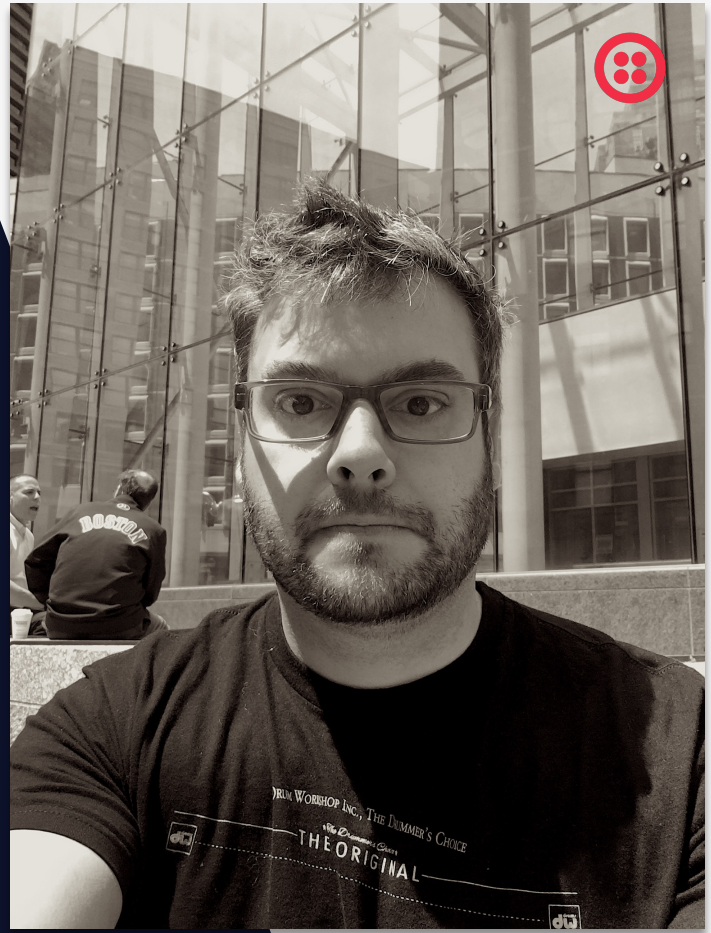
The Path to Predictive Analytics

Scott Haines, Principal Software Engineer,
Voice Insights, **Twilio**

http://bit.ly/odsceastrealtime

@newfront
@twilio
@newfrontcreative



Before we begin

Technoliges

version: 2.7+

version: 2.4.3

version: 0.8.1

@newfront | http://bit.ly/odsceastrealtime

WARM UP

Spark Primer

Exploratory Analysis With Spark SQL

Finding Structure with Unsupervised Learning

http://bit.ly/odsceastrealtime

Spark Primer

- **The Spark Session**: Literally the GodFather of your application
- Controls and Delegates work
- Jobs, Stages and Tasks report back in and if they don't then we kill em off

```
object ChroniclerApp extends App {

  val log: Logger = LoggerFactory.getLogger(clas
  log.info(s"$args")

  val configPath = if (args.length > 0) args(0)
  val config = AppConfig.parse(configPath)

  val sparkConf = new SparkConf()
    .setAppName(config.sparkAppConfig.appName)
    .setJars(SparkContext.jarOfClass(classOf[Chr
    .setAll(config.sparkAppConfig.core)

  log.info(s"sparkConfig: ${sparkConf.toDebugStr

  val session = SparkSession.builder
    .config(sparkConf)
    .enableHiveSupport()
    .getOrCreate()

  session.sparkContext.addSparkListener(SparkApp

  Chronicler(config, session).monitoredRun()

}
```

Spark Primer

- **Creating Projections** on your data is a breeze (once loaded)
- Creating **temporary views can simplify** using SparkSQL

```
spark
.read
.option("inferSchema", "true")
.json(path)
.createOrReplaceTempView("table")

val result = spark.sql(
  "select fieldA, fieldB from table
  where condition>90"
)
```

Spark Primer

- **Loading** data for batch or streaming can be simple.

```
spark
  .read
  .option("inferSchema", "true")
  .json(path)
```

DataFrame Loaded

Row1	ColA	ColB	ColC
Row2	ColA	ColB	ColC

WARM UP

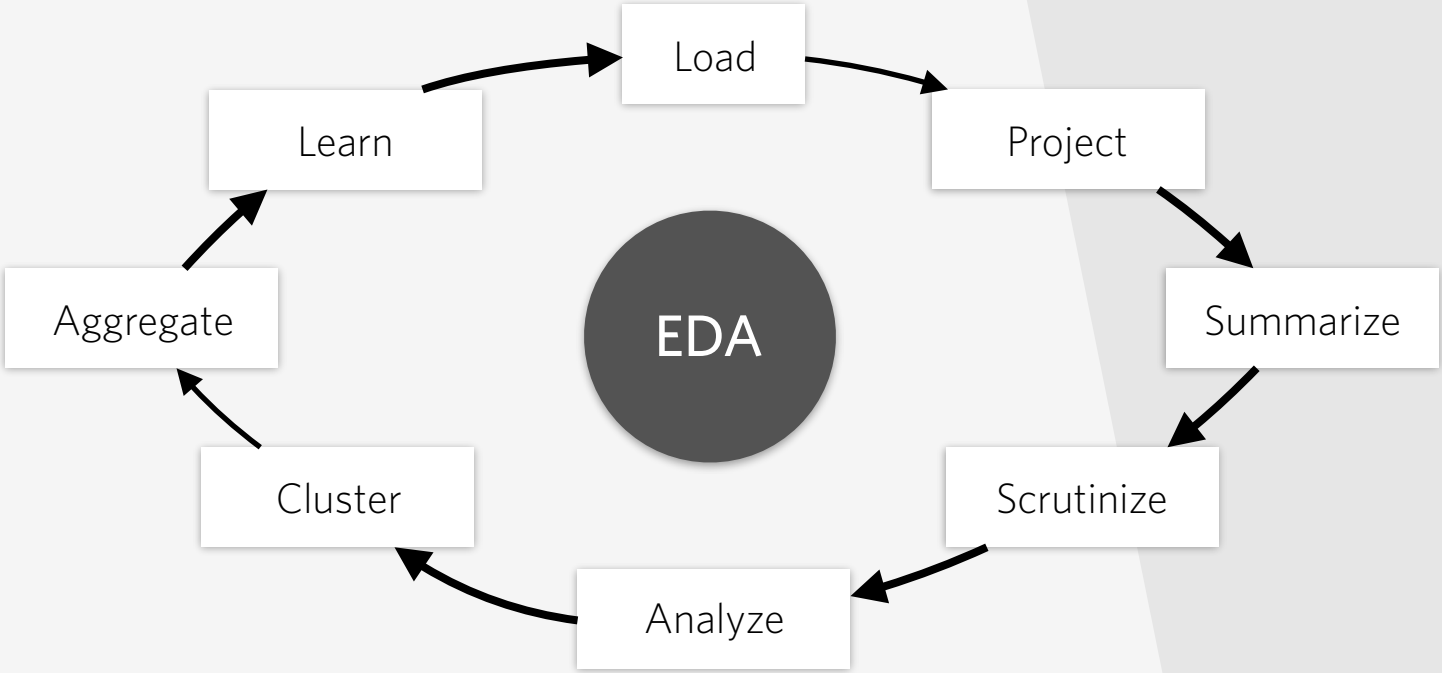
Spark Primer

Exploratory Analysis With Spark SQL

Finding Structure with Unsupervised Learning

http://bit.ly/odsceastrealtime

Exploratory Data Analysis



Exploratory Analytics with Spark SQL

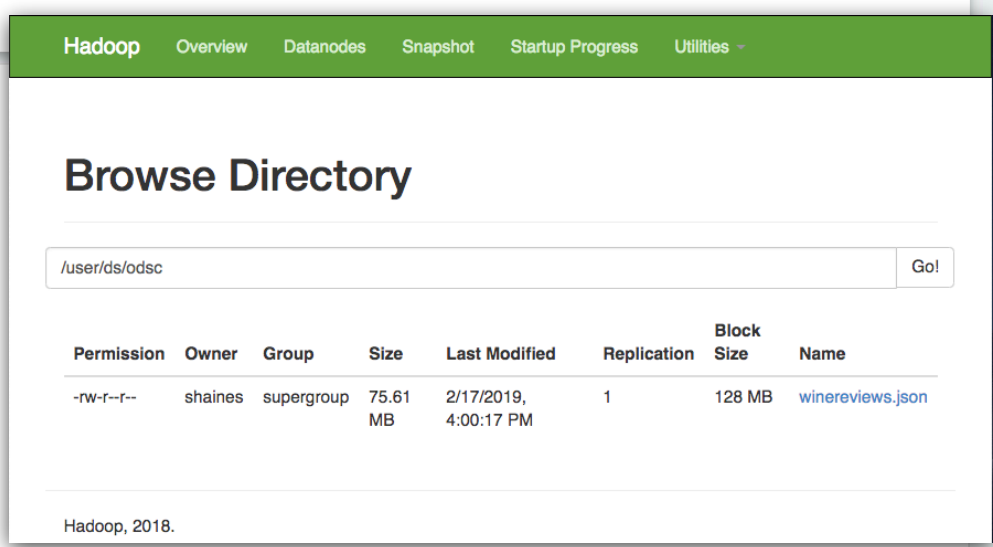
Mental Model

- Understand the fields and types of the data
- Investigate the properties of the data set (missing, distinct, ranges)
- Generate Column Statistics (min, p25, p50, p75, max)
- Field Imputation strategies / cleaning data

Spark Basics : Loading JSON via HDFS

```
/* Load up the Wine Reviews Dataset (via kaggle) */
val dataLocation = "/user/ds/odsc/"
val wineReviewsDf = spark.read.option("inferSchema", "true").json(s"$dataLocation/winereviews.json")
```

SPARK JOB FINISHED




```
wineReviewsDf.printSchema
```

```
root
 |-- country: string (nullable = true)
 |-- description: string (nullable = true)
 |-- designation: string (nullable = true)
 |-- points: string (nullable = true)
 |-- price: long (nullable = true)
 |-- province: string (nullable = true)
 |-- region_1: string (nullable = true)
 |-- region_2: string (nullable = true)
 |-- taster_name: string (nullable = true)
 |-- taster_twitter_handle: string (nullable = true)
 |-- title: string (nullable = true)
 |-- variety: string (nullable = true)
 |-- winery: string (nullable = true)
```

Exploratory Analytics : Column Statistics

1. Create a Temporary SQL Table
2. Enables us to query with simple SQL vs Scala API
3. Enables Zeppelin UI to easily print graphs etc as well.

```
/* Load up the Wine Reviews Dataset (via kaggle) */
val dataLocation = "/user/ds/odsc/"
val wineReviewsDf = spark.read.option("inferSchema", true).json(s"$dat
wineReviewsDf.cache()
```

Took 9 sec. Last updated by anonymous at May 24 2019, 5:56:30 PM.

```
wineReviewsDf.createOrReplaceTempView("reviews")
```

Took 1 sec. Last updated by anonymous at February 18 2019, 10:27:50 AM.

Exploratory Analytics : Finding Missing Data

```
def missingValues(df: DataFrame) = {
    df.schema.map { r => (r.name, df.where(col(r.name).isNull).count) }.toDF("name", "missing")
}
```

FINISHED

```
missingValues(wineReviewsDf).show
```

SPARK JOBS FINISHED

```
+--------------------+-------+
|                name|missing|
+--------------------+-------+
|             country|     63|
|         description|      0|
|         designation|  37465|
|              points|      0|
|               price|   8996|
|            province|     63|
|            region_1|  21247|
|            region_2|  79460|
|         taster_name|  26244|
|taster_twitter_ha...|  31213|
|               title|      0|
|             variety|      1|
|              winery|      0|
+--------------------+-------+
```

Exploratory Analytics : Finding Distinct Values

```
def distinctValues(df: DataFrame) = {
    df.schema.map { r =>
        (r.name, df.select(col(r.name)).where(col(r.name).isNotNull).distinct.count)
        }.toDF("name", "distinct")
}
```

FINISHED

```
distinctValues(wineReviewsDf).show(50, false)
```

SPARK JOBS FINISHED

```
+--------------------+--------+
|name                |distinct|
+--------------------+--------+
|country             |43      |
|description         |119955  |
|designation         |37979   |
|points              |21      |
|price               |390     |
|province            |425     |
|region_1            |1229    |
|region_2            |17      |
|taster_name         |19      |
|taster_twitter_handle|15      |
|title               |118840  |
|variety             |707     |
|winery              |16757   |
```

Exploratory Analytics : Column Statistics

1. Build simple helper functions for exploring the data

```
def columnStats(df: DataFrame) = {
    df.schema.map { r =>
     val colStats = df.select(col(r.name)).where(col(r.name).isNotNull).describe()
     /*colStats.printSchema*/
     colStats.show()
    }
}
```

```
columnStats: (df: org.apache.spark.sql.DataFrame)Seq[Unit]
```

Took 1 sec. Last updated by anonymous at February 18 2019, 10:26:22 AM. (outdated)

```
columnStats(wineReviewsDf)
```

Exploratory Analytics : Column Statistics

1. Build simple helper functions for exploring the data
2. View The summary stats.

```
+-------+---------+
|summary|  country|
+-------+---------+
|  count|   129908|
|   mean|     null|
| stddev|     null|
|    min|Argentina|
|    max|  Uruguay|
+-------+---------+

+-------+--------------------+
|summary|         description|
+-------+--------------------+
|  count|              129971|
|   mean|                null|
| stddev|                null|
|    min|"Chremisa," the a...|
|    max|“Wow” is the firs...|
+-------+--------------------+
```

```
columnStats(wineReviewsDf)
```

```
+-------+------------------+
|summary|       designation|
+-------+------------------+
|  count|             92506|
|   mean|1494.4644378698224|
| stddev|7115.55431803001|
|    min|#19 Phantom Limb ...|
|    max|               “P”|
+-------+------------------+

+-------+-----------------+
|summary|           points|
+-------+-----------------+
|  count|           129971|
|   mean|88.44713820775404|
| stddev|3.0397302029160067|
|    min|              100|
|    max|               99|
+-------+-----------------+
```

Exploratory Analytics : Column Statistics

1. Build simple helper functions for exploring the data
2. View The summary stats.
3. Find Issues with the Data

```
+-------+---------+
|summary|  country|
+-------+---------+
|  count|   129908|
|   mean|     null|
| stddev|     null|
|    min|Argentina|
|    max|  Uruguay|
+-------+---------+
```

```
+-------+--------------------+
|summary|         description|
+-------+--------------------+
|  count|              129971|
|   mean|                null|
| stddev|                null|
|    min|"Chremisa," the a...|
|    max|“Wow” is the firs...|
+-------+--------------------+
```

```
columnStats(wineReviewsDf)
```

```
+-------+------------------+
|summary|       designation|
+-------+------------------+
|  count|             92506|
|   mean|1494.4644378698224|
| stddev|7115.55431803001|
|    min|#19 Phantom Limb ...|
|    max|               “P”|
+-------+------------------+
```

```
+-------+-----------------+
|summary|           points|
+-------+-----------------+
|  count|           129971|
|   mean|88.44713820775404|
| stddev|3.0397302029160067|
|    min|              100|
|    max|               99|
+-------+-----------------+
```

Exploratory Analytics : Fixing Data

```
val fixedReviews = wineReviewsDf
  .withColumn("num_points", 'points.cast("Double"))
  .drop("points")
  .withColumnRenamed("num_points", "points")
```

FINISHED

```
calculateColumnAggregates(fixedReviews)
```

SPARK JOBS FINISHED

```
+-------+------+--------+----+------+---------+-------+
|colname| count|     min| avg|sd_pop|sd_sample|    max|
+-------+------+--------+----+------+---------+-------+
|variety|129970|Abouriou|null|  null|     null|Žilavka|
+-------+------+--------+----+------+---------+-------+

+-------+------+-----+--------+------+---------+-----+
|colname| count|  min|     avg|sd_pop|sd_sample|  max|
+-------+------+-----+--------+------+---------+-----+
| winery|129971|1+1=3|Infinity|   NaN|      NaN|Štoka|
+-------+------+-----+--------+------+---------+-----+

+-------+------+----+----------------+-----------------+-----------------+-----+
|colname| count| min|             avg|           sd_pop|        sd_sample|  max|
+-------+------+----+----------------+-----------------+-----------------+-----+
| points|129971|80.0|88.44713820775404|3.0397185090148677|3.0397302029160067|100.0|
+-------+------+----+----------------+-----------------+-----------------+-----+
```

Exploratory Analytics : Projecting / Viewing Data

```
%sql
select winery, avg(points) as avg_rating from reviews
where country = 'France' AND region_1 = 'Côtes de Provence' AND points > 88 group by winery order by avg_rating desc
```

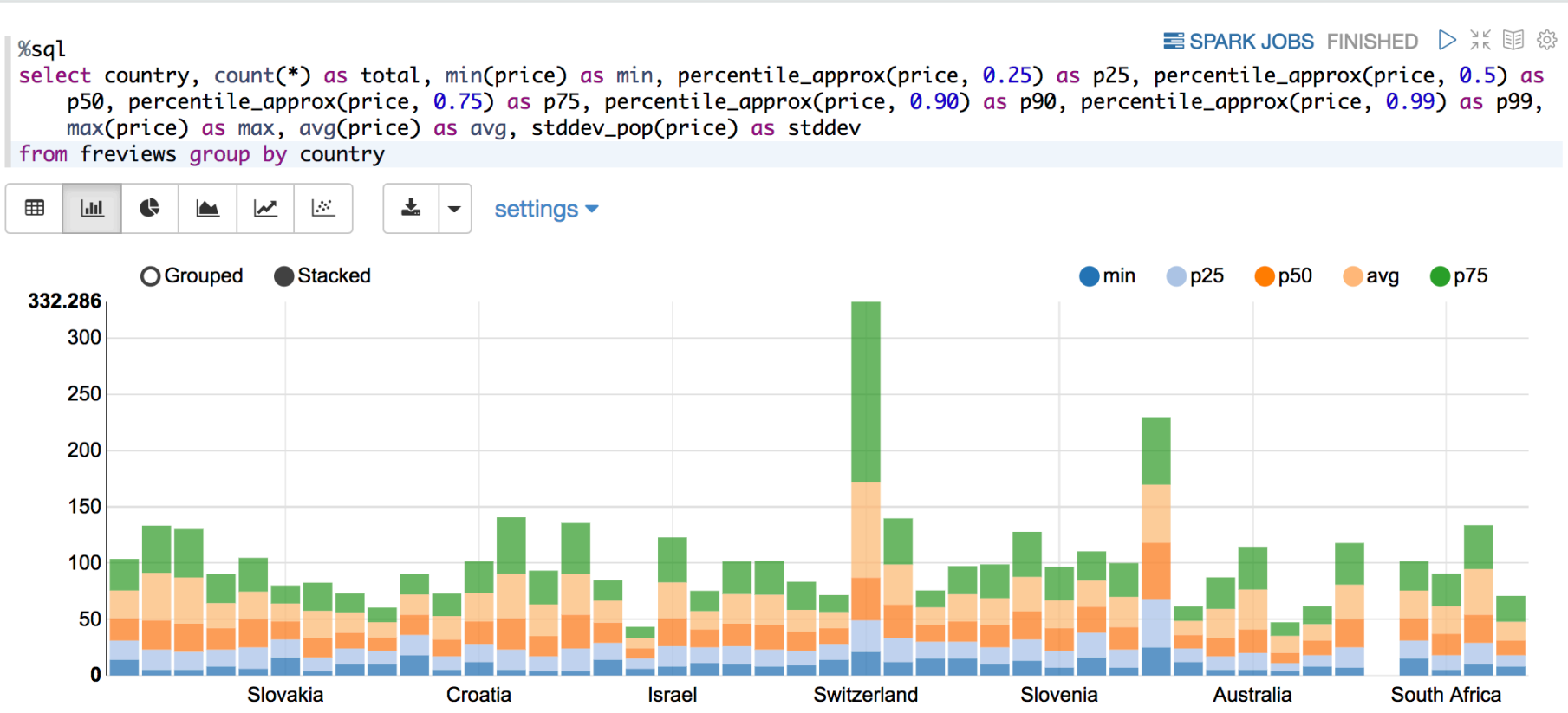
SPARK JOB FINISHED

settings

avg_rating	winery
92	Estandon
92	Château la Vivonne
92	Domaine de la Croix
92	Domaine du Clos Gautier
92	Mirabeau
91.83333333333333	Château Grand Boise
91.125	Château Sainte Marguerite
91	Château Minuty

Took 5 sec. Last updated by anonymous at July 24 2019, 8:20:11 AM. (outdated)

Exploratory Analytics : Percentile Data



Exploratory Analytics : Price Field Imputation

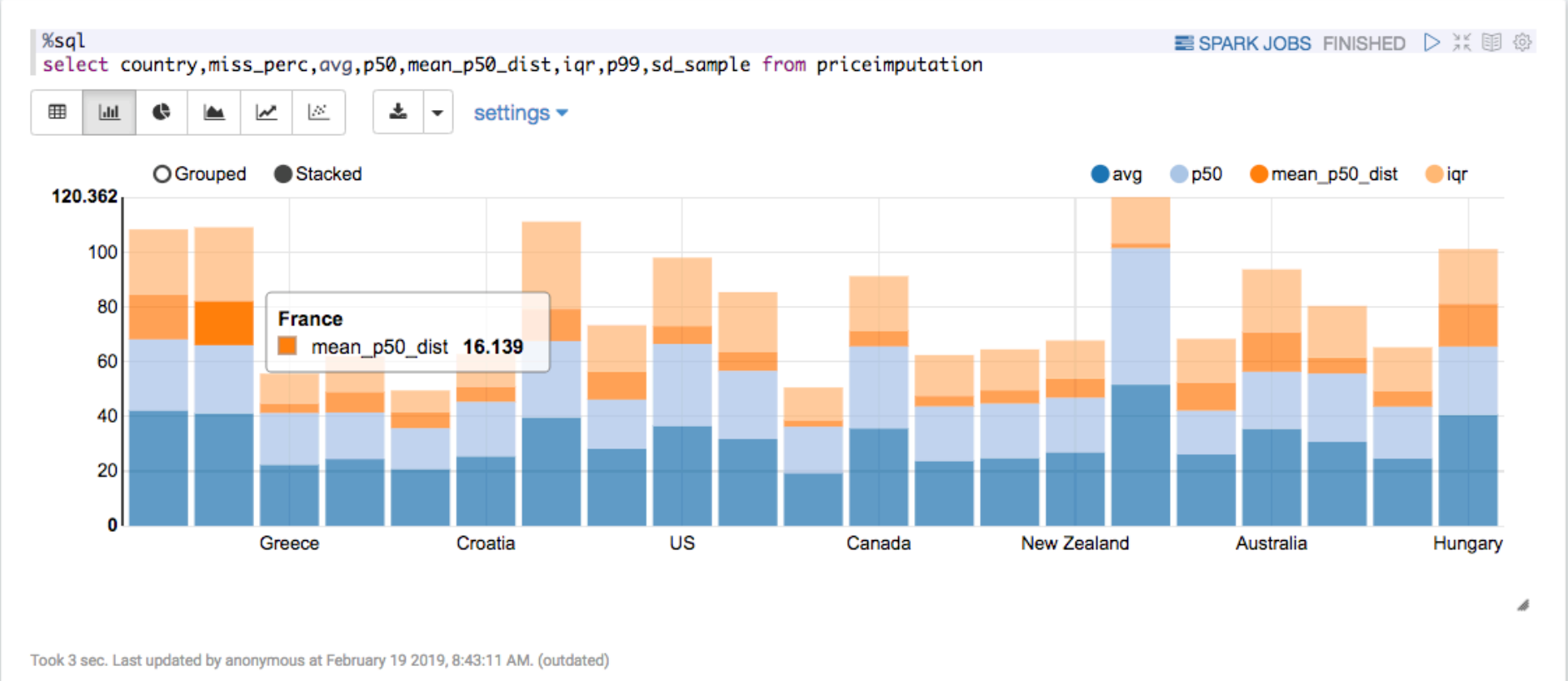
```
/* Price Imputation */

val fullPriceData = spark.sql("select country, count(*) as total, min(price) as min, percentile_approx(price, 0.25) as p25, percentile_approx(price, 0.5)
    as p50, percentile_approx(price, 0.75) as p75, percentile_approx(price, 0.90) as p90, percentile_approx(price, 0.99) as p99, max(price) as max, avg
    (price) as avg, stddev_pop(price) as sd_pop, stddev_samp(price) as sd_sample from winereviews group by country")

val missingPriceRows = fixedReviews
  .where(col("price").isNull.and(col("country").isNotNull)).groupBy("country")
  .agg(count("*") as "missing")

val joinedWithOtherPrices = missingPriceRows
  .join(fullPriceData, "country")
  .withColumn("miss_perc", ((col("missing")/col("total"))*100).cast("Float"))
  .withColumn("iqr", (col("p75")-col("p25")).cast("Float"))
  .withColumn("min_max_dist", (col("max")-col("min")).cast("Integer"))
  .withColumn("median_mean_dist", (col("p50")-col("avg")).cast("Float")).where(col("total") > 1)

joinedWithOtherPrices.createOrReplaceTempView("priceimputation")
```

FINISHED

```
joinedWithOtherPrices.printSchema

root
 |-- country: string (nullable = true)
 |-- missing: long (nullable = false)
 |-- total: long (nullable = false)
 |-- min: double (nullable = true)
 |-- p25: double (nullable = true)
 |-- p50: double (nullable = true)
 |-- p75: double (nullable = true)
 |-- p90: double (nullable = true)
 |-- p99: double (nullable = true)
 |-- max: double (nullable = true)
 |-- avg: double (nullable = true)
 |-- sd_pop: double (nullable = true)
 |-- sd_sample: double (nullable = true)
 |-- miss_perc: float (nullable = true)
 |-- iqr: float (nullable = true)
 |-- min_max_dist: integer (nullable = true)
 |-- median_mean_dist: float (nullable = true)
```

1. Find missing prices by country and join with price statistics data
2. Generate new temp table to visualize data.

Exploratory Analytics : Price Field Imputation



Exploratory Analytics : Price Field Imputation

```
/* Use imputation data to fill in unknown prices */
val missingPriceDf = fixedReviews.where(col("price").isNull.and(col("country").isNotNull))
val normalDf = fixedReviews.where(col("price").isNotNull)

val imputedPrice = missingPriceDf
  .join(joinedWithOtherPrices, "country")
  .withColumn("price", when(col("avg")>col("p50"), col("p50")).otherwise(col("avg")))
  .drop("total","min","p25","p50","p75","p90","p99","max","avg","sd_pop","sd_sample"
  ,"miss_perc","iqR","min_max_dist","median_mean_dist", "missing")
/* both schemas must match for union */
val filledPriceDf = normalDf.union(imputedPrice)
filledPriceDf.cache()
filledPriceDf.createOrReplaceTempView("winereviews")
```

1. Simplified: Mean or Median Fill based on skew
2. Can be smarter - Lookup MICE Strategy

```
filledPriceDf
  .where(col("price").isNull)
  .agg(count("*") as "total").show
```

```
+-----+
|total|
+-----+
|    0|
+-----+
```

WARM UP

Spark Primer

Exploratory Analysis With Spark SQL

Finding Structure with Unsupervised Learning

http://bit.ly/odsceastrealtime

Clustering and Statistics

- Look at the Universal Set vs SubSet Trends
- Find Patterns in the Noise
- Understand where natural thresholds reside

http://bit.ly/odsceastrealtime

Exploratory Analytics : Unsupervised Learning

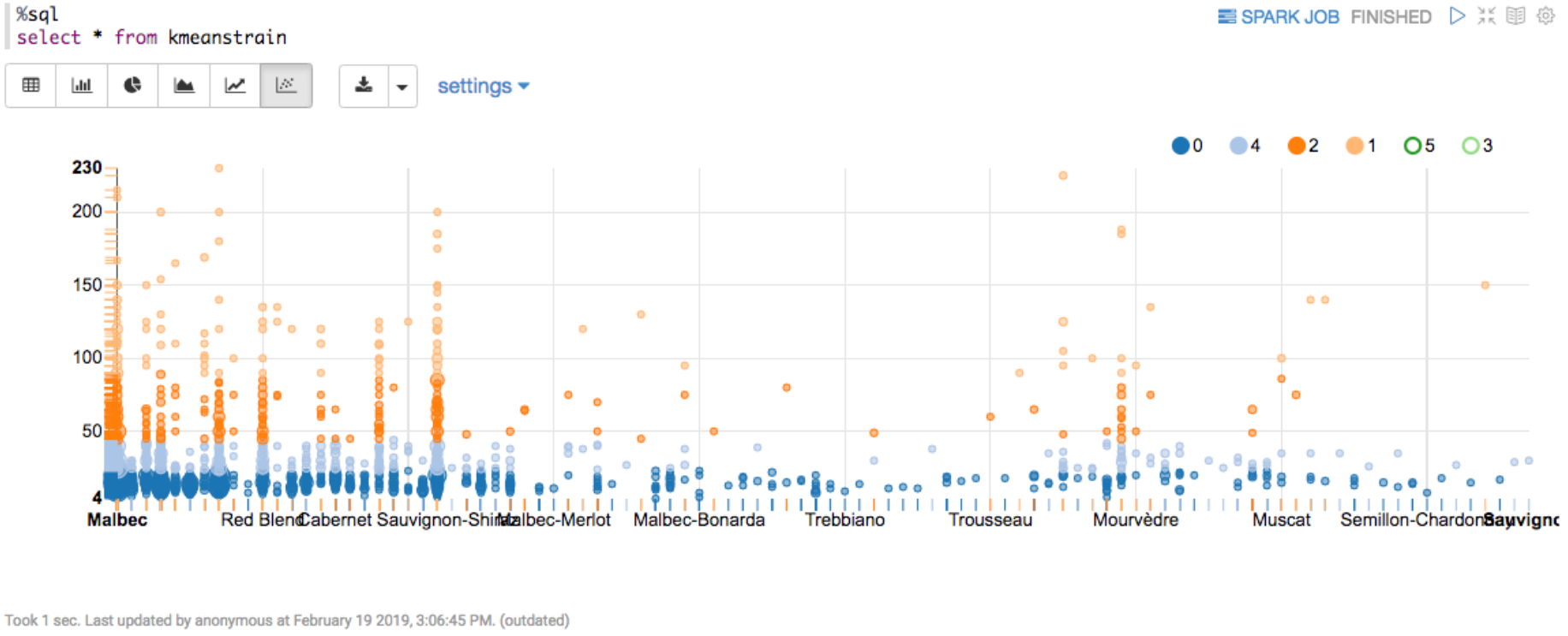
```
/* How to use KMeans Model */

/* which countries can be filled in by varietal */
/* note: indexers can't handle null values */
val normalizedWine = filledPriceDf.where(col("country").isNotNull.and(col("variety").isNotNull))
normalizedWine.cache()

val indexerCountry = new StringIndexer().setInputCol("country").setOutputCol("country_index")
val encoderCountry = new OneHotEncoder().setInputCol("country_index").setOutputCol("country_encoded")
val indexerVariety = new StringIndexer().setInputCol("variety").setOutputCol("variety_index")
val encoderVariety = new OneHotEncoder().setInputCol("variety_index").setOutputCol("variety_encoded")
val wineVectorAssembler = new VectorAssembler().setInputCols(Array("country_encoded", "variety_encoded", "price", "points")).setOutputCol
    ("features")
val transformerPipeline = new Pipeline().setStages(Array(indexerCountry, encoderCountry, indexerVariety, encoderVariety, wineVectorAssembler))

val fittedPipeline = transformerPipeline.fit(normalizedWine)
val Array(trainingData, testData) = normalizedWine.randomSplit(Array(0.7, 0.3))
val transformedTraining = fittedPipeline.transform(trainingData)
transformedTraining.cache()
val kmeans = new KMeans().setK(6).setSeed(1L)
val kmModel = kmeans.fit(transformedTraining)
kmModel.computeCost(transformedTraining)
val transformedTest = fittedPipeline.transform(testData)
transformedTest.cache()
kmModel.computeCost(transformedTest)
kmModel.transform(transformedTest)
```

SPARK JOBS FINISHED

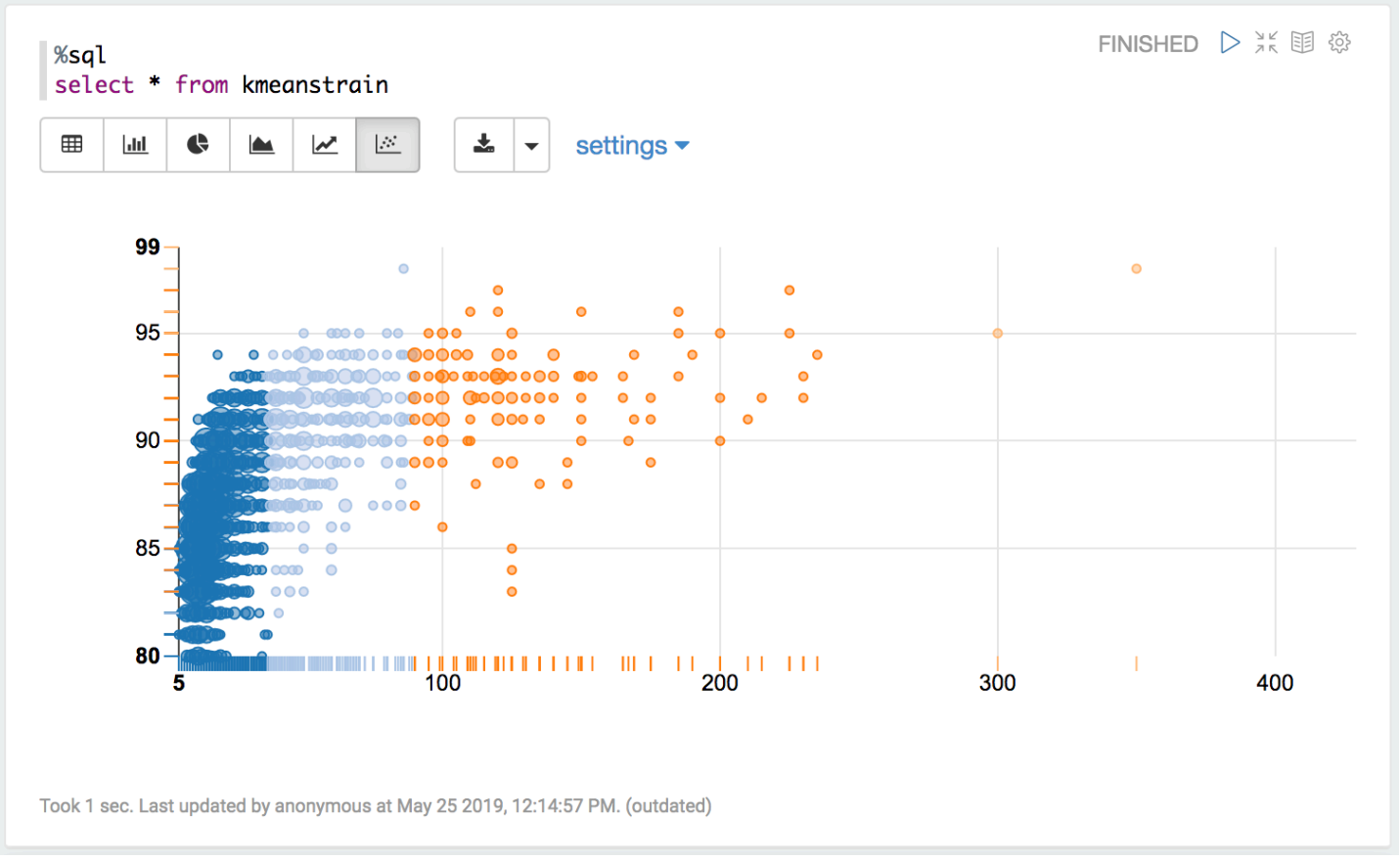
```
kmModel.transform(transformedTest).createOrReplaceTempView("kmeans")
```

FINISHED

Exploratory Analytics : Unsupervised Learning



Exploratory Analytics : Unsupervised Learning



~100 US

After this point, wine is just a name

Clustering and Tagging

```
%spark                                                    FINISHED
val packetLossByCountry = scoresData
  .select("day","account_sid","dimensions_country_code"
      ,"metrics_connect_duration"
      ,"metrics_packet_loss_percentage","score")
  .where(col("day").equalTo(28)
    .and(col("account_sid").isNotNull
      .and(col("dimensions_country_code").notEqual("")
        .and(col("metrics_connect_duration").notEqual(0)
          .and(col("metrics_packet_loss_percentage"
```

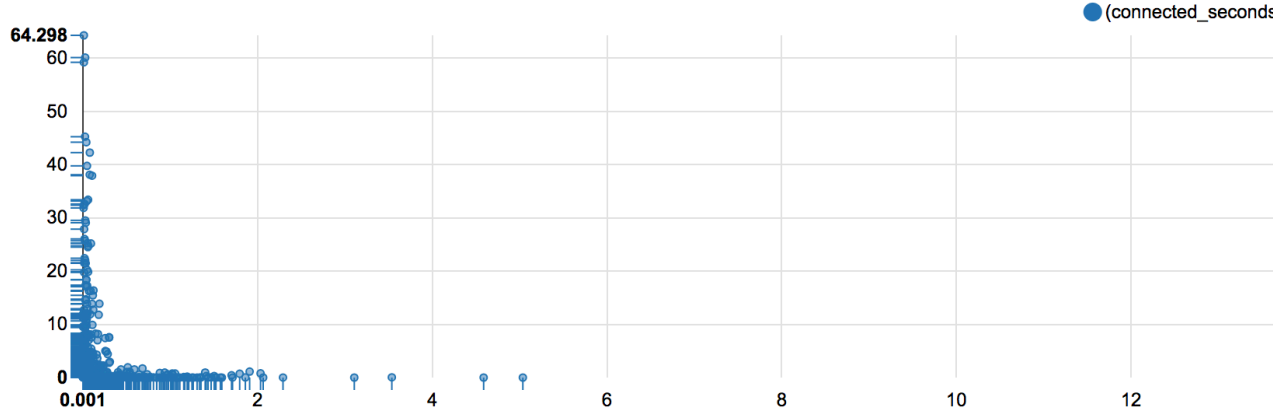
Took 0 sec. Last updated by anonymous at May 01 2019, 1:27:24 AM. (outdated)

~2 SEC

tolerance of bad packet loss.

```
%sql                                                      SPARK JOB FINISHED
select (metrics_connect_duration/1000) as connected_seconds, metrics_packet_loss_percentage as loss_percentage, scor
from loss_by_country
where dimensions_country_code in ("US","MX","JP","BR")
```

settings



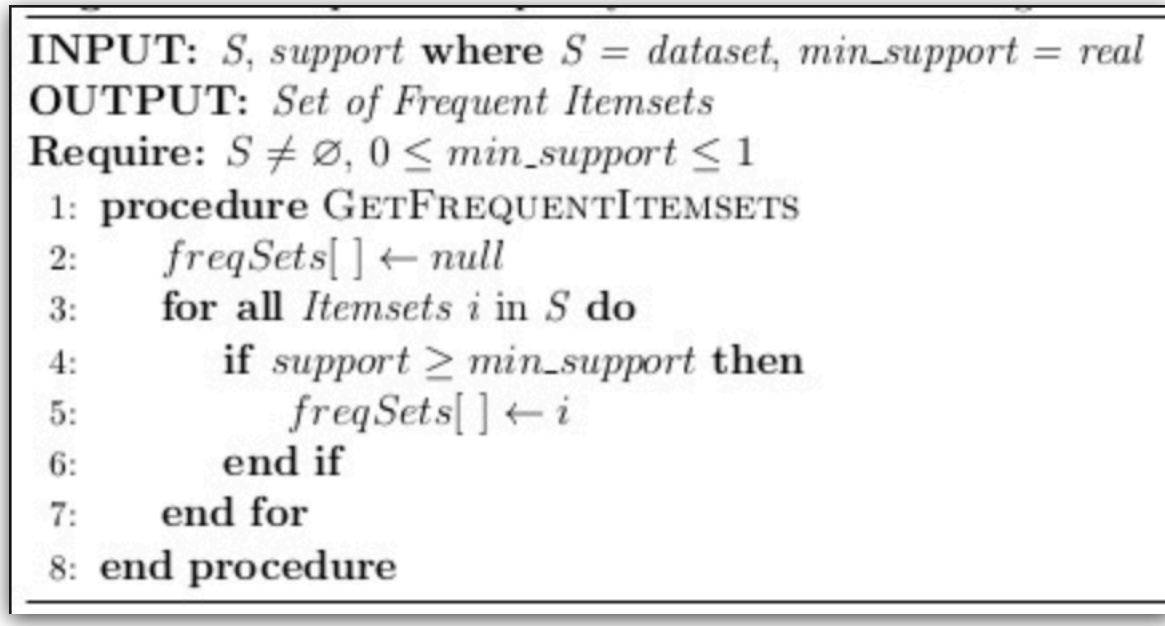
Took 0 sec. Last updated by anonymous at May 01 2019, 1:27:08 AM. (outdated)

https://github.com/newfront/odsc-east-realish-predictions/blob/master/zeppelin/projections_and_naive_bayes.json

Exploratory Analytics : Unsupervised Learning

Apriori Algorithm

1. Well know for the diapers + beer decision.
2. Shoppers who bought X also bought Y
3. Yelp Review Highlights
4. Or Auto Wine Reviews



INPUT: S, $support$ **where** $S = dataset$, $min\_support = real$
OUTPUT: *Set of Frequent Itemsets*
Require: $S \neq \emptyset$, $0 \leq min\_support \leq 1$

```
1: procedure GetFrequentItemsets
2:     freqSets[ ] ← null
3:     for all Itemsets i in S do
4:         if support ≥ min_support then
5:             freqSets[ ] ← i
6:         end if
7:     end for
8: end procedure
```

AKA - Market Basket Analysis

Exploratory Analytics : Unsupervised Learning

```
def tastingNotes(df: DataFrame, minSupport: Double = 0.05, minConfidence: Double = 0.6, freq: Int = 400): String = {
  // can mess around easily with different selection criteria by changing the minSupport and minConfidence coefficents
  val fpg = new FPGrowth()
    .setItemsCol("items")
    .setMinSupport(minSupport)
    .setMinConfidence(minConfidence)

  val remover = new StopWordsRemover()
    .setInputCol("items")
    .setOutputCol("filteredItems")

  // Cleaning up the Wine Descriptions
  val descriptions = df
    .select(col("description")).where(col("description").isNotNull)
    .map { case Row(s:String) =>
      s.replace(",","").replace(".","").replace("wine", "").split(" ").toSet.toSeq
    }
    .toDF("items")

  // remove StopWords
  val filteredDescriptions = remover.transform(descriptions)
  val stopWordsFiltered = filteredDescriptions.select("filteredItems").toDF("items")
  val model = fpg.fit(stopWordsFiltered)
  val freqItems = model.freqItemsets.sort(desc("freq"))
  val notes = freqItems.select("items").where(col("freq")>freq)
  val topWords = notes.flatMap { case Row(notes: Seq[String]) => notes }.groupBy("value").count().sort(desc("count"))
  val tastingNotes = topWords.select("value").collect().map { case Row(s: String) => s }.toSeq.mkString(",")
  tastingNotes
}
```

Exploratory Analytics : Unsupervised Learning

```
val redBlends = spark.sql("select * from winereviews where `variety` == 'Red Blend'")
redBlends.cache
redBlends.explain(true)
val redBlendTastingNotes = tastingNotes(redBlends)
```

palate,**tannins**,aromas,**cherry**,**black**,blend,Cabernet,Sauvignon,flavors,Merlot,**spice**,finish,**alongside**,**plum**,fruit,**offers**,**berry**,red,**pepper**,Syrah,**Drink**,Sangiovese,**ripe**,opens,**blackberry**,acidity,notes,nose,**dried**,delivers,**licorice**,**oak**,Franc,Petit,**dark**,Verdot,**tobacco**,**juicy**,**leather**,firm,**chocolate**,**vanilla**,**clove**,**raspberry**,hint,white,note,Aromas,lead,**currant**,spicy,rich,**tannic**,**touch**,along,**dry**,**earth**,whiff,fresh,Malbec,Made,**espresso**,soft,smooth,herb,fruits,**sweet**,**texture**,wild,cassis,herbal,Petite,**coffee**,Sirah,well,**savory**,**herbs**,shows,**bright**,**mocha**

@newfront

@twilio

@newfrontcreative

blog.twilio.com

THANK YOU

twilio