os.environ['PYSPARK\_SUBMIT\_ARGS'] = '--packages org.apache.spark:spark-streaming-kafka-0-8\_2.11:2.0.2 pyspark-shell'

amount\_tsh - Total static head (amount water available to waterpoint)

date\_recorded - The date the row was entered

funder - Who funded the well

gps\_height - Altitude of the well

installer - Organization that installed the well

longitude - GPS coordinate

latitude - GPS coordinate

wpt\_name - Name of the waterpoint if there is one

num\_private -

basin - Geographic water basin

subvillage - Geographic location

region - Geographic location

region\_code - Geographic location (coded)

district\_code - Geographic location (coded)

lga - Geographic location

ward - Geographic location

population - Population around the well

public\_meeting - True/False

recorded\_by - Group entering this row of data

scheme\_management - Who operates the waterpoint

scheme\_name - Who operates the waterpoint

permit - If the waterpoint is permitted

construction\_year - Year the waterpoint was constructed

extraction\_type - The kind of extraction the waterpoint uses

extraction\_type\_group - The kind of extraction the waterpoint uses

extraction\_type\_class - The kind of extraction the waterpoint uses

management - How the waterpoint is managed

management\_group - How the waterpoint is managed

payment - What the water costs

payment\_type - What the water costs

water\_quality - The quality of the water

quality\_group - The quality of the water

quantity - The quantity of water

quantity\_group - The quantity of water

source - The source of the water

source\_type - The source of the water

source\_class - The source of the water

waterpoint\_type - The kind of waterpoint

waterpoint\_type\_group - The kind of waterpoint

TP1: <https://www.dropbox.com/s/32u49zlg8hm5qvq/tp1.txt?dl=0>

PDF: <https://www.dropbox.com/s/vvo0i3fz222xkcb/Spark_ESTIAM.pdf?dl=0>

PDF HIVE HADDOP:<https://www.dropbox.com/s/5mnsx04q7rn9kxc/Cours%20Estiam%20-%20Big%20Data%20avec%20Hive.pdf?dl=0>

SYLVAIN SCRIPT:<https://www.dropbox.com/s/2t5pkhcdvcnsjcv/tp1.md?dl=0>

PYTHON SPARK HIVE: <https://www.dropbox.com/s/nb8tst9esf42j4w/Spark%20%26%20Hive.ipynb?dl=0>

PYTHON SPARK STREAMING: <https://www.dropbox.com/s/dto64rjy2vuohts/Spark%20Streaming.ipynb?dl=0>

SCALA SCRIPT SPARK STREAMING: <https://www.dropbox.com/s/h0d34oldmcqy6sg/scalaSparkStreamingExample.txt?dl=0>

NOTEBOOK HIVE :

 NOTEBOOK STREAMING :

IMPORT PYTHON vvvv

import sys,os

os.environ["SPARK\_HOME"] = "/usr/hdp/current/spark2-client"

os.environ["\_PYSPARK\_DRIVER\_CALLBACK\_HOST"] = "127.0.0.1"

os.environ["\_PYSPARK\_DRIVER\_CALLBACK\_PORT"] = "8081"

os.environ["PYSPARK\_SUBMIT\_ARGS"] = "--master yarn pyspark-shell"

os.environ["PYSPARK\_PYTHON"] = "python3"

os.environ["PYSPARK\_DRIVER\_PYTHON"] = "jupyter"

sys.path.insert(0,os.path.join(os.environ['SPARK\_HOME'],"python"))

sys.path.insert(0,os.path.join(os.environ['SPARK\_HOME'],"python","lib","py4j-0.10.6-src.zip"))

from pyspark.sql import SparkSession

from pyspark import SparkContext

spark = SparkSession.builder \

.master("yarn") \

.appName("NOMDEVOTREQUEUE") \

.config("spark.yarn.queue", "NOMDEVOTREQUEUE") \

.getOrCreate()

from pyspark.sql.types import IntegerType

mylist = [1, 2, 3, 4]

df = spark.createDataFrame(mylist, IntegerType()).show()

IMPORT SCALA vvvvvv

## SPylon Kernel

%%init\_spark

launcher.\_spark\_home = "/usr/hdp/current/spark2-client"

launcher.num\_executors = 2

launcher.executor\_cores = 2

launcher.driver\_memory = '600m'

launcher.master = "yarn-client"

launcher.conf.spark.app.name = "NOMDEVOTREQUEUE"

launcher.queue = "NOMDEVOTREQUEUE"

/lab/datasets/chicago/chicago-taxi-rides-2016/chicago\_taxi\_trips\_2016\_\*\*.csv

/lab/datasets/wellstraining-set.csv-labels

chicago/chicago-taxi-rides-2016/chicago\_taxi\_trips\_2016\_\*\*.csv

/lab/datasets/chicago/chicago-taxi-rides-2016/data\_dictionary.csv

csv\_full = spark.read.option("inferSchema","true").option("header","true").csv("/lab/datasets/chicago/chicago-taxi-rides-2016/chicago\_taxi\_trips\_2016\_\*\*.csv")

csv\_ok = csv\_full.drop("")

LINK JSON:

json = spark.read.json("/lab/datasets/chicago/chicago-taxi-rides-2016/column\_remapping.json")

DOCUMENTATION SPARK SQL : <https://spark.apache.org/docs/latest/sql-programming-guide.html#untyped-dataset-operations-aka-dataframe-operations>

CONF HIVE

import sys,os

os.environ["SPARK\_HOME"] = "/usr/hdp/current/spark2-client"

os.environ["\_PYSPARK\_DRIVER\_CALLBACK\_HOST"] = "127.0.0.1"

os.environ["\_PYSPARK\_DRIVER\_CALLBACK\_PORT"] = "8081"

os.environ["PYSPARK\_SUBMIT\_ARGS"] = "--master yarn pyspark-shell"

os.environ["PYSPARK\_PYTHON"] = "python3"

os.environ["PYSPARK\_DRIVER\_PYTHON"] = "jupyter"

sys.path.insert(0,os.path.join(os.environ['SPARK\_HOME'],"python"))

sys.path.insert(0,os.path.join(os.environ['SPARK\_HOME'],"python","lib","py4j-0.10.6-src.zip"))

from pyspark.sql import SparkSession

from pyspark import SparkContext

from os.path import expanduser, join, abspath

# warehouse\_location points to the default location for managed databases and tables

spark2 = SparkSession \

    .builder \

    .config("spark.yarn.queue", "ModelFitting") \

    .appName("Python Spark SQL Hive integration example") \

    .config("hive.metastore.warehouse.dir", "hdfs://node2:9083/apps/hive/warehouse") \

    .config("spark.sql.warehouse.dir", "hdfs://node2:9083/apps/hive/warehouse") \

    .config("javax.jdo.option.ConnectionURL", "jdbc:mysql://node2/hive?createDatabaseIfNotExist=true") \

    .config("javax.jdo.option.ConnectionDriverName", "com.mysql.jdbc.Driver") \

    .enableHiveSupport() \

    .getOrCreate()

KAFKA

        curl -O <http://apache.mirrors.ovh.net/ftp.apache.org/dist/kafka/1.1.0/kafka_2.11-1.1.0.tgz>

        tar xvf kafka\_2.11-1.1.0.tgz

        cd kafka\_2.11-1.1.0

        ./bin/kafka-console-consumer.sh --bootstrap-server 213.251.184.52:9092 --topic test

        .\bin\windows\kafka-console-consumer.bat --bootstrap-server 213.251.184.52:9092 --topic test

##### CORRIGEEEEEE dans .ipynb

* {
* "cells": [
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "# TP 1 : Traitement de données avec Spark SQL\n",
* "## I) Initialisation de la session spark"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 1,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "Intitializing Scala interpreter ..."
* ]
* },
* "metadata": {},
* "output\_type": "display\_data"
* },
* {
* "data": {
* "text/plain": [
* "Spark Web UI available at [http://192.27.137.59:4040\n](http://192.27.137.59:4040/n)",
* "SparkContext available as 'sc' (version = 2.2.0, master = local[\*], app id = local-1511344415843)\n",
* "SparkSession available as 'spark'\n"
* ]
* },
* "metadata": {},
* "output\_type": "display\_data"
* },
* {
* "data": {
* "text/plain": [
* "res0: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@626887e2\n"
* ]
* },
* "execution\_count": 1,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "spark"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "## II) Import des données"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 2,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "path: String = C:\\Users\\X170058\\Documents\\Spark\\chicago-taxi-rides-2016\\chicago\_taxi\_trips\_2016\_01.csv\n"
* ]
* },
* "execution\_count": 2,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "//val path = \"C:\\\\Users\\\\X170058\\\\Documents\\\\Spark\\\\chicago-taxi-rides-2016\\\\\*\"\n",
* "val path = \"C:\\\\Users\\\\X170058\\\\Documents\\\\Spark\\\\chicago-taxi-rides-2016\\\\chicago\_taxi\_trips\_2016\_01.csv\""
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 3,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "df\_raw: org.apache.spark.sql.DataFrame = [taxi\_id: int, trip\_start\_timestamp: timestamp ... 18 more fields]\n"
* ]
* },
* "execution\_count": 3,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val df\_raw = spark.read\n",
* "    .format(\"com.databricks.spark.csv\")\n",
* "    .option(\"header\", \"true\") // Use first line of all files as header\n",
* "    .option(\"inferSchema\", \"true\") // Automatically infer data types\n",
* "    .load(path)"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Comparaison des schémas et checking des nulls\n",
* "Ici on utilise la fonction \*\*printSchema\*\* pour vérifier si l'option inferSchema a bien fonctionné, ici on remarque une différence sur la variable \*pickup\_census\_track\*. On regarde alors cette variable et on remarque qu'elle est totalement \*\*null\*\*"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 4,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "root\n",
* " |-- taxi\_id: integer (nullable = true)\n",
* " |-- trip\_start\_timestamp: timestamp (nullable = true)\n",
* " |-- trip\_end\_timestamp: timestamp (nullable = true)\n",
* " |-- trip\_seconds: integer (nullable = true)\n",
* " |-- trip\_miles: double (nullable = true)\n",
* " |-- pickup\_census\_tract: string (nullable = true)\n",
* " |-- dropoff\_census\_tract: integer (nullable = true)\n",
* " |-- pickup\_community\_area: integer (nullable = true)\n",
* " |-- dropoff\_community\_area: integer (nullable = true)\n",
* " |-- fare: double (nullable = true)\n",
* " |-- tips: double (nullable = true)\n",
* " |-- tolls: double (nullable = true)\n",
* " |-- extras: double (nullable = true)\n",
* " |-- trip\_total: double (nullable = true)\n",
* " |-- payment\_type: string (nullable = true)\n",
* " |-- company: integer (nullable = true)\n",
* " |-- pickup\_latitude: integer (nullable = true)\n",
* " |-- pickup\_longitude: integer (nullable = true)\n",
* " |-- dropoff\_latitude: integer (nullable = true)\n",
* " |-- dropoff\_longitude: integer (nullable = true)\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\_raw.printSchema()"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 5,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+-------------------+-------+\n",
* "|pickup\_census\_tract|  count|\n",
* "+-------------------+-------+\n",
* "|               null|1705805|\n",
* "+-------------------+-------+\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\_raw\n",
* ".groupBy(\"pickup\_census\_tract\")\n",
* ".count()\n",
* ".show()"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 6,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "df\_drop: org.apache.spark.sql.DataFrame = [taxi\_id: int, trip\_start\_timestamp: timestamp ... 17 more fields]\n"
* ]
* },
* "execution\_count": 6,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val df\_drop = df\_raw\n",
* ".drop(\"pickup\_census\_tract\")"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Traitement des null values"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 7,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "df: org.apache.spark.sql.DataFrame = [taxi\_id: int, trip\_start\_timestamp: timestamp ... 17 more fields]\n"
* ]
* },
* "execution\_count": 7,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val df = df\_drop.na.drop()"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 8,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "res3: df.type = [taxi\_id: int, trip\_start\_timestamp: timestamp ... 17 more fields]\n"
* ]
* },
* "execution\_count": 8,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "df.cache()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "## III) Analyse SQL des données"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Nombre de taxi fares sur l'année"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 9,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "fares: Long = 615075\n"
* ]
* },
* "execution\_count": 9,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val fares = df.count()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Nombre de taxi fares avec des pourboires supérieurs à 10\\$"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 10,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "res3: Long = 8497\n"
* ]
* },
* "execution\_count": 10,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "df\n",
* ".filter(col(\"tips\") >= 10)\n",
* ".count()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Pourcentage de taxi fares sans pourboires"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 11,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "noTips: Long = 350679\r\n",
* "pourcent: Float = 57.014023\n"
* ]
* },
* "execution\_count": 11,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val noTips = df\n",
* ".filter(col(\"tips\")  === 0)\n",
* ".count()\n",
* "\n",
* "val pourcent = noTips.toFloat/fares\*100"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Ranking décroissant des modes de paiement"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 12,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+------------+------+\n",
* "|payment\_type| count|\n",
* "+------------+------+\n",
* "|        Cash|338751|\n",
* "| Credit Card|271547|\n",
* "|   No Charge|  3137|\n",
* "|     Unknown|  1125|\n",
* "|     Dispute|   515|\n",
* "+------------+------+\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\n",
* ".groupBy(\"payment\_type\")\n",
* ".agg(count(\"payment\_type\") as \"count\")\n",
* ".orderBy(desc(\"count\"))\n",
* ".show()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Moyen de paiement le plus utilisé"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 13,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "top\_payment: org.apache.spark.sql.Row = [Cash,338751]\r\n",
* "res5: String = Cash\n"
* ]
* },
* "execution\_count": 13,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val top\_payment = df\n",
* ".groupBy(\"payment\_type\")\n",
* ".agg(count(\"payment\_type\") as \"count\")\n",
* ".orderBy(desc(\"count\"))\n",
* ".first()\n",
* "\n",
* "top\_payment.getString(0)"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 14,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+------------+------+\n",
* "|payment\_type| count|\n",
* "+------------+------+\n",
* "|        Cash|338751|\n",
* "+------------+------+\n",
* "\n"
* ]
* },
* {
* "data": {
* "text/plain": [
* "count\_df: org.apache.spark.sql.DataFrame = [payment\_type: string, count: bigint]\r\n",
* "dfMax: org.apache.spark.sql.DataFrame = [max\_count: bigint]\r\n",
* "dfTopByJoin: org.apache.spark.sql.DataFrame = [payment\_type: string, count: bigint]\n"
* ]
* },
* "execution\_count": 14,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val count\_df = df\n",
* ".groupBy(\"payment\_type\")\n",
* ".agg(count(\"payment\_type\") as \"count\")\n",
* "\n",
* "val dfMax = count\_df.agg(max($\"count\") as \"max\_count\")\n",
* "\n",
* "val dfTopByJoin = count\_df\n",
* ".join(broadcast(dfMax),col(\"count\") === col(\"max\_count\"))\n",
* ".drop(\"max\_count\")\n",
* "\n",
* "dfTopByJoin.show"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 15,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+------------+------+\n",
* "|payment\_type| count|\n",
* "+------------+------+\n",
* "|        Cash|338751|\n",
* "+------------+------+\n",
* "\n"
* ]
* },
* {
* "data": {
* "text/plain": [
* "import org.apache.spark.sql.expressions.Window\r\n",
* "count\_df: org.apache.spark.sql.DataFrame = [payment\_type: string, count: bigint]\r\n",
* "w: org.apache.spark.sql.expressions.WindowSpec = org.apache.spark.sql.expressions.WindowSpec@1c3fef53\r\n",
* "dfTop: org.apache.spark.sql.DataFrame = [payment\_type: string, count: bigint]\n"
* ]
* },
* "execution\_count": 15,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "import org.apache.spark.sql.expressions.Window\n",
* "val count\_df = df\n",
* ".groupBy(\"payment\_type\")\n",
* ".agg(count(\"payment\_type\") as \"count\")\n",
* "\n",
* "val w = Window.orderBy($\"count\".desc)\n",
* "\n",
* "val dfTop = count\_df.withColumn(\"rn\", row\_number.over(w)).where($\"rn\" === 1).drop(\"rn\")\n",
* "\n",
* "dfTop.show"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Taxi le plus productif"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 16,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "count\_taxis: org.apache.spark.sql.Row = [1769,936]\r\n",
* "id\_max: Int = 1769\r\n",
* "nb\_fares: Long = 936\n"
* ]
* },
* "execution\_count": 16,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val count\_taxis = df\n",
* ".groupBy(\"taxi\_id\")\n",
* ".agg(count(\"taxi\_id\") as \"count\")\n",
* ".orderBy(desc(\"count\"))\n",
* ".first()\n",
* "\n",
* "val id\_max = count\_taxis.getInt(0)\n",
* "val nb\_fares = count\_taxis.getLong(1)"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 18,
* "metadata": {},
* "outputs": [],
* "source": [
* "println(\"Le taxi \" + count\_taxis.getInt(0) + \" a effectué \" + count\_taxis.getLong(1) + \" courses.\")"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Bonus :"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 19,
* "metadata": {},
* "outputs": [],
* "source": [
* "df\n",
* "//.select(\"taxi\_id\",\"trip\_seconds\",\"trip\_miles\")\n",
* ".describe()\n",
* "//.filter(col(\"summary\") === \"count\")\n",
* ".show()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "## IV) Utiliser les Hive UDF Functions sous Spark et en définir de nouvelles\n",
* "<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF>"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Calcul de la durée à partir des timestamps\n"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 55,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+-----+------+\n",
* "|duree| count|\n",
* "+-----+------+\n",
* "|  900|287234|\n",
* "|    0|269308|\n",
* "| 1800| 42431|\n",
* "| 2700| 12131|\n",
* "| 3600|  2923|\n",
* "| 4500|   669|\n",
* "| 5400|   202|\n",
* "| 6300|    63|\n",
* "| 7200|    25|\n",
* "| 8100|    21|\n",
* "| 9000|    17|\n",
* "| 9900|    13|\n",
* "|10800|     7|\n",
* "|11700|     7|\n",
* "|13500|     5|\n",
* "|12600|     5|\n",
* "|15300|     3|\n",
* "|14400|     3|\n",
* "|18900|     2|\n",
* "|28800|     1|\n",
* "+-----+------+\n",
* "only showing top 20 rows\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\n",
* ".withColumn(\"duree\",unix\_timestamp(col(\"trip\_end\_timestamp\"))-unix\_timestamp(col(\"trip\_start\_timestamp\")))\n",
* ".groupBy(\"duree\")\n",
* ".count()\n",
* ".orderBy(desc(\"count\"))\n",
* ".show"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 56,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+------------+-----+\n",
* "|trip\_seconds|count|\n",
* "+------------+-----+\n",
* "|         360|58256|\n",
* "|         300|57279|\n",
* "|         420|55198|\n",
* "|         240|49608|\n",
* "|         480|48667|\n",
* "|         540|41714|\n",
* "|         180|34641|\n",
* "|         600|34625|\n",
* "|         660|28306|\n",
* "|         720|22926|\n",
* "|         780|18222|\n",
* "|         120|17193|\n",
* "|           0|15350|\n",
* "|         840|14483|\n",
* "|         900|11280|\n",
* "|         960| 9047|\n",
* "|          60| 7821|\n",
* "|        1020| 7131|\n",
* "|        1080| 6119|\n",
* "|        1140| 5422|\n",
* "+------------+-----+\n",
* "only showing top 20 rows\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df.groupBy(\"trip\_seconds\")\n",
* ".count()\n",
* ".orderBy(desc(\"count\"))\n",
* ".show"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Calcul de la vitesse moyenne en km/h"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 20,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "speed: (distance: Double, time: Double)Double\n"
* ]
* },
* "execution\_count": 20,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "def speed(distance:Double,time:Double) = {\n",
* "    val kms = distance\*1.60934\n",
* "    val hs = time/3600\n",
* "    kms/hs\n",
* "}"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 17,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "speedUDF: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function2>,DoubleType,Some(List(DoubleType, DoubleType)))\n"
* ]
* },
* "execution\_count": 17,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val speedUDF =spark.udf.register(\"speedUDF\", (distance: Double, time: Double) => distance\*1.60934\*3600/time)"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 18,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "df\_speed: org.apache.spark.sql.DataFrame = [taxi\_id: int, trip\_start\_timestamp: timestamp ... 18 more fields]\n"
* ]
* },
* "execution\_count": 18,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val df\_speed = df.withColumn(\"speed\",speedUDF(col(\"trip\_miles\"),col(\"trip\_seconds\")))"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": null,
* "metadata": {},
* "outputs": [],
* "source": [
* "df\_speed.show"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Taxi fare la plus rapide"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 27,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+-----------+\n",
* "| max(speed)|\n",
* "+-----------+\n",
* "|82211.52456|\n",
* "+-----------+\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\_speed\n",
* ".filter(col(\"trip\_seconds\") !== 0)\n",
* ".agg(max(\"speed\"))\n",
* ".show()"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 25,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+------------+----------+------------------+\n",
* "|trip\_seconds|trip\_miles|             speed|\n",
* "+------------+----------+------------------+\n",
* "|          60|     851.4|       82211.52456|\n",
* "|          60|     170.3|       16444.23612|\n",
* "|         600|     504.0|        4866.64416|\n",
* "|         420|     324.1|        4470.74652|\n",
* "|         540|     246.7|2646.8278533333337|\n",
* "|         180|      74.3|        2391.47924|\n",
* "|         720|     254.6|        2048.68982|\n",
* "|         180|      63.0|         2027.7684|\n",
* "|         600|     201.6|       1946.657664|\n",
* "|         360|     115.0|          1850.741|\n",
* "|          60|      18.9|1824.9915599999997|\n",
* "|         240|      74.7|1803.2654699999998|\n",
* "|          60|      18.4|1776.7113599999998|\n",
* "|          60|      17.3|        1670.49492|\n",
* "|          60|      17.2|1660.8388799999998|\n",
* "|         540|     150.1|1610.4128933333332|\n",
* "|        1080|     294.8|1581.4447733333334|\n",
* "|         240|      64.8|        1564.27848|\n",
* "|          60|      15.6|1506.3422400000002|\n",
* "|          60|      15.0|          1448.406|\n",
* "+------------+----------+------------------+\n",
* "only showing top 20 rows\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\_speed\n",
* ".filter((col(\"trip\_seconds\") !== 0) && (col(\"trip\_miles\") !== 0))\n",
* ".select(\"trip\_seconds\",\"trip\_miles\",\"speed\")\n",
* ".orderBy(desc(\"speed\")).show()"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": null,
* "metadata": {},
* "outputs": [],
* "source": [
* "df\_speed.filter(col(\"trip\_seconds\") === 0).show()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "## V) Travailler avec des timestamps sous Spark"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": null,
* "metadata": {},
* "outputs": [],
* "source": []
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "## VI) Enrichir avec des données JSON externes"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Import d'un fichier JSON"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 9,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "df\_json: org.apache.spark.sql.DataFrame = [company: struct<0: string, 1: string ... 118 more fields>, dropoff\_latitude: struct<0: string, 1: string ... 784 more fields> ... 3 more fields]\n"
* ]
* },
* "execution\_count": 9,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val df\_json = spark.read.json(\"C:/Users/X170058/Documents/Spark/chicago-taxi-rides-2016/column\_remapping.json\")\n",
* ".drop(\"dropoff\_census\_tract\",\"taxi\_id\",\"pickup\_census\_tract\")"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": null,
* "metadata": {
* "scrolled": true
* },
* "outputs": [],
* "source": [
* "df\_json.printSchema"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 17,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "res8: Array[String] = Array(company, dropoff\_latitude, dropoff\_longitude, pickup\_latitude, pickup\_longitude)\n"
* ]
* },
* "execution\_count": 17,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "df\_json.columns"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 64,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "res27: org.apache.spark.sql.DataFrame = [company: struct<0: string, 1: string ... 118 more fields>]\n"
* ]
* },
* "execution\_count": 64,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "df\_json.select(col(\"company\"))"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Deux solutions pour explode le schéma JSON"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 12,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+---+--------------------+\n",
* "|  0|                   0|\n",
* "+---+--------------------+\n",
* "|  0|3623-Arrington En...|\n",
* "|  1|5874 - Sergey Cab...|\n",
* "| 10|KOAM Taxi Associa...|\n",
* "|100|2767 - Sayed M Badri|\n",
* "|101|Dispatch Taxi Aff...|\n",
* "|102|2092 - 61288 Sbei...|\n",
* "|103|4197 - 41842 Roya...|\n",
* "|104| 2192 - Zeymane Corp|\n",
* "|105|3591- 63480 Chuk'...|\n",
* "|106|1247 - 72807 Dani...|\n",
* "|107|Taxi Affiliation ...|\n",
* "|108|     3385 - Eman Cab|\n",
* "|109|Choice Taxi Assoc...|\n",
* "| 11|         Blue Cab Co|\n",
* "|110|3669 - Jordan Tax...|\n",
* "|111|2823 - 73307 Seun...|\n",
* "|112|3897 - 57856 Ilie...|\n",
* "|113|3591 - 63480 Chuk...|\n",
* "|114|1408 - Donald Barnes|\n",
* "|115|6742 - 83735 Tash...|\n",
* "+---+--------------------+\n",
* "only showing top 20 rows\n",
* "\n"
* ]
* },
* {
* "data": {
* "text/plain": [
* "dfExpl: org.apache.spark.sql.DataFrame = [0: string, 1: string ... 118 more fields]\n"
* ]
* },
* "execution\_count": 12,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val dfExpl = df\_json.select(\"company.\*\")\n",
* "\n",
* "dfExpl.columns\n",
* ".map(name => dfExpl.select(lit(name),col(name)))\n",
* "  .reduce(\_ union \_)\n",
* "  .show"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 13,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+---+--------------------+\n",
* "| id|               value|\n",
* "+---+--------------------+\n",
* "|  0|3623-Arrington En...|\n",
* "|  1|5874 - Sergey Cab...|\n",
* "| 10|KOAM Taxi Associa...|\n",
* "|100|2767 - Sayed M Badri|\n",
* "|101|Dispatch Taxi Aff...|\n",
* "|102|2092 - 61288 Sbei...|\n",
* "|103|4197 - 41842 Roya...|\n",
* "|104| 2192 - Zeymane Corp|\n",
* "|105|3591- 63480 Chuk'...|\n",
* "|106|1247 - 72807 Dani...|\n",
* "|107|Taxi Affiliation ...|\n",
* "|108|     3385 - Eman Cab|\n",
* "|109|Choice Taxi Assoc...|\n",
* "| 11|         Blue Cab Co|\n",
* "|110|3669 - Jordan Tax...|\n",
* "|111|2823 - 73307 Seun...|\n",
* "|112|3897 - 57856 Ilie...|\n",
* "|113|3591 - 63480 Chuk...|\n",
* "|114|1408 - Donald Barnes|\n",
* "|115|6742 - 83735 Tash...|\n",
* "+---+--------------------+\n",
* "only showing top 20 rows\n",
* "\n"
* ]
* },
* {
* "data": {
* "text/plain": [
* "dfExpl: org.apache.spark.sql.DataFrame = [0: string, 1: string ... 118 more fields]\r\n",
* "selectExpr: Array[org.apache.spark.sql.Column] = Array(named\_struct(id, 0 AS `id`, NamePlaceholder(), 0 AS `value`) AS `col`, named\_struct(id, 1 AS `id`, NamePlaceholder(), 1 AS `value`) AS `col`, named\_struct(id, 10 AS `id`, NamePlaceholder(), 10 AS `value`) AS `col`, named\_struct(id, 100 AS `id`, NamePlaceholder(), 100 AS `value`) AS `col`, named\_struct(id, 101 AS `id`, NamePlaceholder(), 101 AS `value`) AS `col`, named\_struct(id, 102 AS `id`, NamePlaceholder(), 102 AS `value`) AS `col`, named\_struct(id, 103 AS `id`, NamePlaceholder(), 103 AS `value`) AS `col`, named\_struct(id, 104 AS `id`, NamePlaceholder(), 104 AS `value`) AS `col`, named\_struct(id, 105 AS `id`, NamePlaceholder(), 105 AS `value`) AS..."
* ]
* },
* "execution\_count": 13,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val dfExpl = df\_json.select(\"company.\*\")\n",
* "val selectExpr = dfExpl\n",
* "  .columns\n",
* "  .map(name =>\n",
* "    struct(\n",
* "      lit(name).as(\"id\"),\n",
* "      col(name).as(\"value\")\n",
* "    ).as(\"col\")\n",
* "  )\n",
* "\n",
* "\n",
* "dfExpl\n",
* "  .select(\n",
* "    explode(array(selectExpr: \_\*))\n",
* "  )\n",
* "  .select(\"col.\*\")\n",
* "  .show()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Enrichissement avec la deuxième solution"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 10,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "explodeJSON: (df: org.apache.spark.sql.DataFrame, colname: String)org.apache.spark.sql.DataFrame\n"
* ]
* },
* "execution\_count": 10,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "def explodeJSON(df : org.apache.spark.sql.DataFrame,colname : String)={\n",
* "    val dfExpl = df.select(colname + \".\*\")\n",
* "    val selectExpr = dfExpl\n",
* "      .columns\n",
* "      .map(name =>\n",
* "        struct(\n",
* "          lit(name).as(\"id\"),\n",
* "          col(name).as(\"value\")\n",
* "        ).as(\"col\")\n",
* "      )\n",
* "\n",
* "\n",
* "    dfExpl\n",
* "      .select(\n",
* "        explode(array(selectExpr: \_\*))\n",
* "      )\n",
* "      .select(\"col.\*\")\n",
* "}"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 11,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "pickup\_latitude: org.apache.spark.sql.DataFrame = [id: string, value: string]\r\n",
* "pickup\_longitude: org.apache.spark.sql.DataFrame = [id: string, value: string]\r\n",
* "dropoff\_latitude: org.apache.spark.sql.DataFrame = [id: string, value: string]\r\n",
* "dropoff\_longitude: org.apache.spark.sql.DataFrame = [id: string, value: string]\r\n",
* "company: org.apache.spark.sql.DataFrame = [id: string, value: string]\n"
* ]
* },
* "execution\_count": 11,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val pickup\_latitude = explodeJSON(df\_json,\"pickup\_latitude\")\n",
* "val pickup\_longitude = explodeJSON(df\_json,\"pickup\_longitude\")\n",
* "val dropoff\_latitude = explodeJSON(df\_json,\"dropoff\_latitude\")\n",
* "val dropoff\_longitude = explodeJSON(df\_json,\"dropoff\_longitude\")\n",
* "val company = explodeJSON(df\_json,\"company\")"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 12,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "df\_enrich: org.apache.spark.sql.DataFrame = [taxi\_id: int, trip\_start\_timestamp: timestamp ... 18 more fields]\n"
* ]
* },
* "execution\_count": 12,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "val df\_enrich = df\n",
* "    .join(company,col(\"company\") === col(\"id\"))\n",
* "    .drop(\"id\")\n",
* "    .withColumnRenamed(\"value\",\"company\_name\")\n",
* "\n",
* "    .join(pickup\_latitude,col(\"pickup\_latitude\") === col(\"id\"))\n",
* "    .drop(\"id\",\"pickup\_latitude\")\n",
* "    .withColumnRenamed(\"value\",\"pickup\_latitude\")\n",
* "\n",
* "    .join(pickup\_longitude,col(\"pickup\_longitude\") === col(\"id\"))\n",
* "    .drop(\"id\",\"pickup\_longitude\")\n",
* "    .withColumnRenamed(\"value\",\"pickup\_longitude\")\n",
* "\n",
* "    .join(dropoff\_latitude,col(\"dropoff\_latitude\") === col(\"id\"))\n",
* "    .drop(\"id\",\"dropoff\_latitude\")\n",
* "    .withColumnRenamed(\"value\",\"dropoff\_latitude\")\n",
* "\n",
* "    .join(dropoff\_longitude,col(\"dropoff\_longitude\") === col(\"id\"))\n",
* "    .drop(\"id\",\"dropoff\_longitude\")\n",
* "    .withColumnRenamed(\"value\",\"dropoff\_longitude\")"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 13,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "res4: df\_enrich.type = [taxi\_id: int, trip\_start\_timestamp: timestamp ... 18 more fields]\n"
* ]
* },
* "execution\_count": 13,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "df\_enrich.cache()"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Classement des sociétés de taxis"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 48,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "+----------------------------------------+---------+\n",
* "|company\_name                            |CA       |\n",
* "+----------------------------------------+---------+\n",
* "|Taxi Affiliation Services               |3427225.0|\n",
* "|Dispatch Taxi Affiliation               |958146.0 |\n",
* "|Choice Taxi Association                 |714944.0 |\n",
* "|Blue Ribbon Taxi Association Inc.       |691532.0 |\n",
* "|Northwest Management LLC                |294186.0 |\n",
* "|KOAM Taxi Association                   |222517.0 |\n",
* "|Top Cab Affiliation                     |208510.0 |\n",
* "|Chicago Medallion Leasing INC           |40311.0  |\n",
* "|Chicago Medallion Management            |20619.0  |\n",
* "|Chicago Elite Cab Corp. (Chicago Carriag|7337.0   |\n",
* "|5129 - 87128                            |3654.0   |\n",
* "|6574 - Babylon Express Inc.             |3554.0   |\n",
* "|2092 - 61288 Sbeih company              |3111.0   |\n",
* "|3141 - 87803 Zip Cab                    |3055.0   |\n",
* "|3011 - 66308 JBL Cab Inc.               |3047.0   |\n",
* "|5874 - 73628 Sergey Cab Corp.           |2997.0   |\n",
* "|3623 - 72222 Arrington Enterprises      |2977.0   |\n",
* "|3201 - C&D Cab Co Inc                   |2924.0   |\n",
* "|5724 - 75306 KYVI Cab Inc               |2911.0   |\n",
* "|4197 - 41842 Royal Star                 |2901.0   |\n",
* "+----------------------------------------+---------+\n",
* "only showing top 20 rows\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\_enrich\n",
* ".groupBy(\"company\_name\")\n",
* ".agg(round(sum(\"fare\")).alias(\"CA\"))\n",
* ".orderBy(desc(\"CA\"))\n",
* ".show(false)"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "### Elastic et Kibana\n",
* "#### Dans Kibana, faire un put du schéma de df\_enrich\n",
* "#### Puis utiliser la librairie elastic pour spark pour insérer df\_enrich dans l'index\n",
* "#### Enfin effectuer des viz sous Kibana"
* ]
* },
* {
* "cell\_type": "markdown",
* "metadata": {},
* "source": [
* "PUT taxis\n",
* "{\n",
* "  \"mappings\": {\n",
* "    \"fare\" : {\n",
* "      \"properties\": {\n",
* "        \"taxi\_id\" : {\n",
* "          \"type\": \"text\"\n",
* "        },\n",
* "        \"trip\_start\_timestamp\" :{\n",
* "          \"type\": \"date\"\n",
* "        },\n",
* "        \"trip\_end\_timestamp\" :{\n",
* "          \"type\": \"date\"\n",
* "        },\n",
* "        \"trip\_seconds\" : {\n",
* "          \"type\": \"long\"\n",
* "        },\n",
* "        \"trip\_miles\" : {\n",
* "          \"type\": \"double\"\n",
* "        },\n",
* "        \"dropoff\_census\_tract\" : {\n",
* "          \"type\": \"long\"\n",
* "        },\n",
* "        \"pickup\_community\_area\" : {\n",
* "          \"type\": \"long\"\n",
* "        },\n",
* "        \"dropoff\_community\_area\" : {\n",
* "          \"type\": \"long\"\n",
* "        },\n",
* "        \"fare\" : {\n",
* "          \"type\": \"double\"\n",
* "        },\n",
* "        \"tips\" : {\n",
* "          \"type\": \"double\"\n",
* "        },\n",
* "        \"tolls\" : {\n",
* "          \"type\": \"double\"\n",
* "        },\n",
* "        \"extras\" : {\n",
* "          \"type\": \"double\"\n",
* "        },\n",
* "        \"trip\_total\" : {\n",
* "          \"type\": \"double\"\n",
* "        },\n",
* "        \"payment\_type\" : {\n",
* "          \"type\": \"keyword\"\n",
* "        },\n",
* "        \"company\_name\" : {\n",
* "          \"type\": \"keyword\"\n",
* "        },\n",
* "        \"company\" : {\n",
* "          \"type\": \"long\"\n",
* "        },\n",
* "        \"pickup\_geopoint\" :{\n",
* "          \"type\": \"geo\_point\"\n",
* "        },\n",
* "        \"dropoff\_geopoint\" :{\n",
* "          \"type\": \"geo\_point\"\n",
* "        }\n",
* "      }\n",
* "    }\n",
* "  }\n",
* "}"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 16,
* "metadata": {},
* "outputs": [
* {
* "name": "stdout",
* "output\_type": "stream",
* "text": [
* "root\n",
* " |-- taxi\_id: integer (nullable = true)\n",
* " |-- trip\_start\_timestamp: timestamp (nullable = true)\n",
* " |-- trip\_end\_timestamp: timestamp (nullable = true)\n",
* " |-- trip\_seconds: integer (nullable = true)\n",
* " |-- trip\_miles: double (nullable = true)\n",
* " |-- dropoff\_census\_tract: integer (nullable = true)\n",
* " |-- pickup\_community\_area: integer (nullable = true)\n",
* " |-- dropoff\_community\_area: integer (nullable = true)\n",
* " |-- fare: double (nullable = true)\n",
* " |-- tips: double (nullable = true)\n",
* " |-- tolls: double (nullable = true)\n",
* " |-- extras: double (nullable = true)\n",
* " |-- trip\_total: double (nullable = true)\n",
* " |-- payment\_type: string (nullable = true)\n",
* " |-- company: integer (nullable = true)\n",
* " |-- company\_name: string (nullable = true)\n",
* " |-- pickup\_latitude: string (nullable = true)\n",
* " |-- pickup\_longitude: string (nullable = true)\n",
* " |-- dropoff\_latitude: string (nullable = true)\n",
* " |-- dropoff\_longitude: string (nullable = true)\n",
* "\n"
* ]
* }
* ],
* "source": [
* "df\_enrich.printSchema"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 15,
* "metadata": {},
* "outputs": [
* {
* "data": {
* "text/plain": [
* "import org.elasticsearch.spark.sql.\_\n"
* ]
* },
* "execution\_count": 15,
* "metadata": {},
* "output\_type": "execute\_result"
* }
* ],
* "source": [
* "import org.elasticsearch.spark.sql.\_"
* ]
* },
* {
* "cell\_type": "code",
* "execution\_count": 20,
* "metadata": {},
* "outputs": [],
* "source": [
* "df\_enrich\n",
* ".withColumn(\"pickup\_geopoint\",concat(col(\"pickup\_latitude\"),lit(\",\"),col(\"pickup\_longitude\")))\n",
* ".withColumn(\"dropoff\_geopoint\",concat(col(\"dropoff\_latitude\"),lit(\",\"),col(\"dropoff\_longitude\")))\n",
* ".drop(\"pickup\_latitude\",\"pickup\_longitude\",\"dropoff\_latitude\",\"dropoff\_longitude\")\n",
* ".saveToEs(\"taxis/fare\")"
* ]
* }
* ],
* "metadata": {
* "kernelspec": {
* "display\_name": "spylon-kernel",
* "language": "scala",
* "name": "spylon-kernel"
* },
* "language\_info": {
* "codemirror\_mode": "text/x-scala",
* "file\_extension": ".scala",
* "help\_links": [
* {
* "text": "MetaKernel Magics",
* "url": "<https://github.com/calysto/metakernel/blob/master/metakernel/magics/README.md>"
* }
* ],
* "mimetype": "text/x-scala",
* "name": "scala",
* "pygments\_lexer": "scala",
* "version": "0.4.0"
* }
* },
* "nbformat": 4,
* "nbformat\_minor": 2
* }