

***The Big Bata applications project***  
**Report project: Road network tracking of  
BERRECHID city**

Engineer cycle

Information Systems and Big-Data Engineering (ISBDE)

Department of Mathematics and Computer Science

Realized by :

- AIT OUADIL Kabira
- EL HASSANI Chaimaa
- TIGOUDERN Ayoub
- TERRAF Yassin

Supervised by : Prof.KARIM Lamia

# Table of contents

Introduction .....	2
Project description .....	3
Problematic .....	3
Objectives .....	3
Logo .....	3
Team members .....	3
Technologies used .....	3
Conception .....	5
Scrum : .....	5
Introduction .....	5
Use Case Diagram .....	6
Global Architecture .....	7
Planning and division of tasks .....	7
The first sprint .....	8
The second sprint .....	8
Third sprint .....	9
Technical part .....	9
CROP BERRECHID NETWORK .....	9
Analysis part .....	11
MAKE PREDICTIONS USING SPARK ML .....	13

# Introduction

The road network as a set of interconnected and intersecting roads allowing the passage of people and goods constitutes an important sector in an economy of the city of Berrechid, So our project consists of monitoring in real time the vehicles circulating in the road network of the city of Berrechid, thus the analysis and the detection of the likely congestions and the accidents which can quickly happen, and we will not stop just doing the detection but we will also predict the time when there will be congestion .

Our project is integrated into the big data applications module and aims at Road network tracking of BERRECHID city, we called it "YACK Tracking". It is in order to clarify the idea that we have prepared this report which covers company description, design and technical part.

# Project description

## Problematic

Our problem is to develop an application for monitoring the state of a road network in real time by calculating the flow of a lane to avoid conjections.

## Objectives



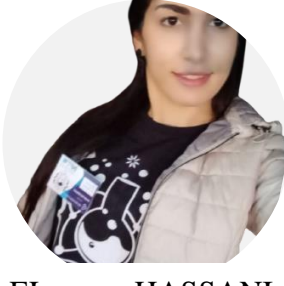

- Detect the coordinates of congestion.
- Detect roads where it is likely to have accidents.
- Real-time monitoring of the state of the road network

## Logo



We chose yack tracking as the name of our company; yack is a combination of the first letters of the names of its founders: Y for Yassine A for Ayoub C for Chaimaa and K for Kabira and Tracking refers to the functionality of our company. Also we made appear a car and a road in our logo which refers to the object of study.

## Team members

 <p>AIT Kabira 21 years old</p>	 <p>TERRAF Yassine 23 years old Manager</p>	 <p>EL HASSANI Chaimaa 23 years old</p>	 <p>TIGOUDERN Ayoub 23 years old</p>
--	--	---	---

SUMO is one of the free open-source traffic simulations tool for building and running sample traffic model systems for T sections and cross intersections. It also provides features to apply different types of entities like vehicles of different sizes ( cars , buses ) & pedestrians in the model.



- Kafka

Kafka is a distributed, publish-subscribe, persistent messaging system for the data it receives, designed to scale easily and support very high data rates.



- MongoDB

**MongoDB** is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents.



### Spark streaming

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streaming can be used to stream live data and processing can happen in real time. Spark Streaming's ever-growing user base consists of household names like Uber, Netflix and Pinterest.

### Spark MLlib : Prediction

#### WHY SPARK ML NOT MAHOUT ?

We chose spark ML for the implementation of ML algorithms because it is scalable, efficient and integrated with spark streaming and spark ML to make real-time prediction on the other hand MAHOUT is based on hadoop and mapreduce it has a low efficiency and limited coverage of a algorithms.



# Conception

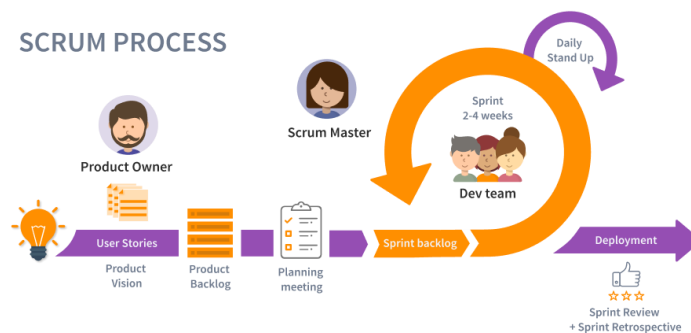
## Scrum :

Scrum is a light weight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems.

The main components of Scrum Framework are:

- The three roles: Scrum Master, Scrum Product Owner and the Scrum Team
- A prioritized Backlog containing the end user requirements
- Sprints
- Scrum Events: Sprint Planning Meeting (WHAT-Meeting, HOW-Meeting), Daily Scrum Meeting, Sprint Review Meeting, Sprint Retrospective Meeting

In our project we worked with the agile scrum method using **JIRA**, we carried out the project in 4 sprints.

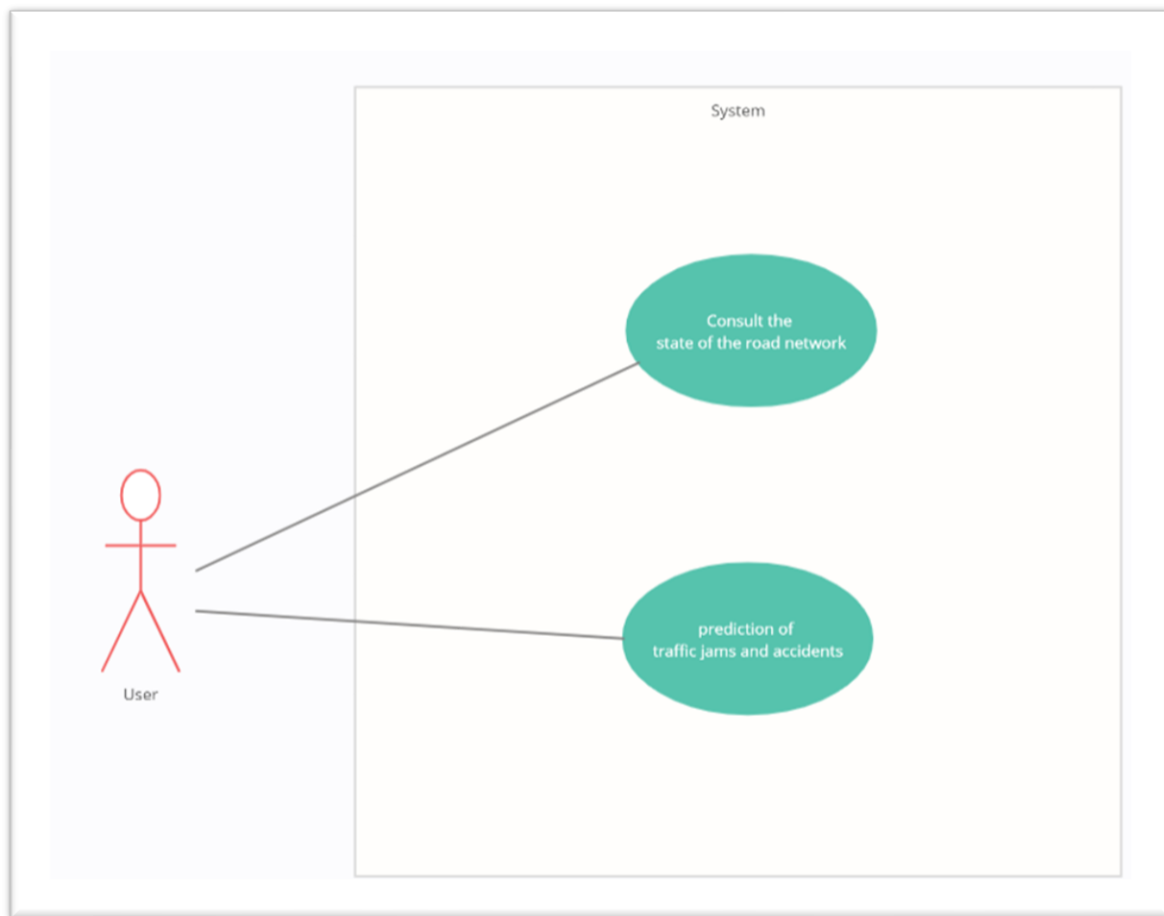


## Introduction

The design stage defines the structures and models to be followed during the implementation phase of the application. This is the phase where we prepare the architecture of the project and where we define the structure of the application and the division of the tasks that we are going to implement.

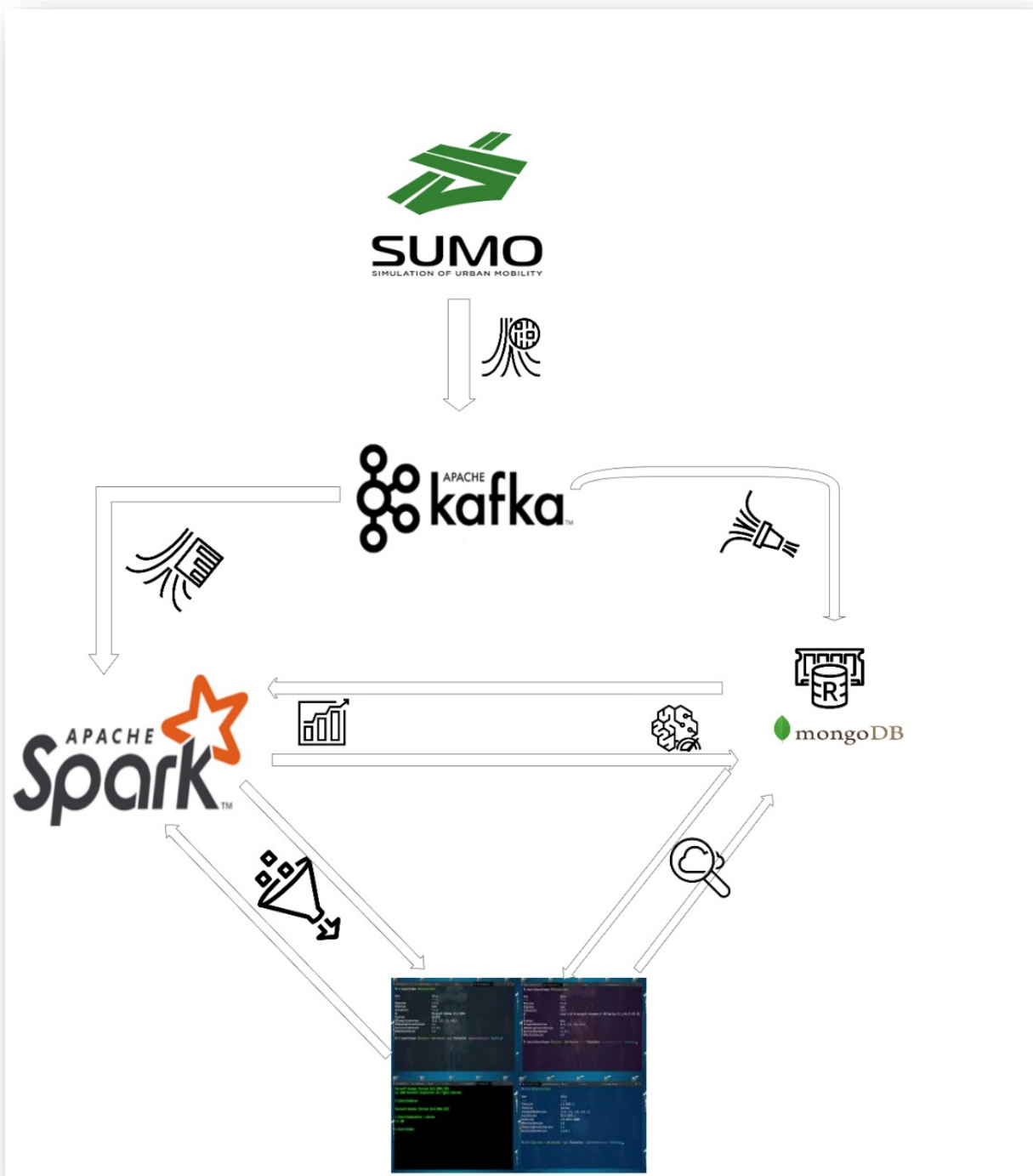
The objective of this project is to set up an application in a Big-data context allowing real-time monitoring of vehicles circulating in the road network of the city of Berrechid.

## Use Case Diagram



The user can consult the analysis and detection of congestion and accidents in real-time, as well as the predilection of the time when there will be congestion.

## Global Architecture

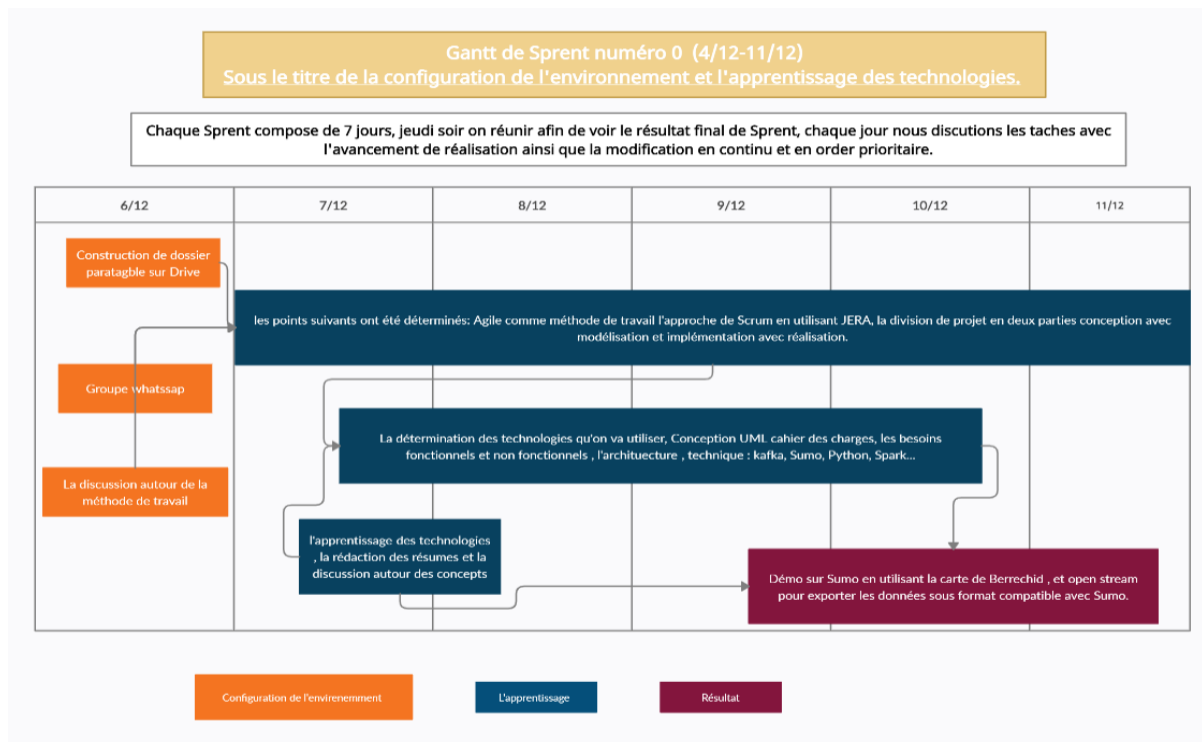


## Planning and division of tasks

We used Gantt charts for planning tasks according to priority order while respecting the hourly volume.

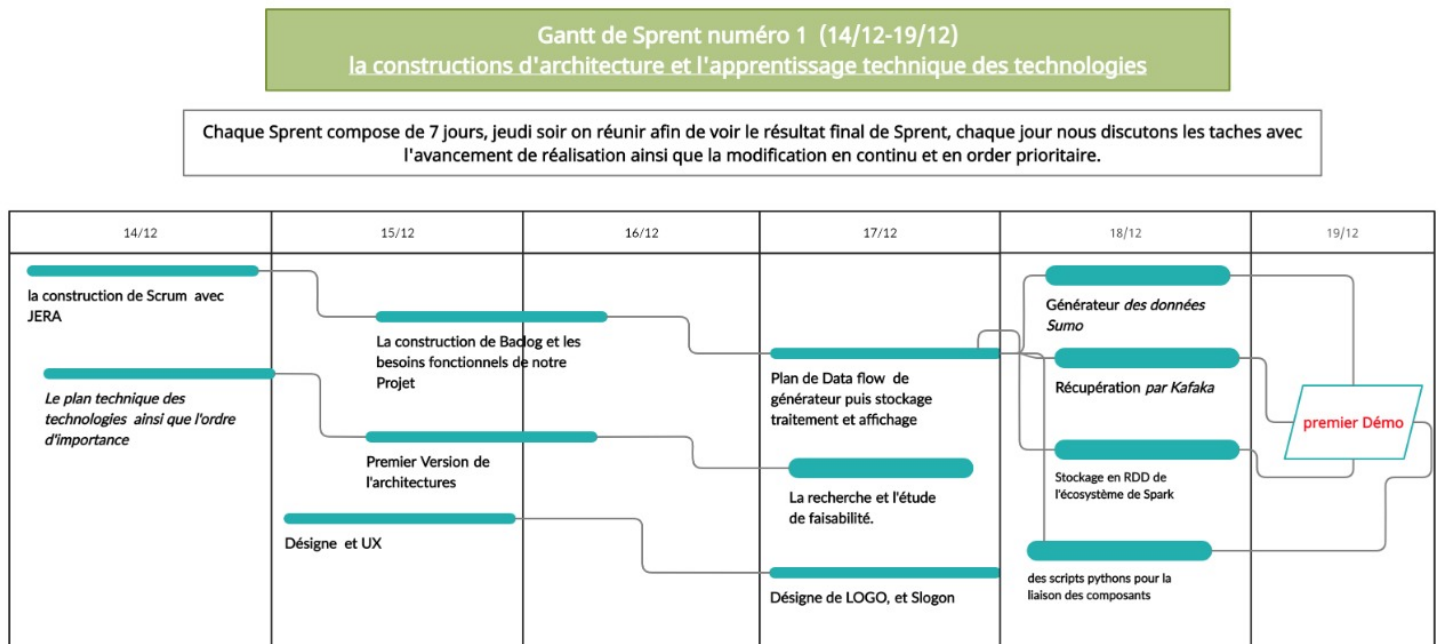


## The first sprint



First Sprint subtitle of the environment setup. In which we carried out the tasks which allowed us to work remotely to respect the constraints of our days.

## The second sprint

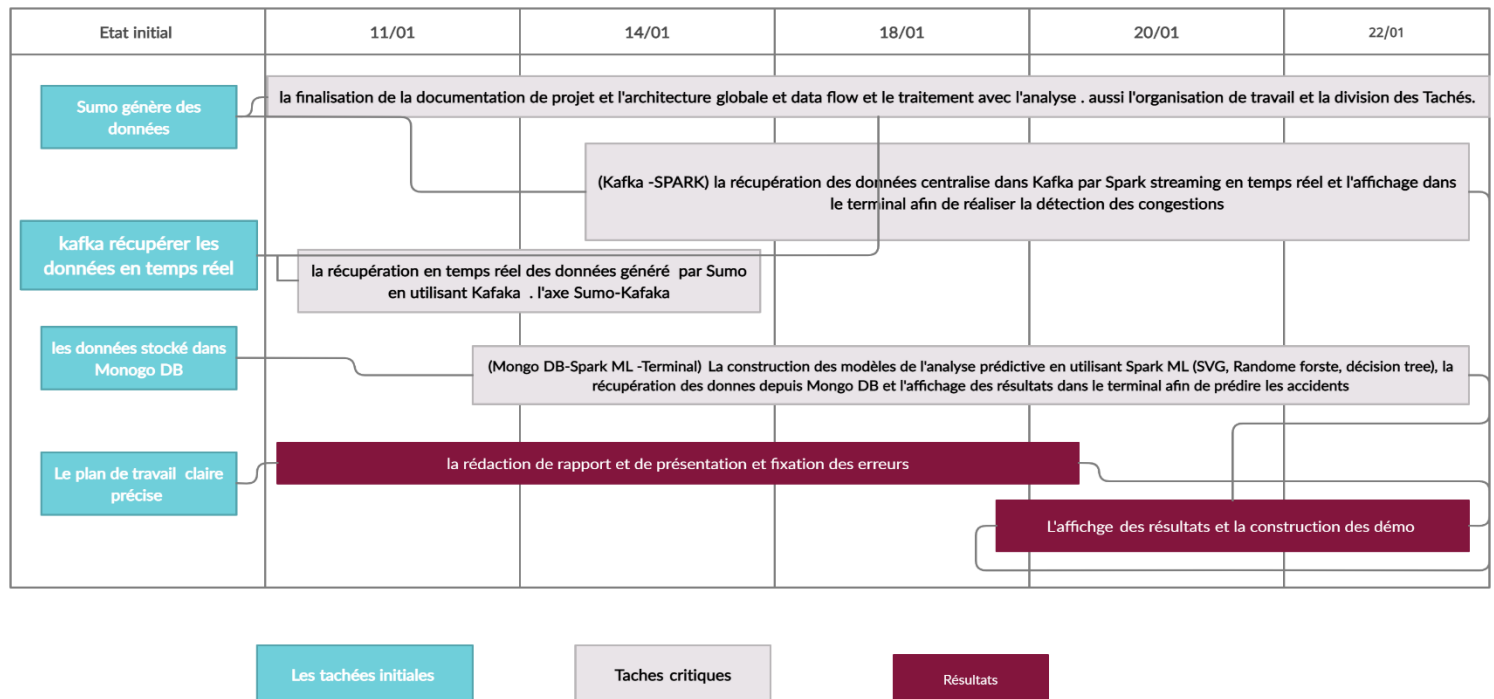


The second sprint subtitle the beginning of the construction of the elements of the axis (Sumo-Kafka) and the implementation of some features.

## Third sprint

Gantt de Sprint numéro 3 (11/01-21/01)  
La dernière Sprint sous titre de l'Analyse et prédiction avec la rédaction de rapport et présentation

Cette Sprint compose de 12 jours pour l'objectif de finaliser le projet et réaliser la partie d'analyse et prédiction et d'utilise Spaark streaming et Spark ML et d'afficher les résultats.



The last sprint we finalized the project with its documentation and we achieved the specifications and the requested objectives.

Conclusion:

This chapter has been devoted to the design of the project and the details of the modeling using UML diagrams and tasks planning.

## Technical part

### CROP BERRECHID NETWORK

First, we used openstreetmap to crop the berrechid network and to be able to extract the map with which we will do the simulation, and since the SUMO simulator takes xml files as input, it is necessary to convert the osm file using the netConvert command to xml files which will be executed on SUMO car tracking simulation software.



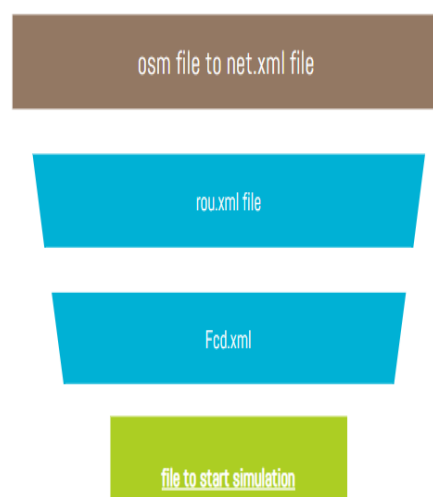
here are all the files that we generated in order to design our first simulation

net.xml: the file which contains all the information on the Berrechid: edges network, traffic lights, junctions.

rou.xml: the file which contains all the roads with which each vehicle passes.

cfg file :contains information about the simulation such as start and end time of simulation

fcd.xml file :is the file which contains for each simulation time all the simulation data: the vehicle, roads, speed, position and many more. so, our key file that will help us to do big data analytics is the fcd.xml file.

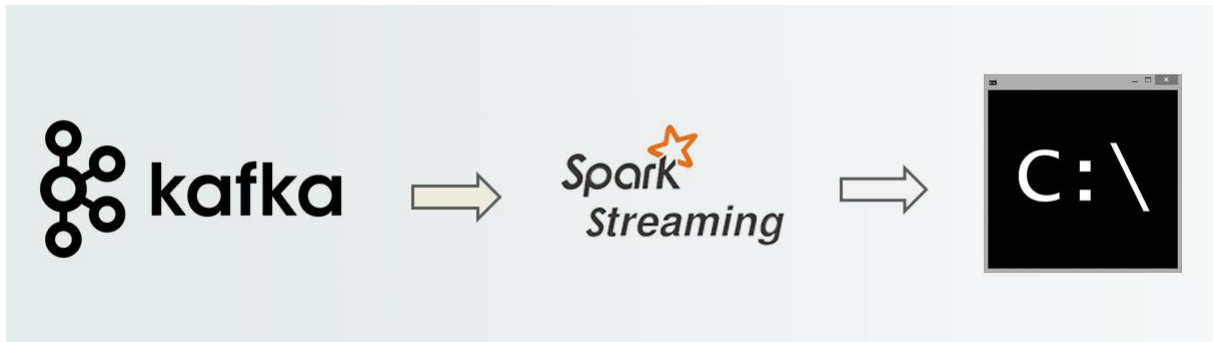


## Analysis part

### Why Spark Streaming?

We can use Spark Streaming to stream real-time data from various sources like Twitter, Stock Market and Geographical Systems and perform powerful analytics to help businesses.

### Processing process



First, we get real-time vehicle data from Kafka to Spark in batch format using script python, and we apply real-time processing.

Here are the list of processing data performed:

- Count number of vehicles per batch
- Detect congestion, thus retrieve the coordinates of each vehicle in congestion.
- Detect accidents.

### Streaming Context

*Streaming Context* consumes a stream of data in Spark. It registers an *Input DStream* to produce a *Receiver* object. It is the main entry point for Spark functionality. Spark provides a number of default implementations of sources like Twitter, AkkaActor and ZeroMQ that are accessible from the context.



```
from pyspark import SparkContext, SparkConf
from pyspark.streaming import StreamingContext
conf = SparkConf().setAppName("KafkaToSpark").setMaster('local')
sc = SparkContext.getOrCreate(conf=conf)
ssc = StreamingContext(sc,20)
```

A Spark Context represents the connection to a Spark cluster

We created Streaming Context from a Spark Context object and get one batch in each 20 seconds.

- ✓ **Count number of vehicles per batch:** using count() transformation

```
#COUNT NUMBER OF ITEMS IN EACH BATCH
dstream.count().map(lambda x:'number of items in this batch: %s' % x).pprint()
```

```
-----
Time: 2021-01-21 17:27:00
-----
number of items in this batch: 10
```

- ✓ **Detect congestion**, this retrieve the coordinates of each vehicle in congestion: by grouping vehicles by time and lane, and comparing this number to the capacity of eachlane, in our case we supposed that medium capacity of all lanes is equal to 8 vehicles.

```
#DETECT CONGESTIONS(8 is CAPACITY MEDIUM LANES)
laneTime_dstream= dstream.map(lambda place: (place['lane'],place['time']))
by_lane_time = laneTime_dstream.countByKey().filter(lambda x : x[1]>8)
by_lane_time.pprint()
```

Here we have congestion in the lane 247021183#1\_1 in second 4 : 10 vehicles > 8

```
-----
Time: 2021-01-21 17:27:00
-----
(('247021183#1_1', '4.00'), 10)
```

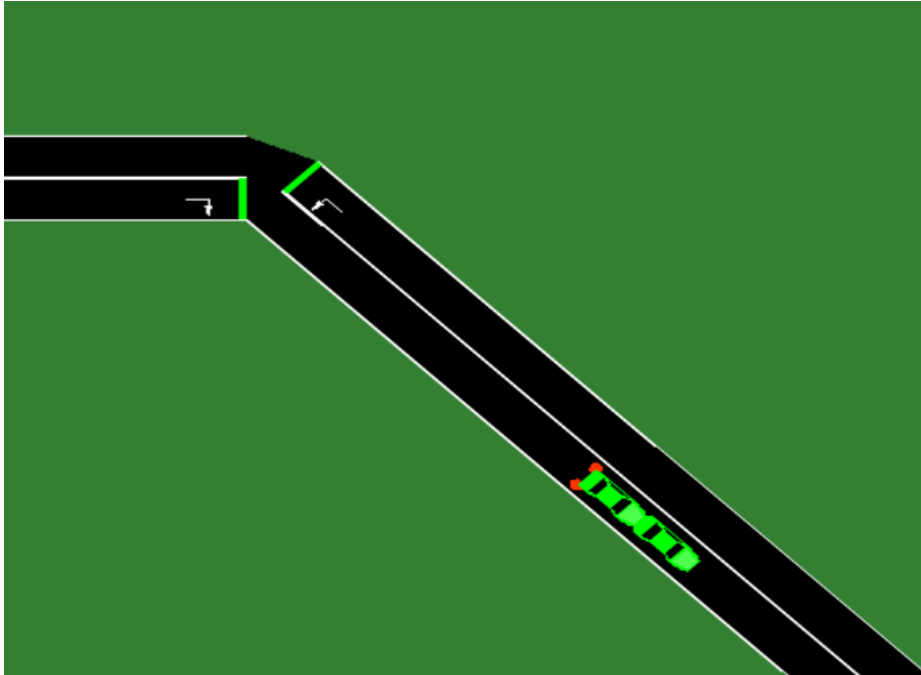
Also we get the coordinates of vehicles in congestion : id,x, and y

```
-----
Time: 2021-01-21 17:27:00
-----
(('247021183#1_1', '4.00'), [(('110', '7589.21', '17277.50'), ('111', '7942.52', '15679.02'), ('112', '7846.86', '15104.38'), ('113', '7671.82', '14686.38'), ('114', '6755.03', '16439.19'), ('115', '6593.46', '15880.89'), ('116', '8032.06', '14225.74'), ('117', '8132.84', '14561.38'), ('118', '6363.41', '16405.12'), ('119', '5434.87', '16157.30')])])
```

- ✓ **Detect accidents** : SUMO traffic simulator is collision free traffic simulator. To simulate accident, cars are forced to make a stop in predefined position. Stops also can be considered important incident in a road segment. a few lanes, and we changed the data in such a way to have an accident.

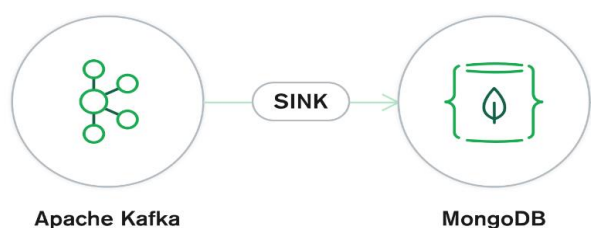
In this capture, we have an accident between two vehicles in the lane gneE2\_0 .

```
-----  
Time: 2021-01-21 20:40:00  
-----  
((('gneE2_0', '0.00', '0'), ({('Vehicle_6', 'bus', '824.54', '473.13'), ('Vehicle_5', 'bus', '828.07', '469.60')}))
```



## MAKE PREDICTIONS USING SPARK ML

For the prediction part and since we need a lot of data to do machine learning algorithms, we stored the data generated by the Kafka consumer in a NOSQL mongoDB database, and we worked with pyspark to run machine learning algorithms by relying on the power of the famous SPARK ML big data library.



In the first step I recovered the data from mongoDB and I converted them into dataframe to facilitate the work

```
>>> #display data
... df.show(truncate=False)
```

_id	angle	id	lane	pos	slope	speed	time	type	x	y
[6003a97e5109bcd379372afe]	341.60	0	246544355#2_0	5.10	0.00	0.00	0.00	DEFAULT_VEHTYPE	6680.11	15695.84
[6003a97e5109bcd379372aff]	341.60	0	246544355#2_0	5	0.00	2.55	1.00	DEFAULT_VEHTYPE	6679.31	15698.26
[6003a97e5109bcd379372b00]	11.32	1	-421239256_0	5.10	0.00	0.00	1.00	DEFAULT_VEHTYPE	7173.99	15765.39
[6003a97e5109bcd379372b01]	341.60	0	246544355#2_0	12.73	0.00	5.08	2.00	DEFAULT_VEHTYPE	6677.70	15703.08
[6003a97e5109bcd379372b02]	11.32	1	-421239256_0	7.02	0.00	1.92	2.00	DEFAULT_VEHTYPE	7174.37	15767.28
[6003a97e5109bcd379372b03]	35.88	2	289430709_0	5.10	0.00	0.00	2.00	DEFAULT_VEHTYPE	8277.37	16491.53
[6003a97e5109bcd379372b04]	339.79	0	:6808302156_4_0	5.99	0.00	7.07	3.00	DEFAULT_VEHTYPE	6675.29	15709.73
[6003a97e5109bcd379372b05]	11.32	1	-421239256_0	10.66	0.00	3.64	3.00	DEFAULT_VEHTYPE	7175.08	15770.85
[6003a97e5109bcd379372b06]	35.88	2	289430709_0	7.62	0.00	2.52	3.00	DEFAULT_VEHTYPE	8278.85	16493.58
[6003a97e5109bcd379372b07]	8.87	3	594754419#0_0	0.20	0.00	0.00	3.00	DEFAULT_VEHTYPE	8986.71	14184.51
[6003a97e5109bcd379372b08]	339.74	0	246544355#3_0	0.14	0.00	8.46	4.00	DEFAULT_VEHTYPE	6672.37	15717.66
[6003a97e5109bcd379372b09]	11.32	1	-421239256_0	16.20	0.00	5.54	4.00	DEFAULT_VEHTYPE	7176.17	15776.29
[6003a97e5109bcd379372b0a]	35.88	2	289430709_0	11.77	0.00	4.15	4.00	DEFAULT_VEHTYPE	8281.28	16496.94
[6003a97e5109bcd379372b0b]	15.45	3	:8090000137_0_0	1.36	0.00	1.36	4.00	DEFAULT_VEHTYPE	8987.31	14185.73
[6003a97e5109bcd379372b0c]	350.47	4	-742341130#0_0	5.10	0.00	0.00	4.00	DEFAULT_VEHTYPE	8202.27	14689.37
[6003a97e5109bcd379372b0d]	336.43	0	:2535725596_5_0	5.75	0.00	10.25	5.00	DEFAULT_VEHTYPE	6668.37	15727.10
[6003a97e5109bcd379372b0e]	358.31	1	:4210333421_4_0	5.16	0.00	7.50	5.00	DEFAULT_VEHTYPE	7176.49	15783.67
[6003a97e5109bcd379372b0f]	35.88	2	289430709_0	17.47	0.00	5.70	5.00	DEFAULT_VEHTYPE	8284.62	16501.56
[6003a97e5109bcd379372b10]	24.98	3	:8090000137_0_0	4.09	0.00	2.73	5.00	DEFAULT_VEHTYPE	8988.43	14188.22
[6003a97e5109bcd379372b11]	350.47	4	-742341130#0_0	7.12	0.00	2.02	5.00	DEFAULT_VEHTYPE	8201.93	14691.36

only showing top 20 rows

Our object is to predict the congestions that can be for any time in the berrechia network, and for this reason we calculated the sum of the vehicles and we have to group them by road, time, position and the coordinates x, y of each vehicle.

For that, we have deployed 5 machine learning algorithms which allow us to do congestion detection and the goal is to compare models accuracies.

- Data preparation:

So first of all we used spark SQL to select time , coordinates x,y of the vehicles lane , position and we have calculated sum of vehicles by alle this variables.

After we have devised data into test and training data and we take as input a list of variables generated by the vector assembler functionand we used a String Indexer as output of the classification.

Is whether yes is a congestion or no is not.

and we used pipeline function to map between input output and dataset used and just after we fit our models

```
>>>
>>> dataCongestion=spark.sql("""SELECT time,x,y,pos,laned,totalVehicules, CASE WHEN totalVehicules > 8 THEN 'Yes' ELSE 'NO' END AS isCongestion FROM CongestionPredic
)
>>>

indexerR = StringIndexer(inputCol="isCongestion", outputCol="label").fit(dataCongestion)
```

- DECISION TREE:

We have started by DecisionTree model we were able to have an accuracy of 100%.

```
>>> # Select (prediction, true label) and compute test error
... evaluatorTree = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
>>> accuracyTree = evaluatorTree.evaluate(predictionsTree)
>>> accuracyTree
1.0
```

```
>>> dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
>>>
>>> # Chain indexers and tree in a Pipeline
... pipeline = Pipeline(stages=[indexerR, assemblerCongestion, dt])
>>>
>>> # Train model. This also runs the indexers.
... modelTree = pipeline.fit(trainingData)
>>>
>>>
>>> # Make predictions.
... predictionsTree = modelTree.transform(testData)
>>>
>>> # Select example rows to display.
... predictionsTree.select("prediction", "label", "features").show(20)
+-----+-----+-----+
|prediction|label|  features|
+-----+-----+-----+
|         0.0|  0.0|[15.0,8.0]|
|         1.0|  1.0|[11.0,11.0]|
|         0.0|  0.0|[12.0,4.0]|
|         0.0|  0.0|[9.0,2.0]|
|         0.0|  0.0|[13.0,0.0]|
|         1.0|  1.0|[16.0,15.0]|
|         0.0|  0.0|[8.0,0.0]|
|         0.0|  0.0|[12.0,0.0]|
|         1.0|  1.0|[10.0,10.0]|
|         0.0|  0.0|[15.0,2.0]|
|         1.0|  1.0|[16.0,12.0]|
|         0.0|  0.0|[8.0,3.0]|
|         0.0|  0.0|[13.0,5.0]|
|         0.0|  0.0|[15.0,1.0]|
|         1.0|  1.0|[16.0,9.0]|
|         0.0|  0.0|[14.0,4.0]|
|         0.0|  0.0|[3.0,3.0]|
|         0.0|  0.0|[13.0,8.0]|
|         1.0|  1.0|[13.0,11.0]|
|         0.0|  0.0|[9.0,0.0]|
+-----+-----+-----+
only showing top 20 rows
```



- RANDOM FOREST CLASSIFIER:

we tested for the randomForest we were able to have the same accuracy .

```
>>> evaluatorForest = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
>>> accuracyForest = evaluatorForest.evaluate(predictionsForest)
>>>
>>> accuracyForest
1.0
```

```
>>> RandomForestModel= RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=10)
>>>
>>>
>>> # Chain indexers and forest in a Pipeline
...
>>> pipelinee = Pipeline(stages=[indexerR ,assemblerCongestion, RandomForestModel])
>>>
>>> # Train model. This also runs the indexers.
... modelForest = pipelinee.fit(trainingData)
>>>
>>> # Make predictions.
... predictionsForest = modelForest.transform(testData)
>>>
>>> predictionsForest.show()
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|time|      x|      y| pos|laned|totalVehicules|isCongestion|label|  features|rawPrediction|probability|prediction|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|15.0|8326.12|16437.65| 5.11| 57|      8|      NO| 0.0|[15.0,8.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
|11.0|8476.76|15766.72| 5.1| 72|     11|     Yes| 1.0|[11.0,11.0]| [0.0,10.0]| [0.0,1.0]|      1.0|
|12.0|8190.57|14759.08| 75.78| 30|      4|      NO| 0.0|[12.0,4.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
| 9.0|8310.29|16536.09| 60.56| 7|      2|      NO| 0.0|[ 9.0,2.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
|13.0|6629.53| 15820.3| 90.77| 36|      0|      NO| 0.0|[13.0,0.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
|16.0|9071.81|14398.32| 6.73| 81|     15|     Yes| 1.0|[16.0,15.0]| [0.0,10.0]| [0.0,1.0]|      1.0|
| 8.0|6652.17|15764.08| 30.15| 36|      0|      NO| 0.0|[ 8.0,0.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
|12.0|6632.52| 15813.8| 83.61| 36|      0|      NO| 0.0|[12.0,0.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
|10.0| 5623.4|14771.31| 5.1| 81|     10|     Yes| 1.0|[10.0,10.0]| [0.0,10.0]| [0.0,1.0]|      1.0|
|15.0|8357.88|16506.09| 4.79| 60|      2|      NO| 0.0|[15.0,2.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
|16.0|6976.46|14738.17| 4.9| 99|     12|     Yes| 1.0|[16.0,12.0]| [0.0,10.0]| [0.0,1.0]|      1.0|
| 9.0| 888.0|14330.43| 15.04| 48|      3|      NO| 0.0|[ 9.0,3.0]| [10.0,0.0]| [1.0,0.0]|      0.0|
```

0  
0.

```
>>> evaluatorSVC = MulticlassClassificationEvaluator(
...     labelCol="label", predictionCol="prediction", metricName="accuracy")
>>>
>>> accuracySVC = evaluatorSVC.evaluate(predictionsSVC)
>>>
>>> accuracySVC
0.9302325581395349
```

```
>>> lsvc = LinearSVC(maxIter=5, regParam=0.1)
>>>
>>> # Fit the model
... lsvcModel = lsvc.fit(trainingData)
>>>
>>>
>>> predictionsSVC = lsvcModel.transform(testData)
>>> predictionsSVC.show()
+-----+-----+-----+-----+-----+
|  features|label|      rawPrediction|prediction|
+-----+-----+-----+-----+-----+
|[14.0,14.0]| 1.0|[-1.1313061619587...|      1.0|
|[13.0,13.0]| 1.0|[-0.9707668839434...|      1.0|
|[14.0,13.0]| 1.0|[-0.8977647268604...|      1.0|
|[15.0,12.0]| 1.0|[-0.5912211346792...|      1.0|
|[12.0,11.0]| 1.0|[-0.5766861708298...|      1.0|
|[15.0,10.0]| 1.0|[-0.1241382644827...|      1.0|
|[13.0,9.0]| 1.0|[-0.0366011435504...|      1.0|
|[15.0,9.0]| 1.0|[0.10940317061548...|      0.0|
|[ 8.0,8.0]| 0.0|[-0.1680704938668...|      1.0|
|[10.0,8.0]| 0.0|[-0.0220661797009...|      1.0|
|[12.0,8.0]| 0.0|[0.12393813446489...|      0.0|
|[16.0,8.0]| 0.0|[0.41594676279668...|      0.0|
```

- NAÏVE BAYES

And for the last algorithm which is the naive bayes we had an accuracy between 0.7 and 0.9

```

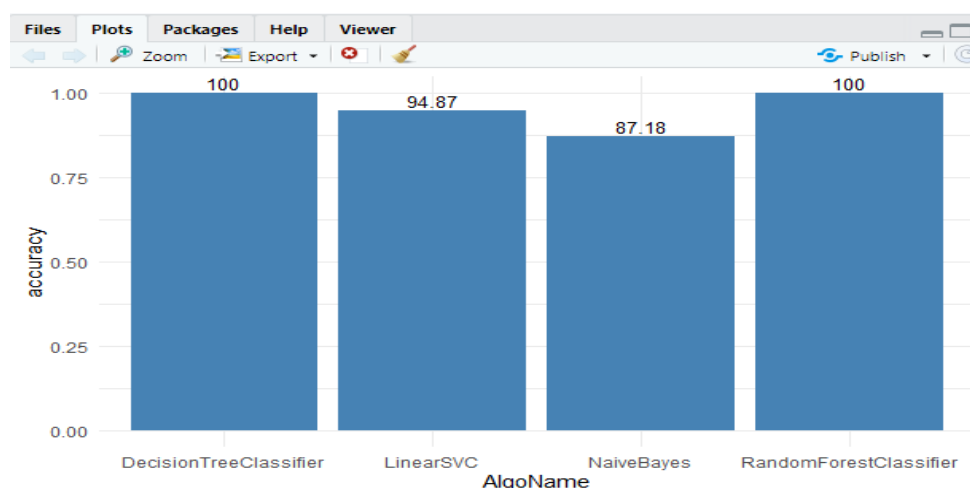
'''
>>> # compute accuracy on the test set
... evaluatorNB = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
...                                                metricName="accuracy")
>>> accuracyNB = evaluatorNB.evaluate(predictionsNB)
>>> print("Test set accuracy = " + str(accuracyNB))
Test set accuracy = 0.8604651162790697

>>> nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
>>>
>>>
>>> modelNB = nb.fit(trainingData)
>>>
>>>
>>> predictionsNB = modelNB.transform(testData)
>>> predictionsNB.show()
+-----+-----+-----+-----+-----+
| features|label| rawPrediction| probability|prediction|
+-----+-----+-----+-----+-----+
|[14.0,14.0]| 1.0| [-23.350884430893...| [0.07769823747997...| 1.0|
|[13.0,13.0]| 1.0| [-21.704368276079...| [0.09777346527360...| 1.0|
|[14.0,13.0]| 1.0| [-22.006376348758...| [0.12802214648072...| 1.0|
|[15.0,12.0]| 1.0| [-20.963876339301...| [0.25741840242147...| 1.0|
|[12.0,11.0]| 1.0| [-18.713344039130...| [0.19546143494755...| 1.0|
|[15.0,10.0]| 1.0| [-18.274860175031...| [0.51287922178753...| 0.0|
|[13.0,9.0]| 1.0| [-16.326335947538...| [0.49992602695497...| 1.0|
|[15.0,9.0]| 1.0| [-16.930352092896...| [0.64725785121133...| 0.0|
|[8.0,8.0]| 0.0| [-13.471787502009...| [0.27626294503068...| 1.0|
|[10.0,8.0]| 0.0| [-14.075803647367...| [0.41198349640034...| 1.0|
|[12.0,8.0]| 0.0| [-14.679819792724...| [0.56255343992290...| 0.0|
|[16.0,8.0]| 0.0| [-15.887852083439...| [0.81246970154157...| 0.0|
|[9.0,7.0]| 0.0| [-12.429287492553...| [0.47403730987901...| 1.0|
|[10.0,7.0]| 0.0| [-12.731295565232...| [0.54976157446403...| 0.0|
|[12.0,7.0]| 0.0| [-13.335311710589...| [0.69147213998854...| 0.0|
|[6.0,6.0]| 0.0| [-10.178755192382...| [0.38712340628465...| 1.0|
|[9.0,6.0]| 0.0| [-11.084779410418...| [0.61100410395931...| 0.0|
|[13.0,6.0]| 0.0| [-12.292811701133...| [0.84106070433234...| 0.0|
|[7.0,5.0]| 0.0| [-9.1362551829255...| [0.59861815183766...| 0.0|
|[14.0,5.0]| 0.0| [-11.250311691676...| [0.92589461920710...| 0.0|
+-----+-----+-----+-----+-----+
only showing top 20 rows
>>>

```

- 1.1. Save models accuracies on a collection mongo DB and compare algorithms accuracies using R script

And as a last action i save the accuracy of each model in a mongoDB collection and we used this collection to make an R language plot that shows the precision of each model.



## Conclusion

this project carried out as part of the Big-data Analytics module project, was for us not only a step to validate the module of our studies, but also an opportunity to take part in an innovative project in the field of big-data and AI application. Our mission was to design and implement a solution capable of real-time monitoring of vehicles circulating in the road network of Berrechid city. Despite the difficulties we encountered during this project, which reside mainly in the novelty of the tools with which we have worked, we finally got to know these new technologies, but unfortunately, the time constraint did not give us the possibility to finalize the project, and that some of its work remains in progress. This project was an opportunity for us to consolidate and implement all of the the knowledge that we have acquired in the Big-data context and to discover the issues and problems relating to the implementation of projects.