Paralelné a distribuované systémy
Projekt

Lucia Kokuľová
1Im, 2018-2019

1 Úvod

Cieľom tohto projektu je vypracovať zadanie využívajúce dataset New York City Taxi Fare Dataset pomocou programu v jazyku Scala. Tento dataset obsahuje viac ako 55 miliónov riadkov, pričom jeden riadkov obsahuje údaje o jednej jazde taxíkom vo formáte

key, //identifikátor jazdy zložený z datetime informácií a jedinečného čísla
fare_amount, // suma zaplatená za jazdu
pickup_datetime, // dátum a čas začiatku jazdy
pickup_longitude, // súradnice začiatku jazdy
pickup_latitude,
dropoff_longitude, // súradnice konca jazdy
dropoff_latitude,
passenger_count. // počet pasažierov v taxíku

Zadanie projektu je rozdelené do dvoch častí. V prvej z nich vyrátame priemernú sumu zaplatenú jedným pasažierom za jeden kilometer jazdy v jednotlivých rokoch, ktoré obsahuje dataset. V druhej časti zadania ukážeme kedy a odkiaľ, resp. kam pasažieri najčastejšie cestujú.

2 Predspracovanie dát

Po určitej dobe kedy som pracovala s datasetom som zistila, že obsahuje mnoho neplatných, chybných dát. Prvým krokom v projekte je preto filtrácia dát, ktorá vyzerá nasledovne:

```
//Read file and create RDD
val textFile = sc.textFile("C:/Users/lucka/Desktop/Taxi/train.csv")
val first = textFile.first()
val data = textFile
   .filter(row => row != first)
   .filter(x => !x.contains(",,"))
   .filter(x => !x.contains("0,0,0,0"))
   .filter(x => !x.contains(",0"))
   .map(x => x.split(","))

//Create dataframe and filter invalid values
val filtered = data
   .toDF()
   .select(concat_ws(",", $"value"))
   .withColumn("_tmp", split($"concat_ws(,, value)", "\\,")).select(
      $"_tmp".getItem(0).as("key"),
      $"_tmp".getItem(1).as("fare_amount"),
      $"_tmp".getItem(2).as("pickup_datetime"),
      $"_tmp".getItem(3).as("pickup_longitude"),
      $"_tmp".getItem(4).as("pickup_latitude"),
      $"_tmp".getItem(5).as("dropoff_longitude"),
      $"_tmp".getItem(6).as("dropoff_latitude"),
      $"_tmp".getItem(7).as("passenger_count")
    ).drop("_tmp")
   .filter(!($"pickup_longitude" === $"pickup_latitude" &&
$"pickup_latitude" === $"dropoff_longitude" && $"dropoff_longitude"
=== $"dropoff_latitude"))
   .filter(!($"pickup_longitude" === $"dropoff_longitude" &&
$"pickup_latitude" === $"dropoff_latitude"))
   .filter(($"pickup_longitude".startsWith("-73.")||
$"pickup_longitude".startsWith("-74.")) &&
$"pickup_latitude".startsWith("40.") &&
($"dropoff_longitude".startsWith("-73.") ||
$"dropoff_longitude".startsWith("-74.")) &&
$"dropoff_latitude".startsWith("40."))
   .filter(!(col("fare_amount") < 0.0))
   .filter(Row => Haversine.distance(Row.getString(3).toDouble,
Row.getString(4).toDouble, Row.getString(5).toDouble,
Row.getString(6).toDouble) >= 1.0)
```

Tieto riadky kódu zaručia že dataset po filtrácii neobsahuje prázdne polia v riadkoch, nulové geografické súradnice a nulový počet pasažierov. Takéto riadky sú bezpredmetné pre naše zadania. Ďalšia filtrácia z dát vylučuje riadky kde sa všetky súradnice rovnajú (ak vzdialenosť jazdy je 0 nemá zmysel rátať cenu za kilometer jazdy) a taktiež riadky kde sa rovnajú počiatočné a konečné body jazdy (rovnaký dôvod). Následne som tiež zabezpečila aby všetky zemepisné dĺžky začínali na "-73." alebo "-74." a zemepisné šírky aby začínali na "40.", čím som vylúčila nezmyselné súradnice a súradnice mimo mesta New York.

3 Riešenie zadania

3.1 Prvá časť zadania

Prvou úlohou bolo vyrátať priemernú cenu na pasažiera na 1 km vzdialenosti. K tomu bolo potrebné vyrátať pre každý riadok vzdialenosti podľa zemepisných dĺžok a šírok (Haversine formula), vyrátať pre každý riadok priemernú cenu na jeden kilometer pre jedného pasažiera a následne zo všetkých riadkov vyrátať priemernú sumu pre celý dataset podľa jednotlivých rokov, ktoré sa v datasete nachádzajú.

Na vyriešenie tohto zadania som potrebovala využiť Haversine formulu na určenie vzdialenosti zo zadaných geografických súradníc, ktorá vyzerá nasledovne:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

V projekte som teda vytvorila objekt *Haversine*, ktorý obsahuje metódu *distance*. Táto metóda ako parametre dostane 4 súradnice a jej výstupom je vzdialenosť v kilometroch.

```
object Haversine {
  import math._

  val R = 6372.8  //radius in km

  // returns distance in km
  def distance(lon1:Double, lat1:Double, lon2:Double, lat2:Double)={
    // calculate differences in radians
    val dLat=(lat2 - lat1).toRadians
    val dLon=(lon2 - lon1).toRadians
    val a = pow(sin(dLat/2),2) + pow(sin(dLon/2),2) *
cos(lat1.toRadians) * cos(lat2.toRadians)
    R * 2 * asin(sqrt(a))
  }
}
```

Následne som pomocou tejto metódy vyrátala priemernú sumu zaplatenú 1 pasažierom za 1 kilometer jazdy pre každý riadok datasetu. Pracovala som teda s dataframe-om, ktorý obsahoval jeden stĺpec - túto priemernú sumu a s druhým dataframe-om, ktorý obsahoval všetky dáta z pôvodného datasetu. Pomocou funkcie *join* som vytvorila jeden dataframe spojením pôvodných dvoch, na ktorý som následne použila funkcie *groupBy* a *aggregate*, čím som dostala výsledok, ktorý obsahuje 4 stĺpce: jednotlivé roky ktoré sa nachádzajú v datasete, počet výskytov týchto rokov, súčet priemerných hodnôt za jednotlivé roky a posledný stĺpec, ktorý je výsledkom riešenia zadania - priemernú sumu za jednotlivé roky pre 1 pasažiera za 1 kilometer.

```
val averageForLine = filtered
   .map(Row => Row.getString(1).toDouble /
Haversine.distance(Row.getString(3).toDouble,
```

```
Row.getString(4).toDouble, Row.getString(5).toDouble,
Row.getString(6).toDouble) / Row.getString(7).toDouble)

val dataYears = filtered.withColumn("year", col("key").substr(0,4))
val df1 = dataYears.withColumn("id",monotonically_increasing_id())
val df2 =
averageForLine.withColumn("id",monotonically_increasing_id())

val joinDF = df1.join(df2, "id")
val average = joinDF.groupBy("year").agg(count("year") as "year
count", sum("value") as "average sum per year",
sum("value")/count("year") as "average per 1 pass / 1
km").sort(asc("year"))
average.show()
```

3.2 Druhá časť zadania

V druhej časti projektu bolo mojou úlohou vyrátať kedy (dni a hodiny) a odkiaľ/kam pasažieri najviac cestujú. K tomu som potrebovala vytriediť dni v mesiaci a jednotlivé hodiny. Ďalší dataframe obsahuje začiatočné a koncové body jázd (zemepisná dĺžka + zemepisná šírka). Vyrátala som teda v akom kalendárnom dni (1 - 31), v akých časoch (hodiny:minúty / hodiny), odkiaľ (súradnice začiatočného bodu) a kam (súradnice koncového bodu) pasažieri najčastejšie cestujú.

```
//most frequent time
val time = filtered
   .map(Row => Row.getString(0).substring(8, 10) + ", " +
Row.getString(0).substring(5, 7) + ", " +
Row.getString(0).substring(11, 16))
val timeDF = time
   .withColumn("_tmp", split($"value", "\\,"))
   .withColumn("day", $"_tmp".getItem(0))
   .withColumn("month", $"_tmp".getItem(1))
   .withColumn("time", $"_tmp".getItem(2))
   .drop("_tmp")
   .drop("value")
timeDF.groupBy("time").agg(count("time") as
"count").sort(desc("count")).show(10)

//most frequent hours
val timeHours = filtered
   .map(Row => Row.getString(0).substring(8, 10) + ", " +
Row.getString(0).substring(5, 7) + ", " +
Row.getString(0).substring(11, 13))
val timeHoursDF = timeHours
   .toDF()
   .withColumn("_tmp", split($"value", "\\,"))
   .withColumn("day", $"_tmp".getItem(0))
   .withColumn("month", $"_tmp".getItem(1))
   .withColumn("time in hours", $"_tmp".getItem(2))
   .drop("_tmp")
   .drop("value")
timeHoursDF.groupBy("time in hours").agg(count("time in hours") as
"count").sort(desc("count")).show(10)
```

```
//most frequent days
val days = filtered
   .map(Row => Row.getString(0).substring(8, 10) + ", " +
Row.getString(0).substring(5, 7) + ", " +
Row.getString(0).substring(11, 13))
val daysDF = days
   .toDF()
   .withColumn("_tmp", split($"value", "\\,"))
   .withColumn("day", $"_tmp".getItem(0))
   .withColumn("month", $"_tmp".getItem(1))
   .withColumn("time in hours", $"_tmp".getItem(2))
   .drop("_tmp")
   .drop("value")
daysDF.groupBy("day").agg(count("day") as
"count").sort(desc("count")).show(10)

//most frequent pickup coordinates
val pickup = filtered
   .map(Row => Row.getString(3) + ", " + Row.getString(4))
val pickupDF = pickup.toDF("pickup coordinates")
pickupDF.groupBy("pickup coordinates").agg(count("pickup
coordinates") as "count").sort(desc("count")).show(10, false)

//most frequent dropoff coordinates
val dropoff = filtered
   .map(Row => Row.getString(5) + ", " + Row.getString(6))
val dropoffDF = dropoff.toDF("dropoff coordinates")
dropoffDF.groupBy("dropoff coordinates").agg(count("dropoff
coordinates") as "count").sort(desc("count")).show(10, false)
```

Vykonanie spomínaných zdrojových kódov (celý skript) trvalo približne 20-25 minút.

4 Výsledky

4.1 Prvá časť zadania

year	year count	average sum per year	average per 1 pass / 1 km
2009	6933907	1.923035157639279E7	2.773378930001915
2010	6761002	1.904462014150704E7	2.816833975423619
2011	7093636	1.988354654386918 2E7	2.8030119594336647
2012	7238592	2.145286430064352E7	2.963679165871418
2013	7130148	2.379274605465092E7	3.3369217658106005
2014	6779067	2.3207515022907626E7	3.423408416365796
2015	3149616	1.0988774619220402E7	3.488925195712875

Výsledný dataframe ukázal, že priemerná suma za 1 kilometer sú približne 3 doláre. Najvyššia priemerná suma bola v roku 2015, pričom ak sa pozrieme na sumy v poradí po rokoch, tak vidíme že cena za 1 kilometer postupne stúpala z 2,77 až na 3,48 dolárov.

4.2 Druhá časť zadania

V akých dňoch sa najviac cestovalo:

day	count
13	1543159
15	1539800
19	1537954
16	1534624
17	1532899
14	1530714
18	1529922
12	1528694
10	1526082
20	1521596

Výsledok ukazuje, že najfrekventovanejšie dni pri cestovaní taxíkmi sú dni v polovici mesiaca, v rozmedzí 13 - 19. Vidíme ale že dni sú pomerne rovnomerne rozložené (počtom) a preto toto nemusí byť relevantný údaj.

V akých časoch (hodiny:minúty / hodiny) sa najviac cestovalo:

time	count
19:16	48720
18:50	48329
19:23	48015
19:12	48003
19:17	47919
19:24	47916
18:42	47849
19:26	47844
19:50	47828
19:28	47825

time in hours	count
19	2809604
18	2698314
20	2693993
21	2656846
22	2607748
23	2323947
17	2221622
14	2214849
12	2142038
13	2139619

Z výsledku vyplýva že najčastejšie časy objednávania taxíkov sú medzi 18:42 a 19:50. To je viditeľné aj z druhého výsledku, ktorý zobrazuje časové údaje v hodinách, pričom najčastejšie sú večerné hodiny medzi 18:00 a 23:00. Najčastejšia hodina (19) sa oproti druhej v poradí (18) vyskytuje v datasete o viac ako 100 000 krát.

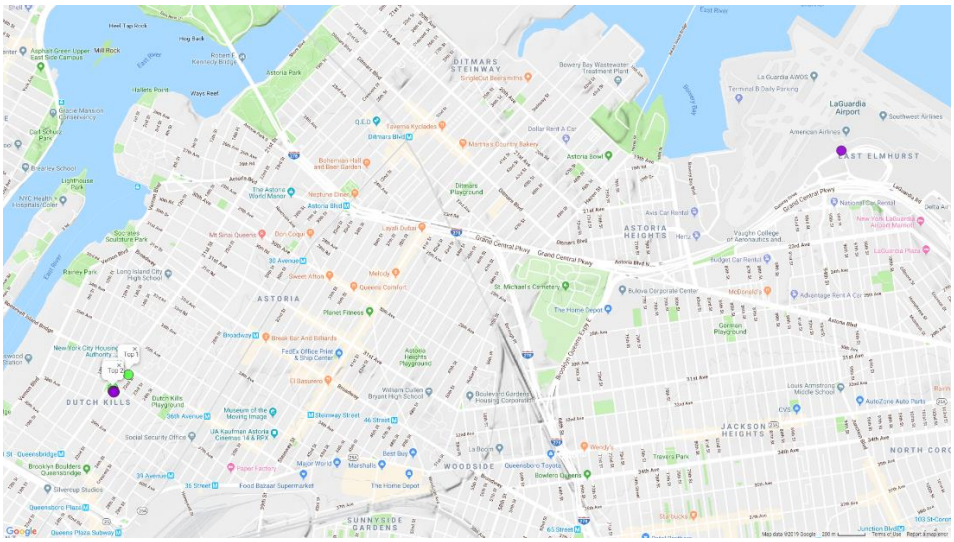
Odkiaľ a kam pasažieri najčastejšie cestovali:

pickup coordinates	count
-73.93649, 40.75936	672
-73.937872, 40.758282	604
-73.937835, 40.758457	560
-73.937715, 40.758277	529
-73.937787, 40.758187	528
-73.937667, 40.758215	392
-73.937778, 40.75825	355
-73.937768, 40.758242	313
-73.937765, 40.758267	281
-73.8745, 40.7741	281

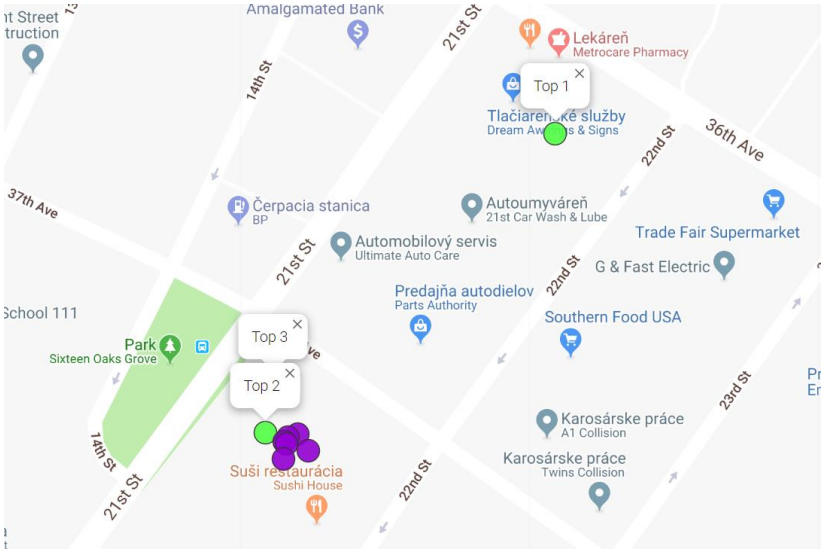
dropoff coordinates	count
-73.8616, 40.7683	122
-73.8615, 40.7682	96
-74.0027, 40.7606	92
-73.8618, 40.7684	90
-73.865, 40.7705	89
-73.8619, 40.7685	86
-74.0028, 40.7605	83
-73.8617, 40.7684	80
-73.8709, 40.7741	76
-73.9947, 40.7504	76

Keďže nám tieto súradnice veľa nehovoria v číselnej podobe, vykreslila som ich pomocou online nástroja a výsledky sú nasledovné:

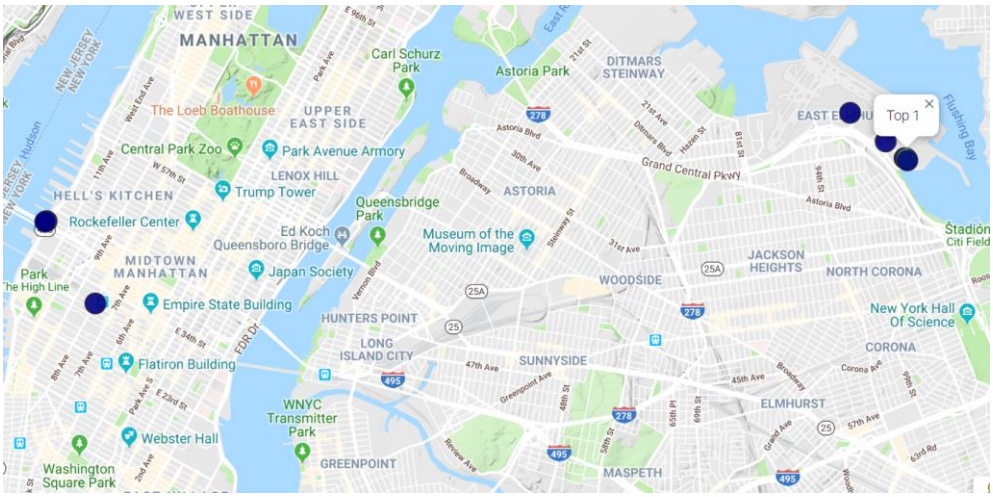
Začiatky jázd (zelenou farbou sú vykreslené 3 najčastejšie súradnice)
Všetky tieto súradnice sa nachádzajú v oblasti Queens.



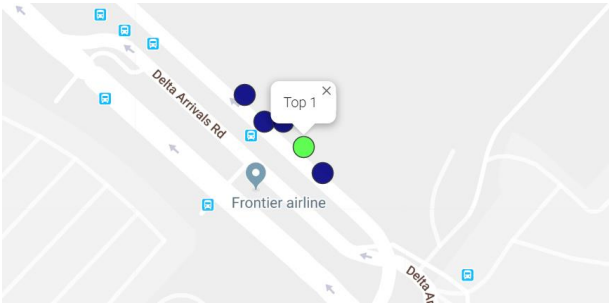
Lokalita troch najčastejších súradníc:



Konečné body jázd: (7 – Queens, 3 – Manhattan)



Najčastejšia lokalita (prvý bod z výsledného dataframe je výrazne častejší ako zvyšné, preto len jeden) sa nachádza pri letisku LaGuardia Airport, spolu s ďalšími 4 bodmi.



5 Záver

V prvej časti sme vyrátali, že priemerná suma pre jedného pasažiera za 1 km jazdy sú približne 3 doláre. Túto sumu sme však rátali pomocou vzdialenosti vyrátanej z Haversine formuly, čo je vzdušná vzdialenosť. Preto tento údaj je veľmi skreslený a nemusí zodpovedať realite. V druhej časti riešenia pri analýze časových údajov (v akých dňoch najviac cestovalo) sú výsledky pomerne rovnomerne rozložené a preto sa nedá hovoriť o najčastejšie vyskytujúcich sa hodnotách. Pri analýze času v hodinách a pri lokalizačných údajov (začiatočné a koncové body jázd) sú už výsledky o niečo rozmanitejšie, preto vieme lepšie porovnať najčastejšie hodnoty v datasete.