capstone

Hi!

Welcome to my capstone project.

I'll be using apache spark to solve this project, because as this course states this shold be at scale (or at least scalable) so I tried to stick to this subject. Please take into account that I'm not really familiar with the topic of the project, but at least this notebook should demonstrated my skills to potentially solve scalable machine learning problems.

Have fun and be kind ^^.

0.- SETUP

Here I'm setting a lot of imports and functions I'm going to be using to handle the data

Finding price numbers, extracting coordinates and the discretization functions I'll use to select the buildings (I'll comment more on that later) and essentially common stuff.

```
// ********************************************************************************
// * SETUP ********************************************************************
// ********************************************************************************

// Setup
val sqlContext = new org.apache.spark.sql.SQLContext(sparkContext)
import sqlContext.implicits._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import scala.util.matching.Regex

val data_seed = 0
val data_path = "../../detroit_data/"

// Cleanup function to find_number
val num_regex = "[0-9]+([,.][0-9]+)?".r
val find_number = udf((data:String) => data match {
  case null => None
  case data => num_regex.findFirstIn(data).map(_.toDouble)
})

// Cleanup function to get the coordinates
val coords_regex = "\\(.*?\\)".r
val get_coordinate = (data:String, index:Integer) => data match {
case null => None
case data => coords_regex
  .findFirstIn(data)
  .map( _.replaceAll("[\\(\\)]", "").split(",")(index).trim.toDouble)
}
val get_latitude  = udf((data:String) => get_coordinate(data, 0))
val get_longitude = udf((data:String) => get_coordinate(data, 1))


// Discretization functions
import scala.math.BigDecimal.RoundingMode
def discretizeCoord(coord:Double, interval:BigDecimal, offset:BigDecimal = BigDecimal(
  val c = BigDecimal(coord.toString)
  val lower_point = (((c + offset)/interval).toInt * interval) - offset
  val center = lower_point + (interval/2)
  return center
}

def bestGrid(lat: Double, lng: Double, interval:BigDecimal) : (BigDecimal, BigDecimal)
  val offset = interval / 2
  // Grid 0 no overlap
  val g0_lat = discretizeCoord(lat, interval)
  val g0_lng = discretizeCoord(lng, interval)
  // Grid 1 half overlap
  val g1_lat = discretizeCoord(lat, interval, offset)
  val g1_lng = discretizeCoord(lng, interval, offset)
  // Compute manhattan distance
  val g0_dist = (g0_lat - lat).abs + (g0_lng - lng).abs
  val g1_dist = (g1_lat - lat).abs + (g1_lng - lng).abs
  // Return
  if (g0_dist <= g1_dist)
    return (g0_lat, g0_lng)
  else
```

```
    return (g1_lat, g1_lng)
}
val bestGridStr = udf((x:Double, y:Double) => Seq(bestGrid(x, y, BigDecimal("0.0001"))
val bestGridLat = udf((x:Double, y:Double) => bestGrid(x, y, BigDecimal("0.0001"))._1)
val bestGridLng = udf((x:Double, y:Double) => bestGrid(x, y, BigDecimal("0.0001"))._2)

// Labeling functions
val labeludf = udf((c: Long, s:Double) => (c > 1 && s > 600))

// Lets define this class to represent geo localized data later on
case class GeoData(lat:Double, lng:Double, value:Double, group:String = "blue")
```

1.- Data load, clean up and brief exploration

1.1.- Blight violations

I'm loading the data through the databricks csv loader. It has a few flaws with new lines inside quotes so I had to use and alternative parser library.

After loading the data I'm essentially extraction prices as Double data types and extracting coordinates.

Finally I create another variable to hold specific detroit data (approx).

```
// ************************************************************************
// * BLIGHT VIOLATION *****************************************************
// ************************************************************************

// ------------------------------------------------------------------------
// Load and cleanup
// ------------------------------------------------------------------------

var blight_violations = sqlContext
  // Read the csv
  .read.format("com.databricks.spark.csv")
  .option("header",      "true")
  .option("inferSchema", "true")      // Automatic schema inference
  .option("parserLib",   "UNIVOCITY") // This configuration solves the multi-line fiel
  .load(data_path + "detroit-blight-violations.csv")
  .cache()  // There is a weird bug with spark and If I want to cache new columns late
  // Parse numerical columns
  .withColumn("JudgmentAmt", find_number($"JudgmentAmt"))
  .withColumn("FineAmt",     find_number($"FineAmt"))
  .withColumn("AdminFee",    find_number($"AdminFee"))
  .withColumn("LateFee",     find_number($"LateFee"))
  .withColumn("StateFee",    find_number($"StateFee"))
  .withColumn("CleanUpCost", find_number($"CleanUpCost"))
  // Extract the geo localization data
  .filter("ViolationAddress LIKE '%(%,%)%'") // Only 3 rows without geodata will be di
  .withColumn("Violation_lat", get_latitude($"ViolationAddress"))
  .withColumn("Violation_lng", get_longitude($"ViolationAddress"))
  .withColumn("Mailing_lat",   get_latitude($"MailingAddress"))
  .withColumn("Mailing_lng",   get_longitude($"MailingAddress"))
  .cache()


var blight_violations_detroit = blight_violations
  // Filter detroit bounds approx.
  .filter("Violation_lat between 42.257 and 42.453")
  .filter("Violation_lng between -83.308 and -82.896")
  .cache()
```

1 second 62 milliseconds

```
// ---------------------------------------------------------------------------
// Schema and data exploration
// ---------------------------------------------------------------------------

blight_violations.printSchema()
blight_violations.count()
widgets.TableChart(blight_violations.take(5))
```

Show: 1 ▼

Search:

TicketID	TicketNumber	AgencyName	ViolName	ViolationStreetNumber	ViolationStreetNam
26288	05000001DAH	Department of Public Works	Group, LLC, Grand Holding	2566	GRAND BLVD
19800	05000025DAH	Department of Public Works	JACKSON, RAECHELLE	19014	ASHTON
19804	05000026DAH	Department of Public Works	TALTON, CAROL ANN	18735	STAHELIN
20208	05000027DAH	Department of Public Works	BONNER, DARRYL E.	20125	MONICA
20211	05000028DAH	Department of Public Works	GREGORY, JAMES LEE	17397	PRAIRIE

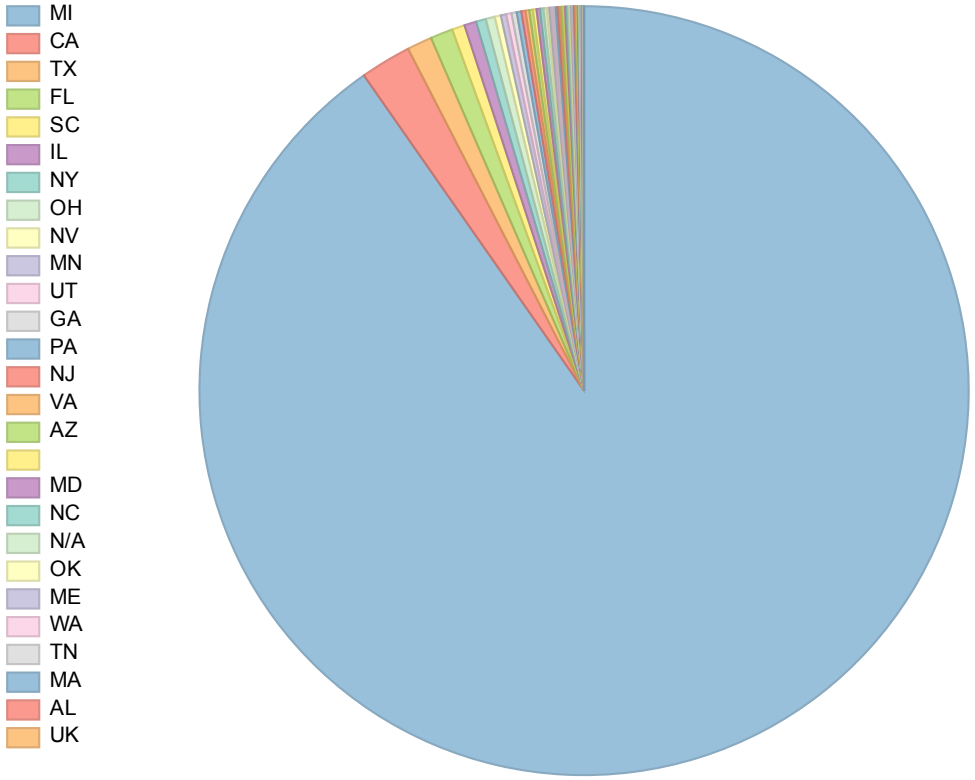
Showing 5 of 5 records

Pages: Previous 1 Next

5 seconds 683 milliseconds

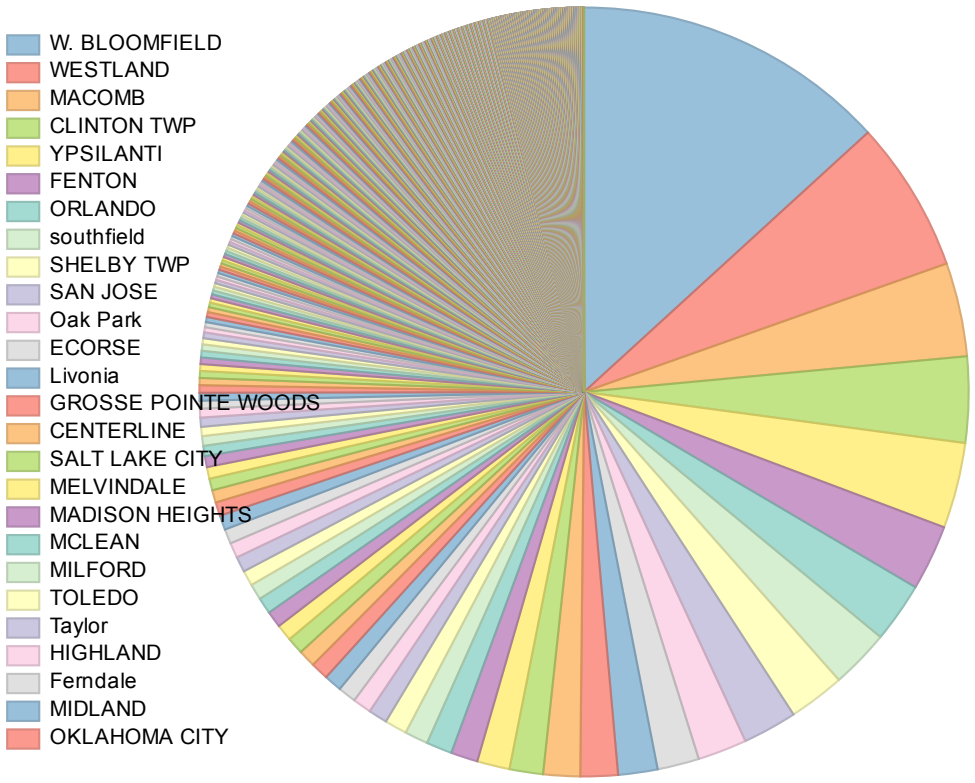
Looks like this is not a really usefull column...

```
widgets.PieChart(blight_violations.groupBy("MailingState").count().collect())
```



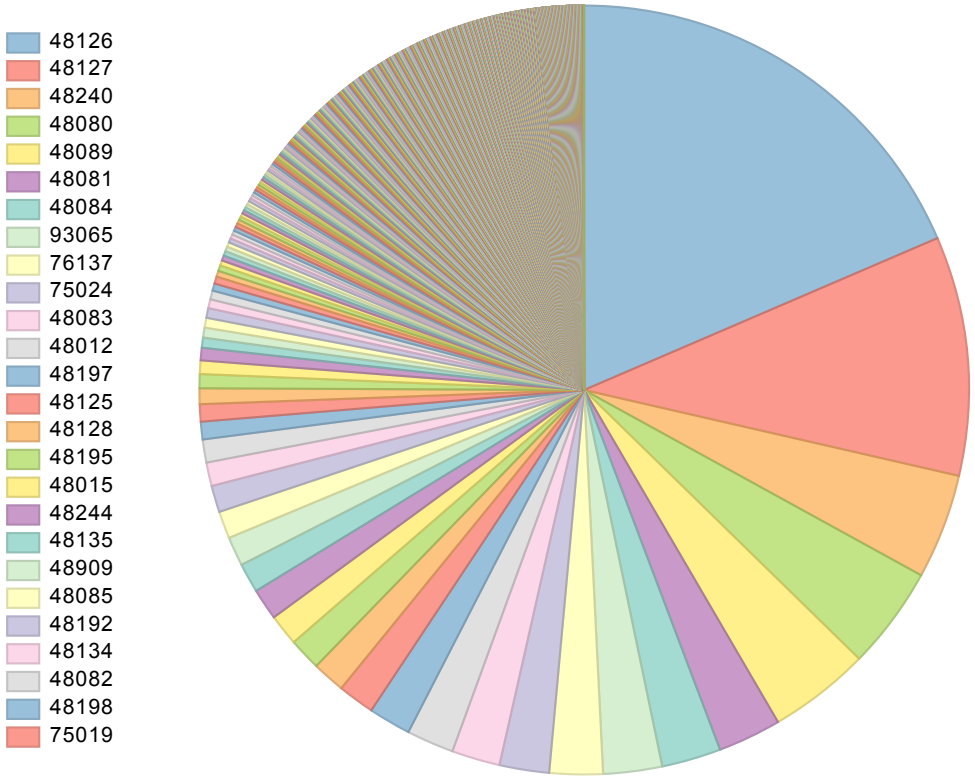
1 second 615 milliseconds

```
widgets.PieChart(blight_violations.groupBy("MailingCity").count().collect())
```



1 second 608 milliseconds

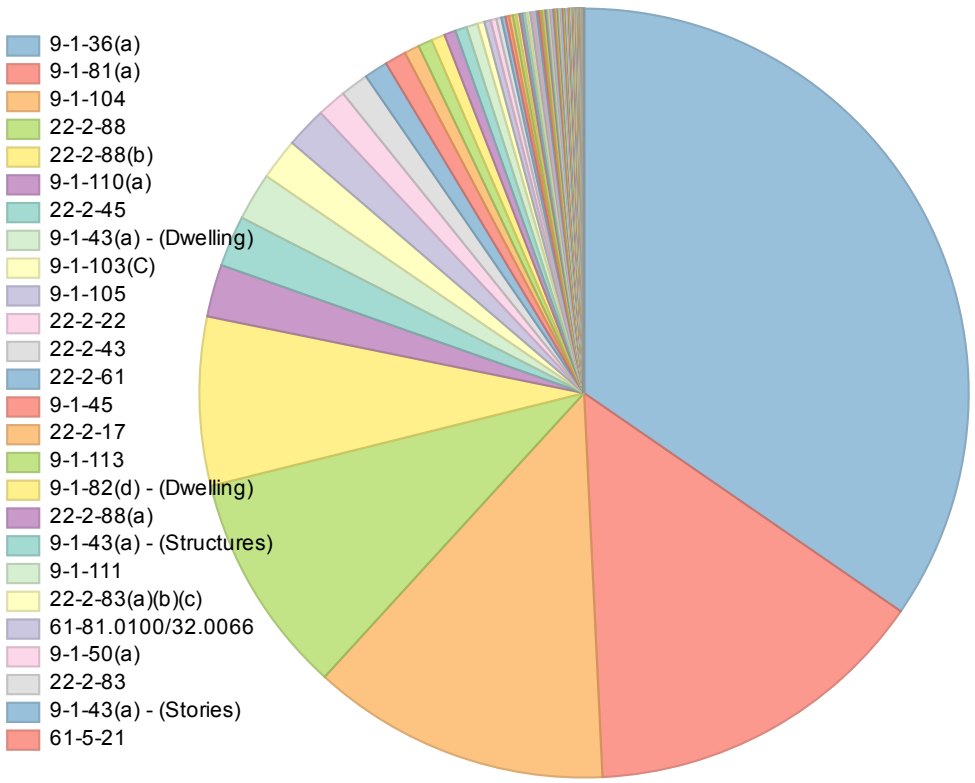
```
widgets.PieChart(blight_violations.groupBy("MailingZipCode").count().collect())
```



1 second 504 milliseconds

Now lets plot a pie chart of all violation codes

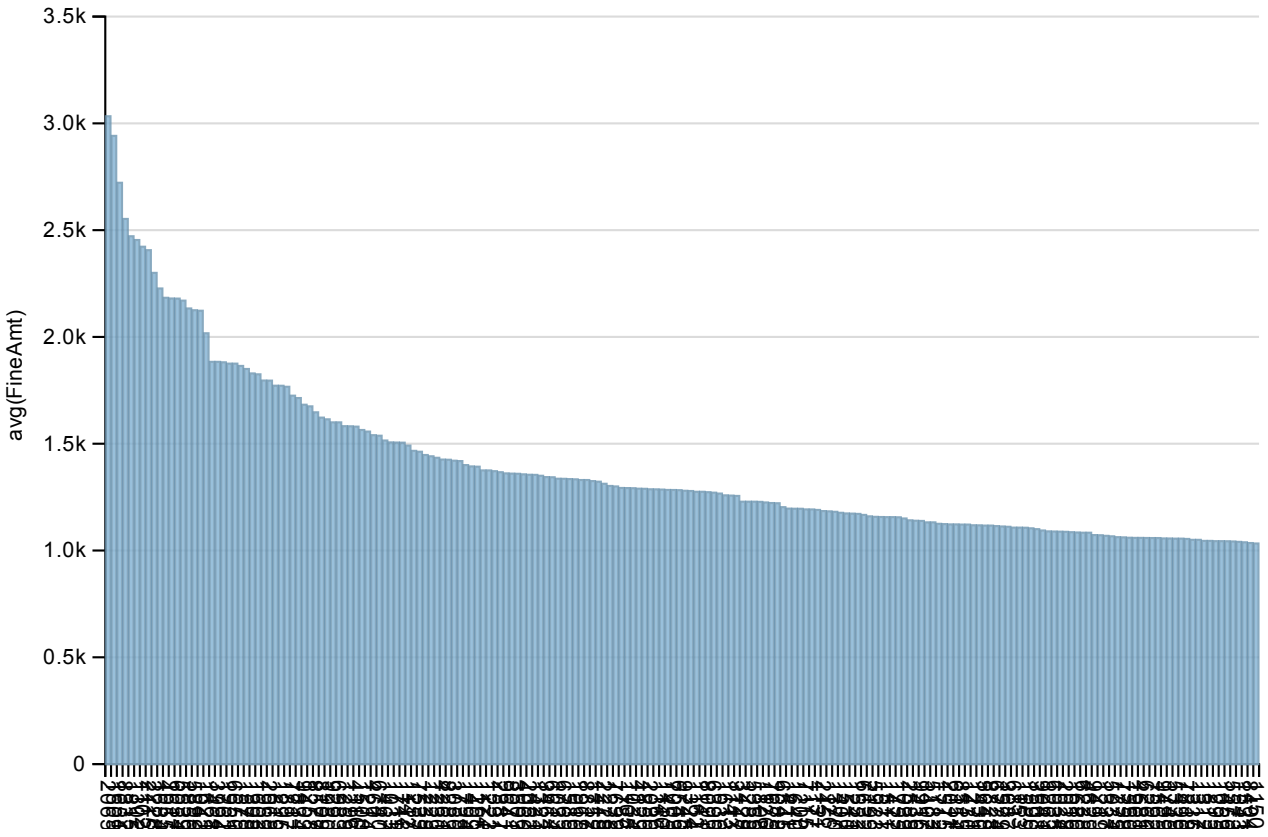
```
widgets.PieChart(blight_violations
  .select("ViolationCode")
  .na.fill(Map("ViolationCode" -> "unknown"))
  .groupBy("ViolationCode")
  .count()
  .collect()
)
```



1 second 51 milliseconds

Plotting a sorted bar chart with the averge fee by street

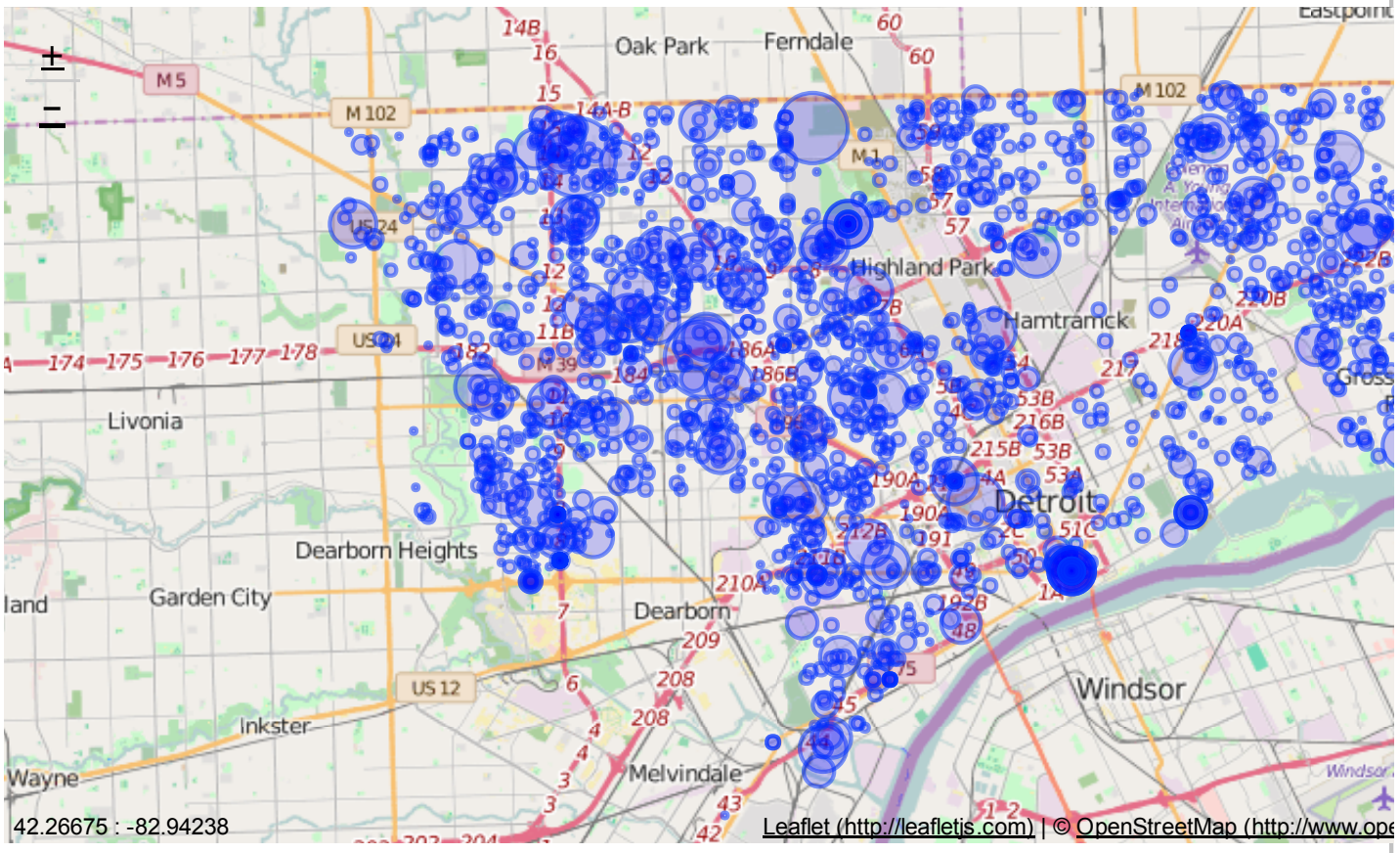
```
// ------------------------------------------------------------------------------
// Average fee by street number
// ------------------------------------------------------------------------------

widgets.BarChart(blight_violations
  .groupBy("ViolationStreetNumber")
  .agg(avg("FineAmt"), count("FineAmt"))
  .filter("count(FineAmt) > 5") // Lets ask for at least 5 ocurrences in that street t
  .drop("count(FineAmt)")
  .sort(desc("avg(FineAmt)"))
  .take(200)
)
```



1 second 473 milliseconds

Finally plotting blight violations geo localized data with its area scaled by the fineamt

```
// ------------------------------------------------------------------------------
// Geo map
// ------------------------------------------------------------------------------

val blight_points = blight_violations_detroit
  .select("Violation_lat", "Violation_lng", "FineAmt")
  .sample(false, 0.008, data_seed)
  .collect()
  .map( r =>
    GeoData(r.getAs[Double]("Violation_lat"),
            r.getAs[Double]("Violation_lng"),
            scala.math.sqrt(r.getAs[Double]("FineAmt") / 3.1416) / 3)
  )
widgets.GeoPointsChart(blight_points,
                       latLonFields=Some(("lat", "lng")),
                       rField=Some("value"),
                       colorField=Some("group"),
                       maxPoints=2000,
                       sizes=(800, 400))
```



4 seconds 622 milliseconds

1.2.- Demolitions

I've had a few problems with the parser for this file and I've had to revert back to de default parser.

Again here extracting coordinate data.

```
// ************************************************************************
// * DEMOLITION ***********************************************************
// ************************************************************************

var demolition_permits = sqlContext.read
  .format("com.databricks.spark.csv")
  // For some reason univocity doesn't work well with tsv
  // I've had to replace \n characters for \b
  // because the default parser doesn't allow new lines even inside quotes
  //.option("parserLib", "UNIVOCITY") // This configuration solves the multi-line fiel
  .option("delimiter", "\t")       // This one is a .tsv
  .option("header", "true")
  .option("inferSchema", "true")    // Automatic schema inference
  .load(data_path + "detroit-demolition-permits.tsv")
  .cache()
  // Extract the geo localization data
  .filter("site_location LIKE '%(%,%)%'") // Only 3 rows without geodata will be disca
  .withColumn("site_lat", get_latitude($"site_location"))
  .withColumn("site_lng", get_longitude($"site_location"))
  .cache()

demolition_permits.printSchema()
demolition_permits.count()
demolition_permits.take(5)
```

Show: 1 ▾ Search:

PERMIT_NO	PERMIT_APPLIED	PERMIT_ISSUED	PERMIT_EXPIRES	SITE_ADDRESS	BETWEE
BLD2015-03955	8/28/15	8/28/15		4331 BARHAM	BETWEE VOIGHT WAVENE
BLD2015-04083	8/28/15	8/28/15		9707 BESSEMORE	BETWEE VINTON GRATIO
BLD2015-03976	8/28/15	8/28/15		5315 BERKSHIRE	BETWEE SOUTHA AND FRANKF
BLD2015-03781	8/28/15	8/28/15		16670 BRINGARD DR	BETWEE CUSHIN SHAKES
BLD2015-03677	8/28/15	8/28/15		1454 BEATRICE	BETWEE TORONT

					GILROY

Showing 5 of 5 records

Pages: Previous 1 Next

1 second 112 milliseconds

1.3.- 311 Calls

Loading 311 calls this time

```
// ****************************************************************************
// * 311 CALLS ****************************************************************
// ****************************************************************************

// ----------------------------------------------------------------------------
// Load and cleanup
// ----------------------------------------------------------------------------

var calls_311 = sqlContext.read
  .format("com.databricks.spark.csv")
  .option("header", "true")
  .option("inferSchema", "true")    // Automatic schema inference
  .option("parserLib", "UNIVOCITY") // This configuration solves the multi-line field
  .load(data_path + "detroit-311.csv")
  .cache()
  // Parse number formats
  .withColumn("lat", $"lat".cast(DoubleType))
  .withColumn("lng", $"lng".cast(DoubleType))
  .withColumn("rating", $"rating".cast(IntegerType))
  // Delete rows with null lat, lng or rating
  .na.drop("all", "lat" :: "lng" :: "rating" :: Nil)
  .cache()
calls_311.printSchema()
calls_311.count()
calls_311.take(5)
```

Show: 1 ▼ Search:

ticket_id	city	issue_type	ticket_status	issue_description	rating	ticket_closed_date_tim
1516722	City of Detroit	Clogged Drain	Acknowledged	Two drains one on each side of street, street floods when ever it rains, drains are clogged down below streets, drains are cleaned at grates by neighbors but drain still clogs and won't drain properly. Drains have been like this for years never got a response from DWSD problem never resolved, trying again.	3	
1525361	City of	Clogged	Acknowledged	standing water on	2	

	Detroit	Drain		lumplin		
1525218	City of Detroit	Clogged Drain	Closed	CITZEN CALLED TO REPORT CLOGGED DRAINS	2	08/15/2015 12:03:43 AM
1525214	City of Detroit	Clogged Drain	Acknowledged	Citizen called DWSD to report clogged drain	3	
1525142	City of Detroit	Clogged Drain	Acknowledged	@ THE CORNER OF GRIGGS & MARGARETA	2	

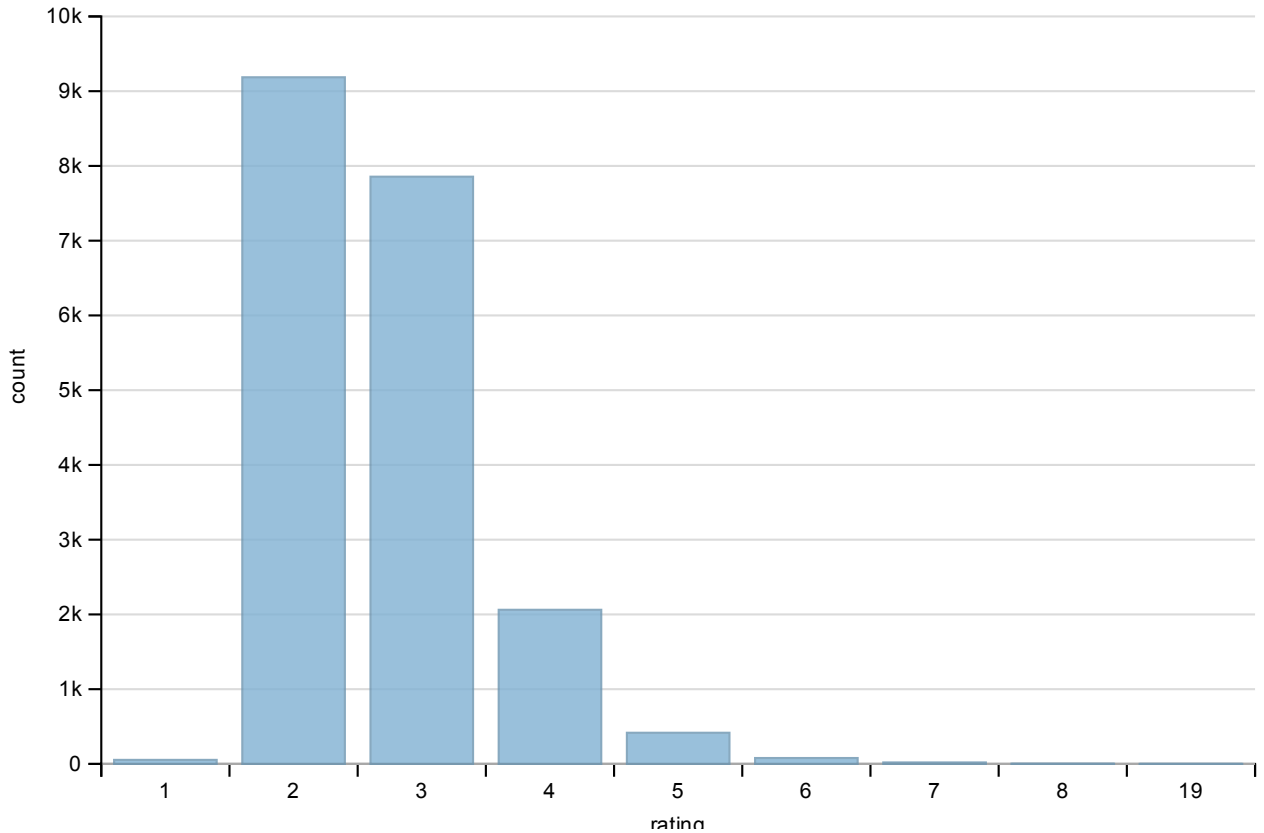
Showing 5 of 5 records

Pages: Previous 1 Next

940 milliseconds

Ploting the ratings in those calls

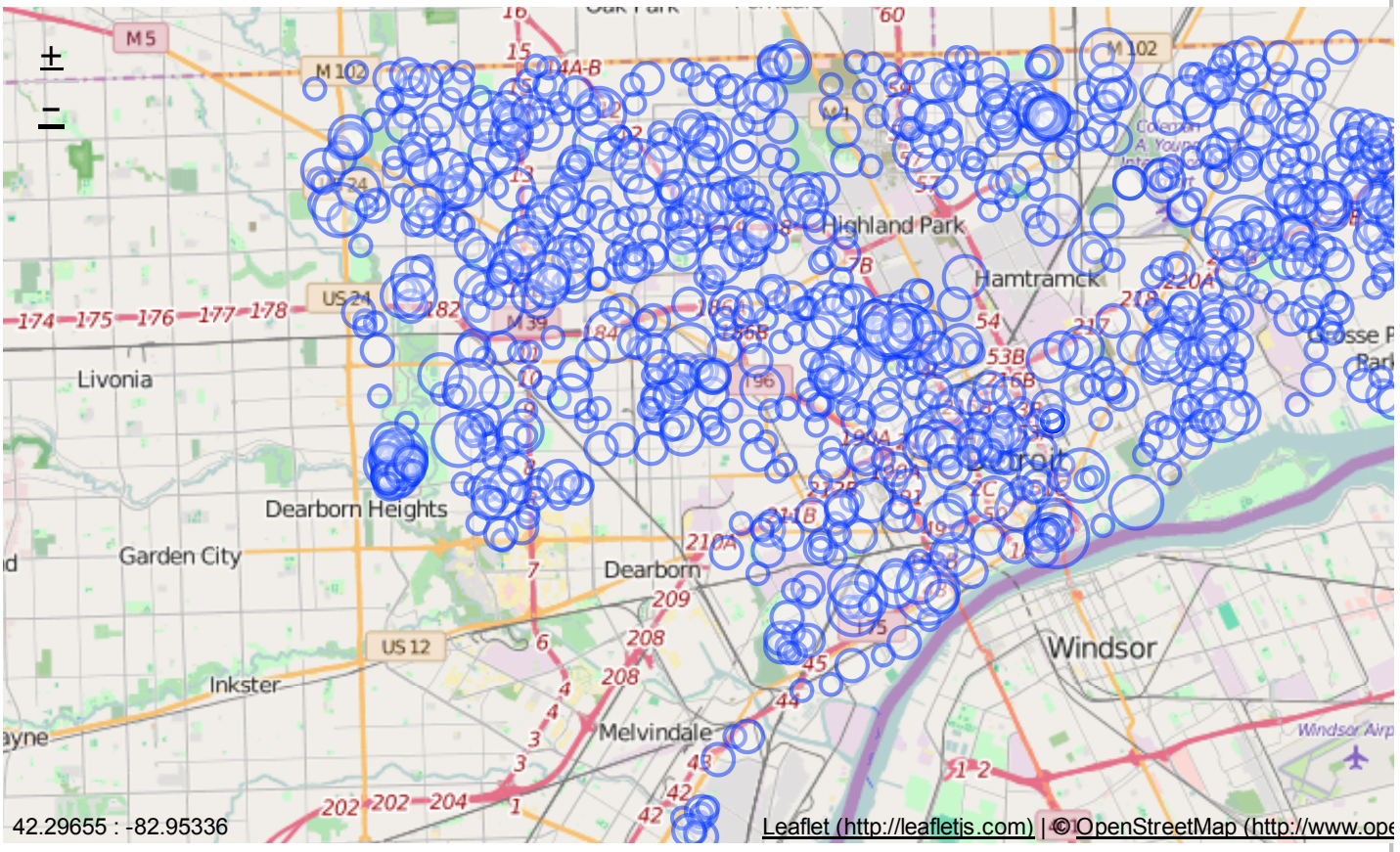
```
// ------------------------------------------------------------------------------
// Bar chart by call rating
// ------------------------------------------------------------------------------

widgets.BarChart(calls_311.na.drop("all", "rating" :: Nil).groupBy("rating").count().c
```



835 milliseconds

Geo localized calls

```
// ------------------------------------------------------------------------------
// Geo map
// ------------------------------------------------------------------------------

val call_points = calls_311
  .select("lat", "lng", "rating")
  .sample(false, 0.05, data_seed)
  .collect()
  .map( r =>
    GeoData(r.getAs[Double]("lat"),
            r.getAs[Double]("lng"),
            r.getAs[Int]("rating").toDouble * 2.5,
           "white")
  )

widgets.GeoPointsChart(call_points,
                       latLonFields=Some(("lat", "lng")),
                       rField=Some("value"),
                       colorField=Some("group"),
                       maxPoints=1000,
                       sizes=(800, 400))
```



682 milliseconds

1.4.- Crimes

Loading crime data

```
// ****************************************************************************
// * CRIMES *******************************************************************
// ****************************************************************************

// ----------------------------------------------------------------------------
// Load and cleanup
// ----------------------------------------------------------------------------

var crimes = sqlContext.read
  .format("com.databricks.spark.csv")
  .option("header", "true")
  .option("inferSchema", "true")    // Automatic schema inference
  .option("parserLib", "UNIVOCITY") // This configuration solves the multi-line field
  .load(data_path + "detroit-crime.csv")
  .cache()

crimes.printSchema()
crimes.count()
crimes.take(5)
```

Show: 1 ▾ Search:

ROWNUM	CASEID	INCINO	CATEGORY	OFFENSEDESCRIPTION	STATEOFFENSEFILE(
53256	1953933	1506030028.1	ASSAULT	ASSAULT AND BATTERY/SIMPLE ASSAULT	13001
17631	1917717	1503010158.1	LARCENY	LARCENY - PARTS AND ACCESSORIES FROM VEHICLE	23006
11207	1910955	1502080223.1	STOLEN VEHICLE	VEHICLE THEFT	24001
116589	2018186	1511090188.1	WEAPONS OFFENSES	WEAPONS OFFENSE (OTHER)	52003
85790	1986862	1508239803.1	LARCENY	LARCENY - PARTS AND ACCESSORIES FROM VEHICLE	23006

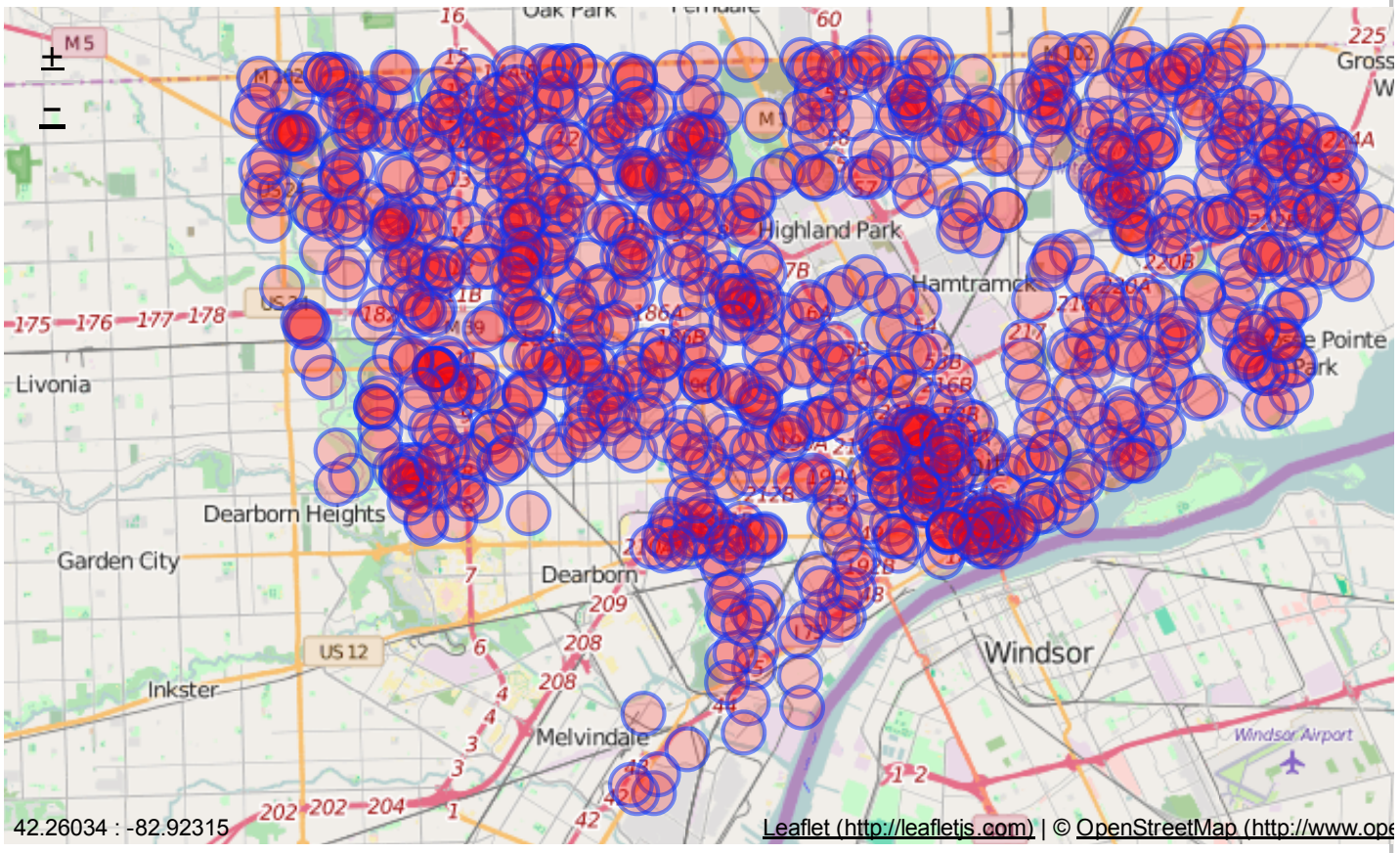
Showing 5 of 5 records

Pages: Previous 1 Next

1 second 607 milliseconds

Geo localized crime data

```
val crime_points = crimes
  .select("LAT", "LON")
  .na.drop("all", "LAT" :: "LON" :: Nil)
  .filter("LAT between -360 and 360")
  .filter("LON between -360 and 360")
  .sample(false, 0.02, data_seed)
  .collect()
  .map( r =>
    GeoData(r.getAs[Double]("LAT"),
            r.getAs[Double]("LON"),
            10.0,
           "red")
  )
widgets.GeoPointsChart(crime_points,
                       latLonFields=Some(("lat", "lng")),
                       rField=Some("value"),
                       colorField=Some("group"),
                       maxPoints=1000,
                       sizes=(800, 400))
```



666 milliseconds

2.- Preparing the data (discretization and labeling)

I've discretized the building using a grid approach.

I'm using 2 with with the same interval and overlaped.

When I assign a "building" to a given set or coordinates, I do it for each grid (the regular and the overlaped one) and then I select the grid cube whose center is closer to the point.

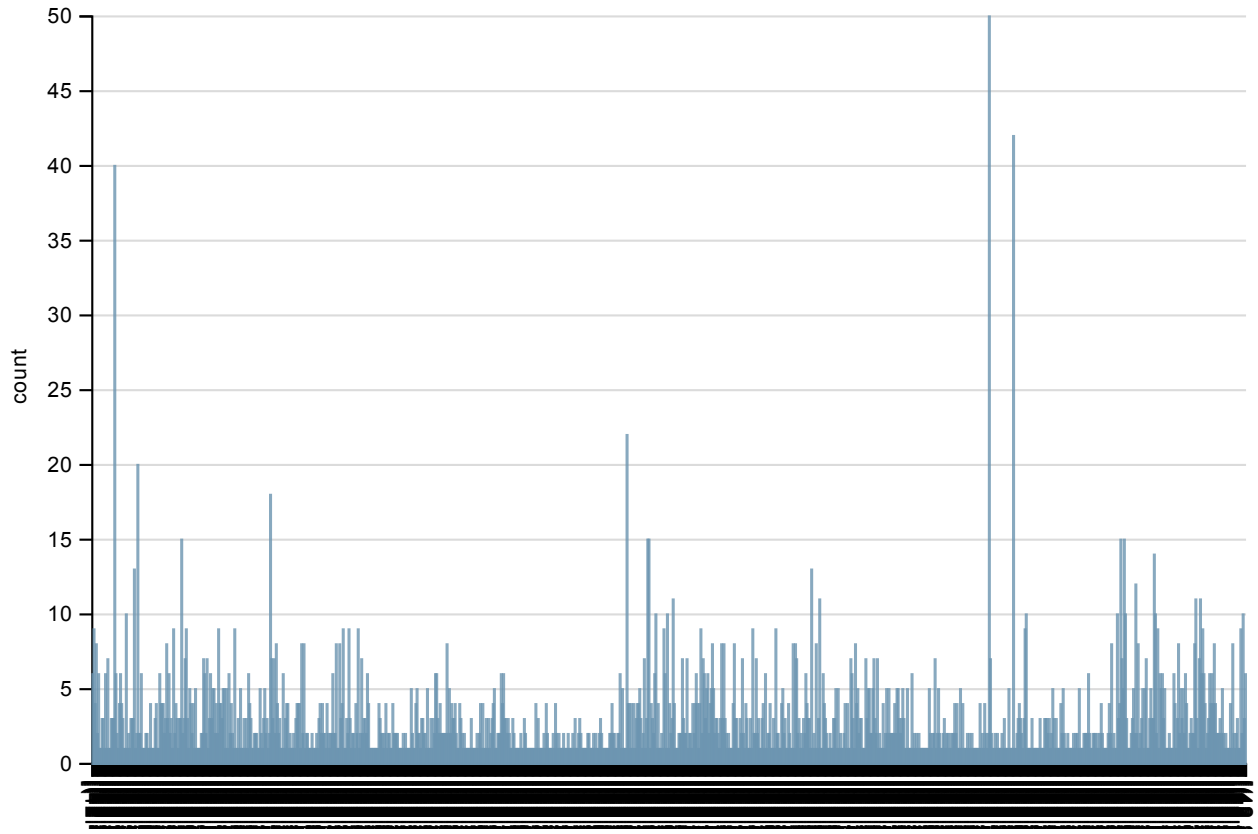
You can check the implentation at the top of this document. I'm using grids with 0.0001 degree interval, but it could be changed at any time at the top of the notebook.

```
// ************************************************************************
// * DISCRETIZATION *******************************************************
// ************************************************************************

// Discretization
blight_violations_detroit = blight_violations_detroit
  .withColumn("building", bestGridStr($"Violation_lat", $"Violation_lng"))
  .withColumn("building_lat", bestGridLat($"Violation_lat", $"Violation_lng"))
  .withColumn("building_lng", bestGridLng($"Violation_lat", $"Violation_lng"))
  .cache()
```

523 milliseconds

Ploting a bar chart to see the count of blight violations per building I've recently created

```
// ------------------------------------------------------------------------------
// Bar chart
// ------------------------------------------------------------------------------

widgets.BarChart(blight_violations_detroit.groupBy("building").count().collect())
```



12 seconds 107 milliseconds

Here I'm using a trivial algorith to assign the blight labels.

I must admit than I'm not familiar with the topic, so this might be the weakest point.

Anyways, the code and the algorithms are trully scalable and configurable so there it goes.

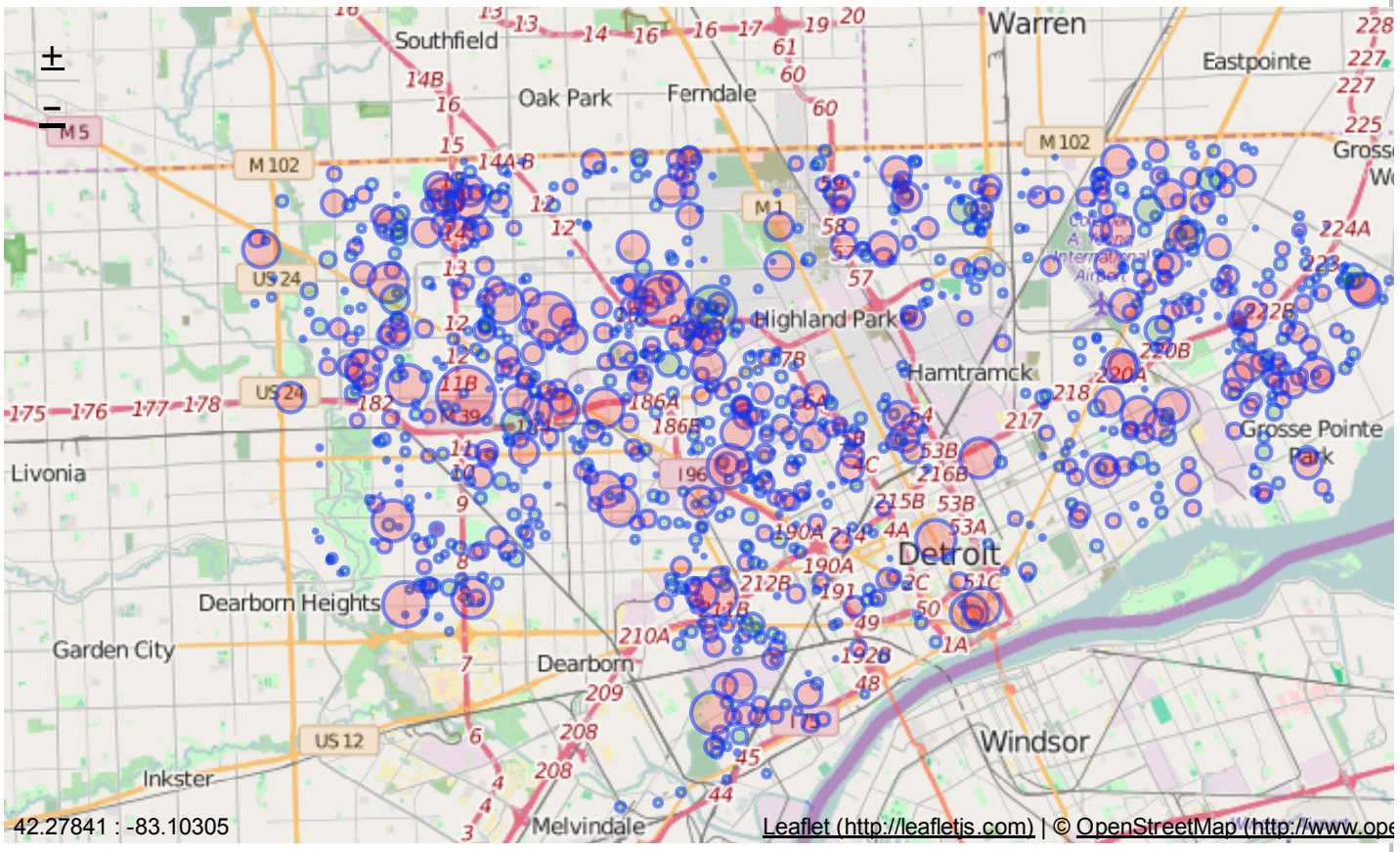
```
// ------------------------------------------------------------------------------
// Labels
// ------------------------------------------------------------------------------

val blight_buildings = blight_violations_detroit.groupBy("building").agg(count("buildi
val blight_labels = blight_buildings.withColumn("label", labeludf($"count(building)",
blight_labels.count() // About 90k buildings detected

blight_violations_detroit = blight_violations_detroit
  .join(blight_labels, blight_violations_detroit("building") === blight_labels("buildi
```

14 seconds 81 milliseconds

Again ploting the data in a map.

It is really hard to see but there are red and green dots in the map.

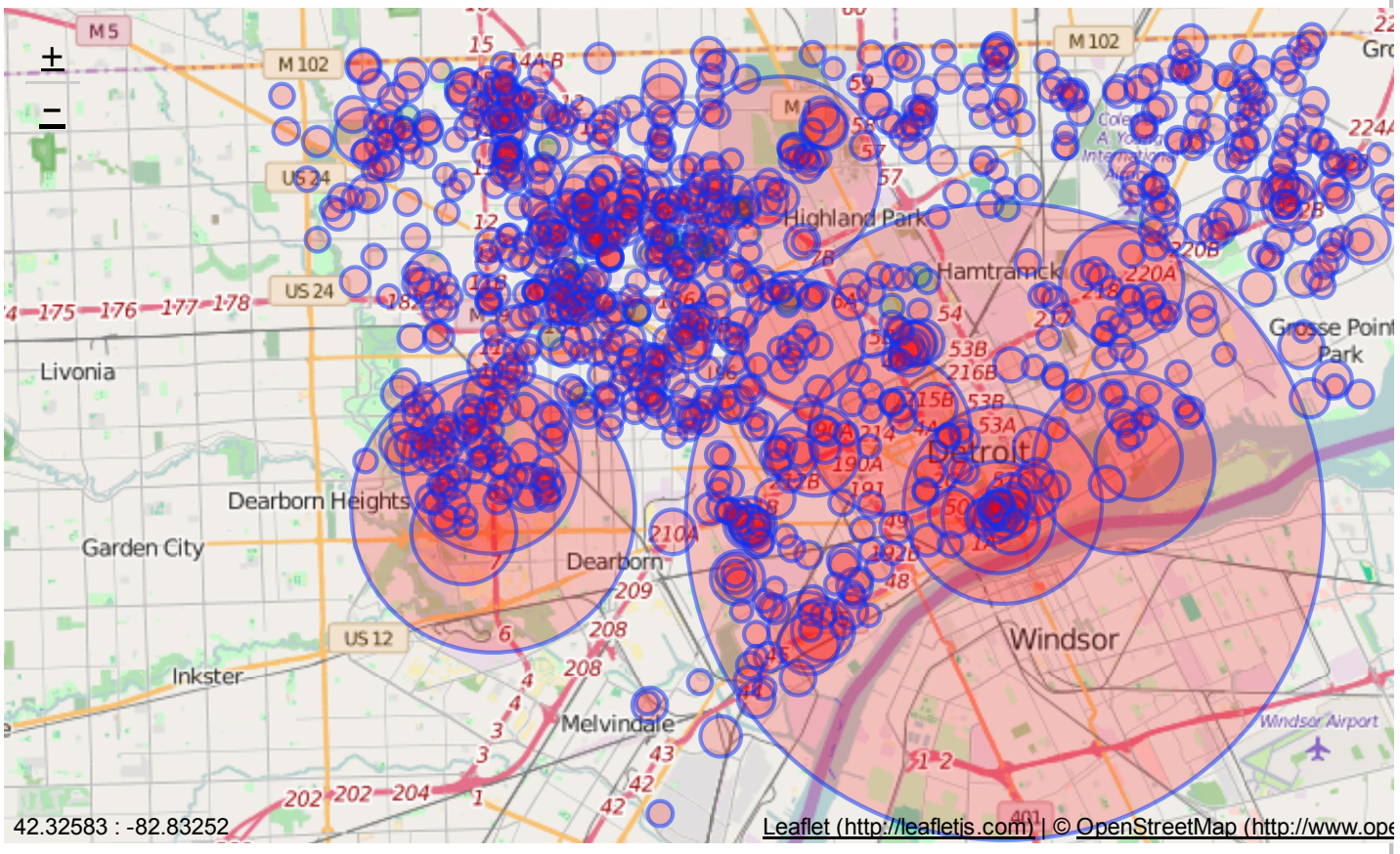
```
// ------------------------------------------------------------------------------
// Geo map sample
// ------------------------------------------------------------------------------

val blight_building_points = blight_violations_detroit
  .select("building_lat", "building_lng", "label", "FineAmt")
  .groupBy("building_lat", "building_lng", "label")
  .agg(sum($"FineAmt"))
  .sample(false, 0.03, data_seed)
  .take(1000) // There are 93489 Buildings derived, so lets take the most costly 1000
  .map( r =>
    GeoData(r.getAs[java.math.BigDecimal]("building_lat").doubleValue(),
            r.getAs[java.math.BigDecimal]("building_lng").doubleValue(),
            scala.math.sqrt(r.getAs[Double]("sum(FineAmt)") / 3.1416) / 5,
            if (r.getAs[Boolean]("label")) "red" else "green")
  )

widgets.GeoPointsChart(blight_building_points,
                       latLonFields=Some(("lat", "lng")),
                       rField=Some("value"),
                       colorField=Some("group"),
                       maxPoints=2000,
                       sizes=(800, 400))
```



25 seconds 74 milliseconds

Now I'm just plotting the most expensive buildings

```
// ------------------------------------------------------------------------------
// Geo map worst blights
// ------------------------------------------------------------------------------

val blight_building_points_bad = blight_violations_detroit
  .select("building_lat", "building_lng", "label", "FineAmt")
  .groupBy("building_lat", "building_lng", "label")
  .agg(sum($"FineAmt"))
  .sort(desc("sum(FineAmt)"))
  .take(1000) // There are 93489 Buildings derived, so lets take the most costly 1000
  .map( r =>
    GeoData(r.getAs[java.math.BigDecimal]("building_lat").doubleValue(),
            r.getAs[java.math.BigDecimal]("building_lng").doubleValue(),
            scala.math.sqrt(r.getAs[Double]("sum(FineAmt)") / 3.1416) / 10,
            if (r.getAs[Boolean]("label")) "red" else "green")
  )

widgets.GeoPointsChart(blight_building_points_bad,
                       latLonFields=Some(("lat", "lng")),
                       rField=Some("value"),
                       colorField=Some("group"),
                       maxPoints=1000,
                       sizes=(800, 400))
```



22 seconds 627 milliseconds

3.- Training a model

First of all we need to "featurize" the dataset and then split it into training and test.

```
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.mllib.linalg.Vectors

// Prepare out training dataset
val assembler = new VectorAssembler()
  .setInputCols(Array("count(building)"))
  .setOutputCol("features")

val dataset = assembler
  .transform(blight_labels)
  .withColumnRenamed("building", "id")
  .select("id", "features", "label")
  .withColumn("label", $"label".cast(DoubleType))
  .na.drop("all", "label" :: Nil ) // everything must be labelled

val split = dataset.randomSplit(Array(0.8, 0.2), seed = data_seed)
val training_data = split(0)
val test_data = split(1)

dataset.groupBy("label").count().collect()
```

Show: 1

Search:

label	count
1	34833
0	58204

Showing 2 of 2 records

Pages: Previous 1 Next

9 seconds 141 milliseconds

Now lets create a pipeline containing a cross validation of 5 folds and train/fit the model.

In this simple case I've used a logistic regression model.

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.tuning.{ParamGridBuilder, CrossValidator}

// Setup our ML pipeline
val lr = new LogisticRegression()
  .setLabelCol("label")
  .setFeaturesCol("features")
  .setMaxIter(10)
val pipeline = new Pipeline().setStages(Array(lr))

val evaluator = new BinaryClassificationEvaluator

val cv = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(evaluator)
  .setEstimatorParamMaps(new ParamGridBuilder().build()) // No parameter search
  .setNumFolds(5)

// Run cross-validation, and fit the estimator
val cvModel = cv.fit(training_data)
```

38 seconds 998 milliseconds

Testing the accuracy of the model.

It is completely misleading. as I said before I didn't really understand the domain so the label selection is the guilty in this case.

```
val predictions = cvModel.transform(test_data)
evaluator.evaluate(predictions)
```

0.9555435870709632

2 seconds 853 milliseconds

I'm on my way to upload the last part using a random forest model. It will be up in 24h.