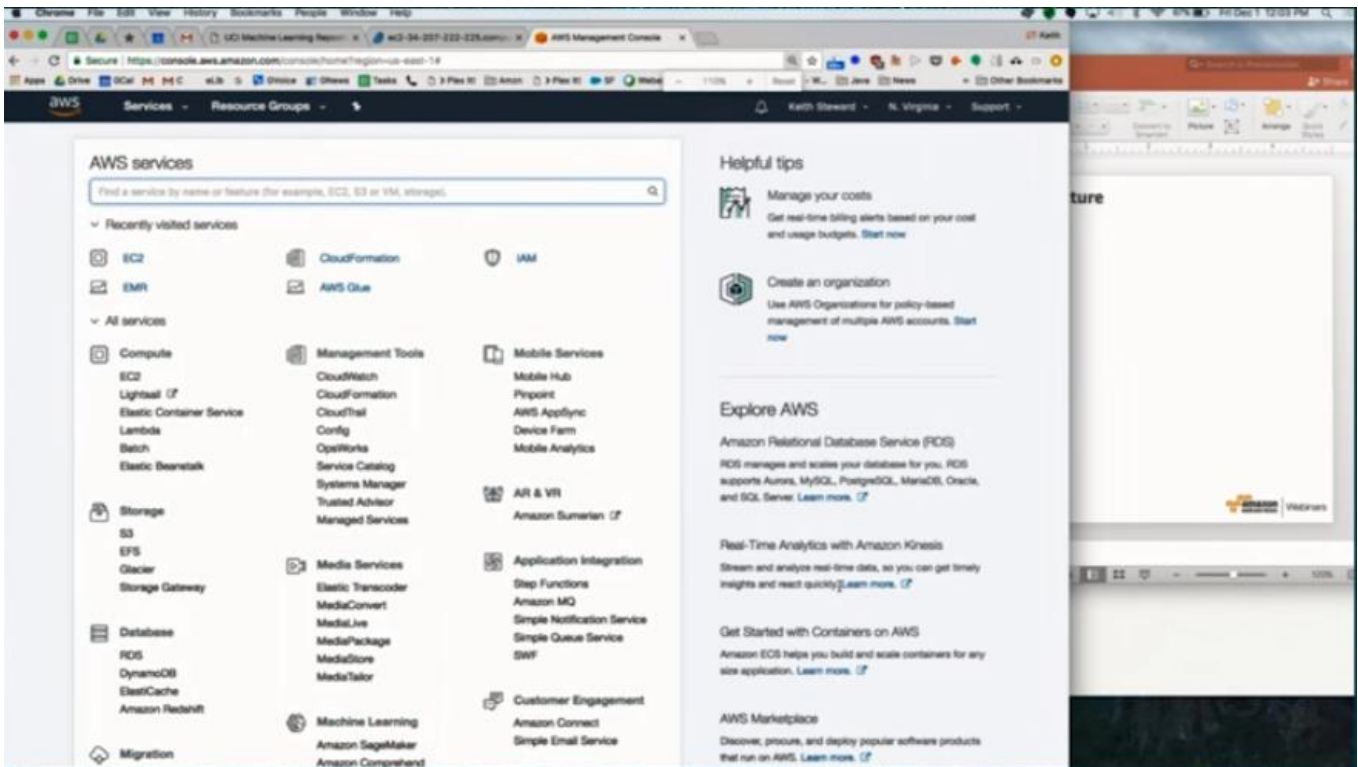
## Demo: Predictive Analytics architecture



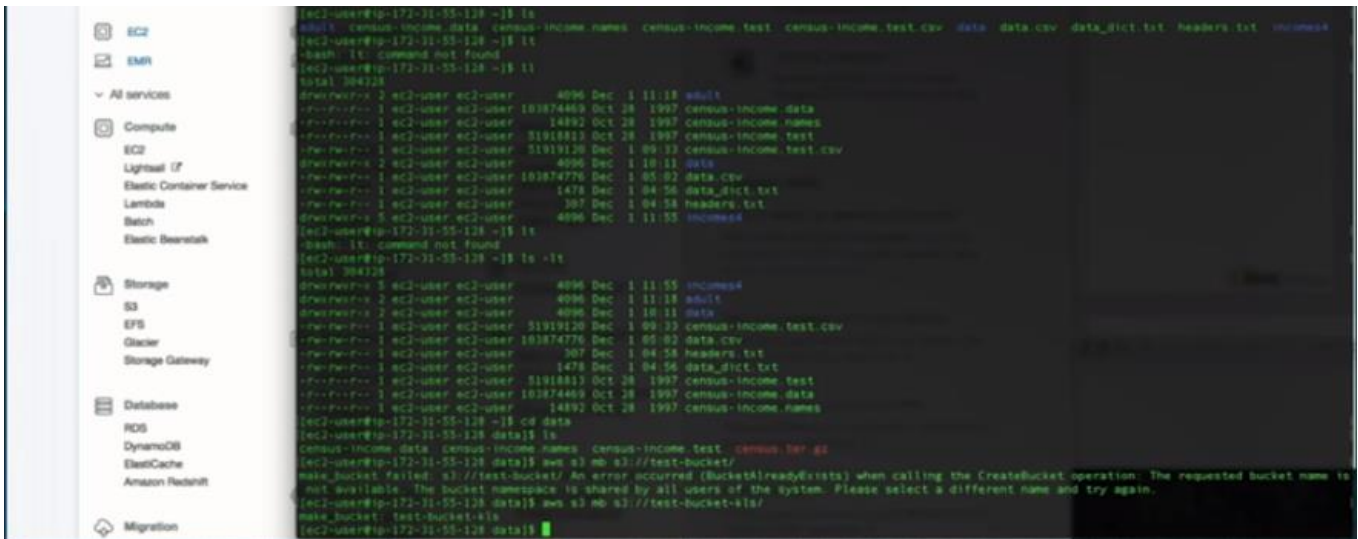
Then we can visualize our query results and create reports to share

## Demo: Predictive Analytics architecture

*Let's Build It!*



Spin up an EC2 instance an SSH into it



Use **wget** to pull down the census data and store it in an S3 bucket using the AWS S3 CLI with the **\$ aws s3 mb s3://test-bucket-41s/training** command and then use the **\$ aws s3 cp census-income-data s3://test-bucket-41s/training/** to copy the data into the bucket.

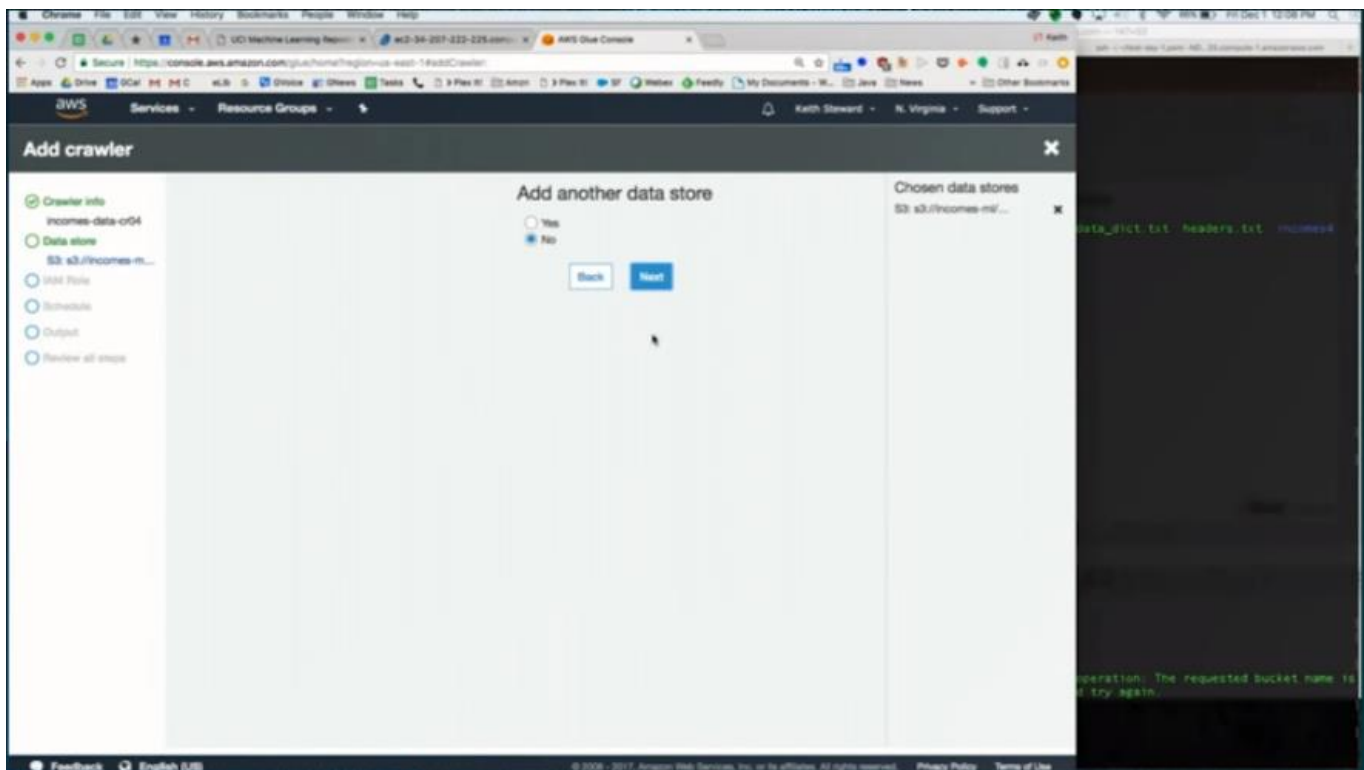
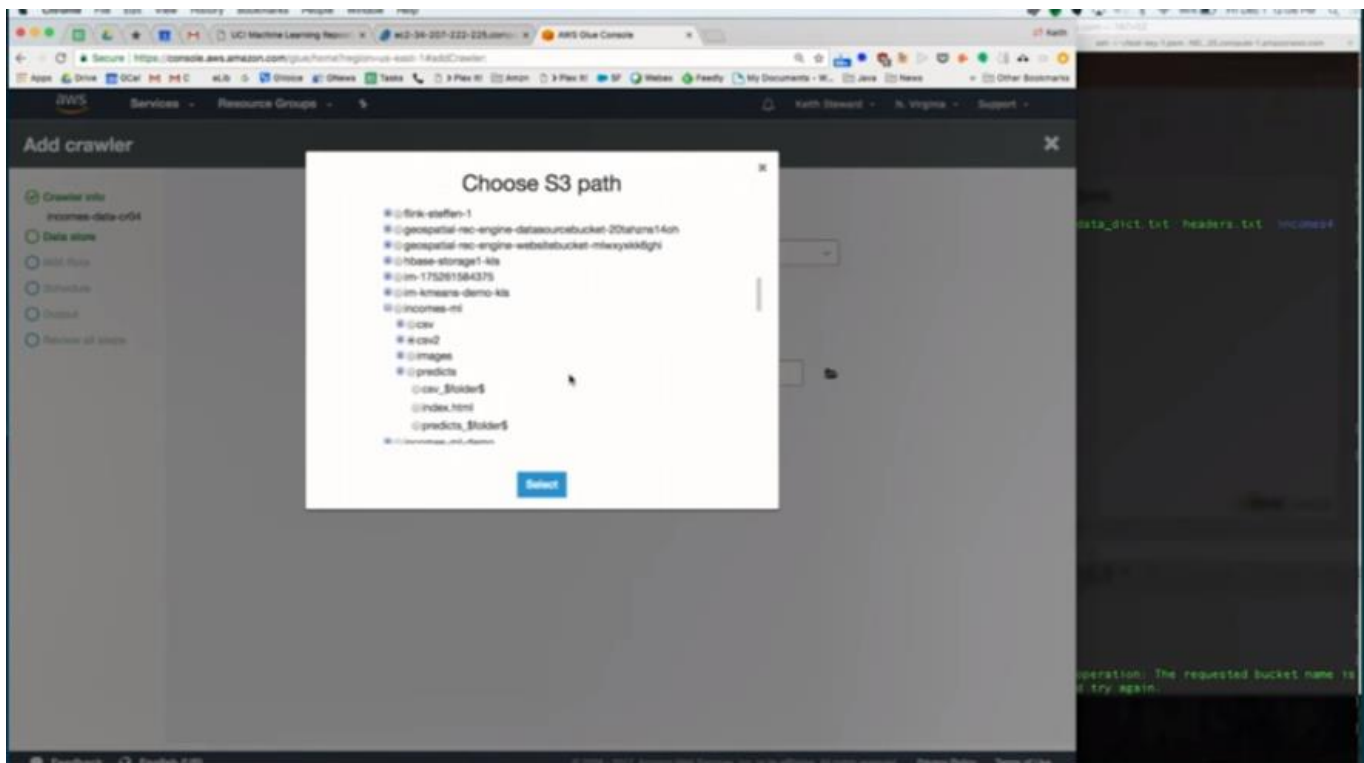
The screenshot shows the AWS Glue console interface. On the left is a navigation menu with options like Data catalog, Databases, Tables, Connections, Crawlers, Classifiers, ETL, Jobs, Triggers, Dev endpoints, Tutorials, Add crawler, Explore table, Add job, and Resources. The main area displays the details for a table named 'civ2'. The table's description is 'Incomes3\_db', its classification is 'civ', and its location is 's3://incomes-milwau/'. It was last updated on Oct 21, 2017. The input format is 'org.apache.hadoop.mapred.TextInputFormat' and the output format is 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'. The SerDe is 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'. The SerDe parameters are 'field.delim = ' and 'skip.header.line.count = 1'. The table properties include 'delimiter = ' and 'CrawlerSchemaSerializerVersion = 1.0'. The schema is shown as a table with three columns: 'age' (bigint), 'workclass' (string), and 'lnwgt' (bigint).

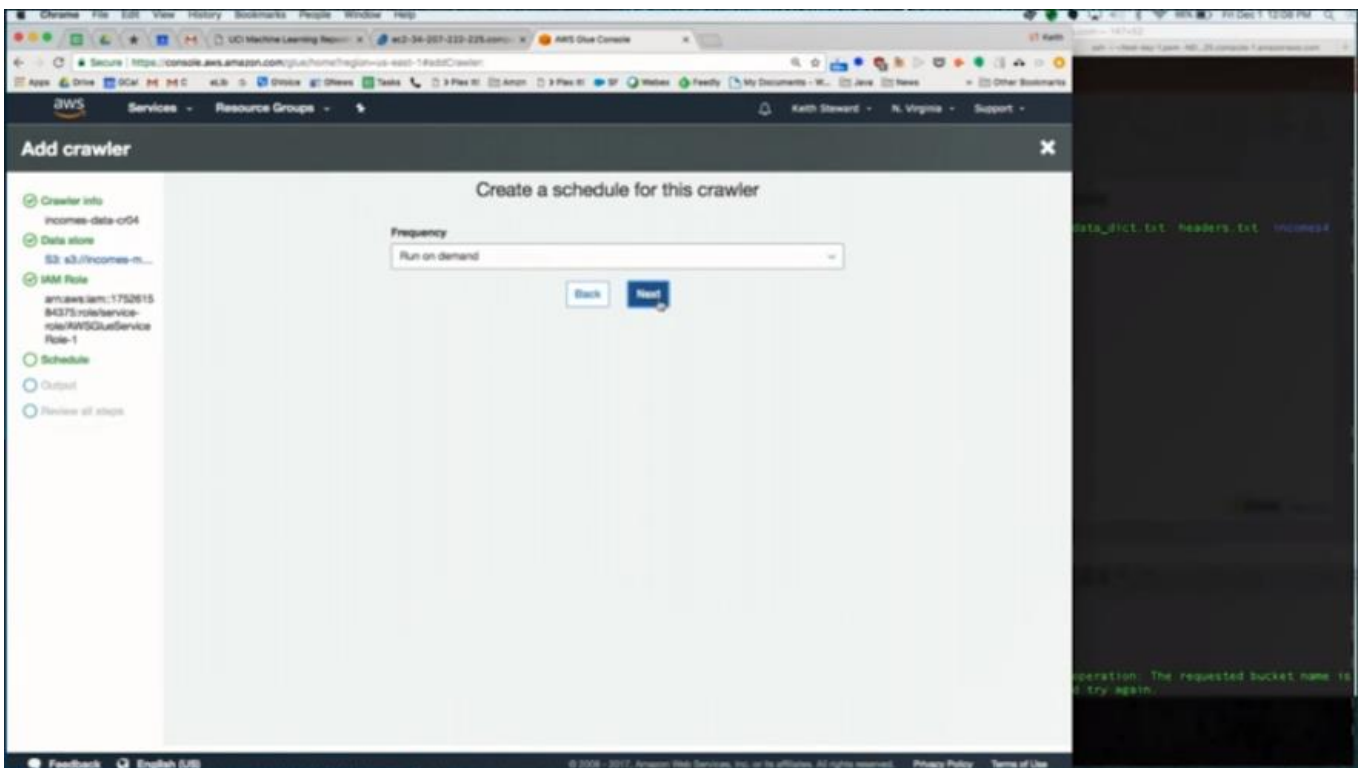
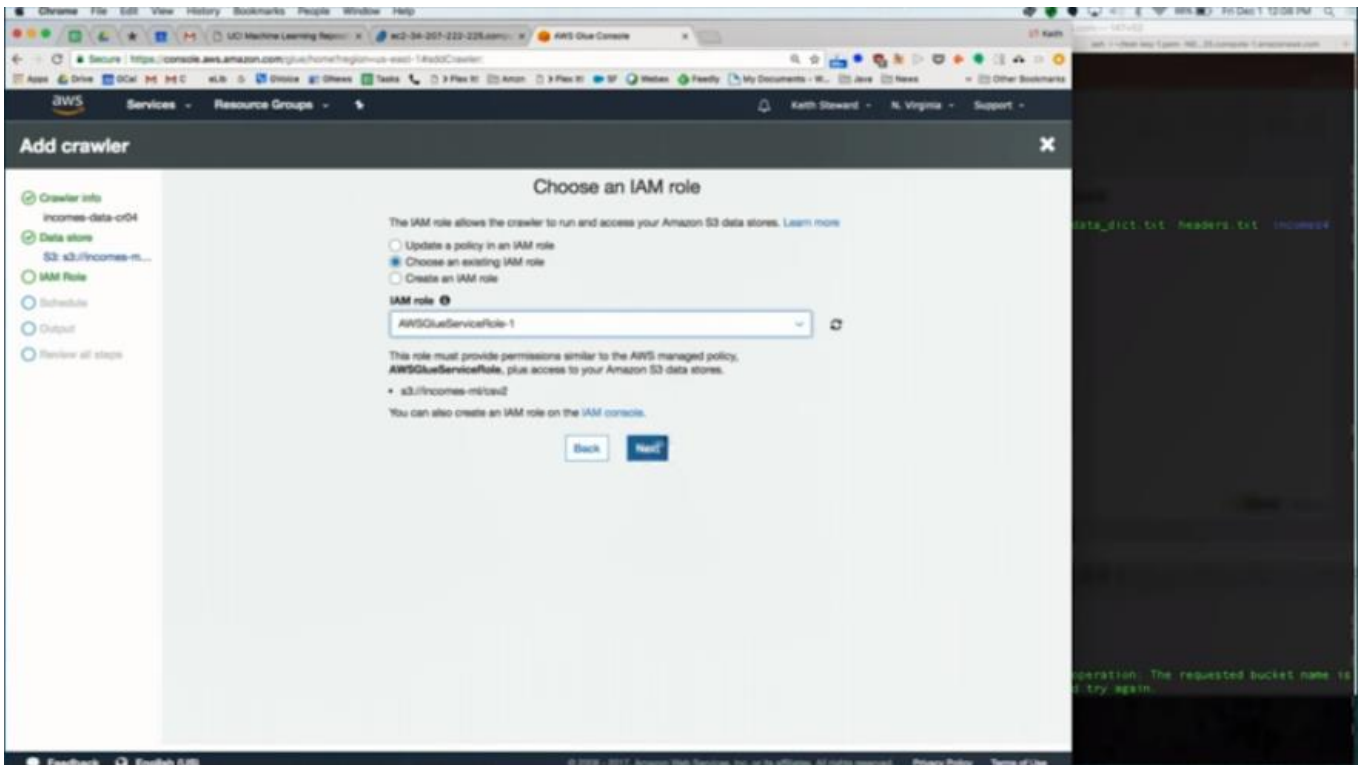
Column name	Data type	Key
1	age	bigint
2	workclass	string
3	lnwgt	bigint

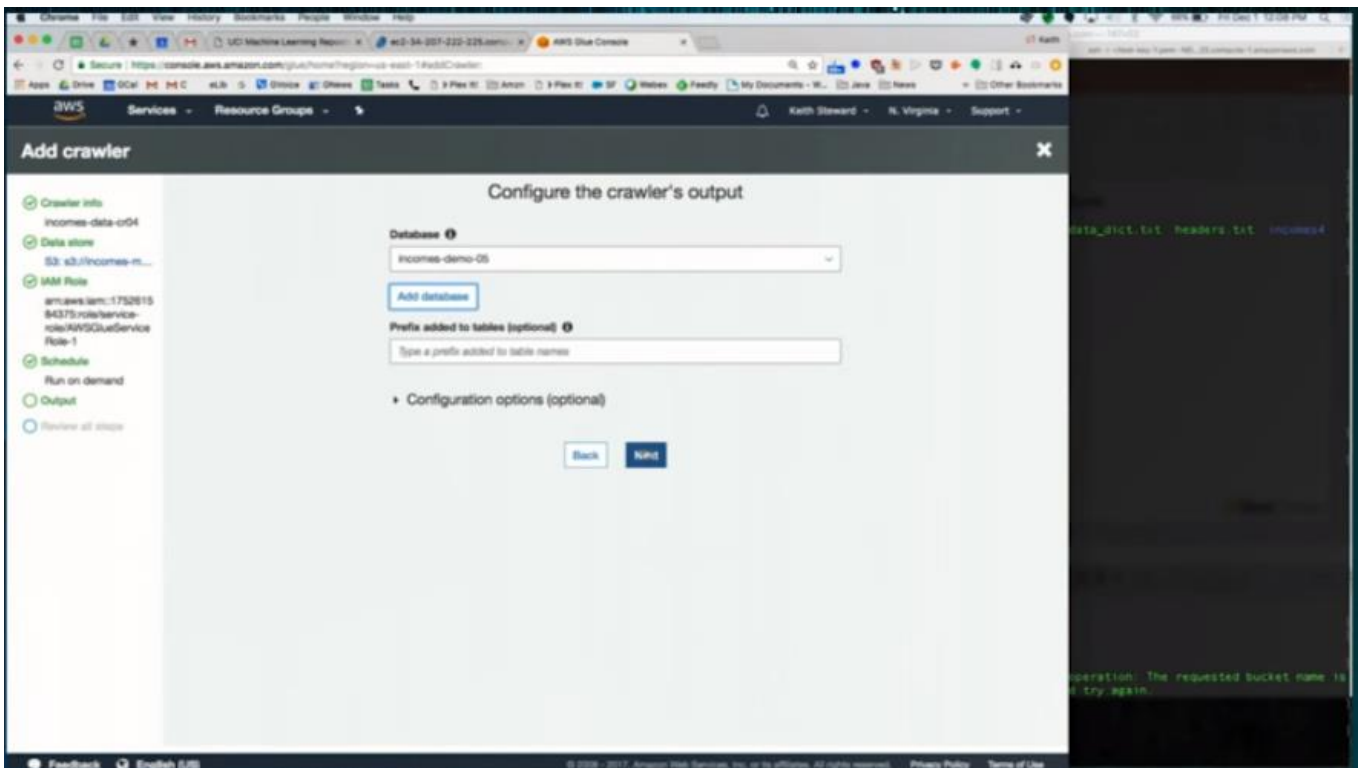
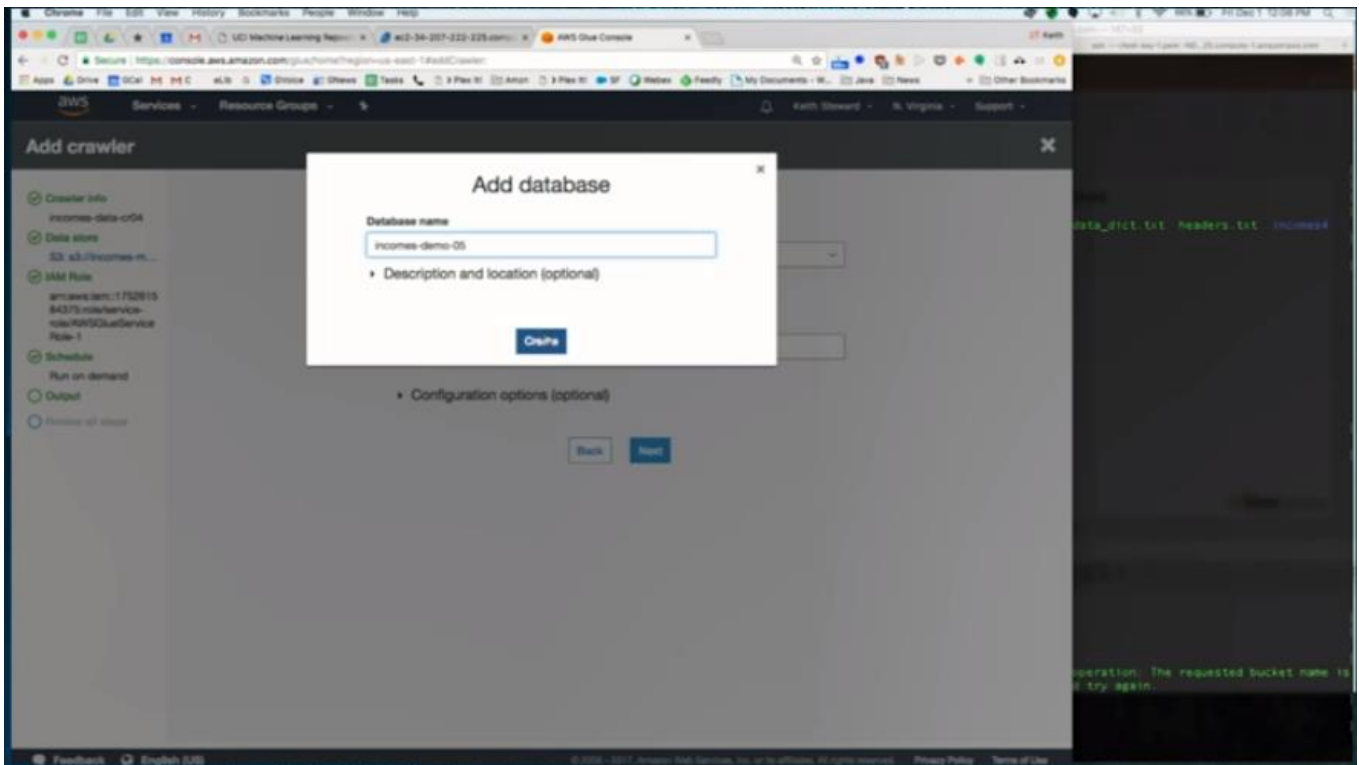
We then run AWS Glue on the data in S3 to schematize it and get us a table with all the data attributes as columns

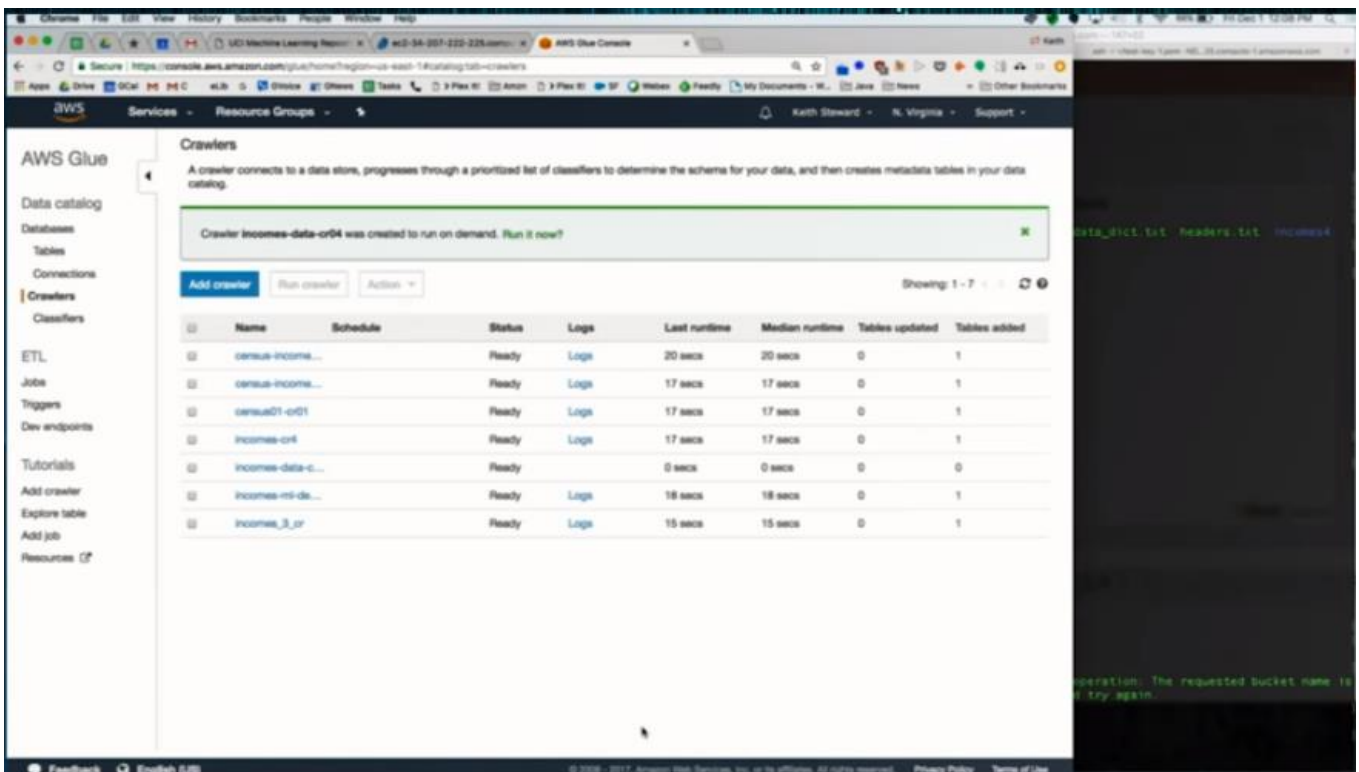
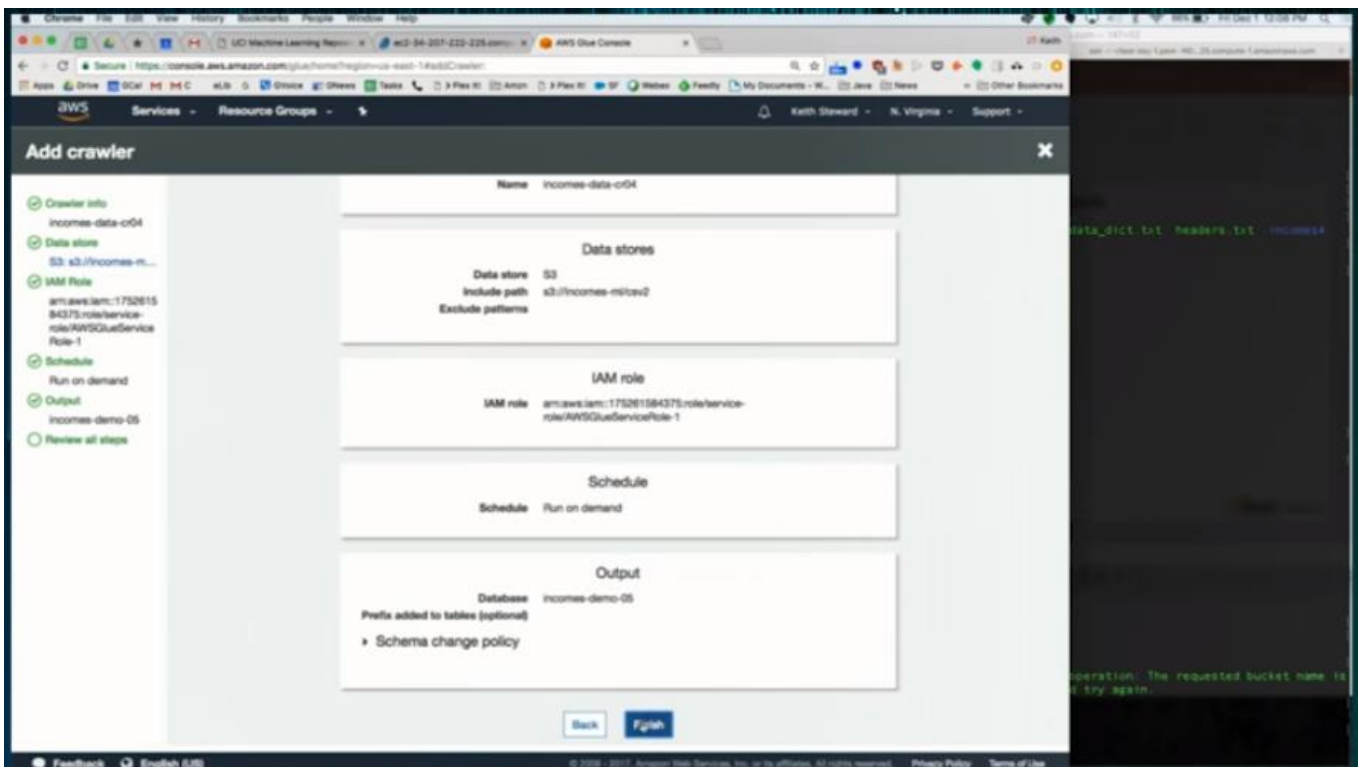
The screenshot shows the 'Add crawler' wizard in the AWS Glue console. The 'Data store' is set to 'S3'. The 'Crawl data in' section has 'Specified path in my account' selected. The 'Include path' is set to 's3://Bucket-name/folder-name/file-name'. The 'Exclude patterns (optional)' section is expanded. The 'Next' button is highlighted.













**Crawlers**

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawler "incomes-data-crawler" is now running.

[Add crawler](#) [Run crawler](#) [Action](#)

Showing: 1 - 7

Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
census-income...		Ready	<a href="#">Logs</a>	20 secs	20 secs	0	1
census-income...		Ready	<a href="#">Logs</a>	17 secs	17 secs	0	1
census01-cr01		Ready	<a href="#">Logs</a>	17 secs	17 secs	0	1
incomes-cr01		Ready	<a href="#">Logs</a>	17 secs	17 secs	0	1
incomes-data-c...		Starting		0 secs	0 secs	0	0
incomes-mi-da...		Ready	<a href="#">Logs</a>	18 secs	18 secs	0	1
incomes_3_or		Ready	<a href="#">Logs</a>	15 secs	15 secs	0	1

**Databases**

A database is a set of associated table definitions, organized into a logical group.

[Add database](#) [View tables](#) [Action](#)

Showing: 1 - 19

Name	Description
adult-incomes-db01	adult incomes data from UCI ML repository at <a href="http://archive.ics.uci.edu/ml/datasets/Adult">http://archive.ics.uci.edu/ml/datasets/Adult</a>
adult-incomes-db02	
adult-incomes-db03	<a href="http://archive.ics.uci.edu/ml/datasets/Adult">http://archive.ics.uci.edu/ml/datasets/Adult</a> UCI ML Repository
adult-incomes-db04	<a href="http://archive.ics.uci.edu/ml/datasets/Adult">http://archive.ics.uci.edu/ml/datasets/Adult</a> UCI ML Repository
airflights-db1	
bank-marketing	
census-income-2000-db	data from <a href="http://archive.ics.uci.edu/ml/datasets/Census+Income+%28KOO%29">http://archive.ics.uci.edu/ml/datasets/Census+Income+%28KOO%29</a> ML repository.
default	
flightdelays-db	
flightdelays-db1	
flights-db	
incidents	
incomes-demo-05	
incomes-mi-demo-db1	
incomes3_db	
incomes_db02	
instate0504	
nyctstrips	

When the AWS Glue crawler finishes, it creates a database with a bunch of tables to store the results in for us

Chrome File Edit View History Bookmarks People Window Help

Secure | https://console.aws.amazon.com/glue/home?region=us-east-1#database:name=incomes3\_db

Services Resource Groups

Keith Steward N. Virginia Support

### AWS Glue

Data catalog

Databases

Tables

Connections

Crawlers

Classifiers

ETL

Jobs

Triggers

Dev endpoints

Tutorials

Add crawler

Explore table

Add job

Resources

Databases > incomes3\_db

Edit database Delete database

Name	Description	Location
incomes3_db		

Tables in incomes3\_db

Feedback English (US) © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Chrome File Edit View History Bookmarks People Window Help

Secure | https://console.aws.amazon.com/glue/home?region=us-east-1#database:name=incomes3\_db

Services Resource Groups

Keith Steward N. Virginia Support

### AWS Glue

Data catalog

Databases

Tables

Connections

Crawlers

Classifiers

ETL

Jobs

Triggers

Dev endpoints

Tutorials

Add crawler

Explore table

Add job

Resources

#### Tables

A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Add tables Action Database: incomes3\_db Save view Showing: 1 - 3

Name	Database	Location	Classification	Last updated	Deprecated
census_training	incomes3_db	s3://census01-4/sa/training/	csv	1 December 201...	
training_csv2	incomes3_db	s3://incomes-m/sa/csv2/	csv	1 December 201...	
csv2	incomes3_db	s3://incomes-m/sa/csv2/	csv	21 October 2017...	

Feedback English (US) © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Chrome File Edit View History Bookmarks People Window Help

Secure | https://console.aws.amazon.com/glue/home?region=us-east-1#table:name=csv2;namespace=incomes3\_db

Services Resource Groups

Keith Steward N. Virginia Support

### AWS Glue

Tables > csv2

Last updated 21 Oct 2017 Table Version [Current version]

Edit table Delete table View properties Compare versions Edit schema

Data catalog

Databases

Tables

Connections

Crawlers

Classifiers

ETL

Jobs

Triggers

Dev endpoints

Tutorials

Add crawler

Explore table

Add job

Resources

Name: csv2

Description:

Database: incomes3\_db

Classification: csv

Location: s3://incomes-m/csv2/

Connection:

Deprecated: No

Last updated: Sat Oct 21 21:04:46 GMT-700 2017

Input format: org.apache.hadoop.mapred.TextInputFormat

Output format: org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat

serde serialization lib: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

serde parameters: field.delim ,

skip.header.line.count 1 sizeKey 3974463 objectCount 1

UPDATED\_BY\_CRAWLER incomes\_3\_or delimiter ,

CrawlerSchemaSerializerVersion 1.0 recordCount 26993 averageRecordSize 139

CrawlerSchemaDeserializerVersion 1.0 compressionType none typeOfData file

Schema

Showing: 1 - 15 of 15

Secure | https://console.aws.amazon.com/glue/home?region=us-east-1#table:name=csv2;namespace=incomes3\_db

Services Resource Groups

Keith Steward N. Virginia Support

### AWS Glue

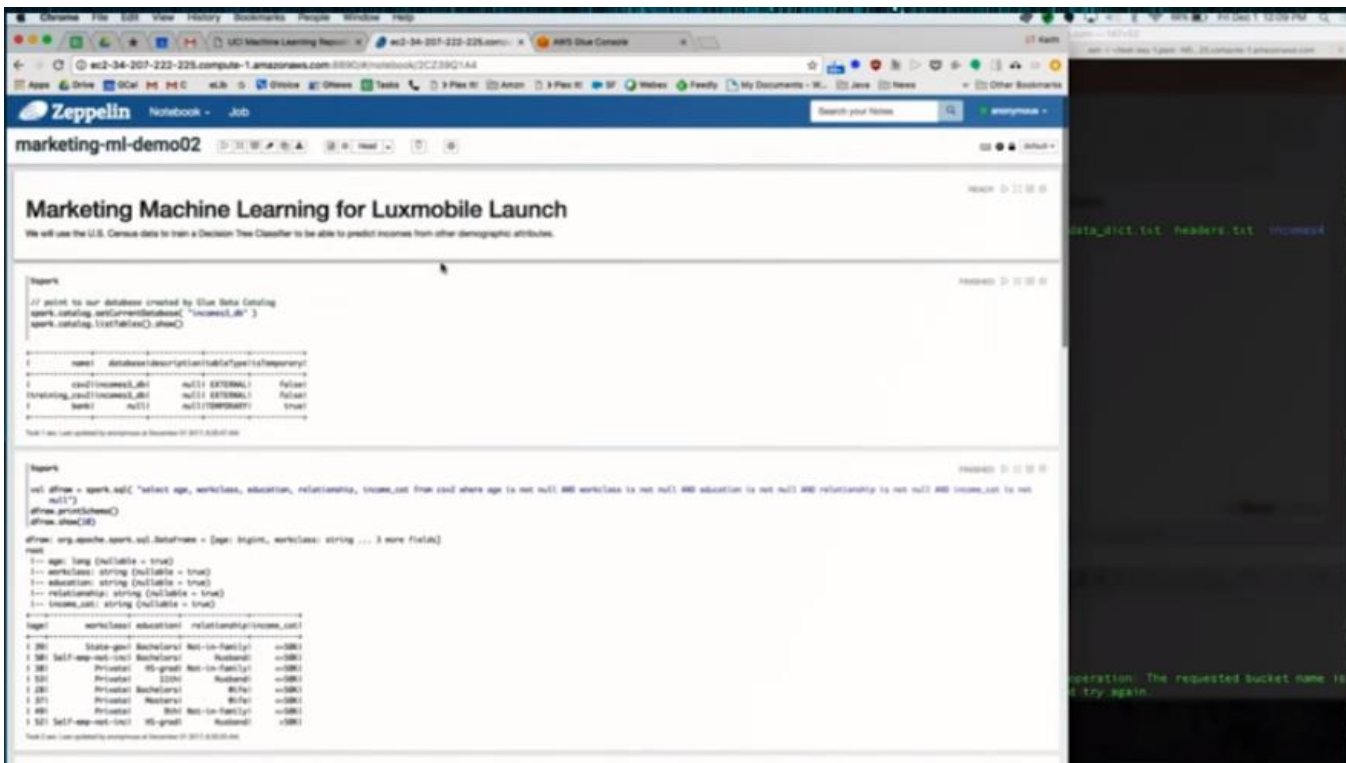
CrawlerSchemaDeserializerVersion 1.0 compressionType none typeOfData file

Schema

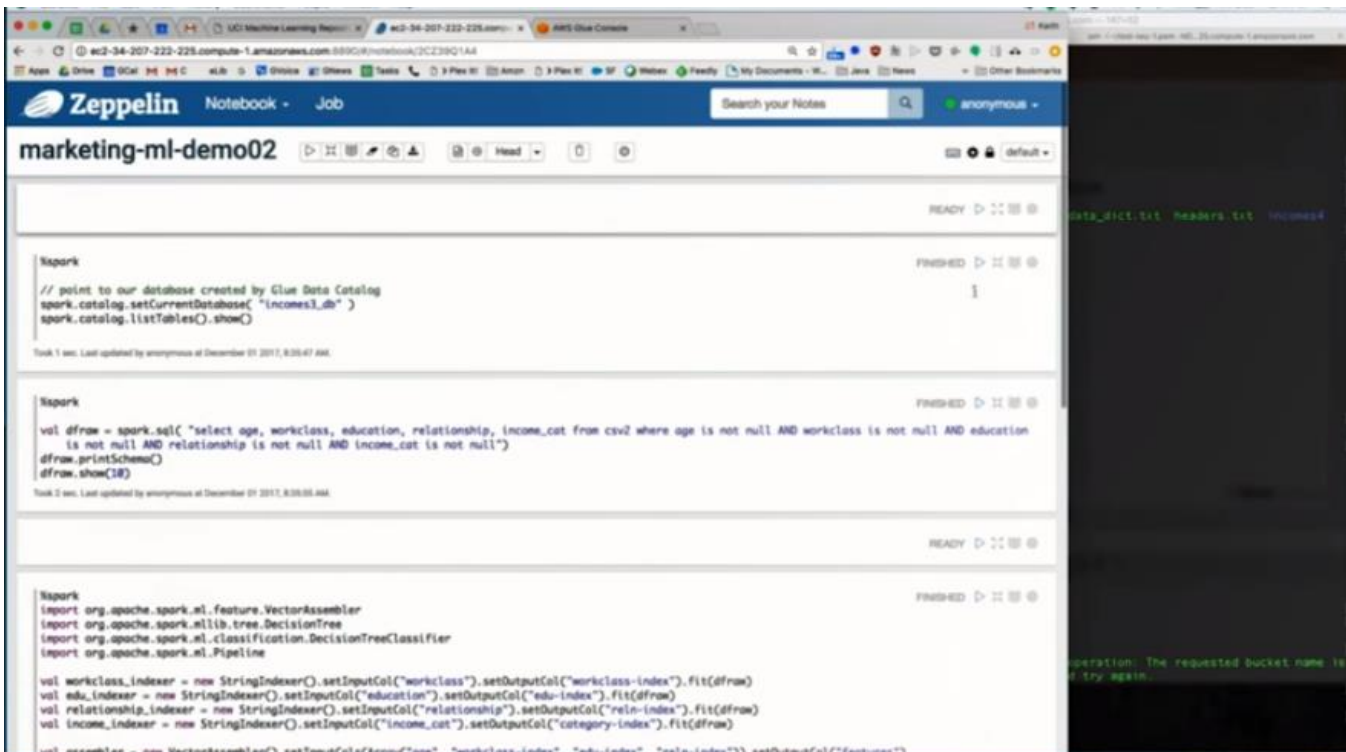
Showing: 1 - 15 of 15

	Column name	Data type	Key
1	age	bigint	
2	workclass	string	
3	fnlwgt	bigint	
4	education	string	
5	education_num	bigint	
6	marital_status	string	
7	occupation	string	
8	relationship	string	
9	race	string	
10	sex	string	
11	capital_gain	bigint	
12	capital_loss	bigint	
13	hrs_per_week	bigint	
14	native_country	string	
15	income_cat	string	

It has now converted the CSV data into a suitable schema for us



We then spin up an EMR cluster for our ML process with Spark and Zeppelin using the Web UI to have the applications running, we can then open up the Zeppelin notebook that is running on the master node in the EMR cluster as above



We are then running Spark with the notebook to run Spark code, this will distribute all the processing across all the nodes within the 10-node cluster we have. We can then start running Spark code as above



**Zeppelin Notebook - Job**

marketing-ml-demo02

Task 1: `spark`  
 // point to our database created by Glue Data Catalog  
`spark.catalog.setCurrentDatabase("incomes3_db")`  
`spark.catalog.listTables().show()`

name	database	description	tableType	isTemporary
census_training	incomes3_db		EXTERNAL	false
csv2	incomes3_db		EXTERNAL	false
training_csv2	incomes3_db		EXTERNAL	false
bank			TEMPORARY	true

Task 2: `spark`  
`val dfraw = spark.sql("select age, workclass, education, relationship, income_cat from csv2 where age is not null AND workclass is not null AND education is not null AND relationship is not null AND income_cat is not null")`  
`dfraw.printSchema()`  
`dfraw.show(10)`

Task 3: `spark`  
`import org.apache.spark.ml.feature.VectorAssembler`

**Zeppelin Notebook - Job**

marketing-ml-demo02

Task 1: `spark`  
 // point to our database created by Glue Data Catalog  
`spark.catalog.setCurrentDatabase("incomes3_db")`  
`spark.catalog.listTables().show()`

name	database	description	tableType	isTemporary
census_training	incomes3_db		EXTERNAL	false
csv2	incomes3_db		EXTERNAL	false
training_csv2	incomes3_db		EXTERNAL	false
bank			TEMPORARY	true

Task 2: `spark`  
`val dfraw = spark.sql("select age, workclass, education, relationship, income_cat from csv2 where age is not null AND workclass is not null AND education is not null AND relationship is not null AND income_cat is not null")`  
`dfraw.printSchema()`  
`dfraw.show(10)`

Task 3: `spark`  
`import org.apache.spark.ml.feature.VectorAssembler`

Next, we run another query to create a data frame by running Spark SQL across the Glue data catalog table called csv2 and pull out specific fields we need.

The screenshot shows a Zeppelin Notebook interface with a blue header bar containing the Zeppelin logo, 'Notebook - Job', and a search bar. The notebook title is 'marketing-ml-demo02'. Below the title, there's a toolbar with icons for execution, undo, redo, and other actions. The main content area displays a Spark SQL query and its results. The query is as follows:

```
val dfraw = spark.sql("select age, workclass, education, relationship, income_cat from csv2 where age is not null AND workclass is not null AND education is not null AND relationship is not null AND income_cat is not null")
dfraw.printSchema()
dfraw.show(10)
```

The results are displayed in a table with the following columns: age, workclass, education, relationship, and income\_cat. The table shows 10 rows of data, with the first row being (39, State-gov, Bachelors, Not-in-family, <=50K). The results are truncated with 'only showing top 10 rows'.

Task 1 sec. Last updated by anonymous at December 01 2017, 12:11:42 PM.

Using the data frame and running the query will put the result data in-memory for us to do further work with.

The screenshot shows a Zeppelin Notebook interface with a blue header bar containing the Zeppelin logo, 'Notebook - Job', and a search bar. The notebook title is 'marketing-ml-demo02'. Below the title, there's a toolbar with icons for execution, undo, redo, and other actions. The main content area displays a Scala code snippet for creating a ML pipeline. The code is as follows:

```
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.Pipeline

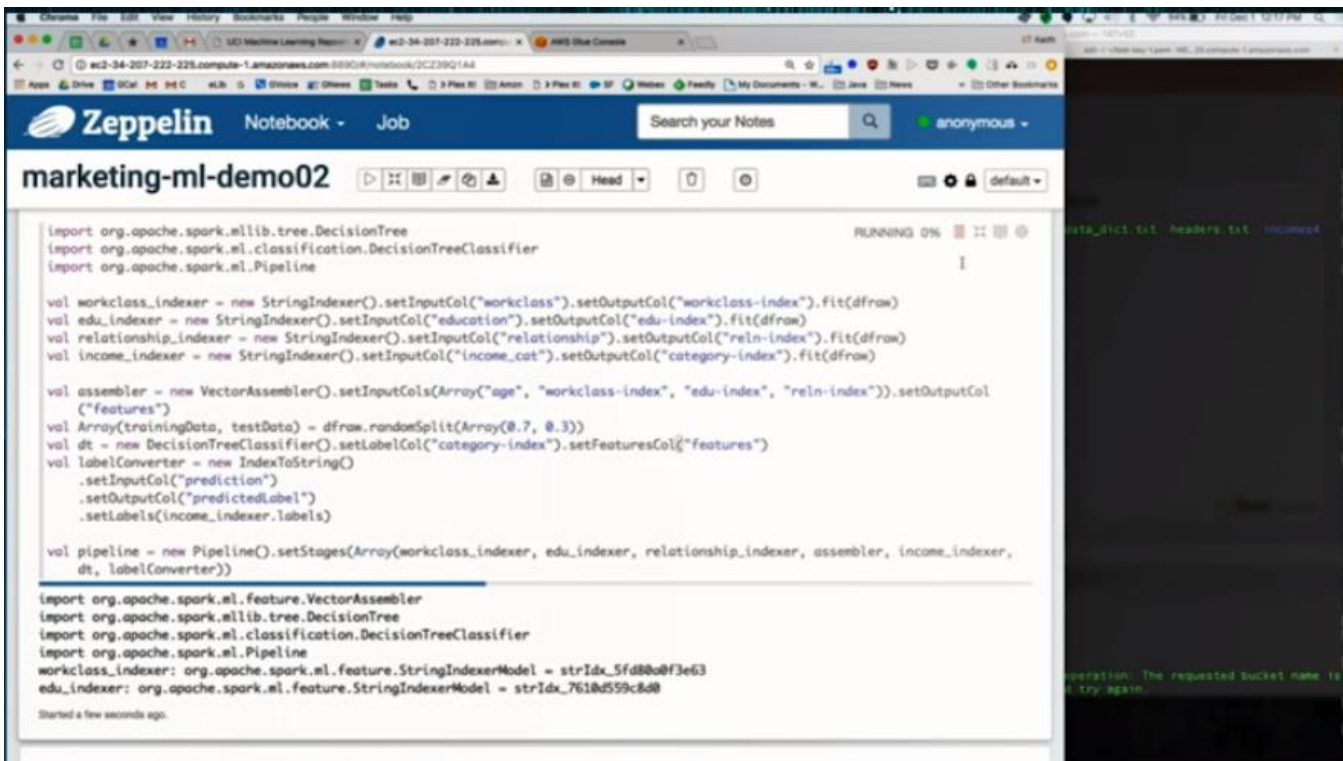
val workclass_indexer = new StringIndexer().setInputCol("workclass").setOutputCol("workclass-index").fit(dfraw)
val edu_indexer = new StringIndexer().setInputCol("education").setOutputCol("edu-index").fit(dfraw)
val relationship_indexer = new StringIndexer().setInputCol("relationship").setOutputCol("reln-index").fit(dfraw)
val income_indexer = new StringIndexer().setInputCol("income_cat").setOutputCol("category-index").fit(dfraw)

val assembler = new VectorAssembler().setInputCols(Array("age", "workclass-index", "edu-index", "reln-index")).setOutputCol("features")
val Array(trainingData, testData) = dfraw.randomSplit(Array(0.7, 0.3))
val dt = new DecisionTreeClassifier().setLabelCol("category-index").setFeaturesCol("features")
val labelConverter = new IndexToString()
  .setInputCol("prediction")
  .setOutputCol("predictedLabel")
  .setLabels(income_indexer.labels)

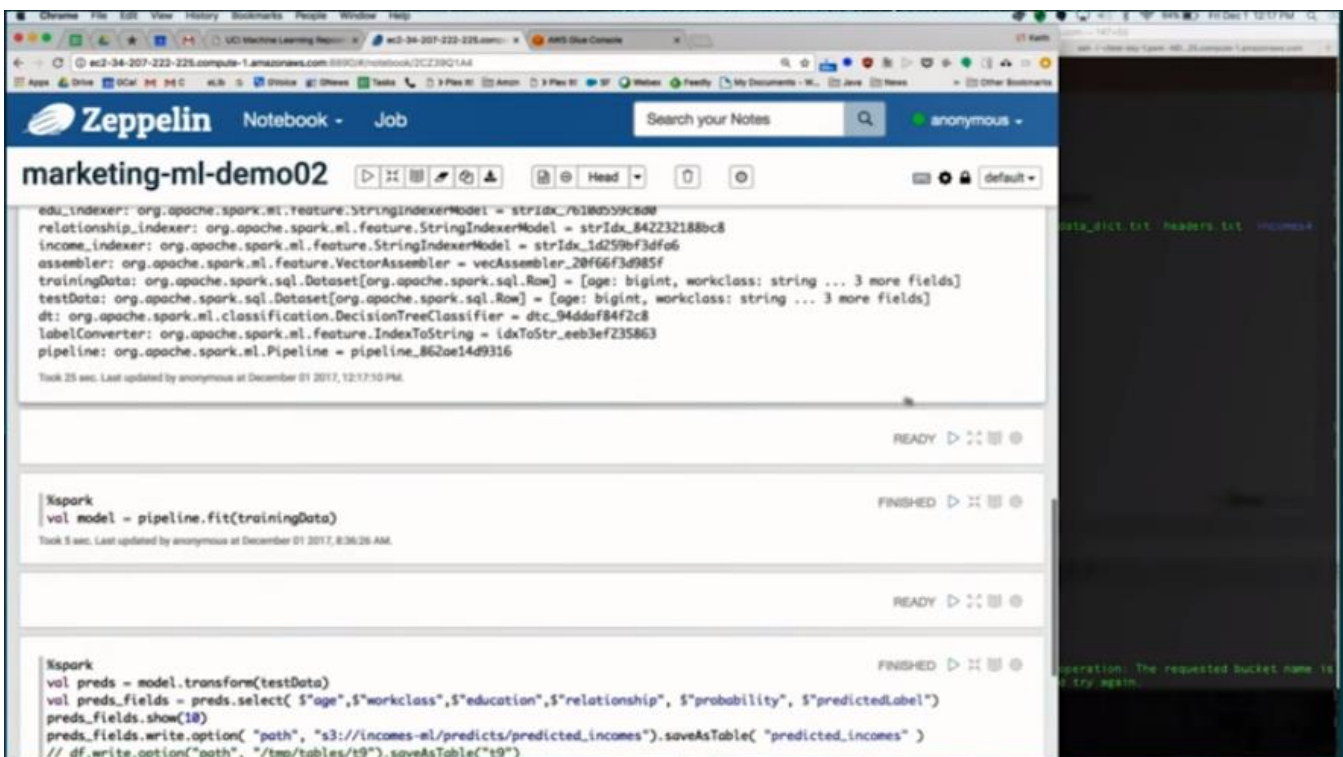
val pipeline = new Pipeline().setStages(Array(workclass_indexer, edu_indexer, relationship_indexer, assembler, income_indexer, dt, labelConverter))
```

Task 7 sec. Last updated by anonymous at December 01 2017, 8:36:15 AM.

We can now start to do the training of the model in this step.



We then run the pipeline query again





The screenshot shows a Zeppelin Notebook interface with the title "marketing-ml-demo02". The top section contains a code block with Spark configuration and data loading code. Below this, a task titled "fit the DecisionTreeClassifier to the training data" is shown in a "RUNNING" state with a progress bar at 0%. The code for this task is:

```
%spark
val model = pipeline.fit(trainingData)
```

Below the running task, there is a "READY" task and a "FINISHED" task, both with their respective code blocks.

Now that we have the new data columns, we now need to fit the pipeline to the training model

The screenshot shows the same Zeppelin Notebook interface. The task titled "fit the DecisionTreeClassifier to the training data" is now in a "FINISHED" state. The code for this task is:

```
%spark
val model = pipeline.fit(trainingData)

model: org.apache.spark.ml.PipelineModel = pipeline_862ae14d9316
```

Below the finished task, there is a "READY" task and a "FINISHED" task, both with their respective code blocks.

We now have a trained model



The screenshot shows the Zeppelin Notebook interface for a notebook titled "marketing-ml-demo02". The first code block is executed and shows the following Spark code:

```
%spark
val model = pipeline.fit(trainingData)

model: org.apache.spark.ml.PipelineModel = pipeline_862ae14d9316
```

The output of the code block is "FINISHED".

We can now make some predictions on a different set of data to infer the income level of the samples using the trained model, we then write the results back to an S3 bucket using a Hive table with the results

The screenshot shows the Zeppelin Notebook interface for the same notebook. The second code block is executed and shows the following Spark code:

```
%spark
val preds = model.transform(testData)
val preds_fields = preds.select($"age", $"workclass", $"education", $"relationship", $"probability", $"predictedLabel")
preds_fields.show(10)
preds_fields.write.option("path", "s3://incomes-ml/predicts/predicted_incomes").saveAsTable("predicted_incomes")
// df.write.option("path", "/tmp/tables/t9").saveAsTable("t9")
// df.write.saveAsTable("database.test")
//preds.show(25)
```

The output of the code block is "FINISHED".

Below the code block, the following data is displayed:

```
preds: org.apache.spark.sql.DataFrame = [age: bigint, workclass: string ... 12 more fields]
preds_fields: org.apache.spark.sql.DataFrame = [age: bigint, workclass: string ... 4 more fields]
```

	age	workclass	education	relationship	probability	predictedLabel
1	17	Private	11th	Own-child	[0.99562636676038...	<=50K
1	17	Private	11th	Own-child	[0.99562636676038...	<=50K
1	17	Private	11th	Own-child	[0.99562636676038...	<=50K
1	17	Private	11th	Own-child	[0.99562636676038...	<=50K
1	17	Private	9th	Not-in-family	[0.96861782127659...	<=50K
1	18	7	12th	Own-child	[0.99562636676038...	<=50K
1	18	Private	11th	Own-child	[0.99562636676038...	<=50K
1	19	7	Some-college	Own-child	[0.99562636676038...	<=50K
1	19	Private	HS-grad	Other-relative	[0.99562636676038...	<=50K
1	19	Private	Some-college	Own-child	[0.99562636676038...	<=50K

Zeppelin Notebook - Job

### marketing-ml-demo02

```
%spark
val preds = model.transform(testData)
val preds_fields = preds.select($"age", $"workclass", $"education", $"relationship", $"probability", $"predictedLabel")
preds_fields.show(10)
preds_fields.write.option("path", "s3://incomes-ml/predicts/predicted_incomes").saveAsTable("predicted_incomes")
// df.write.option("path", "/tmp/tables/t9").saveAsTable("t9")
// df.write.saveAsTable("database.test")
//preds.show(25)

preds: org.apache.spark.sql.DataFrame = [age: bigint, workclass: string ... 12 more fields]
preds_fields: org.apache.spark.sql.DataFrame = [age: bigint, workclass: string ... 4 more fields]
```

age	workclass	education	relationship	probability	predictedLabel
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	9th	Not-in-family	[0.96861782127659...	<=50K
18	Private	7th	Own-child	[0.99562636676038...	<=50K
18	Private	11th	Own-child	[0.99562636676038...	<=50K
19	Private	Some-college	Own-child	[0.99562636676038...	<=50K
19	Private	HS-grad	Other-relative	[0.99562636676038...	<=50K
19	Private	Some-college	Own-child	[0.99562636676038...	<=50K

only showing top 10 rows

Took 20 sec. Last updated by anonymous at December 01 2017, 12:19:48 PM.

Zeppelin Notebook - Job

### marketing-ml-demo02

```
%spark
// write the hi income individuals to s3
val pred_records = preds_fields.filter(col("predictedLabel").like(">50K"))
// df.filter(col("line").like("ERROR"))
pred_records.show(10)

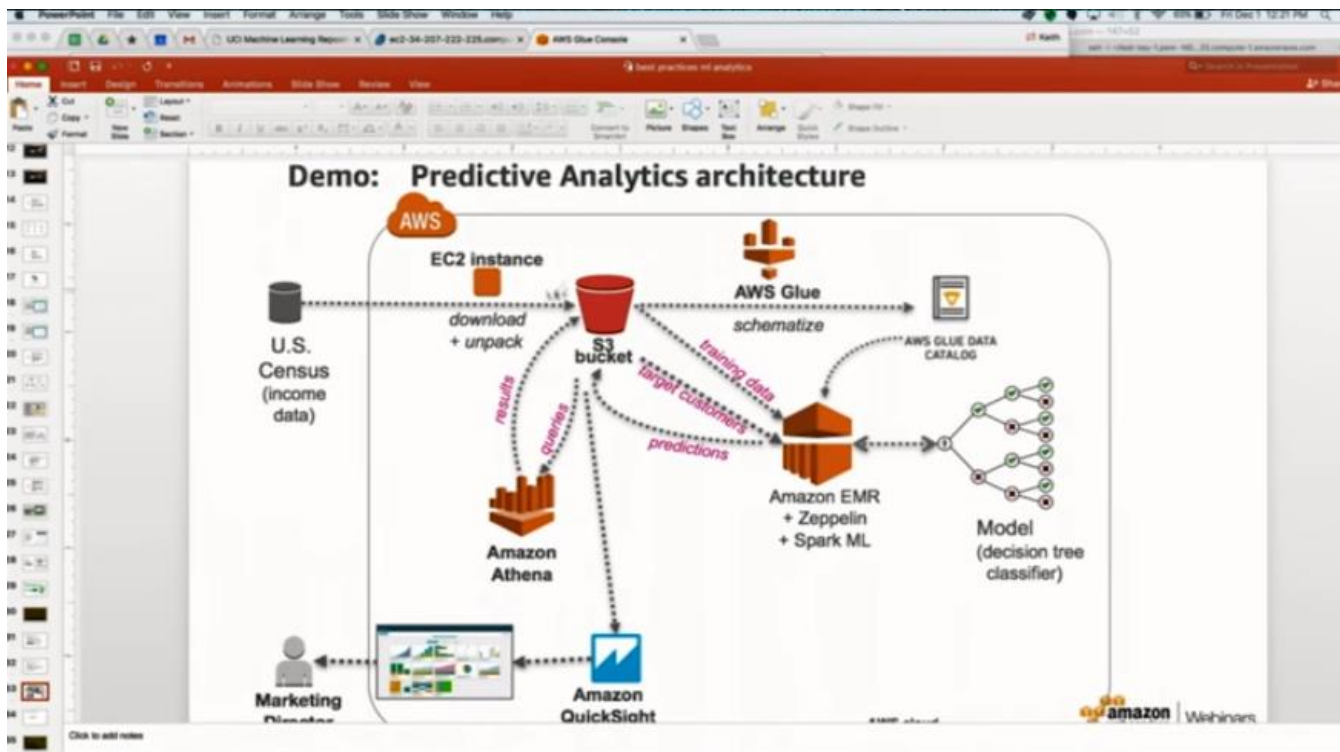
// .write.format("json").save("s3a://...")

%spark
```

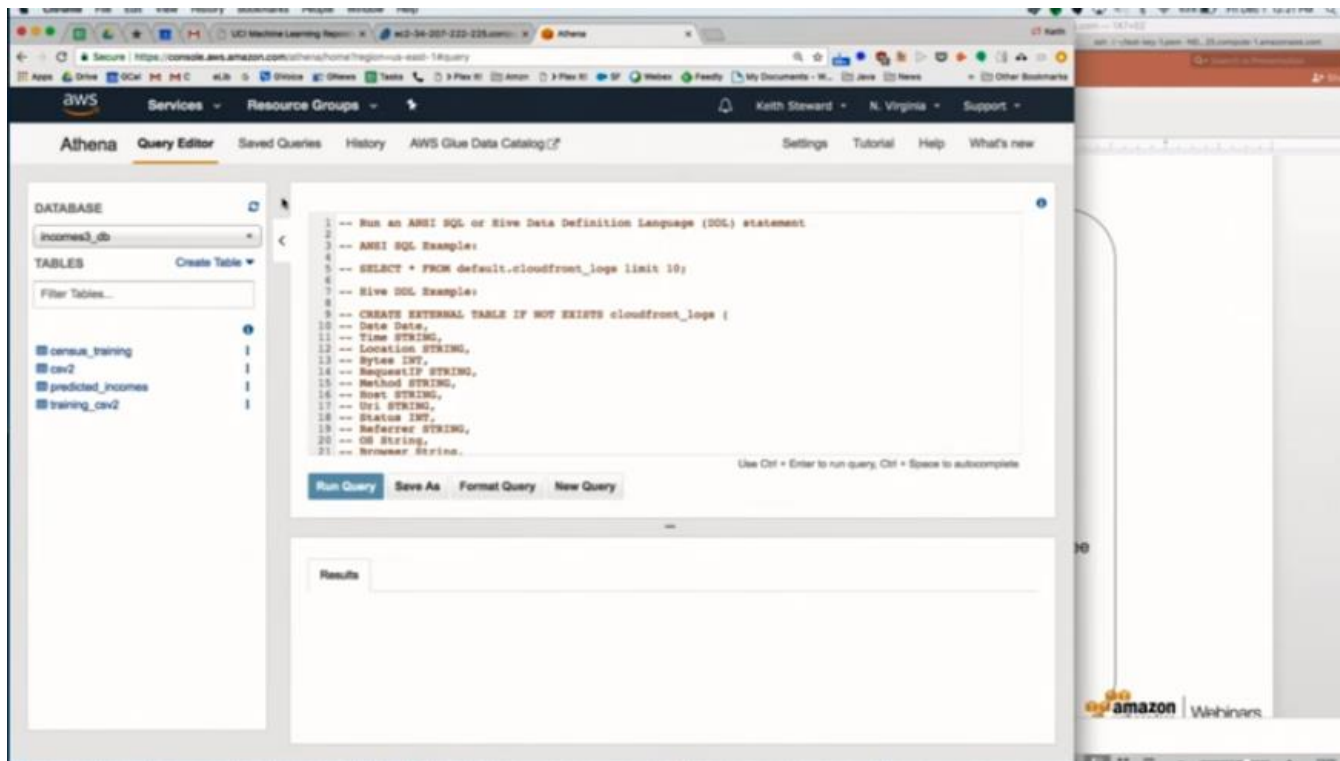
age	workclass	education	relationship	probability	predictedLabel
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	11th	Own-child	[0.99562636676038...	<=50K
17	Private	9th	Not-in-family	[0.96861782127659...	<=50K
18	Private	7th	Own-child	[0.99562636676038...	<=50K
18	Private	11th	Own-child	[0.99562636676038...	<=50K
19	Private	Some-college	Own-child	[0.99562636676038...	<=50K
19	Private	HS-grad	Other-relative	[0.99562636676038...	<=50K
19	Private	Some-college	Own-child	[0.99562636676038...	<=50K

only showing top 10 rows

Took 20 sec. Last updated by anonymous at December 01 2017, 12:19:48 PM.



We can now use Athena to query the result data in S3





Query Editor interface showing a query against the 'cav2' table in the 'incomes3\_db' database. The query is:

```
select * from cav2 limit 100;
```

The results are displayed in a table with columns: age, workclass, fnlwgt, education, education\_num, marital\_status, occupation, relationship, race. The first five rows of data are shown:

age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race
1	workclass		education		marital_status	occupation	relationship	race
2	39	State-gov	Bachelors		Never-married	Adm-clerical	Not-in-family	White
3	50	Self-emp-not-inc	Bachelors		Married-civ-spouse	Exec-managerial	Husband	White
4	36	Private	HS-grad		Divorced	Handlers-cleaners	Not-in-family	White
5	53	Private	11th		Married-civ-spouse	Handlers-cleaners	Husband	Black

We can now write a query against the original CSV data from the census site

Query Editor interface showing a query against the 'predicted\_incomes' table in the 'incomes3\_db' database. The query is:

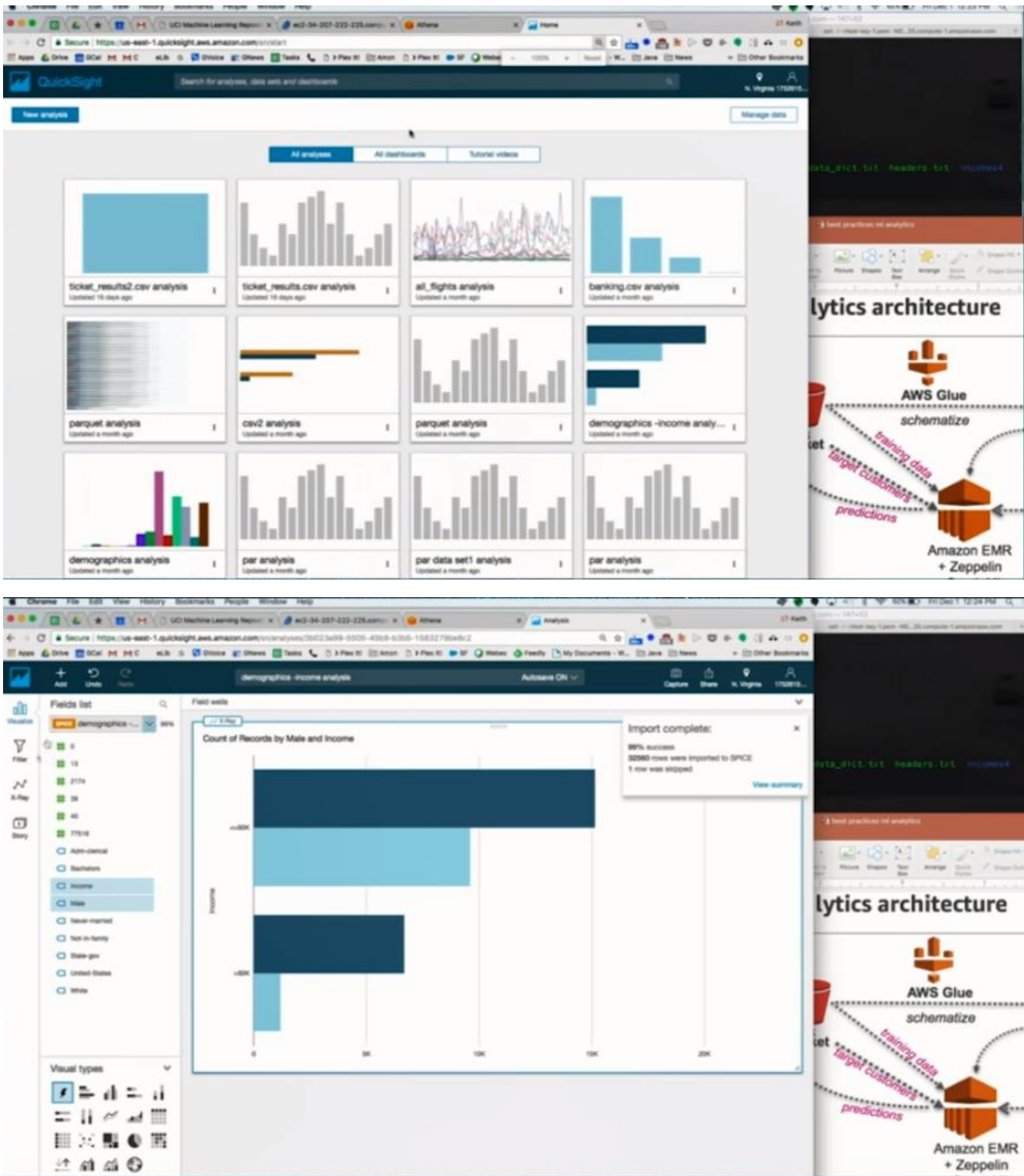
```
select * from predicted_incomes limit 100;
```

The results are displayed in a table with columns: education, relationship, probability, predictedlabel. The first five rows of data are shown:

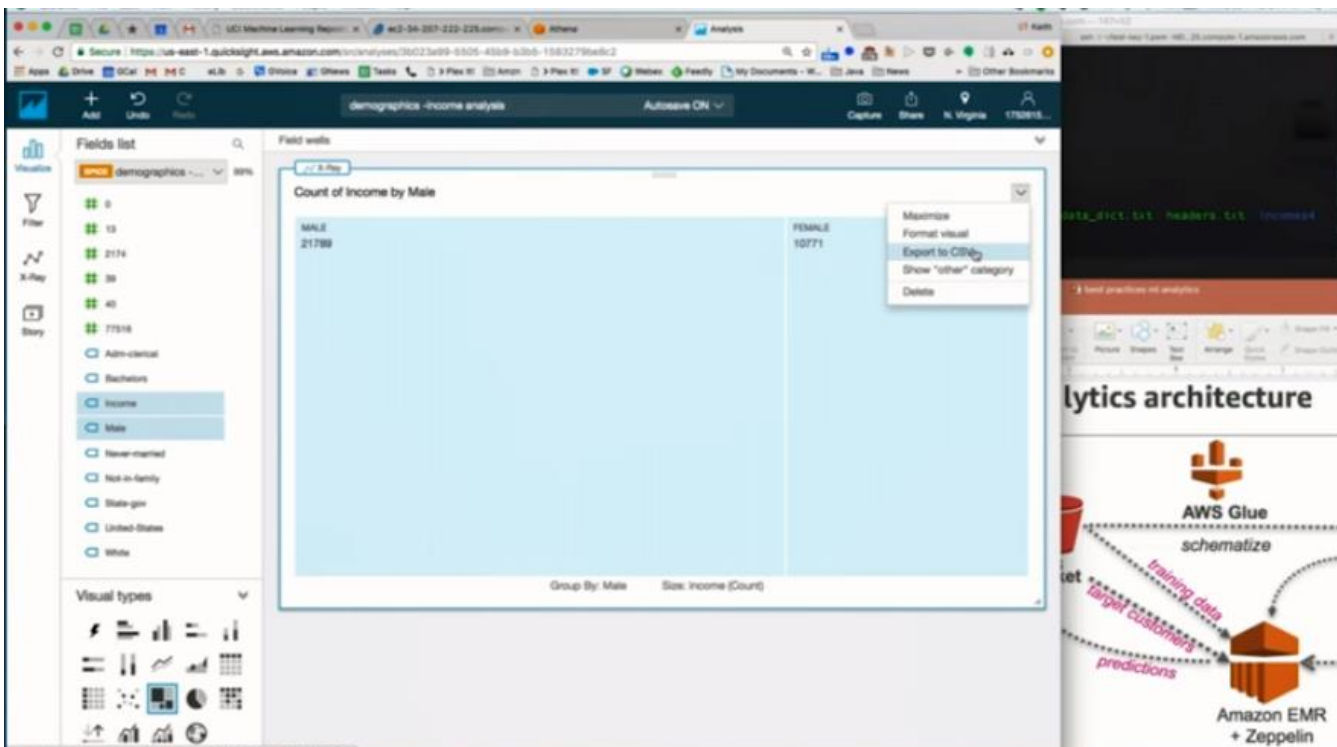
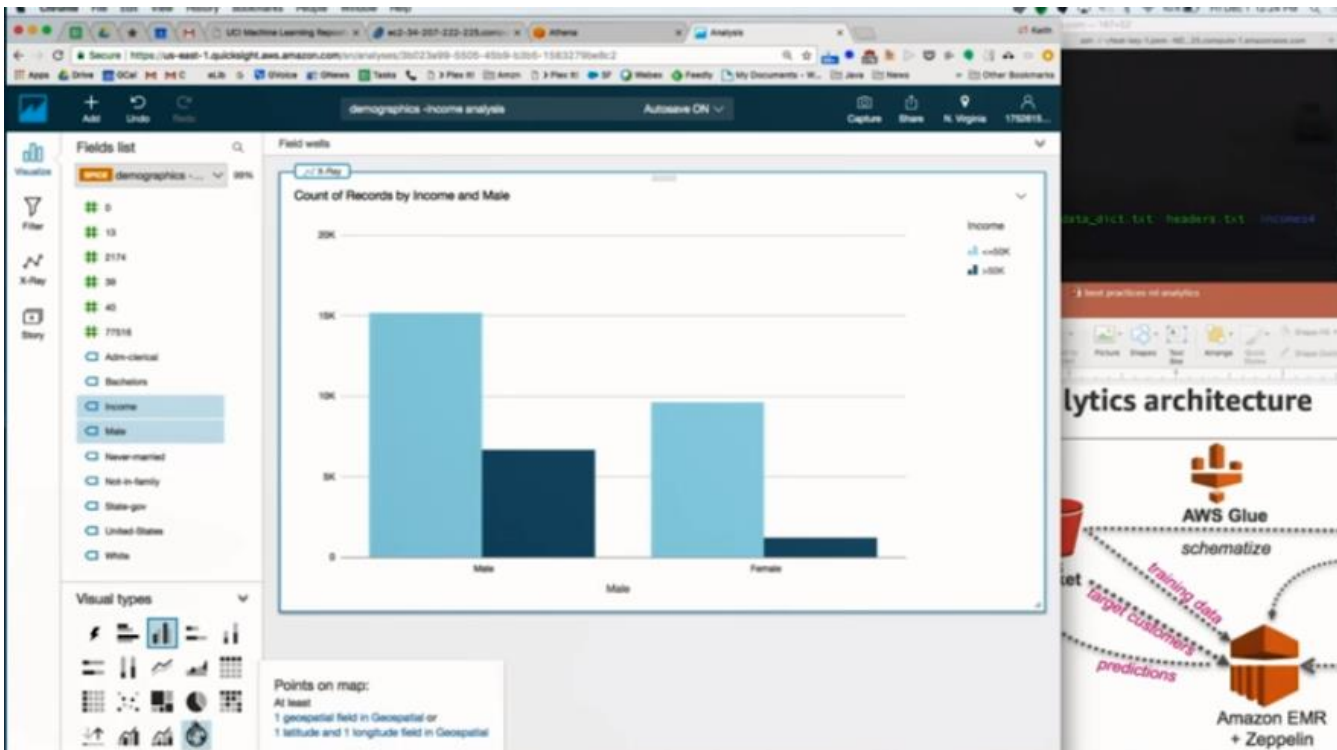
education	relationship	probability	predictedlabel
12th	Oen-child	{type=1, size=null, indices=null, values={0.9956263667603874, 0.0043736332396126214}}	<=50K
11th	Oen-child	{type=1, size=null, indices=null, values={0.9956263667603874, 0.0043736332396126214}}	<=50K
HS-grad	Oen-child	{type=1, size=null, indices=null, values={0.9956263667603874, 0.0043736332396126214}}	<=50K
Some-college	Oen-child	{type=1, size=null, indices=null, values={0.9956263667603874, 0.0043736332396126214}}	<=50K
Some-college	Oen-child	{type=1, size=null, indices=null, values={0.9956263667603874, 0.0043736332396126214}}	<=50K

We then run a query against the new predicted\_table data from the ML process too as above





We then use QuickSight to pull the S3 data into the SPICE engine for visualizations like above to share



## lytics architecture

