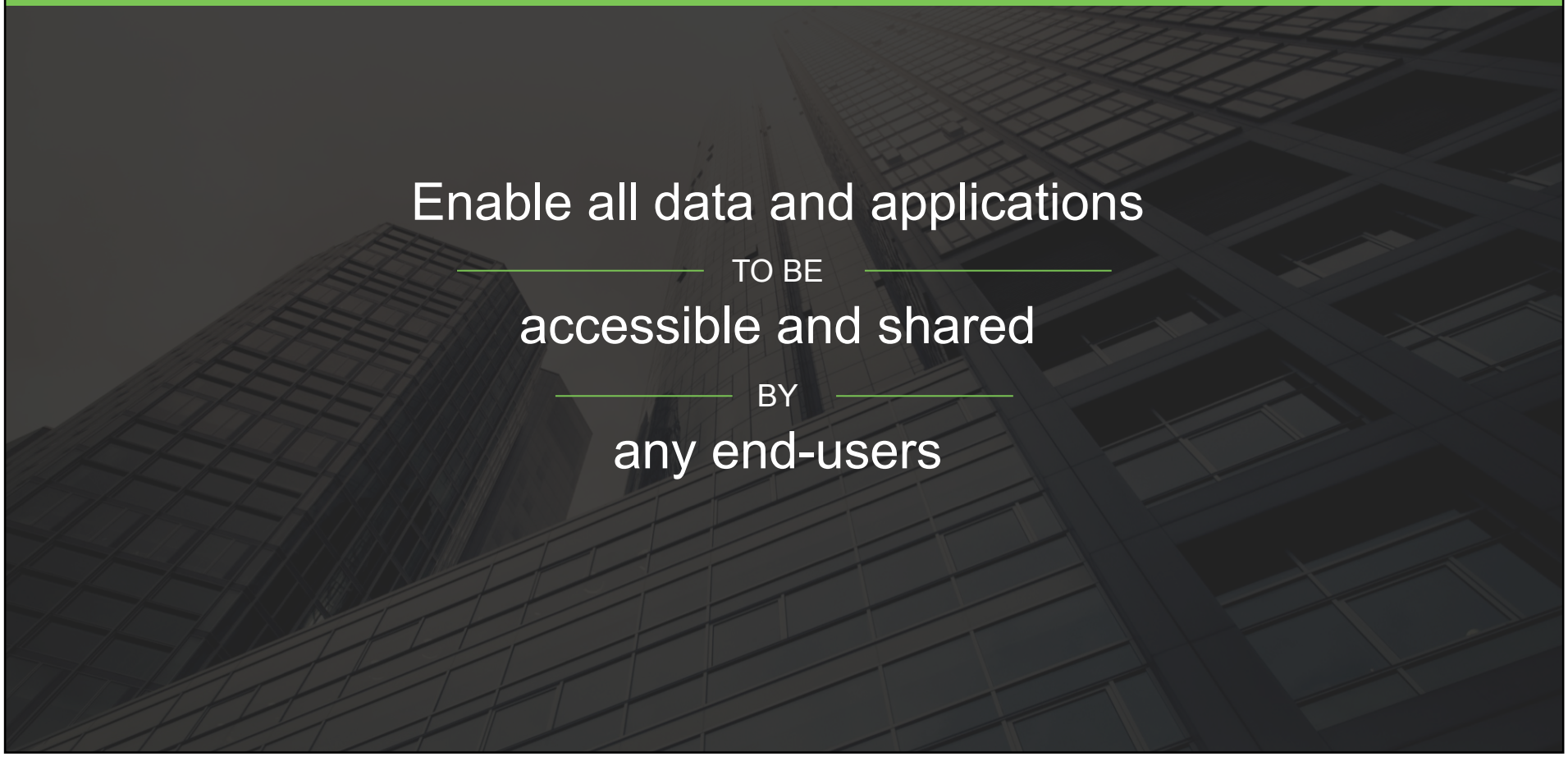
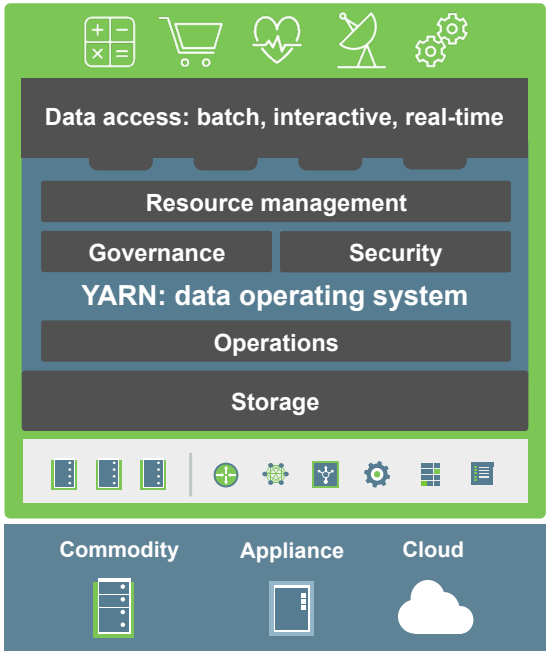

# Hadoop® and Spark® – Perfect Together

Arun C. Murthy (@*acmurthy*)
Co-Founder, Hortonworks

# Data Operating System



# Data Operating System: Open Enterprise Hadoop




## Hadoop/YARN-powered data operating system

100% open source, multi-tenant data platform for any application, any data set, anywhere.

## Built on a centralized architecture of shared enterprise services:

Scalable tiered storage

Resource and workload management

Trusted data governance & metadata management

Consistent operations

Comprehensive security

Developer APIs and tools



Hortonworks®

# Why We Love Spark at Hortonworks

## Elegant Developer APIs
DataFrames, Machine Learning, and SQL

## Made for Data Science
All apps need to get predictive at scale and fine granularity

## Democratize Machine Learning
Spark is doing to ML on Hadoop what Hive did for SQL on Hadoop

## Community
Broad developer, customer and partner interest

## Realize Value of Data Operating System
A key tool in the Hadoop toolbox




Hortonworks

# Spark and Hadoop – How Can We Do Better?

## Resource Management
YARN for multi-tenant, diverse workloads with predictable SLAs

## Tiered Memory Storage
HDFS in-memory tier – External BlockStore for RDD Cache

## SparkSQL & Hive for SQL
Interop with modern Metastore/HS2, optimized ORC support, advanced analytics e.g. Geospatial

## Spark & NoSQL
Deep integration with HBase via DataSources/Catalyst for Predicate/Aggregate Pushdown

## Connect The Dots – Algorithms to Use-Cases
Higher-level ML Abstractions – E.g. OneVsRest

Validation, Tuning, Pipeline assembly, Model export/retraining …




Hortonworks

# Spark and Hadoop – How Can We Do Better?

## Ease of Use
Apache Zeppelin for interactive notebooks

## Metadata & Governance
Apache Atlas for metadata & Apache Falcon support for Spark pipelines

## Security & Operations
Apache Ranger managed authorization and deployment/ management via Apache Ambari

## Deployable Anywhere
Linux, Windows, on-premises or cloud

## Self-Service Spark in the Cloud
Easy launch of Data Science clusters via Cloudbreak and Ambari – for Azure, AWS, GCP, OpenStack, Docker



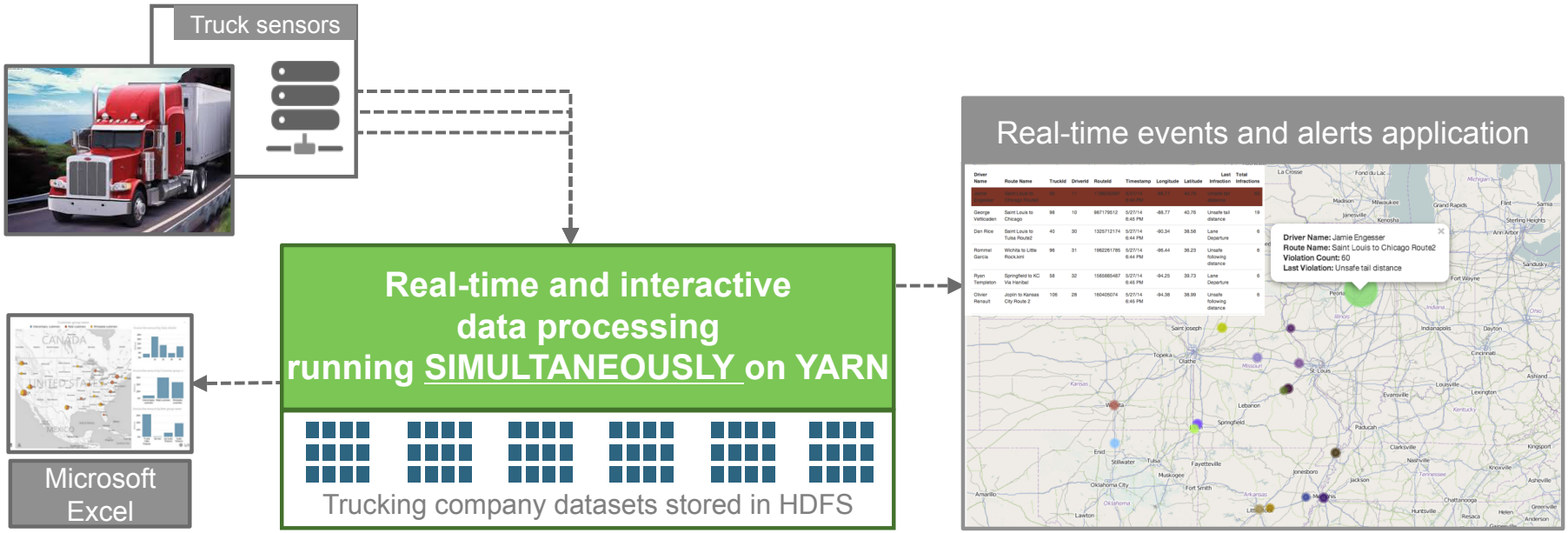

Hortonworks

# Let's Talk Real-World Use-Cases



Hortonworks®

# Last Year: Real-time and Interactive Application





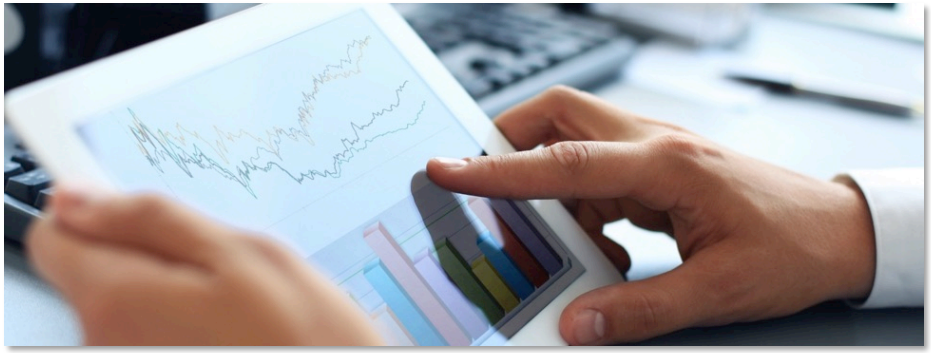

Hortonworks®

# Truck Fleet Use Case: Real-time, Predictive App

## Chief Data Officer's Needs

Increase safety and reduce liabilities

Anticipate driver violations BEFORE they happen and take precautionary actions



## Application Team's Response

Enrich app with weather data and driver profiles

Explore data for predictive model features

Train and generate predictive model

Plug in model to original application so it can predict driver violations in real-time



Hortonworks®

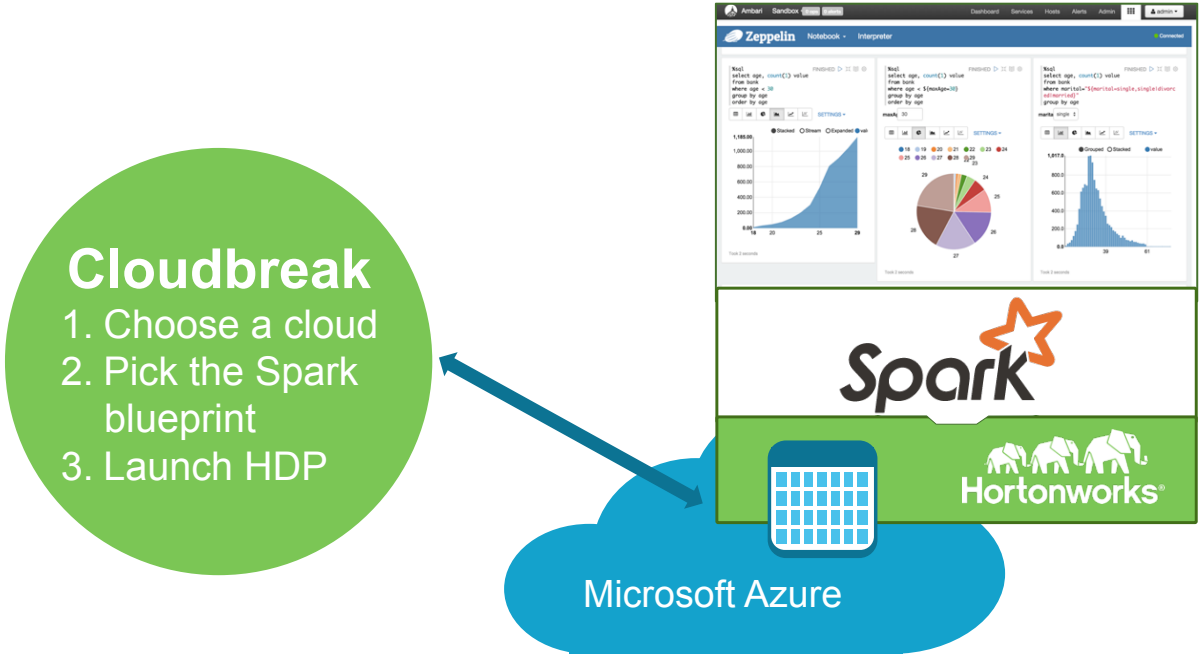
# Data Scientist: Explore Data & Build Model in Cloud

## Click-thru Demo

Provision data science environment in the cloud

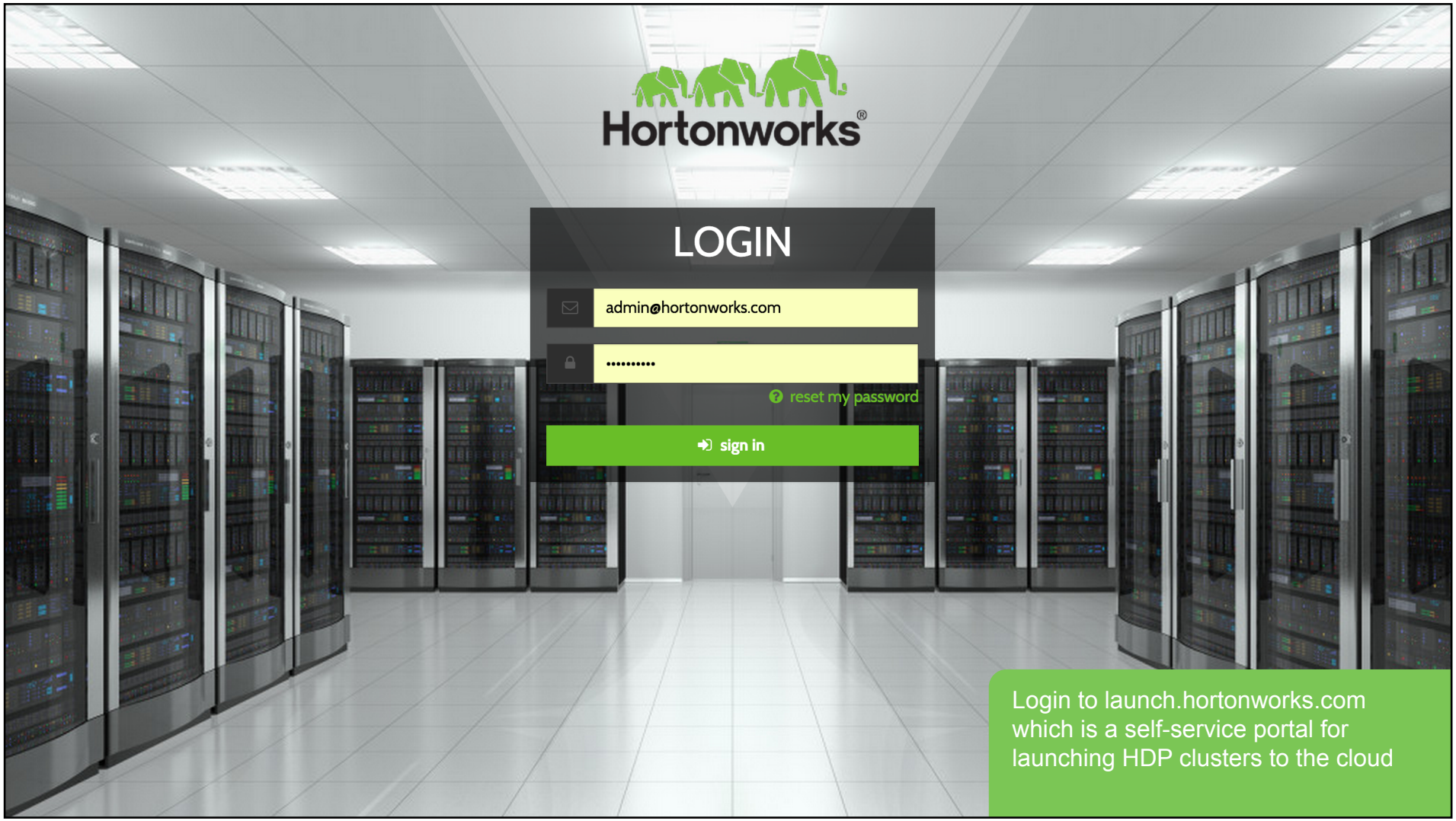
Use data science notebook to explore data

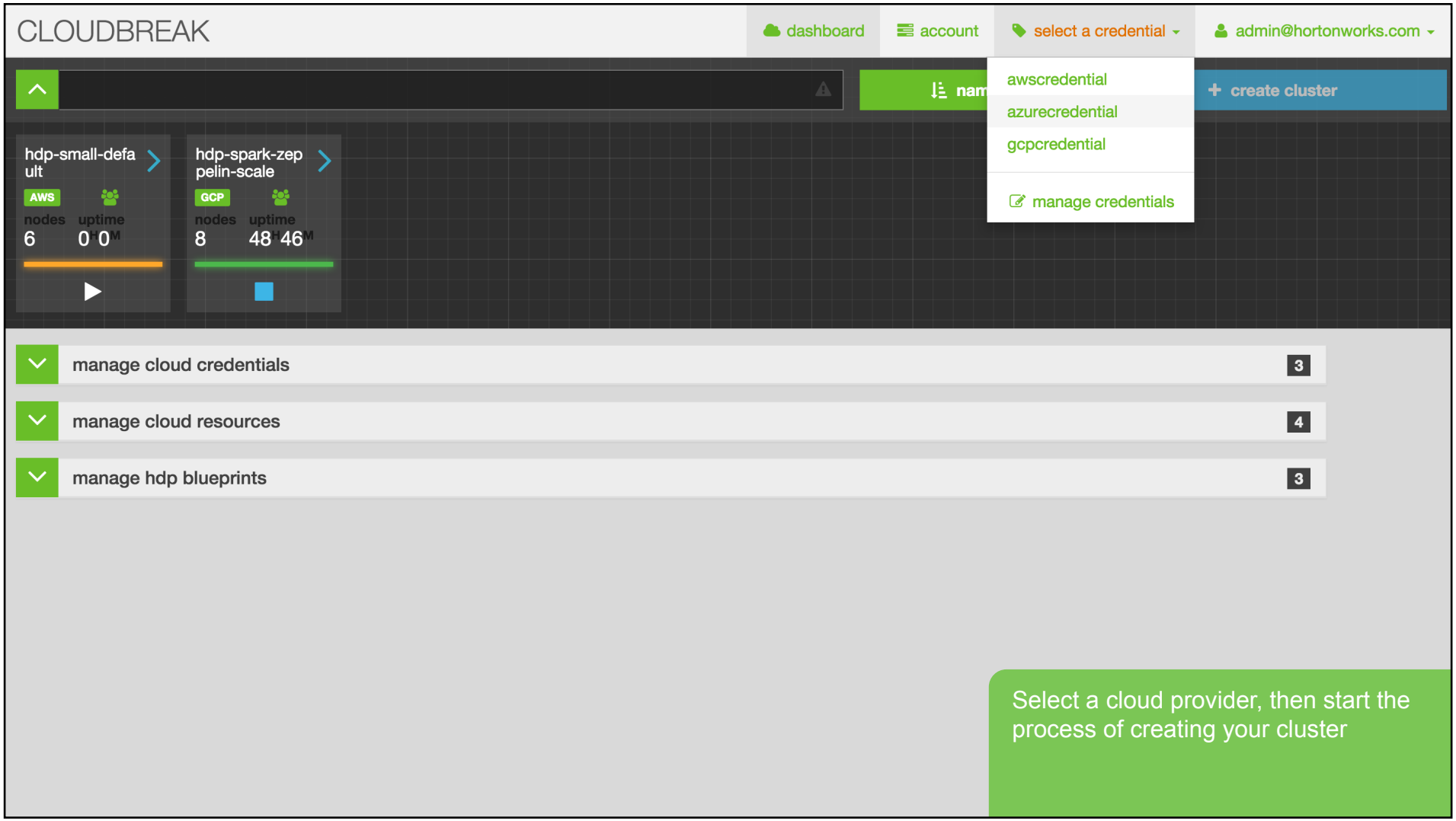
Run algorithms to create predictive model

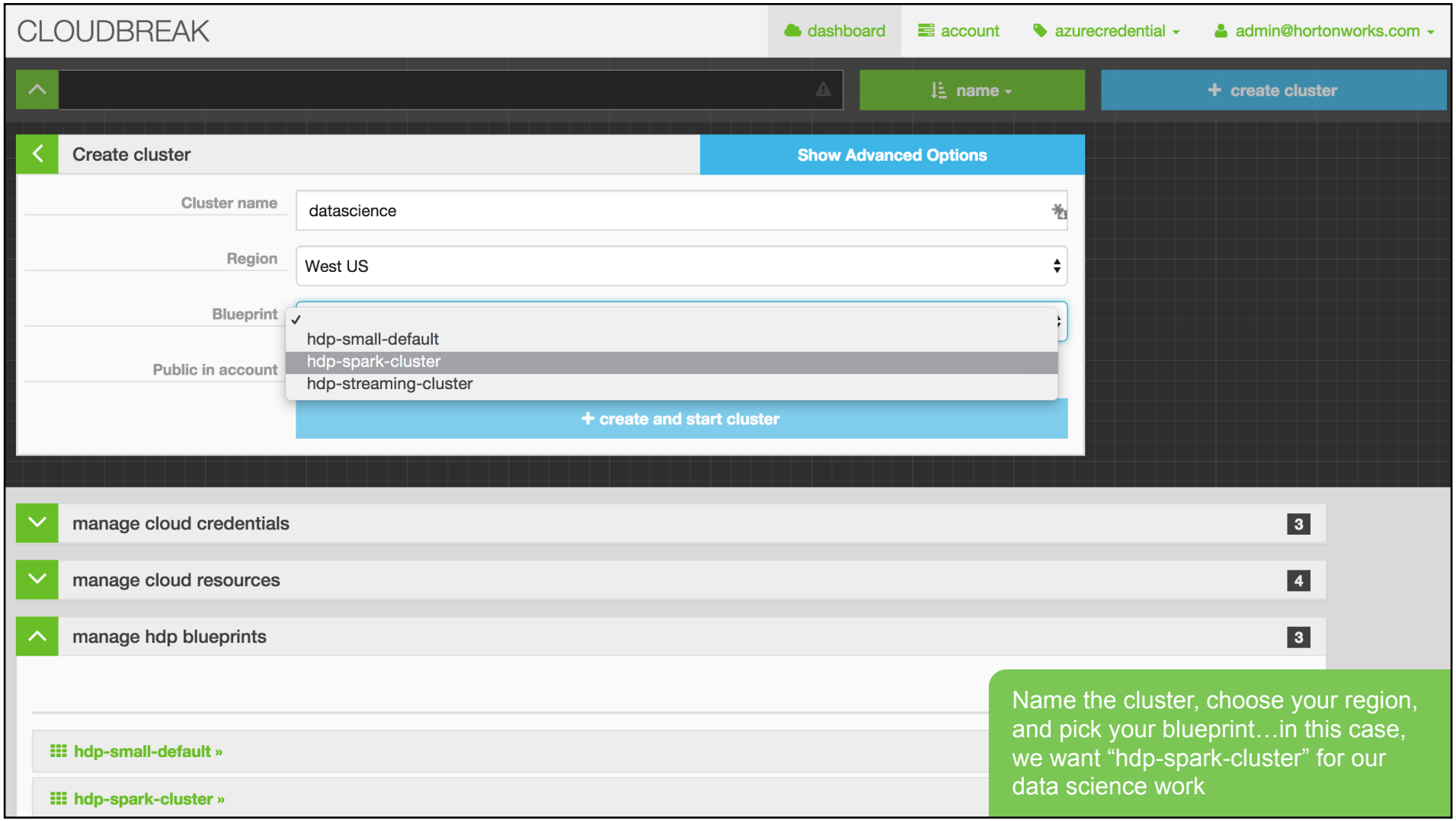


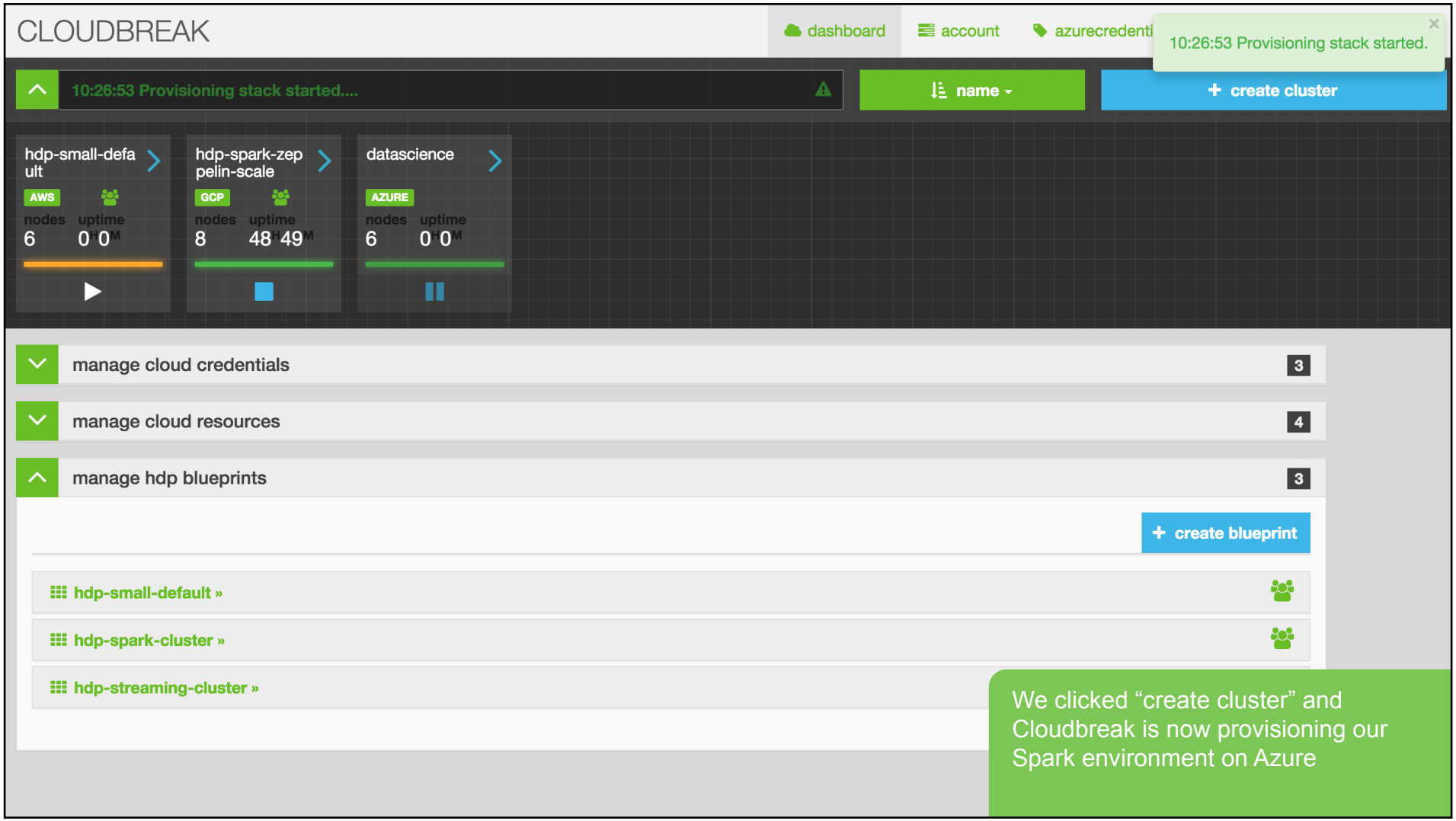



Hortonworks


Hortonworks
LOGIN
admin@hortonworks.com
reset my password
sign in
Login to launch.hortonworks.com
which is a self-service portal for
launching HDP clusters to the cloud


CLOUDBREAK
dashboard
account
select a credential
admin@hortonworks.com
awscredential
azurecredential
gcpcredential
manage credentials
create cluster
hdp-small-default
AWS
nodes
6
uptime
0 0
hdp-spark-zeppelin-scale
GCP
nodes
8
uptime
48 46
manage cloud credentials
3
manage cloud resources
4
manage hdp blueprints
3
Select a cloud provider, then start the process of creating your cluster


CLOUDBREAK
dashboard
account
azurecredential
admin@hortonworks.com
name
+ create cluster
Create cluster
Show Advanced Options
Cluster name
datascience
Region
West US
Blueprint
hdp-small-default
hdp-spark-cluster
hdp-streaming-cluster
Public in account
+ create and start cluster
manage cloud credentials
3
manage cloud resources
4
manage hdp blueprints
3
hdp-small-default »
hdp-spark-cluster »
Name the cluster, choose your region, and pick your blueprint…in this case, we want “hdp-spark-cluster” for our data science work


CLOUDBREAK
dashboard
account
azurecredenti
10:26:53 Provisioning stack started.
10:26:53 Provisioning stack started....
name
create cluster
hdp-small-default
AWS
nodes 6
uptime 0 0
hdp-spark-zeppelin-scale
GCP
nodes 8
uptime 48 49
datascience
AZURE
nodes 6
uptime 0 0
manage cloud credentials 3
manage cloud resources 4
manage hdp blueprints 3
create blueprint
hdp-small-default »
hdp-spark-cluster »
hdp-streaming-cluster »
We clicked “create cluster” and Cloudbreak is now provisioning our Spark environment on Azure

Zeppelin Notebook Interpreter Connected

FINISHED

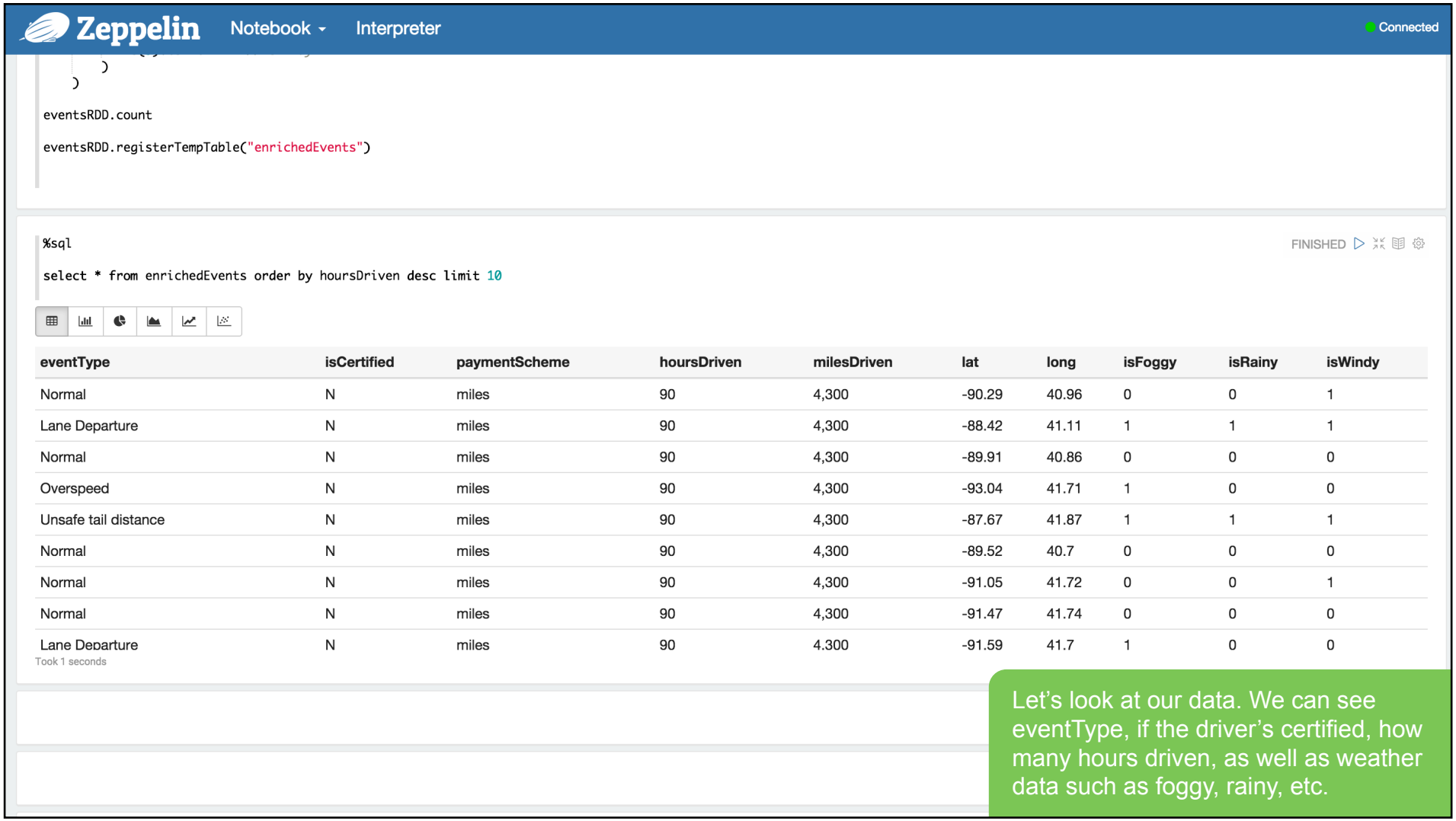
Let's do some Exploratory Data Analysis so that we have a better grasp on the nature of this data set.

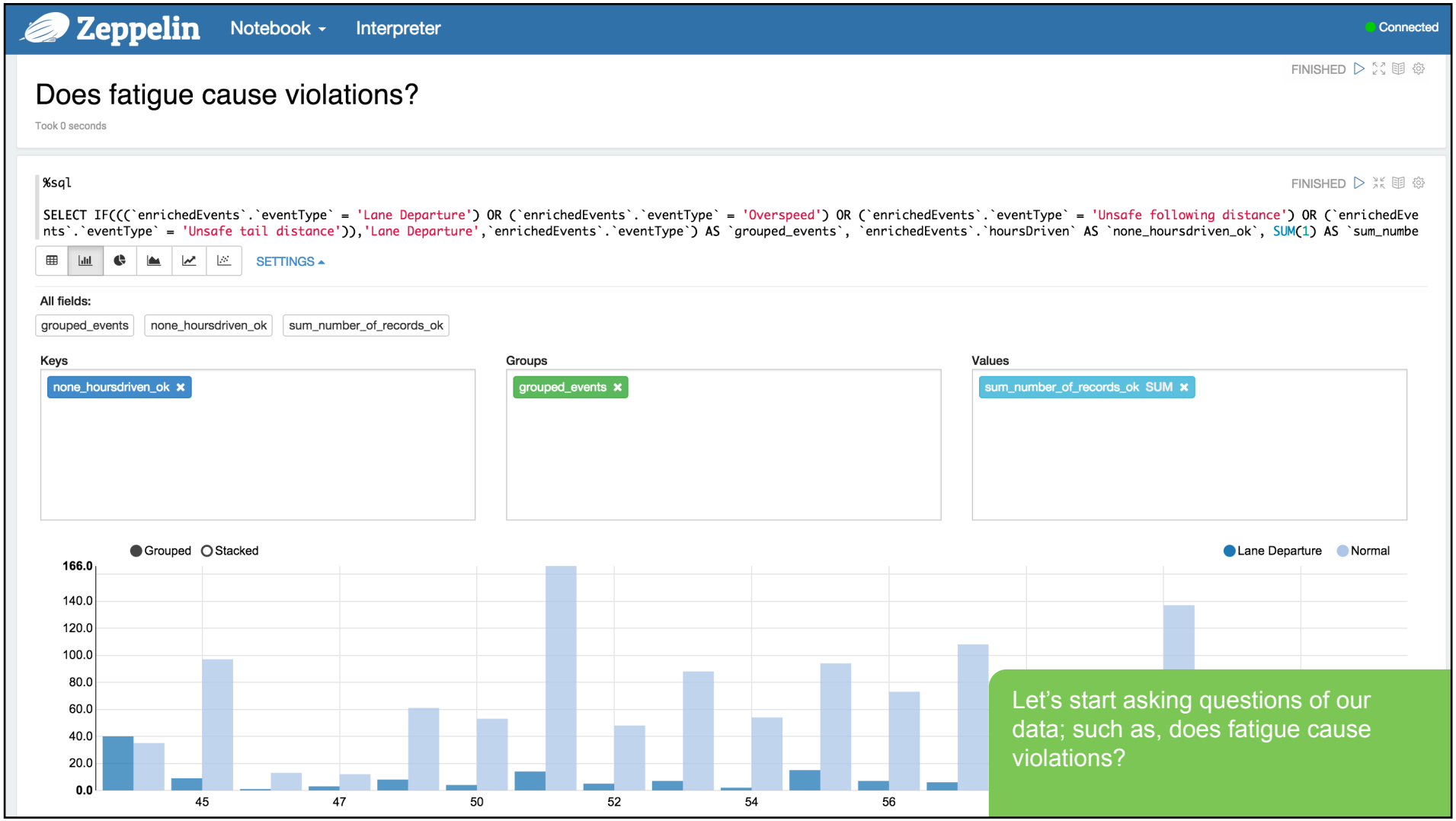
Took 0 seconds

FINISHED

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val eventsFile = sc.textFile("hdfs:///user/root/dhruv/keynoteDemo/enrichedEvents")

case class Event(eventType: String,
                 isCertified: String,
                 paymentScheme: String,
                 hoursDriven: Int,
                 milesDriven: Int,
                 lat: Float,
                 long: Float,
                 isFoggy: Int,
                 isRainy: Int,
                 isWindy: Int)

def toBool(in: String) = if (in == "1") true else false
def certToBool(in: String) = if (in == "Y") true else false
def scaleEvents(hrs: Int) = if (hrs = 0) hrs = 45

val eventsRDD = eventsFile.map(s => s.split(",")).map(
    s => Event(s(0),         //eventType
          s(1).replaceAll("\"", ""),  //isCertified
          s(2).replaceAll("\"", ""),  //paymentScheme
          scaleEvents(s(3).toInt),     //hoursDriven
          s(4).toInt,     //milesDriven
          s(5).toFloat,   //latitude
          s(6).toFloat,   //longitude
          s(7).toInt,     //isFoggy
          s(8).toInt,     //isRainy
          s(9).toInt      //isWindy
      )
    )

eventsRDD.count

eventsRDD.registerTempTable("enrichedEvents")
```

We can now access Zeppelin which is a data science notebook for Spark that's similar to iPython notebook


Zeppelin
Notebook
Interpreter
Connected
)
)
eventsRDD.count
eventsRDD.registerTempTable("enrichedEvents")
%sql
FINISHED
select * from enrichedEvents order by hoursDriven desc limit 10
eventType isCertified paymentScheme hoursDriven milesDriven lat long isFoggy isRainy isWindy
Normal N miles 90 4,300 -90.29 40.96 0 0 1
Lane Departure N miles 90 4,300 -88.42 41.11 1 1 1
Normal N miles 90 4,300 -89.91 40.86 0 0 0
Overspeed N miles 90 4,300 -93.04 41.71 1 0 0
Unsafe tail distance N miles 90 4,300 -87.67 41.87 1 1 1
Normal N miles 90 4,300 -89.52 40.7 0 0 0
Normal N miles 90 4,300 -91.05 41.72 0 0 1
Normal N miles 90 4,300 -91.47 41.74 0 0 0
Lane Departure N miles 90 4,300 -91.59 41.7 1 0 0
Took 1 seconds
Let's look at our data. We can see eventType, if the driver's certified, how many hours driven, as well as weather data such as foggy, rainy, etc.


Zeppelin
Notebook
Interpreter
Connected
FINISHED
Does fatigue cause violations?
Took 0 seconds
%sql
FINISHED
SELECT IF(((`enrichedEvents`.`eventType` = 'Lane Departure') OR (`enrichedEvents`.`eventType` = 'Overspeed') OR (`enrichedEvents`.`eventType` = 'Unsafe following distance') OR (`enrichedEve
nts`.`eventType` = 'Unsafe tail distance')),'Lane Departure',`enrichedEvents`.`eventType`) AS `grouped_events`, `enrichedEvents`.`hoursDriven` AS `none_hoursdriven_ok`, SUM(1) AS `sum_numbe
SETTINGS
All fields:
grouped_events
none_hoursdriven_ok
sum_number_of_records_ok
Keys
none_hoursdriven_ok
Groups
grouped_events
Values
sum_number_of_records_ok SUM
Grouped
Stacked
Lane Departure
Normal
166.0
140.0
120.0
100.0
80.0
60.0
40.0
20.0
0.0
45
47
50
52
54
56
Let's start asking questions of our data; such as, does fatigue cause violations?

Zeppelin Notebook Interpreter Connected

FINISHED

# Does fatigue cause violations?

Took 0 seconds

FINISHED

```
%sql

SELECT IF(((`enrichedEvents`.`eventType` = 'Lane Departure') OR (`enrichedEvents`.`eventType` = 'Overspeed') OR (`enrichedEvents`.`eventType` = 'Unsafe following distance') OR (`enrichedEve
nts`.`eventType` = 'Unsafe tail distance')),'Lane Departure',`enrichedEvents`.`eventType`) AS `grouped_events`, `enrichedEvents`.`hoursDriven` AS `none_hoursdriven_ok`, SUM(1) AS `sum_numbe
```

SETTINGS

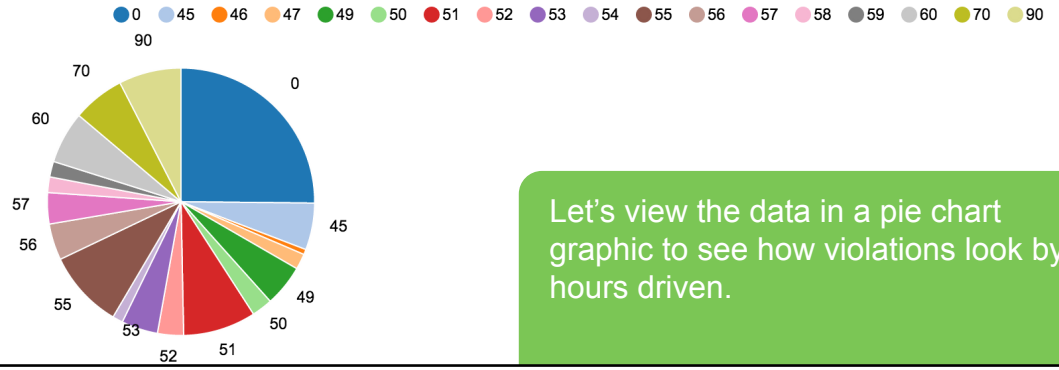
All fields:

grouped_events | none_hoursdriven_ok | sum_number_of_records_ok

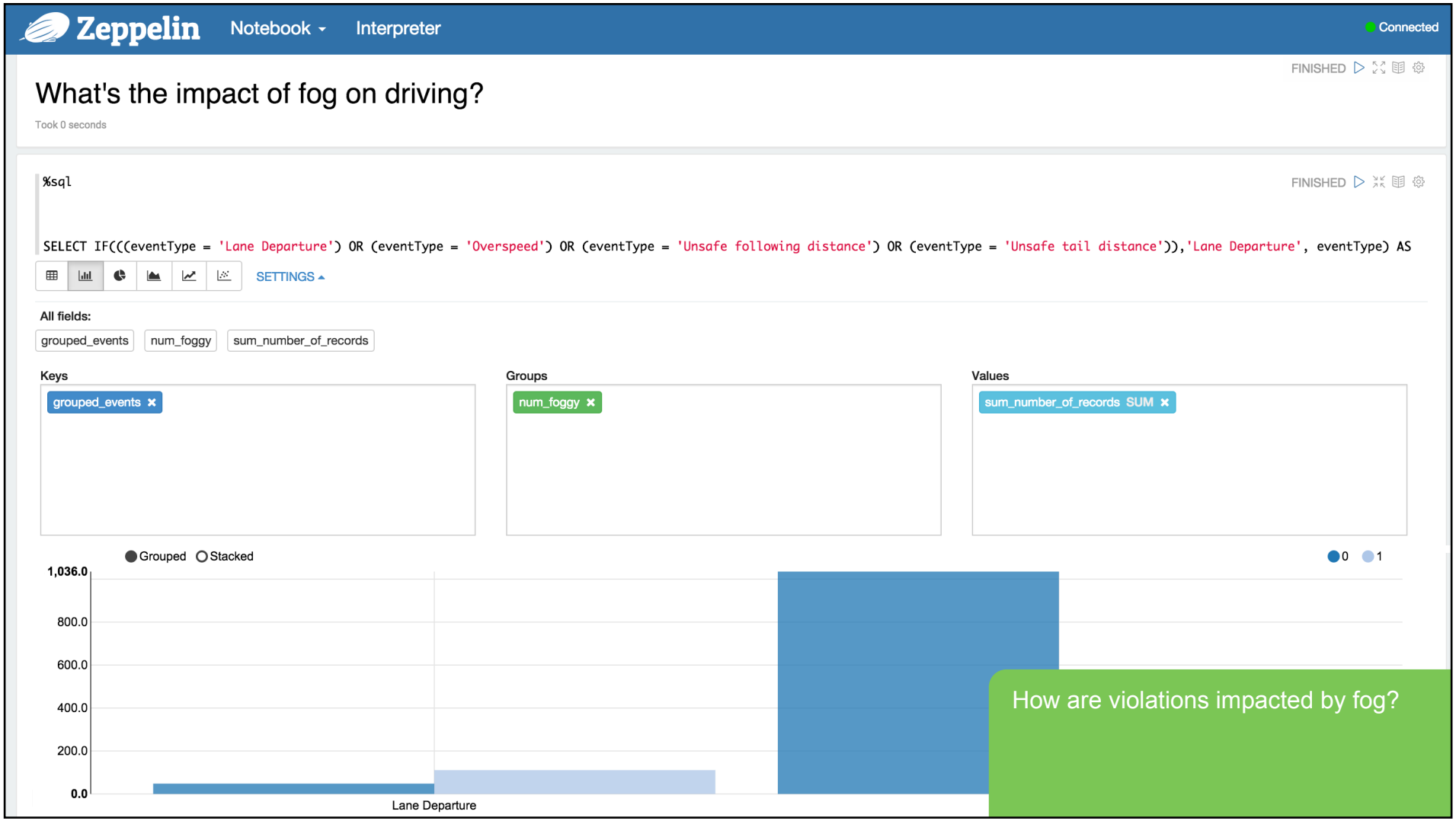
Keys: none_hoursdriven_ok

Groups: grouped_events

Values: sum_number_of_records_ok SUM




Let's view the data in a pie chart graphic to see how violations look by hours driven.


Zeppelin
Notebook
Interpreter
Connected
FINISHED
What's the impact of fog on driving?
Took 0 seconds
%sql
FINISHED
SELECT IF(((eventType = 'Lane Departure') OR (eventType = 'Overspeed') OR (eventType = 'Unsafe following distance') OR (eventType = 'Unsafe tail distance')),'Lane Departure', eventType) AS
SETTINGS
All fields:
grouped_events
num_foggy
sum_number_of_records
Keys
grouped_events
Groups
num_foggy
Values
sum_number_of_records SUM
Grouped
Stacked
0
1
1,036.0
800.0
600.0
400.0
200.0
0.0
Lane Departure
How are violations impacted by fog?


Zeppelin
Notebook
Interpreter
Connected
FINISHED
Does location have any impact on incidents?
Took 0 seconds
FINISHED
%sql
select lat, long, eventType, isFoggy from enrichedEvents where eventType not in ('Normal') and isFoggy = 1 group by lat, long, eventType, isFoggy
SETTINGS
All fields:
lat
long
eventType
isFoggy
xAxis
long
yAxis
lat
group
eventType
size
Overspeed
Unsafe following distance
Lane Departure
Unsafe tail distance
Unsafe tail distance
-87.67
-89.74
-92
-94
-95.99
34.78
35
35.5
36
36.5
37
37.5
38
38.5
39.1
39.5
40
40.5
41
41.5
42
42.25
Took 2 seconds
import org.apache.spark.mllib.util.MLUtils;
import org.apache.spark.mllib.optimization;
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD;
Does location have an impact on incidents?

Zeppelin Notebook Interpreter Connected

FINISHED

# Done with data analysis, let's build a regression model to predict violations

Took 0 seconds

FINISHED

```
import org.apache.spark.mllib.util.MLUtils;
import org.apache.spark.mllib.optimization;
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD;
import org.apache.spark.mllib.optimization.SquaredL2Updater;
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics;

    val examples = MLUtils.loadLabeledData(sc,"hdfs:///user/root/dhruv/keynoteDemo/trainingData").cache()

    val splits = examples.randomSplit(Array(0.8, 0.2))
    val training = splits(0).cache()
    val test = splits(1).cache()

    val numTraining = training.count()
    val numTest = test.count()
    println(s"Training: $numTraining, test: $numTest.")

    val updater = new SquaredL2Updater()

     val model = {
        val algorithm = new LogisticRegressionWithSGD()
        algorithm.optimizer.setNumIterations(200).setStepSize(1.0).setUpdater(updater).setRegParam(0.1)
        algorithm.run(training).clearThreshold()
     }

    val rprediction = model.predict(test.map(_.features))
    val rpredictionAndLabel = rprediction.zip(test.map(_.label))
    val rmetrics = new BinaryClassificationMetrics(rpredictionAndLabel)

    println("\n");
    println("Certification Weight " + model.weights(0))
    println("Wage Plan Weight " + model.weights(1))
    println("Fatigue by Hours Weight " + model.weights(2))
    println("Fatigue by Miles Weight " + model.weights(3))
    println("Foggy weather Weight " + model.weights(4))
    println("Rainy weather Weight " + model.weights(5))
    println("Windy weather Weight " + model.weights(6))
    println("\n")
    println("Intercept " + model.intercept)
    println(s"Test areaUnderPR = ${rmetrics.areaUnderPR()}.")
```

OK, we've learned enough about the data and what features we want to include in our model. So we'll run a logistic regression on training data.

Zeppelin Notebook Interpreter Connected

FINISHED

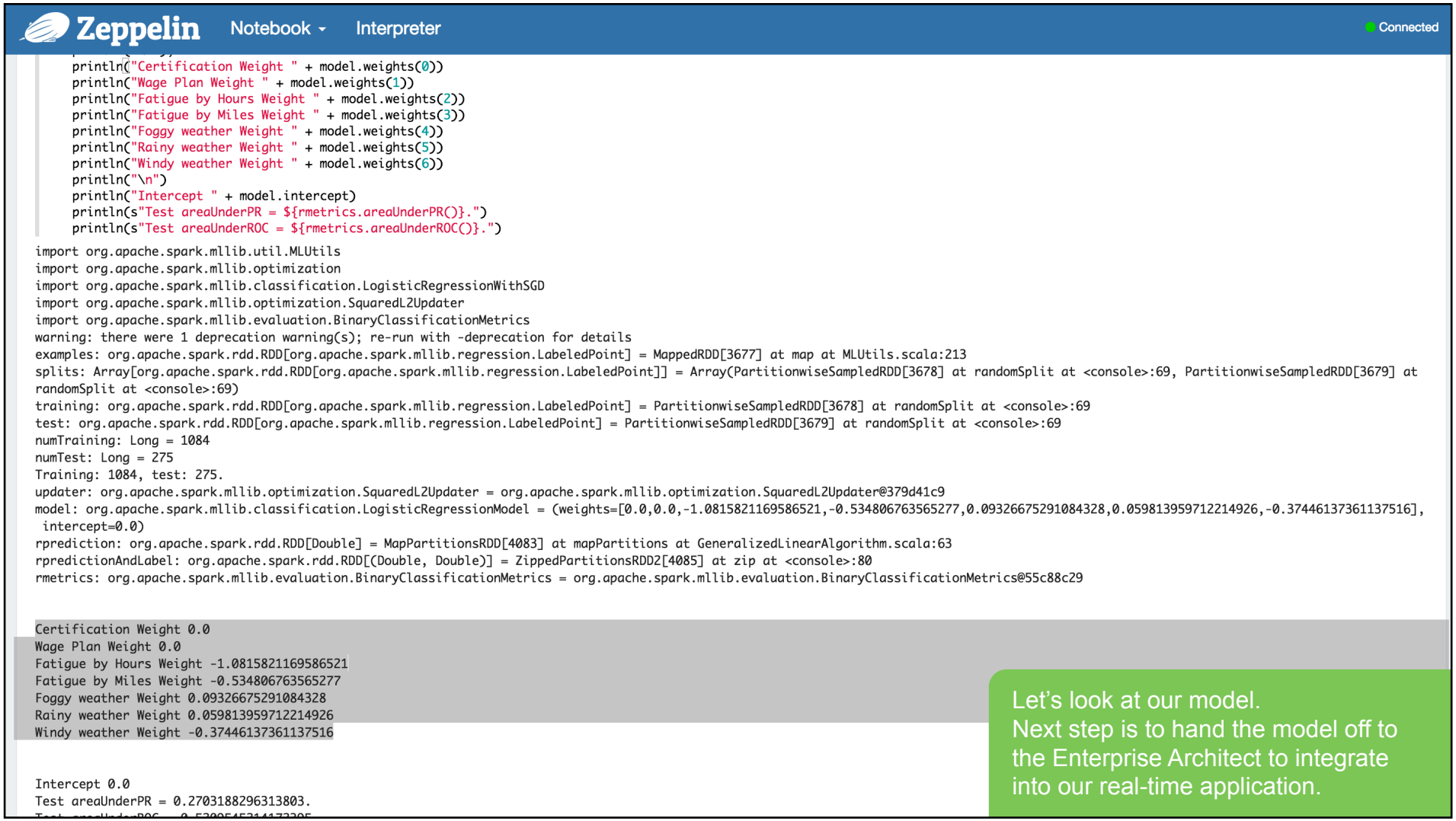
## Done with data analysis, let's build a regression model to predict violations

Took 0 seconds

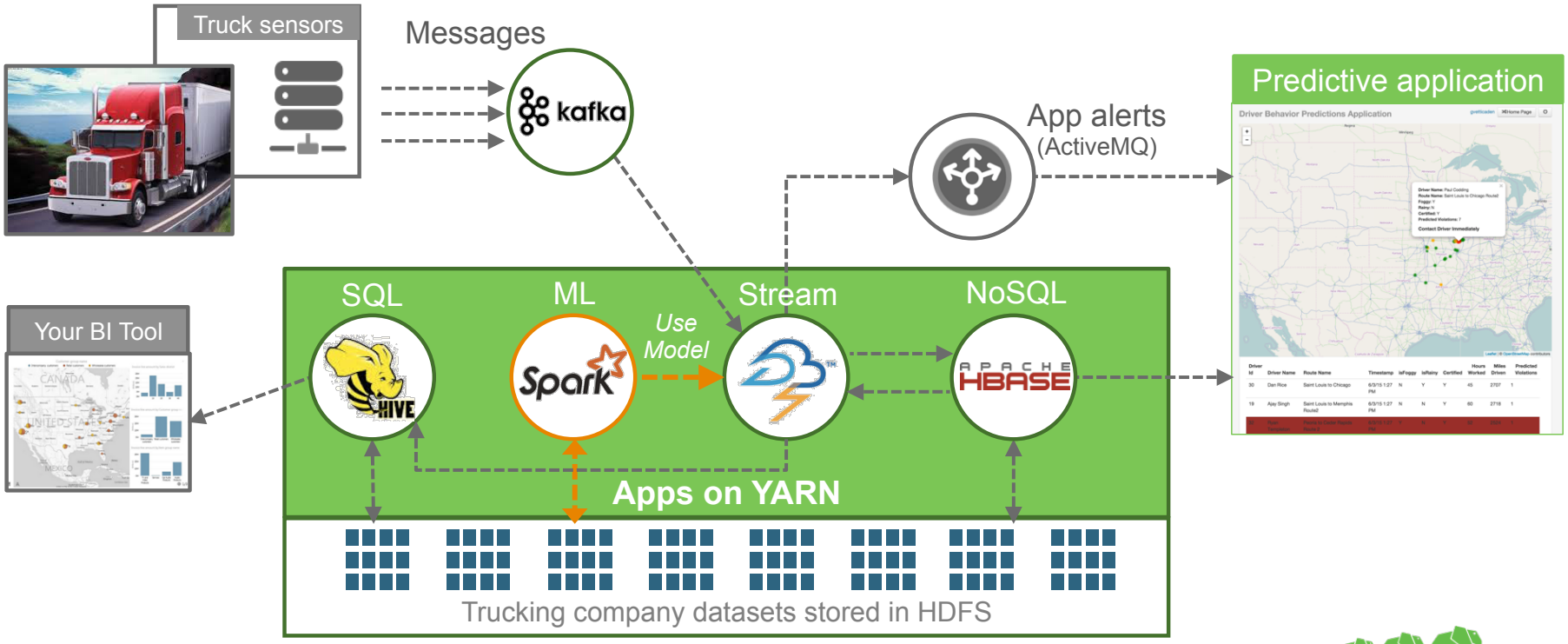
RUNNING 0%

```
import org.apache.spark.mllib.util.MLUtils;
import org.apache.spark.mllib.optimization;
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD;
import org.apache.spark.mllib.optimization.SquaredL2Updater;
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics;

    val examples = MLUtils.loadLabeledData(sc,"hdfs:///user/root/dhruv/keynoteDemo/trainingData").cache()

    val splits = examples.randomSplit(Array(0.8, 0.2))
    val training = splits(0).cache()
    val test = splits(1).cache()

    val numTraining = training.count()
    val numTest = test.count()
    println(s"Training: $numTraining, test: $numTest.")

    val updater = new SquaredL2Updater()

     val model = {
        val algorithm = new LogisticRegressionWithSGD()
        algorithm.optimizer.setNumIterations(200).setStepSize(1.0).setUpdater(updater).setRegParam(0.1)
        algorithm.run(training).clearThreshold()
     }

    val rprediction = model.predict(test.map(_.features))
    val rpredictionAndLabel = rprediction.zip(test.map(_.label))
    val rmetrics = new BinaryClassificationMetrics(rpredictionAndLabel)

    println("\n");
    println("Certification Weight " + model.weights(0))
    println("Wage Plan Weight " + model.weights(1))
    println("Fatigue by Hours Weight " + model.weights(2))
    println("Fatigue by Miles Weight " + model.weights(3))
    println("Foggy weather Weight " + model.weights(4))
    println("Rainy weather Weight " + model.weights(5))
    println("Windy weather Weight " + model.weights(6))
    println("\n")
    println("Intercept " + model.intercept)
    println(s"Test areaUnderPR = ${rmetrics.areaUnderPR()} ")
```

Let's run our code


Zeppelin
Notebook
Interpreter
Connected
println("Certification Weight " + model.weights(0))
println("Wage Plan Weight " + model.weights(1))
println("Fatigue by Hours Weight " + model.weights(2))
println("Fatigue by Miles Weight " + model.weights(3))
println("Foggy weather Weight " + model.weights(4))
println("Rainy weather Weight " + model.weights(5))
println("Windy weather Weight " + model.weights(6))
println("\n")
println("Intercept " + model.intercept)
println(s"Test areaUnderPR = ${rmetrics.areaUnderPR()}.")
println(s"Test areaUnderROC = ${rmetrics.areaUnderROC()}.")
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.optimization
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD
import org.apache.spark.mllib.optimization.SquaredL2Updater
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
warning: there were 1 deprecation warning(s); re-run with -deprecation for details
examples: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MappedRDD[3677] at map at MLUtils.scala:213
splits: Array[org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]] = Array(PartitionwiseSampledRDD[3678] at randomSplit at <console>:69, PartitionwiseSampledRDD[3679] at randomSplit at <console>:69)
training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = PartitionwiseSampledRDD[3678] at randomSplit at <console>:69
test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = PartitionwiseSampledRDD[3679] at randomSplit at <console>:69
numTraining: Long = 1084
numTest: Long = 275
Training: 1084, test: 275.
updater: org.apache.spark.mllib.optimization.SquaredL2Updater = org.apache.spark.mllib.optimization.SquaredL2Updater@379d41c9
model: org.apache.spark.mllib.classification.LogisticRegressionModel = (weights=[0.0,0.0,-1.0815821169586521,-0.534806763565277,0.09326675291084328,0.059813959712214926,-0.37446137361137516], intercept=0.0)
rprediction: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[4083] at mapPartitions at GeneralizedLinearAlgorithm.scala:63
rpredictionAndLabel: org.apache.spark.rdd.RDD[(Double, Double)] = ZippedPartitionsRDD2[4085] at zip at <console>:80
rmetrics: org.apache.spark.mllib.evaluation.BinaryClassificationMetrics = org.apache.spark.mllib.evaluation.BinaryClassificationMetrics@55c88c29
Certification Weight 0.0
Wage Plan Weight 0.0
Fatigue by Hours Weight -1.0815821169586521
Fatigue by Miles Weight -0.534806763565277
Foggy weather Weight 0.09326675291084328
Rainy weather Weight 0.059813959712214926
Windy weather Weight -0.37446137361137516
Intercept 0.0
Test areaUnderPR = 0.2703188296313803.
Let’s look at our model.
Next step is to hand the model off to the Enterprise Architect to integrate into our real-time application.

# Real-time and Predictive Application Architecture






Hortonworks®


HDP and Spark…
Perfect Together