ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/336936987

# Geospatial and temporal data analysis on NYC taxi trip data

**Article** *in* Journal of Data Analysis and Information Processing · December 2016

CITATIONS
0

READS
4

**10 authors**, including:

Irina Matijosaitiene
Saint Peter's University

**25** PUBLICATIONS **50** CITATIONS

SEE PROFILE

**Some of the authors of this publication are also working on these related projects:**

Artificial Intelligence and Machine Learning for Predicting Urban Mobility and Territory Segmentation View project

CPTED for Safer Society View project

All content following this page was uploaded by Irina Matijosaitiene on 01 November 2019.

The user has requested enhancement of the downloaded file.

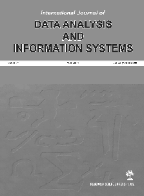
Research Science Press

International Journal of Data Analysis and Information Systems

VOLUME 8 • NUMBER 2 • DECEMBER 2016

journal homepage : serialsjournals.com

ISSN : 2229-5887




# Geospatial and Temporal Data Analysis on the New York City Taxi Trip Data

**Venkatesh Subramaniam*, Gowtham Bharath Srungarapu*, Irina Matijosaitiene*, Mohit Supe*, Aditi Agarwal*, Peng Zhao*, Xin Wang*, Edward Kwartler** and Sylvain Jaume***

* *Saint Peter's University, USA*

** *Liberty Mutual Inc., USA*

Authors emails

*sylvain@csail.mit.edu*

© 2016 Research Science Press. All rights reserved

ABSTRACT

*The New York City taxi business is one of the interesting fields for data analysis. The analysis is done on spark and it concentrate on the duration of wait time for the drivers after a successful ride based on location. The data used is from NYC Taxi and Limousine department. The geospatial and temporal data is made to good use in spark and the insights are derived. The final result that is desired is in the form of location and average wait time for next passenger. Hence the data can be used to find the good place to get the customer or lesser wait time in order to get the next customer. The results are pretty assuring and sensible. For example, it is possible to get customers quicker in Manhattan than in Bronx and the output exactly shows that.*

## INTRODUCTION

Taxi business is one of the largest in New York City (Farber, 2008). The way the yellow taxi works is that the drivers should be working in shift to limit the number of taxicab that is running in the city (Camerer *et al.*, 1997). The reason they work in shift is because they want to limit the number of taxicabs that run at a given time in the road (Ferreira *et al.*, 2013). The other reason is everyone in the taxicab business has to get equal number of customers and the profit that everyone makes is almost equal (Harcourt *et al.*, 2006). Hence this approach has got a lot of appreciation from the cab drivers and has been followed (Ryza *et al.*, 2015). There are two types of taxi cabs that run in New York City (Splechtna *et al.*, 2016). One is New York City Yellow Taxi cab that we are analyzing in this project. The other type is New York City Green Taxi cab which is also popular but not as popular as yellow taxi cab. That is the reason we analyze only New York City yellow taxi cab in this analysis. The main objective of this project is to find the time when the cab is occupied and when it is not. It can be found by using the passenger destination location. For example, if a passenger gets down at Bronx, NY, the time taken for the next passenger is 5 min and if the passenger gets down in Long Island, NY, the time taken for next passenger is 40 min, the result will be that Bronx is better place to get next customer quickly.

## DATA

The data is available from January 2013. The size of the data is 2.5GB before unzipping and uncompressing. Hence setting up Spark for 2.5GB of compressed data is not a reasonable task. The data is in Comma Separated Format (CSV). The variables include latitude, longitude, temporal data such as date and time. The data after unzipping and uncompressing is around 20GB. So for analysis purpose, we take just the data for January 2013 data. The reason being that the data is huge and analyzing just one-month data looks reasonable. Therefore the code runs faster and we can see how the model is built quicker and make modifications and check the process again. In other words, for checking the code and de bugging and finding how good is the model, we are choosing a

month data. The data seems to be sufficient to start the analysis.

## METHOD

The process for the project is as follows. We know what we need to have in order to complete the analysis. We set up all the tools and libraries needed to complete the analysis. The process needed to set the environment is connecting to Amazon Web Services EMR, Uploading Data In The S3 Bucket, installing spark and running spark and Importing The Data From S3 Bucket. Each of these steps are clearly explained on how to set up in mac system. Windows system also follows similar procedure but the only change is in place of terminal, you have to use command prompt or anaconda prompt. The connectivity with Amazon Web Services EMR has to be done with Putty. Putty is a connectivity terminal similar to the terminal command. The key pair generated, call pem file, is the tricky part because the permission can be given easily in terminal but have to follow a different approach in Putty. Hence if using Windows, follow the instruction given in the manual developed by the Putty developers to establish the connection with the Amazon Web Services EMR.

### Connecting to Amazon Web Services EMR

The Steps for connecting to Amazon Web Services EMR involves three steps. The first step is to create an EMR instance. The second one is to create a create a key pair. The third step is to connect to the local system using terminal. We shall go in detail for each step. To create the key value pair, go to key pair options in the home page. Click on new key pair. Give a new name to it. Download the key pair and know the location where it is saved. To create a EMR instance, login into your Amazon Web Services console site. Select EMR option from the Analytics section. Then click on create cluster. The quick option windows opens up. Under Software Configuration, choose Spark: Spark 2.0.1 on Hadoop 2.7.3 YARN with Ganglia 3.7.2 and Zeppelin 0.6.2. Under Hardware Configuration section, choose the number of clusters to be 4. Under Security and Access window, select the key pair you created and click next. Go to advanced settings and check for the master node and slave nodes. To connect to the local server, open the terminal. Give the key pair access power by typing the following command CHMOD 600 /keyname.pem. Then connect to the EMR by clicking the ssh – I key name.pem Hadoop@ec2 – 35 – 160 – 198 – 2.us – west – 2. compute amazonaws. com. It will ask a prompt message whether to continue, enter yes. The terminal will run successful and EC2 image will be displayed along with a successful message as shown in Figure 1.

### Uploading Data to the S3 Bucket:

To access the huge data that we will use, we upload it to a S3 Bucket. S3 is available from Storage and Content Delivery section. S3 is nothing but a storage place for data that we will use for our analysis. The steps to upload the data into S3 is very simple. After clicking the S3 icon under the Storage and Content Delivery

```
[mbp-p203-ds10:~ vsubramaniam$ cd Downloads/
[mbp-p203-ds10:Downloads vsubramaniam$ chmod 600 nyc.pem
[mbp-p203-ds10:Downloads vsubramaniam$ ssh -i nyc.pem hadoop@ec2-35-160-198-2.us-west-2.compute.amazonaws.com
 The authenticity of host 'ec2-35-160-198-2.us-west-2.compute.amazonaws.com (35.160.198.2)' can't be established.
 ECDSA key fingerprint is SHA256:VhvPhDvPwBYsXYz2gAf86L00YXvvbBo+vUF/0EI+xfg.
 Are you sure you want to continue connecting (yes/no)? yes
 Warning: Permanently added 'ec2-35-160-198-2.us-west-2.compute.amazonaws.com,35.160.198.2' (ECDSA) to the list of known hosts.
 Last login: Fri Nov 11 17:52:58 2016

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

 https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/

 EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
 E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
 EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
   E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
   E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
   E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
   E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
   E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
   E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
   E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
 EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
 E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
 EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

 [hadoop@ip-172-31-2-29 ~]$
```

**Figure 1:** EMR installation

section, a new page opens. Click the create bucket icon. Give a name to be your folder name. Click create icon. Then go inside the folder and click on upload icon. Browse through your computer to select the data and click upload. After the data is uploaded successful, check if the data is available in your S3 Bucket. In our case, as the data is huge we subset our data. The Data uploaded is for one month. The original data had data for 2 years but to test the code against a smaller chunk of data, uploaded January 2013 data in the S3 bucket. The S3 Bucket after uploading is shown in Figure 2.

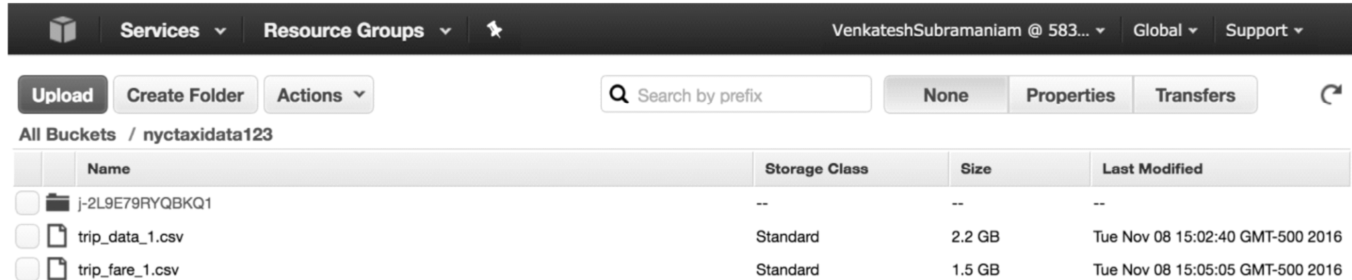



**Figure 2:** S3 Bucket

## Running Spark

After the data is successfully uploaded, we run spark in our local system. The spark is run on the terminal. So, open the terminal and ran the command spark-shell. The command runs and prompts whether to run spark, enter yes. The command will run successfully and spark is shown in the terminal. The command output is shown in Figure 3.

```
[hadoop@ip-172-31-2-29 ~]$ spark-shell
\Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
16/11/11 18:18:49 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
16/11/11 18:19:03 WARN SparkContext: Use an existing SparkContext, some configuration may not take effect.
Spark context Web UI available at http://172.31.2.29:4040
Spark context available as 'sc' (master = yarn, app id = application_1478635431473_0005).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.0.1
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_111)
Type in expressions to have them evaluated.
Type :help for more information.
```

**Figure 3:** Spark Installation

## Running Spark from Databricks

The reason for running the Databricks is because the output is more clear and easy to understand. The same command works in the terminal as well. So open the Databricks site and open a notebook selecting the Scala language if you are using Databricks. If you are using terminal, then you don’t need to do anything as running spark in terminal will enable you to run Scala code. If you are running directly with Databricks, you do not need to set up the Spark in the terminal as Databricks runs in the web and has its own Spark set up with the notebook.

## Importing the Data from a S3 Bucket

To connect with the S3 Bucket from Databricks needs a command. The tricky part is to know the difference between S3, S3n and S3a. The letter change makes a big difference because the algorithm it uses to interface with the Databricks is different. The S3 is a block based overlay on top of the Amazon S3 Bucket whereas S3n and S3a is a object based overlay on the top of Amazon S3 bucket. The difference between S3a and S3n is that S3n can handle the objects up to 5GB in capacity whereas S3a can handle the objects up to 5 Tera Bytes. When we compare the performance between S3a and S3n, the performance for S3a is higher than S3n. The reason is because they are multi part upload. The difference is illustrated in the table in Figure 4.

| S3 | S3n | S3a |
| --- | --- | --- |
| Block-based overlay on top of Amazon S3 | Object based with up to 5GB file data | Object based with up to 5TB file data |
| Only the data should be present and not any other file | Any Number of data can be present | Any number of data can be present |

**Figure 4:** Difference between S3, S3n and S3a

The command to connect the DataBricks with S3 Bucket is as follows
val data = sc.textFile("s3n://nyctaxidata123/nyc.pem")

## Choosing the Libraries

The advantage of using Scala is because it has got a variety of libraries and library dependencies that can make our analysis process easy and accurate. The disadvantage of the same point is one should have a knowledge about what packages to use in order to make full use of the Spark with Scala combination. There are many similar packages with same functionality but the algorithm used to obtain results is different. Changing between these packages can actually affect your results because the algorithm is important. So, we use our packages carefully to make our analysis easy and accurate. The coding is based on few existing libraries. The library Kyro is used to serialization so there is a serializable interface to work on. The disadvantages of using Maven and SBT to deal with the external dependencies are because we want our application to be interactive. So we choose libraries with lesser dependencies. Java libraries with Scala Wrappers are the best solution to the above problem. There is date and time data , so as a beginner one would choose Java temporal class but the disadvantage is they use space for small operations. So using NScalaTime and JodaTime is better as they have advantage over the Java class.

## Reading the Temporal Data

Temporal Data is a data consisting of time frame details such as Date, Time, Year, Quarter, Decade. The time data is always important component of analysis because if they are present, the scope of analysis increases to different dimension. The data is read using NScalaTime and JodaTime libraries. The import and manipulation of data is illustrated in Figure 5.

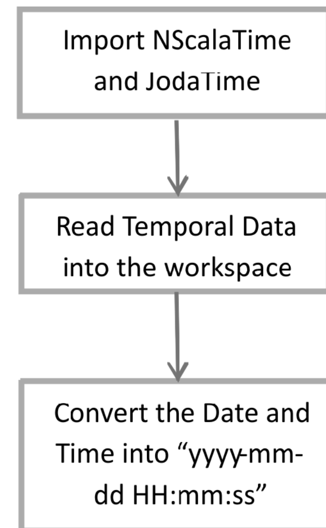



**Figure 5:** Workflow for importing the Temporal Data

The libraries to read the temporal data are nscala_time and JodaTime. The nscala_time library is built around joda library. It is nothing but Scala Wrapper which is used to add, subtract or find duration between dates. The usage is also simple, that is you can call by simply calling a function. JodaTime is a standard date and time format based for creating new packages. It has a simple API, which allows users to access with multiple calendar systems. As you can see in the below example, with the help of nscala library, a new Date Time is created. The operation of addition is performed on the two date variables. The second operation performed is subtraction on the two date variables. Following that, the next operation performed is duration between two date columns. This is just an example to explain the functionality of the nscala library. The .get will render the date in the format specified by the function used before the .get function. This is the advantage of using temporal data on the Scala platform. Joda Time library is a dependency, so it is added as a central package in the DataBricks notebook.

```
> import com.github.nscala_time.time.Imports._

import com.github.nscala_time.time.Imports._
Command took 5.23 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:43:00 PM on My Cluster (6 GB)

> val dt1 = new DateTime(2014, 9, 4, 9, 0)

dt1: org.joda.time.DateTime = 2014-09-04T09:00:00.000Z
Command took 0.70 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:43:15 PM on My Cluster (6 GB)

> dt1.dayOfYear.get

res8: Int = 247
Command took 0.10 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:17:19 PM on My Cluster (6 GB)

> val dt2 = new DateTime(2014, 10, 31, 15, 0)

dt2: org.joda.time.DateTime = 2014-10-31T15:00:00.000Z
Command took 0.16 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:43:23 PM on My Cluster (6 GB)
```

**Figure 6:** Code used to import the temporal data

```
> import java.text.SimpleDateFormat

import java.text.SimpleDateFormat
Command took 0.09 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:43:52 PM on My Cluster (6 GB)

> val format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")

format: java.text.SimpleDateFormat = java.text.SimpleDateFormat@4f76f1a0
Command took 0.11 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:43:57 PM on My Cluster (6 GB)

> val date = format.parse("2014-10-12 10:30:44")

date: java.util.Date = Sun Oct 12 10:30:44 UTC 2014
Command took 0.16 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:44:00 PM on My Cluster (6 GB)

> val datetime = new DateTime(date)

datetime: org.joda.time.DateTime = 2014-10-12T10:30:44.000Z
Command took 0.14 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:44:03 PM on My Cluster (6 GB)

> val d = new Duration(dt1, dt2)

d: org.joda.time.Duration = PT4946400S
Command took 0.11 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 5:44:41 PM on My Cluster (6 GB)
```

**Figure 7:** Code used to format the temporal data

## Reading the Geospatial Data

Geospatial data is a data that consists of location information such as latitude, longitude, zip code, borough, city, state, country, continent or any other parameter used to locate a place. There are two types of geospatial data, one is vector and the other is raster. The data provided by the nyctaxi is a vector with GeoJSON format. The obstacle here is to make sure that the coordinates are present in the New York Borough. There is no existing library that can solve the above problem, so the existing Esri Geometry API is edited to the convince of the problem. The Esri Geometry API has GeometryEngine that is used to find relationship between coordinates. The main advantage of using Scala for geospatial data is to use the esri geometry. The .com esri geometry is a API used for 3rd party data processing solutions. The main usage is for map reduce users or using map reduce applications for Hadoop system. It is used to import the geometry from the shape files. The shape file is nothing but a map built based a geometric shape. The geometric shapes can be polygon, triangle, square, rectangle, circle and their like. The spatial operations such as union, difference, intersect, clip, cut and extension of boundaries can be done. The map relationships can also be explored with the help of the esri.

```
> import com.esri.core.geometry.Geometry
  import com.esri.core.geometry.GeometryEngine
  import com.esri.core.geometry.SpatialReference

import com.esri.core.geometry.Geometry
import com.esri.core.geometry.GeometryEngine
import com.esri.core.geometry.SpatialReference
Command took 0.15 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 6:40:00 PM on My Cluster (6 GB)
```

```
> class RichGeometry(val geometry: Geometry,
   val spatialReference: SpatialReference =
   SpatialReference.create(4326)) {
   def area2D() = geometry.calculateArea2D()
   def contains(other: Geometry): Boolean = {
   GeometryEngine.contains(geometry, other, spatialReference)
   }
   def distance(other: Geometry): Double =
   GeometryEngine.distance(geometry, other, spatialReference)
  }

defined class RichGeometry
Command took 0.29 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 6:40:03 PM on My Cluster (6 GB)
```

**Figure 8:** Code used to load the geospatial data

## Using the GeoJSON package

GeoJSON is a JavaScript Object Notation for representing geographical locations. The features include points (address and location), line strings (streets, highway and boundaries), polygon (countries and region of land) and other types related to location. The layer is added and the vector locations are added to the layer. The spray package is a light lightweight JSON file implemented in Scala. It is used for efficient JSON parser. The spray package also allows you to convert String JSON documents, JSON Abstract Syntax Trees (AST) with base type JsValue and instances of arbitrary Scala types. The Conversion can be between String JSON documents and JSON Abstract Syntax Trees (AST) with base type JsValue or JSON Abstract Syntax Trees(AST) with base type JsValue and String JSON documents or JSON Abstract Syntax Trees(AST) with base type JsValue and instances of arbitrary Scala types or instances of arbitrary Scala types and String JSON documents or JSON Abstract Syntax Trees (AST) with base type JsValue or String JSON documents and instances of arbitrary Scala types or instances of arbitrary Scala types and String JSON documents. It utilizes SJSONs Scala-colloquial sort class-based way to deal with interface a current sort T with the rationale how to (de)serialize its occurrences to and from JSON.

(Indeed shower json even reuses some of SJSONs code, see the "Credits" segment underneath). This approach has the benefit of not requiring any change (or even access) to Ts source code. All deserialization rationale is appended 'all things considered'. There is no reflection included, so the subsequent transformations are quick. Scala's superb sort derivation lessens verbosity and standard to a base, while the Scala compiler will set aside a few minutes that you gave all required (de)serialization rationale. In shower jsons wording a "JsonProtocol" is only a group of verifiable estimations of sort JsonFormat, whereby each JsonFormat contains the rationale of how to change over occasion of T to and from JSON. All JsonFormats of a convention should be "mece" (fundamentally unrelated, all things considered thorough), i.e. they are not permitted to cover and together need to traverse assorted types required by the application. This may sound more entangled than it is. splash json accompanies a Default Json Protocol, which as of now covers the greater part of Scala's esteem sorts and additionally the most essential reference and gathering sorts. For whatever length of time that your code utilizes simply these you just need the Default Json Protocol.

```
> object RichGeometry {
   implicit def wrapRichGeo(g: Geometry) = {
   new RichGeometry(g)
   }
  }

warning: there were 1 feature warning(s); re-run with -feature for details
defined module RichGeometry
warning: previously defined class RichGeometry is not a companion to object RichGeometry.
Companions must be defined together; you may wish to use :paste mode for this.
Command took 0.18 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 6:40:06 PM on My Cluster (6 GB)

> import RichGeometry._

import RichGeometry._
Command took 0.08 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 7:01:02 PM on My Cluster (6 GB)

> import spray.json.JsValue

import spray.json.JsValue
Command took 0.39 seconds -- by vsubramaniam@saintpeters.edu at 11/10/2016, 6:40:12 PM on My Cluster (6 GB)
```

**Figure 9:** Code used to process the geospatial data

## Handling noise in the data

The size of the data is so large that it is impossible to find invalid values in them. So we create a exception in the code to jump to next value if there is a invalid entry. It can be done by creating a function called safeParse() and apply these conditions. The other option is to use try catch to filter invalid records. This method is feasible only when there is small fraction of the data is invalid. The filtering and mapping is done together to get rid of invalid records with the help of collect statement. The collect statement takes partial function as a argument. Most of the invalid records in the data is missing values, so it is easy to identify and get rid of it. For temporal data, the time when a passenger takes the ride will always be earlier to the time when he/she reached the destination. If the condition is not true, then the record is invalid. These kind of problems may occur because of the data quality distortion when it is imported into the workspace. To overcome the problem, we define a function called hours(). The function is calculated by the difference between (destination time - pick off time). For example, if the pick up time is Monday morning 10 am and drop time is Monday morning 9.30 am. The record is nothing more than noise because the event is highly impossible. The pick up time should always be less than the drop off time.

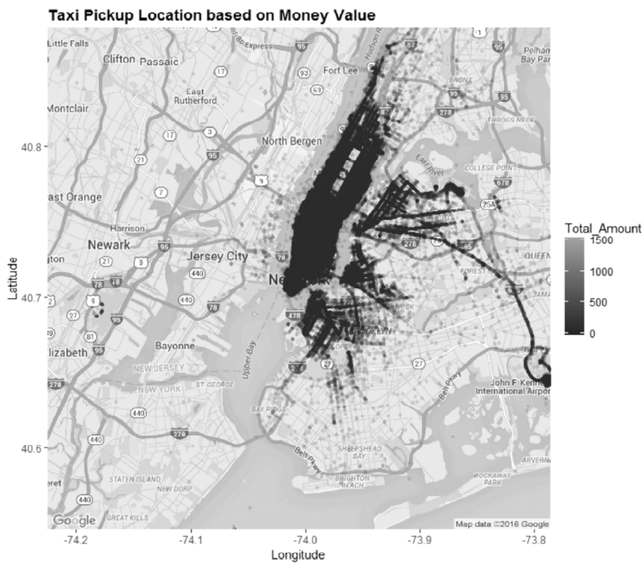



**Figure 10:** Plot of the geospatial data for NYC taxi

To be on a lighter note, it is only possible if pick up guy and drop him off using a time travel. Handling this kind of noise is important because these factors will affect the accuracy of the model which build at the end.

Hours() = Time when reached destination – Time when the journey started

The negative values are removed because it doesn't make any sense as the start time will never be greater than end time.

## Location Analysis

The data is concerned about five boroughs, so the data which has a destination outside these borough will be invalid data as the taxi won't be used for distance greater than two boroughs. The objective can be achieved by converting the pickup latitude, longitude and drop off latitude, longitude and convert it into borough. Then a conditional statement will parse the records that are in those five boroughs. Filtering of the invalid records are done and the final data is good for analysis. To elaborate on that, if we have a data that is outside USA or outside the five borough is present, the location outside the scope of the dataset we are dealing with. So if the present data is for London, London is not in the country we analyze or present in the five boroughs we specifically analyze. So, we remove the location data from the remaining date. So, after this cleaning process, the data will have location as either of these five boroughs. This cleaning process is followed by the sessionization of the data so we do not loose sight of any data.

## Objective re-declaration

The main objective of the problem is to find the wait time for a driver after dropping a passenger based on the location or borough. For that, we use a concept called sessionization. It means keeping the records of a single driver as a entity. This tool is helpful to find insights about the data and the relationship between the entries. The analysis uses the behavior of the data to find patterns.

## Building sessions

The session in spark is built by using groupBy statement. The data is grouped based on the driver id or driver name. The method works better when there is small amount of records in each entity. The reason why it works for small data is because the records should be in memory for computation. So the alternative approach is secondary sort on a composite key of identifier and time stamp. This approach accepts RDD key value pair that we want to operate, input of value and extraction of secondary key to do sorting, optional splitting function that takes input as sorted value and split same key to multiple groups ie. Records from the same driver, number of partitions in RDD. The secondary key mentioned in the explanation is nothing but the start time of the trip. The innocent approach to make sessions in Spark is to play out a groupBy on the identifier we need to make sessions for and afterward sort the occasions post-rearrange by a timestamp identifier. On the off chance that we just have a little number of occasions for every element, this approach will work sensibly well.

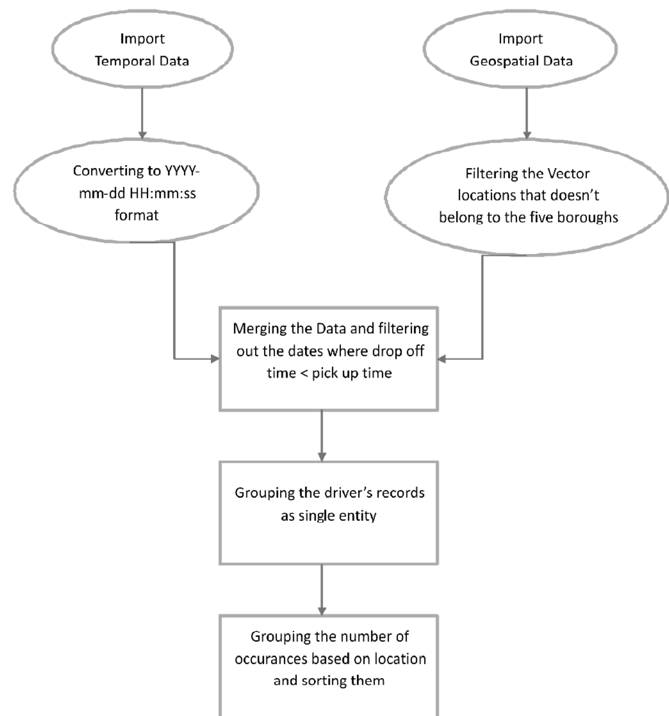



**Figure 11:** Workflow showing the analysis of the NYC taxi data

Since this approach requires every one of the occasions for any specific substance to be in memory in the meantime, it will not scale as the quantity of occasions for every substance gets bigger and bigger. We require a method for building sessions that does not require the majority of the occasions for a specific element to be held in memory at the same time for sorting. In MapReduce, we can construct sessions by playing out an optional sort, where we make a composite key made up of an identifier and a timestamp esteem, sort the greater part of the records on the composite key, and after that utilization a custom practitioner and gathering function to guarantee that the greater part of the records for a similar identifier show up in a similar yield parcel. Luckily, Spark can likewise bolster this same auxiliary sort design by making utilization of its repartition And Sort Within Partitions change. In the repo, we've given a usage of a group By Key And Sort Values arrangement that does precisely this. Since the workings of this usefulness are generally orthogonal to the ideas this section is covering, we are overlooking the violent subtle elements here.

## Analysis

Deciding the threshold for the wait time of the taxi driver based on the location or borough is a bigger challenge because fixing the threshold is important for deriving insights. So splitting the data with different threshold and doing the analysis is always a better option. Running the analysis with threshold as 4 hours and again running the same analysis with the threshold, as 3 hours and comparing would give a clear picture. Creating a pipeline for this operation is an expensive work, so we use session data containing entity and each entity as driver records for all operations. So, we store the session data into Hadoop Distributed File System (HDFS) so that it can be called easily when it is needed. To see the time taken for the driver to find next passenger, we create a function called boroughDuration method. The complete workflow of the analysis is shown in Figure 11.

boroughDuration = drop off time of Nth trip – pick up time of the N + 1 th trip

We find the difference between drop off time and next pick up time. We aggregate the duration of wait time of each drivers based on the location or borough so the output will be average wait time and the location or borough. We have to make sure that the boroughDuration is not negative because the negative values suggest that it is not taking the previous trip drop off time and the pick up time of the new trip. So, after applying the filter the output looks good to analyze. The error is being studied with the help of Spark's StatCounter. But there was no pattern found that would help explain the negative values. The final output is sorted so that the order specifies the lesser number of wait time.

```
import org.apache.spark.util.StatCounter
boroughDurations.filter {
 case (b, d) => d.getMillis >= 0
}.mapValues(d => {
 val s = new StatCounter()
 s.merge(d.getStandardSeconds)
}).
reduceByKey((a, b) => a.merge(b)).collect().foreach(println)
```

**Figure 12:** Code used to produce the output

## RESULTS

The sorted output shows the order in wait time. So, Manhattan has the lowest wait time with an average of 10 min. Staten Island has a worst wait time of 45 min. The results were used to fine drivers who rejected the passengers who wanted to travel to these high wait time borough or locations based on this insight. With this information there are many things that can made sense out of in the taxi business. The output gives a lot of information about the wait time. The first element is the name of the borough the result is being displayed. The second element is the number of records that were analyzed for this borough. The third element is the mean. that is the mean time it takes to get the next customer. The fourth element is the standard deviation. The standard deviation is nothing but the amount of

deviance that can be expected from the analyzed mean. The fifth element is the maximum wait time. The maximum wait time is the value for the highest wait time taken by a taxi cab driver to get a next customer for that borough. The sixth element is the minimum wait time. The minimum wait time is the value for the lowest wait time taken by a taxi cab driver to get a next customer for that borough.

The above code helps you find the duration taken by the taxi driver to get the next customer. The sorted output is shown in the table in Figure 13.

```
(Some(Bronx),(count: 56951, mean: 1945.79,
  stdev: 1617.69, max: 14116, min: 0))
(None,(count: 57685, mean: 1922.10,
  stdev: 1903.77, max: 14280, min: 0))
(Some(Queens),(count: 557826, mean: 2338.25,
  stdev: 2120.98, max: 14378.000000, min: 0))
(Some(Manhattan),(count: 12505455, mean: 622.58,
  stdev: 1022.34, max: 14310, min: 0))
(Some(Brooklyn),(count: 626231, mean: 1348.675465,
  stdev: 1565.119331, max: 14355, min: 0))
(Some(Staten Island),(count: 2612, mean: 2612.24,
  stdev: 2186.29, max: 13740, min: 0.000000))
```

**Figure 13:** Final output in the terminal window

## DISCUSSION

There are so many research projects on the New York City Taxi data, but the general analysis is similar to this workflow but with few minor changes (Chirigati et al., 2016). There is an amazing idea of equal income to drivers. The main objective is to make all the cab drivers to earn the same money by giving the busiest locations to the drivers who has earned less in the previous day. That is implemented by grouping the number of occurrences of rides based on location and sorting. This will give the most profitable location and this was done by (Farber, H. S., 2008). The other interesting concept is labor supply. Depending on the demand, supply the drivers to a specific location or not spending too much resources on a location that doesn't require more than fewer taxi cabs. This was implemented by grouping the the number of occurrences of rides based on location and sorting. But this time using the same data to allocate resources. This project was cited from (Camerer, 1997). The workflow of the Geospatial and Temporal Data Analysis on the New York City Taxi Trip Data also follows similar method but the objective is different (Cibulski et al., 2016). The scope of the project could have been different, like using the data to find the effective working hours for the drivers based on the peak hours and dry hours. The efficient way of shift working can be made with the help of the data (Huang et al., 2016). An examination concentrates on an especially vital urban information set: taxi trips. Taxicabs are profitable sensors and data connected with taxi outings can give remarkable understanding into a wide range of parts of city life, from financial movement and human conduct to versatility designs (He et al., 2016). Be that as it may, investigating these information presents many difficulties. A research project proposed another model that permits clients to outwardly question taxi trips (Xu et al., 2016). Other than standard examination inquiries, the model backings birthplace goal questions that empower the investigation of versatility over the city. They demonstrate that this model can express an extensive variety of spatio-transient inquiries, and it is additionally adaptable in that can questions be made as well as various conglomerations and visual representations can be connected, permitting clients to investigate and think about results (Al-Dohuki et al., 2017). They have assembled a versatile framework that executes this model, which underpins intuitive reaction times, makes utilization of a versatile level-of-detail rendering system to produce mess free representation for vast results, and shows shrouded subtle elements to the clients in a rundown using overlay heat maps (Zeng et al., 2016). The data taken as driver sample can be used to predict the efficient working hours so that the cab drivers works smart and not hard. The other project idea that can be done is the using the time data to find the quarter of the year where the business is good and when the business is bad, so that there can be part time drivers who can work when the period is good and do some other work when the period is dry. Hence, with the help of these kinds of information, the taxi business industry can reach different heights.

## CONCLUSION

The workflow of the Geospatial and Temporal Data Analysis on the New York City Taxi Trip Data also follows similar method to that of research papers but the efficiency of filtering out invalid data and having a seamless flow process makes the difference. The spark is well utilized by applying all the necessary data manipulation and modifying the existing libraries to the project requirement. The advantage of using the Scala package to effective use for the project objective is the highlighting factor. The improvements can be made in the areas like sorting the final output to make better sense out of the result. Using the output to something bigger would been another improvement, For example, using the output to find the best time of the year where the business is at its peak or the period where the drivers found it hard to get customers.

### *References*

[1] Al-Dohuki, S., Wu, Y., Kamw, F., Yang, J., Li, X., Zhao, Y., ... & Wang, F. (2017). SemanticTraj: A New Approach to Interacting with Massive Taxi Trajectories. IEEE Transactions on Visualization and Computer Graphics, 23(1), 11-20.

[2] Camerer, C., Babcock, L., Loewenstein, G., & Thaler, R. (1997). Labor supply of New York City cabdrivers: One day at a time. The Quarterly Journal of Economics, 407-441.

[3] Chirigati, J. F. A. B. F., & Zhao, H. V. K. (2016). Exploring What not to Clean in Urban Data: A Study Using New York City Taxi Trips. Data Engineering, 63.

[4] Cibulski, L., Graèanin, D., Diehl, A., Splechtna, R., Elshehaly, M., Delrieux, C., & Matkoviæ, K. (2016). ITEA—interactive trajectories and events analysis: exploring sequences of spatio-temporal events in movement data. The Visual Computer, 1-11.

[5] Farber, H. S. (2008). Reference-dependent preferences and labor supply: The case of New York City taxi drivers. The American Economic Review,98(3), 1069-1082.

[6] Ferreira, N., Poco, J., Vo, H. T., Freire, J., & Silva, C. T. (2013). Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips. IEEE Transactions on Visualization and Computer Graphics, 19(12), 2149-2158.

[7] Harcourt, B. E., & Ludwig, J. (2006). Broken windows: New evidence from New York City and a five-city social experiment. The University of Chicago Law Review, 271-320.

[8] He, X., Raval, N., & Machanavajjhala, A. (2016). A demonstration of VisDPT: visual exploration of differentially private trajectories. Proceedings of the VLDB Endowment, 9(13), 1489-1492.

[9] Huang, X., Zhao, Y., Ma, C., Yang, J., Ye, X., & Zhang, C. (2016). TrajGraph: A Graph-Based Visual Analytics Approach to Studying Urban Network Centralities Using Taxi Trajectory Data. IEEE transactions on visualization and computer graphics, 22(1), 160-169.

[10] Ryza, S., Laserson, U., Owen, S., & Wills, J. (2015). Advanced Analytics with Spark: Patterns for Learning from Data at Scale. " O'Reilly Media, Inc.".

[11] Splechtna, R., Diehl, A., Elshehaly, M., Delrieux, C., Graèanin, D., & Matkoviæ, K. (2016, September). Bus Lines Explorer: Interactive Exploration of Public Transportation Data. In Proceedings of the 9th International Symposium on Visual Information Communication and Interaction (pp. 30-34). ACM.

[12] Xu, X., Su, B., Zhao, X., Xu, Z., & Sheng, Q. Z. (2016). Effective Traffic Flow Forecasting Using Taxi and Weather Data. In Advanced Data Mining and Applications: 12th International Conference, ADMA 2016, Gold Coast, QLD, Australia, December 12-15, 2016, Proceedings 12 (pp. 507-519). Springer International Publishing.

[13] Zeng, W., Fu, C. W., Arisona, S. M., Schubiger, S., Burkhard, R., & Ma, K. L. (2016, November). A visual analytics design for studying crowd movement rhythms from public transportation data. In SIGGRAPH ASIA 2016 Symposium on Visualization (p. 4). ACM.

View publication stats