# APACHE SPARK: EVERYTHING YOU NEED TO KNOW FROM A DEVELOPER IN THE TRENCHES

IBM Runtime Technologies

**Recurring theme…**

**Finding the valuable information amongst the masses of it!**

***Like panning for gold…***

## Important disclaimers

## What I'll cover...

- Brief introduction and my role
- Spark and the value proposition
- Worked examples
- IBM platform nuances
- Spark on the mainframe
- Machine learning value and caveats
- Real world example
- Sharing ideas
- How to find out more

## What it really comes down to?

- Open source distributed analytics and more
- Mix and match powerful features to get results fast – stick to the one runtime
- Information here is aimed at getting you up to speed with Spark and able to do useful things ASAP

## Brief introduction and my role

- **Software developer based here in Hursley**
- Working directly with Spark since November '14
- Conferences, meetups, quality evaluation, bug fixing, performance, customer problems, education, training support teams, fixing obscure bugs in the ecosystem…
- Deep in Spark/Java code daily
- Plenty of horror stories
- github.com/a-roberts
- https://www.linkedin.com/in/adam-c-roberts/

# But this talk is not...

A deep dive

Exclusively about performance

Touching on the horror stories to get to this point

A complete encyclopaedia of all things Spark!

## You'll be...

- Learning about IBM’s involvement
- Learning about the capabilities for your developers
  Data scientists, software developers, analysts…
- Learning realistic use cases and examples
- Learning the pros and cons and Spark from someone deep in the internals – honest opinions given here…
- Asking for more details! Ask or find me on LinkedIn, see my talks:
  *“DIY Analytics with Apache Spark”*
  *“Using GPUs to handle Big Data with Java”*
  *“Apache Spark Performance Observations”*

## What I want you to take away

- **Confidence** using Spark

  - yes, there are IBM offerings out there using but this is a technology talk and not a sales pitch!
- **Ideas** for your next projects
- **Knowledge** of what is and isn't feasible

## Audience that'll benefit the most?

- Totally new to Spark
- Heard of it, not quite ready to deploy
- Experimenting with it, want something concrete to chew on!

Info heavy – you will be receiving this deck! Useful reference material and advice

*Brief introduction and my role*

## For newcomers

Open source project (the most active for big data) offering distributed…

- **Machine learning**
- Graph processing
- Core operations (map, reduce, joins)
- In-memory caching

- **SQL queries with DataFrames/Datasets**

## Why Spark?

- ✓ Java/Scala/Python/R APIs
- ✓ Easy ML and scalability
- ✓ Versatility
- ✓ Active community
- ✓ `git clone ... && mvn -T 1C -Dskiptests package`

## Things Spark can handle

- ✓ *File formats you could use with Hadoop*
- ✓ *Anything there's a Spark package for*
- ✓ *json, csv, parquet…*
- ✓ *Data in DB2, IMS, VSAM files*

## Things Spark can be used with

- ✓ *Kafka for streaming*
- ✓ *Hive tables*
- ✓ *Cassandra as a database*
- ✓ *Hadoop (using HDFS with Spark)*
- ✓ *Notebooks (Jupyter, Zeppelin), R studio…*

## Not every problem is a Spark one!

- Can you get away with using spreadsheet software?
- Have you **really** got a large amount of data?
- Data preparation is **still** very important!

  How will you properly handle negative, null, or otherwise strange values in your data?
- Will you benefit from massive concurrency?
- Is the data in a format you can work with?
- Needs transforming first (and is it worth it)?

## What do we need to obtain value?

1) Data
2) Know how to write Java, Scala, Python or R
3) The more compute power the better
4) **Ideas**

## What can we realistically *expect*?

1) More informed decisions
   - Confirming or disproving **theories**
   - Brand new **insights**
2) Discovery of new facts
   - **Trend** identification
   - **Anomaly** identification

## Spark's versatility

1) ML
2) Streaming
3) SQL APIs
4) Core functions
5) Graph processing

## IBM's involvement

1) Contributors
2) Committers
3) Evangelism
4) Working with customers
5) New features and ecosystem
6) Use in our offerings

## “Sounds good, code examples?”

## Starting simple: finding a data source

- Your own data already in IMS/DB2/VSAM
- Your own plaintext files on disk
- **Kaggle's** a good place to practice...
- I'll be showing Scala code eventually

- Data I’ll process here is for educational purposes only: *road_accidents.csv*

- Data I'm using is licensed under the Open Government License for public sector information

*Worked examples*

## Features of the data ("columns")

"accident_index","vehicle_reference","**vehicle_type**","towing_and_articulation",
"**vehicle_manoeuvre**","vehicle_location",restricted_lane","junction_location","skidding_and_overturning","hit_object_in_carriageway","vehicle_leaving_carriageway","hit_object_off_carriageway","1st_point_of_impact","was_vehicle_left_hand_drive?","journey_purpose_of_driver","sex_of_driver","age_of_driver","age_band_of_driver","engine_capacity_(cc)","propulsion_code","age_of_vehicle","driver_imd_decile","driver_home_area_type","vehicle_imd_decile","NUmber_of_Casualities_unique_to_accident_ind ex","No_of_Vehicles_involved_unique_to_accident_index","location_easting_osgr","location_northing_osgr","longitude","latitude","police_force","accident_severity","number_of_vehicles","number_of_casualties","date","day_of_week","time","local_authority_(district)","local_authority_(highway)","1st_road_class","1st_road_number","road_type",
"speed_limit","junction_detail","junction_control"," 2nd_road_class","2nd_road_number","pedestrian_crossing-human_control","pedestrian_crossing-physical_facilities",
"light_conditions","**weather_conditions**","road_surface_conditions","special_conditions_at_site","carriageway_hazards","urban_or_rural_area","did_police_officer_attend_scene_of_accident","lsoa_of_accident_location","casualty_reference","casualty_class","sex_of_casualty","**age_of_casualty**","age_band_of_casualty","casualty_severity","pedestrian_location","pedestrian_movement","car_passenger","bus_or_coach_passenger","pedestrian_road_maintenance_worker","casualty_type","casualty_home_area_type","casualty_imd_decile"

## Values ("rows")

```
"201506E098757",2,9,0,18,0,8,0,0,0,0,3,1,6,1,45,7,1794
,1,11,-1,1,-1,1,2,384980,394830,-
2.227629,53.450014,6,3,2,1,"42250",2,1899-12-30
12:56:00,102,"E08000003",5,0,6,30,3,4,6,0,0,0,1,1,1,0,
0,1,2,"E01005288",NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA, NA,NA

"201506E098766",1,9,0,9,0,8,0,0,0,0,4,1,6,2,25,5,1582, 2,1,-1,-
1,-1,1,2,383870,394420,-
2.244322,53.446296,6,3,2,1,"14/03/2015",7,1899-12-30
15:55:00,102,"E08000003",3,5103,3,40,6,2,5,0,0,5,1,1,1
,0,0,1,1,"E01005178",NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA, NA,NA,NA
```

## Type of vehicle involved in the most accidents?

Spark way to figure this out?

*groupBy** **vehicle_type**

*sort*** the results on count

**vehicle_type** maps to a code

First place: car
Distant second: pedal bike
Close third: van/goods HGV <= 3.5 T
Distant last: electric motorcycle

## Which actions should I be extra careful with?

*groupBy** **manoeuvre**

*sort*** the results on count

**manoeuvre** maps to a code

First place: going ahead (!)
Distant second: turning right
Distant third: slowing or stopping
Last: reversing

## What weather should I be avoiding?

*groupBy* **weather_conditions**

*sort* the results on count

**weather_conditions** maps to a code

First place: fine with no high winds
Second: raining, no high winds
Distant third: fine, with high winds
Distant last: snowing, high winds

## *“Why * and **?”*

## *org.apache.spark functions that can run in a distributed manner*

# Age group with the most accidents?

**More complex Spark code example – I'm using Scala**

- Forced **mutability** consideration (val or var)
- Not mandatory to declare types (or “return ...”)
- Check out “*Scala for the Intrigued*” on YouTube
- JVM based

Scala main method I’ll be using

```
object AccidentsExample {

   def main(args: Array[String]) : Unit = {

   }
}
```

## Spark code...

```
val session = SparkSession.builder().appName("Accidents").master("local[*]")
```

## Creating a DataFrame: as though our data's in an SQL table

```
   val sqlContext =  session.getOrCreate().sqlContext

   val allAccidents = sqlContext.read.format("com.databricks.spark.csv").
   option("header",  "true").load(myHome  +  "/datasets/road_accidents.csv")
```

## allAccidents.show would give us a table like...

| accident_index | vehicle_reference | vehicle_type | towing_and_articulation |
|---|---|---|---|
| 201506E098757 | 2 | 9 | 0 |
| 201506E098766 | 1 | 9 | 0 |

## Group our data and save the result

```
val myAgeDF = groupCountSortAndShow(allAccidents, "age_of_casualty",  true)

myAgeDF.coalesce(1).write.option("header",
"true").format("csv").save("victims")

Runtime.getRuntime().exec("python plot_me.py" )
```

...

```
def groupCountSortAndShow(df: DataFrame, columnName: String, toShow:  Boolean):DataFrame = {
  val ourSortedData = df.groupBy(columnName).count().sort("count")
   if(toShow)
     ourSortedData.show()

   ourSortedData
 }
```

*“Hold on…
what’s that getRuntime().exec
stuff?!”*

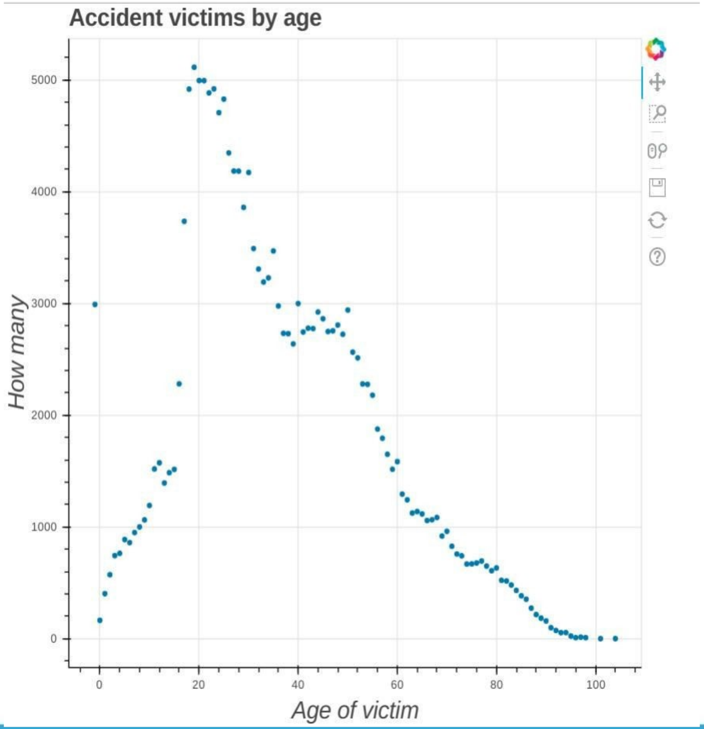
*Worked examples*

## It’s calling my Python code

```
import glob, os, pandas
from bokeh.plotting import figure, output_file, show
path = r'victims'
all_files = glob.glob(os.path.join(path, "*.csv"))

df_from_each_file = (pandas.read_csv(f) for f in all_files)

df = pandas.concat(df_from_each_file, ignore_index=True)

plot = figure(plot_width=640,plot_height=640,title='Accident victims by age',

  x_axis_label='Age of victim', y_axis_label='How many')

plot.title.text_font_size = '16pt'

plot.xaxis.axis_label_text_font_size = '16pt'

plot.yaxis.axis_label_text_font_size = '16pt'

plot.scatter(x=df.age_of_casualty, y=df['count'])

output_file('victims.html') show(plot)
```

*Worked examples*

## Bokeh gives us a graph like this



## Recap – what just happened?

1) DataFrame created from a CSV file
2) Grouped it by accident type
3) Count the number of elements per type
4) Save results to a different CSV file
5) Plot what's in that CSV file with Bokeh

## Why do it this way?

1) CSV file could be a whopper; it's small in my example
2) Spark functions used for the scalability
3) Easy to configure where this runs by modifying `.setMaster`

## How to set up a cluster?

1) SSH keys on machines to prevent password prompts
2) On the machine where the master will run add those machine names in `conf/slaves`
3) Run `sbin/start-master.sh`
4) Check web UI on port 8080 on the master

*“Where should I run Spark?”*

## Does it matter?

- **Linux on Intel**?

  Cheap and commonplace, great for serverless/containerised work
- **Linux on Power LE**?

  Raw compute power, FPGAs/GPUs/RDMA as well
- **Linux on Z**?

  Fast IO, raw compute power, RDMA, FPGAs
- **z/OS**?

  Security, access to data, access to HiperSockets

*“Tell me more about z/OS?”*

## Bringing Spark to the data

- Why go off-platform when running Java is cheap?
- IBM Machine Learning for z/OS
- IBM Open Data Analytics for Z
- Plenty of bug fixes required for this one (big-endian HW!)
- Look out for z/OS specific benefits:

  WLM and E2E encryption?

- SHARE workshops
- https://pages.github.ibm.com/zos-ecosystem/IzODA/blogs/
  a big thanks to Edrian Irizarry and Nicholas Marion
- What value can a systems programmer working with mainframes get from Spark?

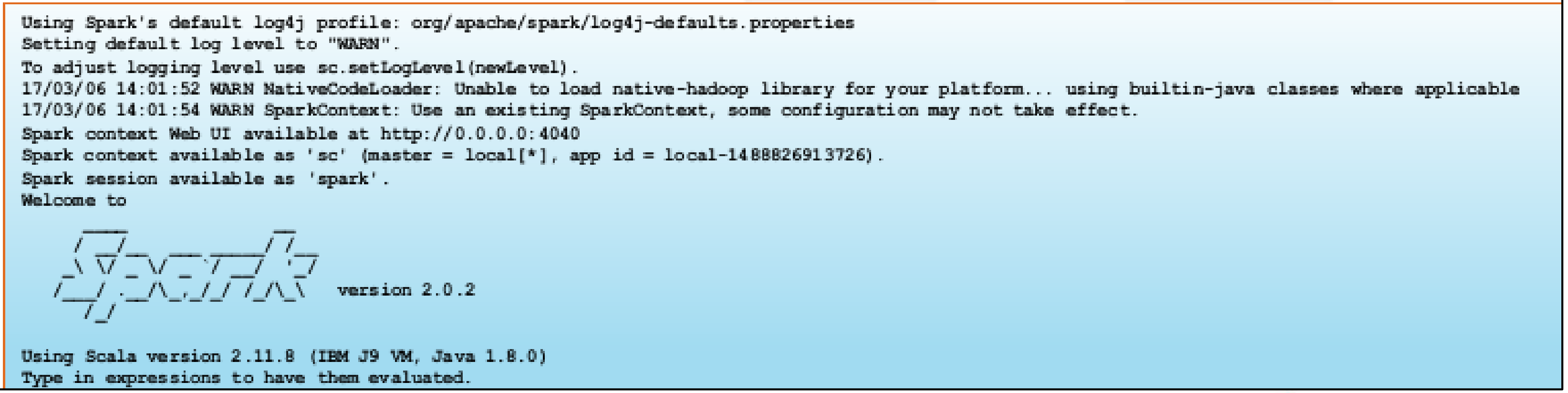
## Example for a sys prog, taken from SHARE

The scenario:

- Catch a run-away job that is generating SMF records and filling our logstream! SYSLOG isn’t helpful, we need to do some ad-hoc processing of SMF30 to figure this out
- SMF data is pre-staged in JSON files locally on a zFS

## First, we want to get into spark-shell

```
spark-shell
```



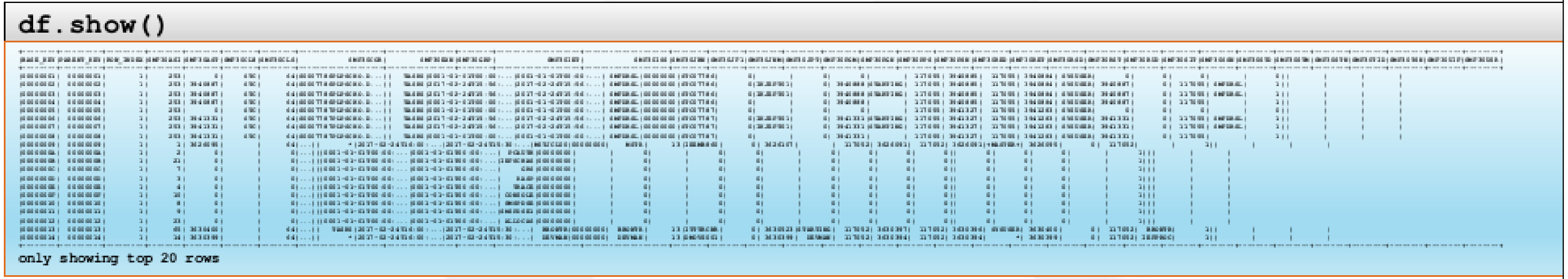
```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
17/03/06 14:01:52 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/03/06 14:01:54 WARN SparkContext: Use an existing SparkContext, some configuration may not take effect.
Spark context Web UI available at http://0.0.0.0:4040
Spark context available as 'sc' (master = local[*], app id = local-1488826913726).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.0.2
      /_/

Using Scala version 2.11.8 (IBM J9 VM, Java 1.8.0)
Type in expressions to have them evaluated.
```

Now that we’re in the spark-shell, lets load our data.

```
val df = spark.read.json("SMF30.json")
```

```
df: org.apache.spark.sql.DataFrame = [BASE_KEY: string, PARENT_KEY: string ... 32 more fields]
```

Using our existing spark session, Spark will read a json file and put it into a DataFrame called df.



```
df.show()
```

```
[illegible]
only showing top 20 rows
```

show() will display the contents of the DataFrame. Only 20 rows will be displayed.

Lets take a look at our headers.

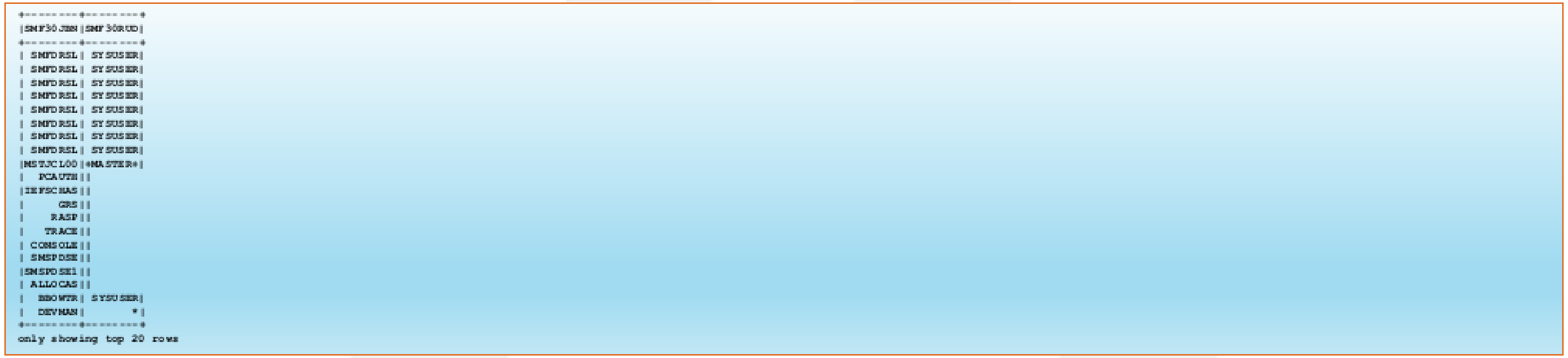
```
df.printSchema()
```

```
root
 |-- BASE_KEY: string (nullable = true)
 |-- PARENT_KEY: string (nullable = true)
 |-- ROW_INDEX: long (nullable = true)
 |-- SMF30ASI: long (nullable = true)
 |-- SMF30AST: long (nullable = true)
 ...
 |-- SMF30JBN: string (nullable = true)
 |-- SMF30JF1: string (nullable = true)
 |-- SMF30JNM: string (nullable = true)
 |-- SMF30JPT: long (nullable = true)
 |-- SMF30PGM: string (nullable = true)
 ...
 |-- SMF30RUD: string (nullable = true)
 |-- SMF30SIT: long (nullable = true)
 ...
```

printSchema() displays the headers of our DataFrame. In our DataFrame, we care about the Job Name (SMF30JBN) and the RACF User ID (SMF30RUD)

## Let's limit our data to only the columns we care about.

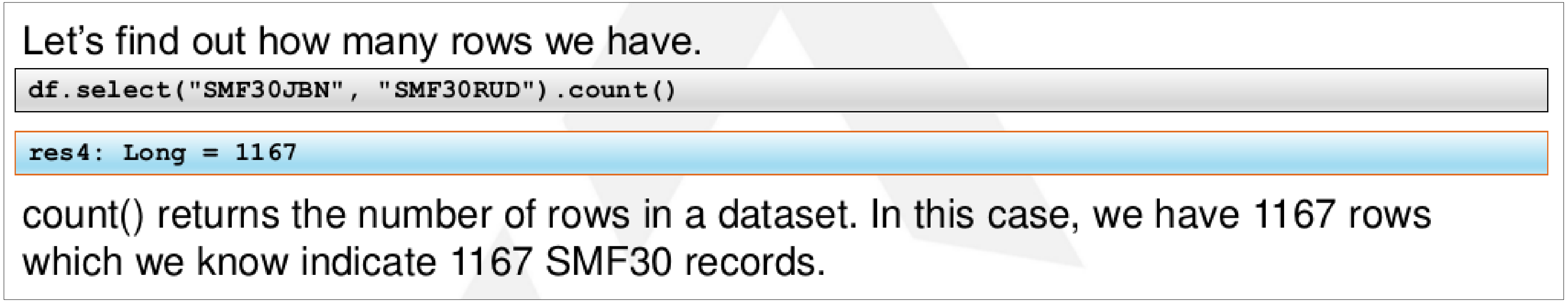
```
df.select("SMF30JBN", "SMF30RUD").show()
```



```
+--------+--------+
|SMF30JBN|SMF30RUD|
+--------+--------+
| SMFDRSL| SYSUSER|
| SMFDRSL| SYSUSER|
| SMFDRSL| SYSUSER|
| SMFDRSL| SYSUSER|
| SMFDRSL| SYSUSER|
| SMFDRSL| SYSUSER|
| SMFDRSL| SYSUSER|
| SMFDRSL| SYSUSER|
|MSTJCL00|*MASTER*|
|  PCAUTH||
|IEFSCHAS||
|     GRS||
|    RASP||
|   TRACE||
| CONSOLE||
| SMSPDSE||
|SMSPDSE1||
| ALLOCAS||
|  BBOWTR| SYSUSER|
|  DEVMAN|       *|
+--------+--------+
only showing top 20 rows
```

select() allows you to choose a column we're interested in.

## *Spark on the mainframe*



Let's find out how many rows we have.

```
df.select("SMF30JBN", "SMF30RUD").count()
```

```
res4: Long = 1167
```

count() returns the number of rows in a dataset. In this case, we have 1167 rows which we know indicate 1167 SMF30 records.

## Do we have duplicate Job Names and RACF User IDs?

```
val jobs = df.groupBy("SMF30JBN").count()
```

```
jobs: org.apache.spark.sql.DataFrame = [SMF30JBN: string, count: bigint]
```

```
val users = df.groupBy("SMF30RUD").count()
```

```
users: org.apache.spark.sql.DataFrame = [SMF30RUD: string, count: bigint]
```

groupBy() will group the Dataset using the specified column. By adding count() on the end, we create a new count column associated with each grouping.

## Let's order and display our data, first by JOB

```
jobs.orderBy(jobs.col("count").desc).limit(10).show()
```

```
+--------+-----+
|SMF30JBN|count|
+--------+-----+
| SATURN9|  116|
| SATURN1|  114|
| SATURN8|  114|
| SATURN4|  110|
| SATURN7|  110|
| SATURN6|  110|
…
```

orderBy() creates a new dataset sorted by the given expression. We sort by our new "count" column, in descending order. Limiting to the top 10.

## Now lets order by user

```
users.orderBy(users.col("count").desc).limit(10).show()
```

```
+--------+-----+
|SMF30RUD|count|
+--------+-----+
|  SATURN| 1004|
| SYSUSER|   22|
|  CANSID|   15|
|       *|   11|
| STASKID|   11|
|     TCP|   10|
...
```

It is pretty clear who caused the spike. The count is 50 times greater than the second highest.

## What did we find?

Looking back at our original problem.

- All these jobs came from one user ID
- The suffix of the jobnames as a random number
  - This indicates they were z/OS UNIX processes

Conclusion

- The user's script went into some kind of loop.

## How about machine learning?

Plenty of algorithms available

- These can run in a highly parallel way
- As with previous examples we'll be proving something simple
- Same process used each time

## Recipe for success with machine learning

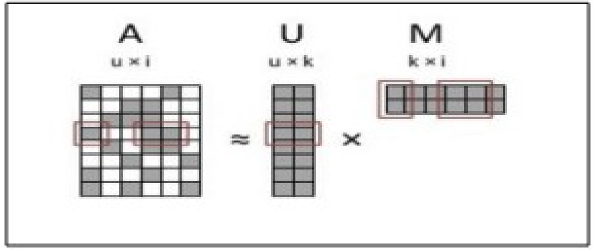
1) Define the objectives
2) Pick a suitable **algorithm** for said objective
3) Prepare the data
4) Separate into training and test data
5) Build a model with the training data (in parallel using Spark)
6) Use that model on the test data
7) Evaluate the model – test for accuracy
8) Experiment with parameters until it’s suitably accurate

## What's especially useful?

**Recommendation algorithms**

*Alternating Least Squares*

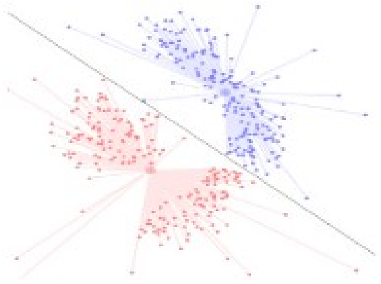
Movie recommendations on Netflix?
Recommended purchases on Amazon?
Similar songs with Spotify?
Recommended videos on YouTube?



**Clustering algorithms**

*K-means* (unsupervised learning (no labels, cheap))

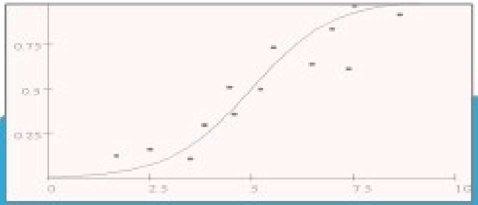
Produce n clusters from data to determine which cluster a new item can be categorised as
Identify anomalies: transaction fraud, erroneous data



**Classification algorithms**

*Logistic regression*

Create model that we can use to predict where to plot the next item in a sequence (above or below our line of best fit)
Healthcare: predict adverse drug reactions based on known interactions with similar drugs
Spam filter (binomial classification)
**Naive Bayes**

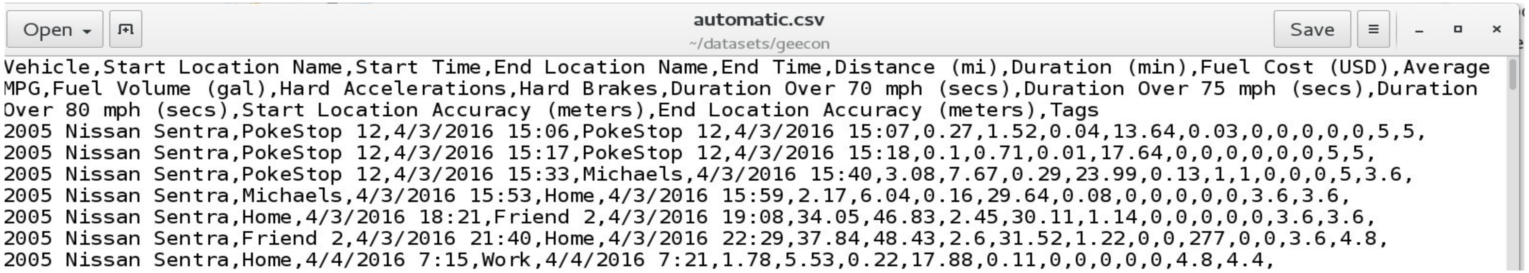


## *Machine learning example*

## Analysing data from wearable devices

- Colleague travelling the US for four years
- Wearing devices that track location (including on foot), physical activity, car journeys
- Can we process this data to gain some **meaningful insights about the person?**

## Exploring the data first



```
Vehicle,Start Location Name,Start Time,End Location Name,End Time,Distance (mi),Duration (min),Fuel Cost (USD),Average
MPG,Fuel Volume (gal),Hard Accelerations,Hard Brakes,Duration Over 70 mph (secs),Duration Over 75 mph (secs),Duration
Over 80 mph (secs),Start Location Accuracy (meters),End Location Accuracy (meters),Tags
2005 Nissan Sentra,PokeStop 12,4/3/2016 15:06,PokeStop 12,4/3/2016 15:07,0.27,1.52,0.04,13.64,0.03,0,0,0,0,0,5,5,
2005 Nissan Sentra,PokeStop 12,4/3/2016 15:17,PokeStop 12,4/3/2016 15:18,0.1,0.71,0.01,17.64,0,0,0,0,0,0,5,5,
2005 Nissan Sentra,PokeStop 12,4/3/2016 15:33,Michaels,4/3/2016 15:40,3.08,7.67,0.29,23.99,0.13,1,1,0,0,0,5,3.6,
2005 Nissan Sentra,Michaels,4/3/2016 15:53,Home,4/3/2016 15:59,2.17,6.04,0.16,29.64,0.08,0,0,0,0,0,3.6,3.6,
2005 Nissan Sentra,Home,4/3/2016 18:21,Friend 2,4/3/2016 19:08,34.05,46.83,2.45,30.11,1.14,0,0,0,0,0,3.6,3.6,
2005 Nissan Sentra,Friend 2,4/3/2016 21:40,Home,4/3/2016 22:29,37.84,48.43,2.6,31.52,1.22,0,0,277,0,0,3.6,4.8,
2005 Nissan Sentra,Home,4/4/2016 7:15,Work,4/4/2016 7:21,1.78,5.53,0.22,17.88,0.11,0,0,0,0,0,4.8,4.4,
```

```scala
val autoData = sqlContext.read.format("com.databricks.spark.csv").
  option("header", "true").
  option("inferSchema", "true").
  load(myHome + "/datasets/geecon/automatic.csv").
  withColumnRenamed("End Location Name", "Location").
  withColumnRenamed("End Time", "Time")
```

## Checking our data is sensible

```
val colsWeCareAbout =
  "Distance (mi)",
  "Duration (min)",
  "Fuel Cost (USD)")

for (col <- colsWeCareAbout) {
  summarise(autoData, col)
}
```

```
def summarise(df: DataFrame, columnName: String)
    {averageByCol(df, columnName)
    minByCol(df, columnName)
    maxByCol(df, columnName)
  }

  def averageByCol(df: DataFrame, columnName: String)
    {println("Printing the average " + columnName)
    df.agg(avg(df.col(columnName))).show()
  }

  def minByCol(df: DataFrame, columnName: String)
    {println("Printing the minimum " + columnName)
    df.agg(min(df.col(columnName))).show()
  }

  def maxByCol(df: DataFrame, columnName: String)
    {println("Printing the maximum " + columnName)
    df.agg(max(df.col(columnName))).show()
  }
```

*Average distance (in miles): 6.88, minimum: 0.01, maximum: 187.03*
*Average duration (in minutes): 14.87, minimum: 0.2, maximum: 186.92*
*Average fuel Cost (in USD): 0.58, minimum: 0.0, maximum: 14.35*

## Preparing the data

**Let's say we want to analyse the subject's sleep**

*If duration is less than eight hours*
good sleep = true, else false (*I'm using 1 for true and 0 for false)*

Then we'll determine the most influential features on the value being true or false. This can reveal the interesting stuff!

We'll first check there are **any** recorded good sleeps

```
val onlyDurations = sleepData.select("s_duration")
val NUM_HOURS = 8
val THRESHOLD = 60 * 60 * NUM_HOURS
val onlyGoodSleeps = onlyDurations.filter($"s_duration" >= THRESHOLD)


// Don't care if the sleep duration wasn't even recorded or it's 0
 val onlyRecordedSleeps = onlyDurations.filter($"s_duration" > 0)

 println("Found " + onlyRecordedSleeps.count() + " valid recorded " +
   "sleep times and " + onlyGoodSleeps.count() +"
   were " + NUM_HOURS + " or more hours in
   duration")
```

```
Found 538 valid recorded sleep times and 129 were 8 or more
                   hours in duration
```

## Biggest influences on sleep length?

ChiSq algorithm used

| Feature | Description from Jawbone API docs |
|---|---|
| s_count | Number of primary sleep entries logged |
| s_awake_time | Time the user woke up |
| s_quality | Proprietary formula, don't know |
| s_asleep_time | Time when the user fell asleep |
| s_bedtime | Seconds the device is in sleep mode |
| s_deep | Seconds of main “sound sleep” |
| s_light | Seconds of “light sleeps” during the sleep period |
| m_workout_time | Length of logged workouts in seconds |
| n_light | Seconds of light sleep during the nap |
| n_sound | Seconds of sound sleep during the nap |

## Correlations?

Dataset has these features

- **s_duration**: how long was the subject sleeping for?
- *m_active_time: how long was the subject considered active for?*
- *m_calories: how many calories were burned during an activity?*
- *m_distance: how far did the subject travel?*
- *m_steps: how many steps did the subject make?*
- *m_total_calories: total number of calories burned*
- ***n_bedtime (hmm)**: what time did they go to bed**?***
- *n_awake_time: what time did they wake?*

https://jawbone.com/up/developer/types

```
val compareToCol = "s_duration"
for (col <- sleepData.columns) {
   If (! col.equals(compareToCol)) { // don't compare to itself...
      val corr = sleepData.stat.corr(col, compareToCol)

      if (corr > 0.8) {
         println("Very strong positive correlation for " + col + " and " +
            compareToCol)
      } else if (corr >= 0.5) {
         println("Positive correlation for " + col + " and " + compareToCol)
      }
   }
}
```

```
Very strong positive correlation for n_bedtime and s_asleep_time
```

## And something we *know* isn't related?

```
Val shouldBeLow = sleepData.stat.corr("goal_body_weight", "s_duration")
println("Correlation between goal body weight and sleep duration: " + shouldBeLow)
```

```
Correlation between goal_body_weight and
  s_asleep time: -0.02
```

## Where would this be useful?

*Relationship between elements in that big table of yours?*

***Easy ones: when X goes up, does Y aswell?***

*When the temperature goes up do our revenues increase at the mall?*

*When the temperature goes down do coat sales go up?*

*When it's NFL season do jersey sales go up?*

***More sophisticated?***

*When X stock goes down, does Y's stock go up?*

*If it does, under what conditions?*

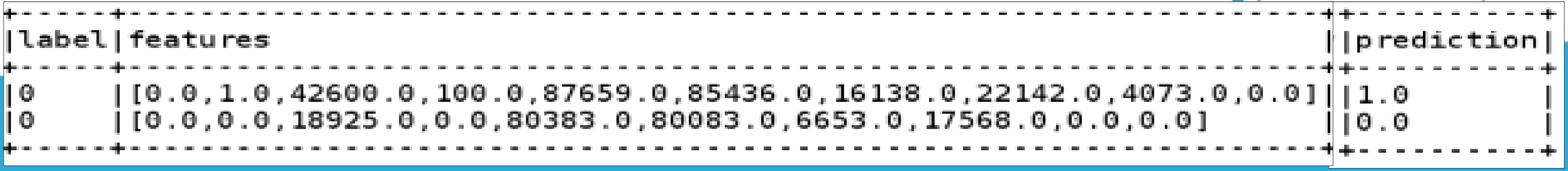
## Predicting a good night's sleep?

```
val Array(trainingSleepData, testSleepData) = thePreparedData.randomSplit(Array(0.7, 0.3))

val sleepModel = new NaiveBayes().fit(trainingSleepData)

val predictions = sleepModel.transform(testSleepData)

val evaluator = new MulticlassClassificationEvaluator()
.setLabelCol("label")
.setPredictionCol("prediction")
.setMetricName("accuracy")

val accuracy = evaluator.evaluate(predictions)
println("Test set accuracy for labelled sleep data = " + accuracy)
```

```
Test set accuracy for labelled sleep data = 0.81 ...
```

## Testing the model with new data

```
val somethingNew = sqlContext.createDataFrame(Seq(

 // Good sleep: high workout time, achieved a good amount of deep sleep, went to bed
 after midnight and woke at almost noon!

 (0, Vectors.dense(0, 1, 42600, 100, 87659, 85436, 16138, 22142, 4073, 0)),

 // Bad sleep, woke up early (5 AM), didn't get much of a deep sleep, didn't workout,
 bedtime 10.20 PM

 (0, Vectors.dense(0, 0, 18925, 0, 80383, 80083, 6653, 17568, 0, 0))

)).toDF("label","features")

sleepModel.transform(somethingNew).show()
```



```
+-----+--------------------------------------------------------------+  +----------+
|label|features                                                      |  |prediction|
+-----+--------------------------------------------------------------+  +----------+
|0    |[0.0,1.0,42600.0,100.0,87659.0,85436.0,16138.0,22142.0,4073.0,0.0]|  |1.0       |
|0    |[0.0,0.0,18925.0,0.0,80383.0,80083.0,6653.0,17568.0,0.0,0.0]  |  |0.0       |
+-----+--------------------------------------------------------------+  +----------+
```

## Why does predicting an outcome matter?

Informed decisions leading to **preventive** measures instead of **proactive** ones

- Logistical planning?
- Emergency responses?
- Catering to high demand?

## That NaiveBayes algorithm?

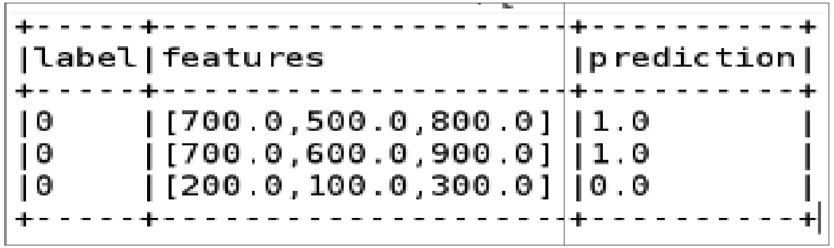
- Powerful algorithm that's easy to try out
- Using evidence provided, guess what a label will be (1 or 0)

bayes_data.txt (libSVM format)

```
1 1:700 2:525 3:825
0 1:150 2:140 3:410
1 1:825 2:550 3:900
0 1:245 2:225 3:315
0 1:200 2:125 3:400
1 1:700 2:525 3:825
0 1:150 2:140 3:410
1 1:825 2:550 3:900
```

0 = the label (category 0 or 1), e.g. 0 = *low scoring athlete*, *1 = high scoring*, *so 1*:**x** = the score for a sporting event 1
2:**x** = the score for a sporting event 2
3:**x** = the score for a sporting event 3 etc

## Adding new data to test it



```
+-----+--------------------+-----------+
|label|features            |prediction|
+-----+--------------------+-----------+
|0    |[700.0,500.0,800.0] |1.0        |
|0    |[700.0,600.0,900.0] |1.0        |
|0    |[200.0,100.0,300.0] |0.0        |
+-----+--------------------+-----------+
```

Naive Bayes correctly classifies the data (giving it the right labels)

“How about a real world application?”

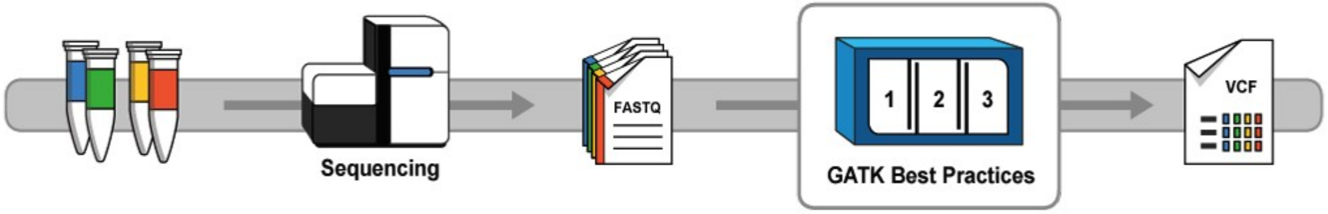
Real world example

## Genomics Analytics Toolkit (GATK)

*https://software.broadinstitute.org/gatk/download/beta*

BROAD INSTITUTE

**MIT/Harvard - world's leading biomedical and genomic research center**



Real world example

**High-throughput Sequencing (HTS) data analysis workflow**

- Pipeline of a variety of tools with a primary focus on variant discovery and genotyping
- 31,000 registered users of GATK (providers, clinicians, research centers, focused on **personalized medicine**)
- Requires extensive local compute and storage infrastructure to process vast amount of data required to conduct personalized analysis

- **Code/data owner:** Broad Institute (open source; GATK4 (Spark-enabled) currently in **Alpha version**); reference input data shared with collaborators
- **Taxonomy:** elasticity, compute and I/O bottlenecks, genomics, Spark
- **APIs/platform:** Standard Spark API (scheduler-agnostic); components calling native code needs porting
- **IBM team:** Carlos Costa (WRC), Yinhe Cheng (Life Sciences, Systems)**,** Frank Liu (ARL)
- **Product alignment:** Watson Health Cloud

Real world example

## GATK4: Spark based

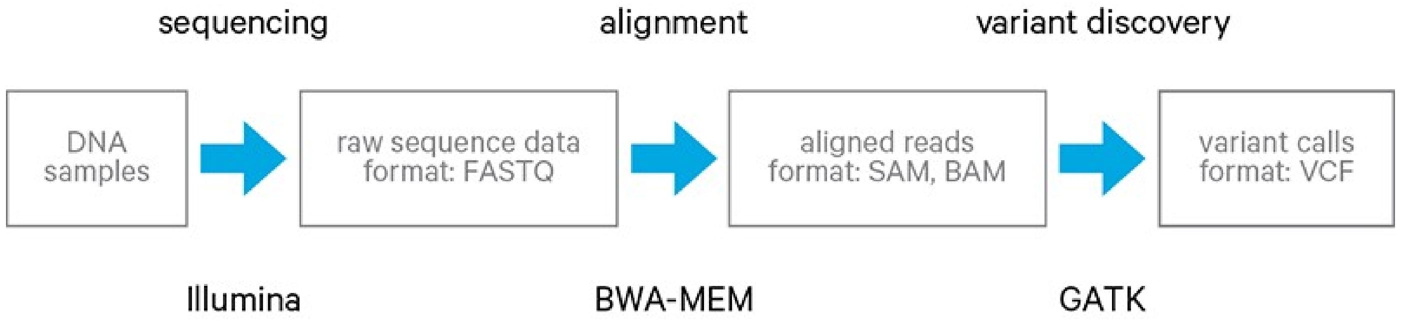
- Spark being used to facilitate parallelism and in-memory computation to speedup methods
- Broad Institute working with collaborators to develop **cloud-hosted** options to expand access and facilitate usage
- Collaborators to provide cloud services enabling GATK4 through **software-as-a-service (SaaS)** mechanism
- Critical to accelerate precision medicine by lowering the **cost** of genome sequencing
- GATK4 will extend the range of use cases supported to include cancer, structural variation, copy number variation, and more
- Collaborators closely interacting with GATK developers and influencing the project
- Genome analysis can be seen as one step in potentially larger workflows applying machine learning to personalized **treatment plans**

**GATK Collaboration Members**

https://www.broadinstitute.org/news/8065

Real world example

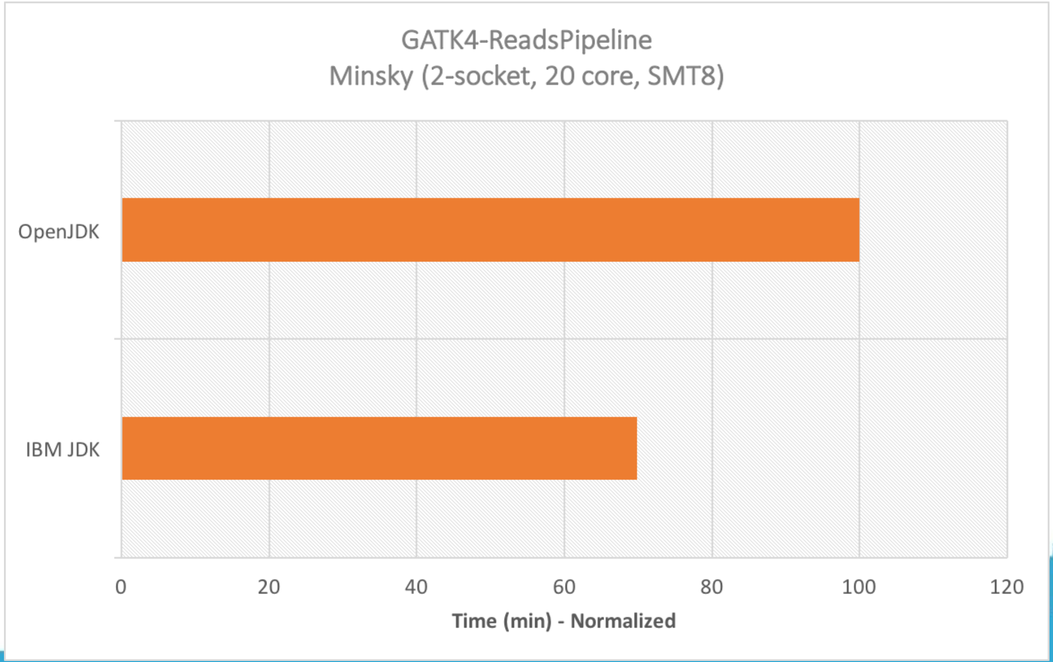
## GATK4 on POWER


sequencing
alignment
variant discovery
DNA samples
raw sequence data format: FASTQ
aligned reads format: SAM, BAM
variant calls format: VCF
Illumina
BWA-MEM
GATK

- On-going effort to port, profile and accelerate end-to-end pipeline for whole genome on POWER
- Most of major methods (including low-level dependencies) running on POWER
- Detailed profiling of scalability of steps/methods
- Significant speedup with tuning IBM's Spark package on POWER

Real world example

## GATK4 performance



**POWER8 Minsky**
**8335-GTB**

**2-socket, 10-core SCM, SMT8**
IB EDR (100 GB/s)

Real world example

## IBM, Spark and Wimbledon

- Command centre featured Spark streaming using the PageRank algorithm to determine which Tweeters were the most influential
- Used in production for **Wimbledon 2016**
- Watch the video **here**

*“So why isn't everybody using Spark?”*

## Use cases Spark really *sucks* at...

- Embarrassingly parallel and not much data? Use GPUs
- Tiny short-lived workloads (JVM startup cost)
- Trivial workloads (spreadsheet software    )
- Applications relying on some other slow service provider
- Ones we don't have the ML algorithm for in Spark!

## Use cases Spark *excels* at

- PoCs; mix and match APIs, get things done fast
- Data science – exploration, discovery
- Building scalable ML applications
- Database interrogation

## Ideas by sector

- **Banking**

  Reduce churn
- **Healthcare**

  Drug-drug interactions
- **Automotive**

  Processing vast amounts of data e.g. from self-driving cars?

  Heard they'll generate 1 GB data per sec!

**Stock trading insights with Linux on Z**
YouTube link

**Churn reduction example on z/OS**
***YouTube link***

- **Network security**

  Known list of rules for bad traffic; find in your logs
- **Automation**

  Process logs to retrieve only anomalies
- **Data mining**

  Finding the “golden nuggets” in vast data sources
- **Hypothesis testing**

  Somebody tells you X is true – (dis)prove it

*“I have plenty of ideas now...what's next?”*

## Having an analytics tool lets you focus on

**Infrastructure** when you’re ready to scale beyond your laptop

- Setting up a huge HA cluster: a talk on its own
- Who sets up then maintains the machines? Automate it all?
- How many machines do you need? RAM/CPUs?
- Who ensures all software is up to date (CVEs?)
- Access control lists?
- Hosting costs/providers?
- Reliability, fault tolerance, backup procedures…

## Summary

- Analytics doesn’t need to be complicated: Spark’s useful for the heavy lifting (a good computation engine)
- Sometimes best to just plot as you go – saves plenty of time. Quickly evaluate and test new theories
- Other harder things to worry about: writing a distributed machine learning algorithm shouldn’t be one of them

## And if you want to *really* show off with Spark?

- Use **GPUs** to train models faster
  - DeepLearning4J?
  - Writing your own kernels/C/JNI code (or a Java API like CUDA4J/Aparapi?)
- Use **RDMA** to reduce network transfer times
  - Zero copy: RoCE or InfiniBand?
- Tune the **JDK**, the **OS**, the **hardware**
- Continuously evaluate performance: **Spark** itself or the JVM, use
  - -Xhealthcenter, your own **metrics**, various **libraries**…
- Go **tackle something huge** – join the alien search
- Combine Spark Streaming with MLlib to gain insights fast
  - More informed decision making

## Points to consider...

Where would Spark fit in to your systems? A replacement or supplementary?

Give it a try with your own data and you might be surprised with the outcomes

It’s free and open source, active community!

## Points to take home...

- **Built-in Spark functions** are aplenty – try and stick to these
- You can **plot your results by saving to a csv/json** and using your existing favourite plotting libraries easily
- DataFrame (or Datasets) combined with ML = powerful APIs
  - **Filter** your data – decide how to handle nulls!
  - **Pick and use** a suitable ML algorithm
  - **Plot** results: I suggest Bokeh

*“Great! So where can I find out more?”*

## Useful resources

Official Spark **docs** and **mailing lists**
Check out **Data Science Experience**
**IBM Spectrum Conductor with Spark**
**IBM Machine Learning for z/OS**
**IBM Open Data Analytics for Z**
**z/OS Platform for Apache Spark**
**developerWorks**
Your IBM contacts!

# Any questions?