Spark Streaming from Kafka Topic

READY

```
%dep

// Load all the dependencies
// This will let us connect Spark Streaming to Kafka topics

z.load("spark-streaming-kafka-0-10_2.11-2.2.1.jar")
z.load("spark-sql-kafka-0-10_2.11-2.1.1.jar")
z.load("kafka-clients-0.11.0.1.jar")
```

```
res0: org.apache.zeppelin.dep.Dependency = org.apache.zeppelin.dep.Dependency@525f997
```

READY

```
sc.version
```

```
res1: String = 2.2.1
```

READY

```
import org.apache.spark.streaming._
import org.apache.spark.sql.types._
```

```
import org.apache.spark.streaming._
import org.apache.spark.sql.types._
```

READY

```
/*
    6371.0 is the mean radius of the Earth in km
    3958.761 is the mean radius of the Earth in miles
*/

def haversineDistance(pointA: (Double, Double), pointB: (Double, Double)): Double = {
  val deltaLat = math.toRadians(pointB._1 - pointA._1)
  val deltaLong = math.toRadians(pointB._2 - pointA._2)
  val a = math.pow(math.sin(deltaLat / 2), 2) + math.cos(math.toRadians(pointA._1)) * math.cos(math.toRadians(poi
  val greatCircleDistance = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
  6371.0 * greatCircleDistance
}
```

```
haversineDistance: (pointA: (Double, Double), pointB: (Double, Double))Double
```

READY

```
val kafkaStream = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "localhost:9092")
  .option("subscribe", "live_flights")
  .option("startingOffsets","latest")
  .load()
```

```
kafkaStream: org.apache.spark.sql.DataFrame = [key: binary, value: binary ... 5 more fields]
```

READY

```
kafkaStream.printSchema
```

```
root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

READY

```
val pointMontreal = (45.4690, -73.7378)
```

```
pointMontreal: (Double, Double) = (45.469,-73.7378)
```

READY

```
val dataStream = kafkaStream.selectExpr("CAST(value AS STRING)").as[String]
```

```
dataStream: org.apache.spark.sql.Dataset[String] = [value: string]
```

```
case class DistSchema(
    ac_number: String,
    ac_distance_km: Double)
```

READY

```
defined class DistSchema
```

```
val distance = dataStream.map(row => row.split("[:,}]")).map(row => (row(2), haversineDistance((row(11).toDouble,
            .map(row => DistSchema(row._1,row._2))
```

READY

```
distance: org.apache.spark.sql.Dataset[DistSchema] = [ac_number: string, ac_distance_km: double]
```

```
distance.printSchema
```

READY

```
root
 |-- ac_number: string (nullable = true)
 |-- ac_distance_km: double (nullable = false)
```

```
distance
      .writeStream
      .format("console")
      .outputMode("append")
      .start()
      .awaitTermination()
```

READY

READY