Report Big Data project: Big Sales

Carlo Di Raddo - Mat 1007219
Demetrio Andriani - Mat 10033010

Marzo 2023

Contents

1 Introduzione

1.1 Descrizione del Dataset

Il dataset analizzato con il nome "Liquor Sales.csv" (`https://www.kaggle.com/datasets/pigment/big-sales-data`), contiene informazioni sulle vendite di bevande alcoliche effettuate da diversi negozi in America nel periodo che va da Gennaio 2012 a Settembre 2020.

La grandezza del dataset è di 4.77 GB e contiene molte informazioni sulle transazioni di vendita, tra cui il tipo di bevanda alcolica venduta, il prezzo di vendita, la quantità venduta, la data e l'ora della transazione e il negozio in cui la transazione è stata effettuata.

Il dataset può essere utilizzato per analizzare il mercato delle bevande alcoliche in America e per comprendere le tendenze di vendita nel corso del tempo. Ad esempio, si può valutare la popolarità di diversi tipi di bevande alcoliche in base alla loro quota di mercato, oppure si può esaminare come le vendite di alcune bevande alcoliche sono influenzate da eventi stagionali come le festività.

Inoltre, il dataset può essere utilizzato per fare previsioni sulle vendite future di bevande alcoliche, ad esempio utilizzando tecniche di analisi delle serie temporali per identificare tendenze e stagionalità.

In generale, il dataset "Liquor Sales.csv" è una risorsa preziosa per coloro che vogliono studiare il mercato delle bevande alcoliche in America o per coloro che vogliono effettuare analisi di mercato e previsioni.

Il dataset è composto da una serie di righe, ognuna delle quali rappresenta un singolo evento. Per ogni evento sono registrati diversi attributi, tra cui:

- la data di vendita del prodotto.
- il nome del negozio che ha effettuato la vendita,
- la città in cui si trova il negozio,
- le coordinate geografiche del negozio (latitudine e longitudine),
- il paese in cui si trova la città,
- la categoria del prodotto venduto,
- il formato del prodotto,
- il volume della bottiglia in millilitri,
- il numero di bottiglie vendute,
- il prezzo del prodotto in dollari.
- ecc ecc...

1.2 Obbiettivi dell'analisi

L'obbiettivo del progetto è analizzare il dataset, al fine di rilevare informazioni utili sui dati raccolti. Di seguito vengono riportate le **query esplorative** che ci siamo posti e a cui si è cercato di dare una risposta analizzando il dataset:

1. Capire che il numero degli stores number e degli stores name è diverso (es: negozio1(nome1,nome2,nome3));
2. Capire che il numero degli items number e degli items name è diverso;
3. Capire che il numero degli country number e degli country name è diverso;
4. Capire che il numero degli city number e degli city name è diverso;
5. Capire che il numero degli category number e degli category name è diverso;
6. Capire che il numero degli vendor number e degli vendor name è diverso;

Successivamente abbiamo svolto delle **query più complesse**:

1. In Media, quanti prodotti sono stati venduti per negozio?;
2. In Media, quanti litri sono stati vednuti per stato?;
3. In Media, quante bottiglie sono state vednute per categoria?;
4. items più venduti in termini di fatturato e unità;
5. Categorie più vendute in termini di fatturato e unità;

Per poi terminare con **i due job**:

1. (job1) svolgere un'analisi sul prezzo medio di ogni singolo prodotto per ogni anno,
2. (job2) svolgere una classificazione dei prodotti in base al fatturato e calcolare il fatturato totale per ogni categoria e fascia di prezzo.

2 Setting per l'analisi

2.1 Configurazione del cluster

Prima di parlare nel dettaglio delle varie query, è necessario, anche se pur brevemente, parlare della configuarzione del cluster. Nella nostra analisi abbiamo preferito una configurazione di default, utilizzata anche durante i laboratori. Esso è composto da 1 master e 2 nodi slave, entrambi con 4 core e 16 GB di RAM.Per quanto osservato in teoria, si sono ritenuti ottimali i setting del cluster e di conseguenza si è deciso di mantenerli invariati.

2.2 Preprocessing dei dati

Di seguito una porzione del preprocessing dei dati:

[2]:
```scala
import org.apache.spark.sql.SaveMode

object LiquorSalesParser {



  /** Function to parse stores records
   *
   *  @param line line that has to be parsed
   *  @return tuple containing storeNumber, storeName,, productsID none in case of input errors
   */

  def parseStores(line: String) : Option[(Long, String, String)] = {
    try {
      val input = line.split(',')
      Some(input(2).trim.toLong, input(3),  input(16))
    } catch {
      case _: Exception => None
    }
  }
```

Figure 1: parse

Il parse, come svolto in figura, è stato fatto per tutti i vari records di nostro interesse.

3 Query esplorative

Dopo la piccola digressione sul preprocessing dei dati, di seguito viene analizzato il dataset attraverso query esplorative. Vi sono due esempi delle query esplorative eseguite, una relativa agli store ed una relativa agli items:

```
/*Da questa analisi possiamo capire che il numero degli stores number e degli stores name è diverso => che il nome di un negozio può
essere cambiato nel corso degli anni => negozio#1(nome1,nome2,nome3)*/

val rddStoresCached = rddStores.cache()
"Number of Stores group by their name: " + rddStoresCached.map(x => (x._2)).distinct().count()
"Number of Stores group by their code: " + rddStoresCached.map(x => (x._1)).distinct().count()
```

```
Last executed at 2023-03-01 16:31:17 in 1m 43.67s
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
rddStoresCached: rddStores.type = MapPartitionsRDD[2] at flatMap at <console>:26
res9: String = Number of Stores group by their name: 2632
res10: String = Number of Stores group by their code: 2484
```

Figure 2: query esplorativa 1

```
/*Da questa analisi possiamo capire che il numero degli items number e degli items name è diverso*/

val rddItemsCached = rddItems.cache()
"Number of Items group by their name: " + rddItemsCached.map(x => (x._2)).distinct().count()
"Number of Items group by their code: " + rddItemsCached.map(x => (x._1)).distinct().count()
```

```
Last executed at 2023-03-01 16:33:11 in 1m 53.66s
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
rddItemsCached: rddItems.type = MapPartitionsRDD[5] at flatMap at <console>:26
res13: String = Number of Items group by their name: 7834
res14: String = Number of Items group by their code: 8322
```

Figure 3: query esplorativa 2

4 Query complesse

Dopo aver svolto semplici query esplorative di nostro interesse, ci siamo concentrati su query, sempre esplorative, ma di complessità livemente maggiore. In figura le 5 query esplorative più complesse eseguite, in particolare :

1. **In Media, quanti prodotti sono stati venduti per negozio;**

   Viene calcolato il numero medio di prodotti venduti per negozio attraverso l'utilizzo di una serie di operazioni sui dati contenuti nel dataset. In particolare, vengono mappati gli identificativi dei negozi e per ognuno di essi, viene creato un valore pari a 1. Questi valori vengono poi sommati insieme utilizzando la funzione "reduceByKey". Con la funzione "aggregate" si ottiene una struttara del (numeroItemsVenduti, numeroNegozi). Infine basta dividere il numeroItemsVenduti per numeroNegozi ed ottenere così la media dei prodotti venduti per singolo negozio

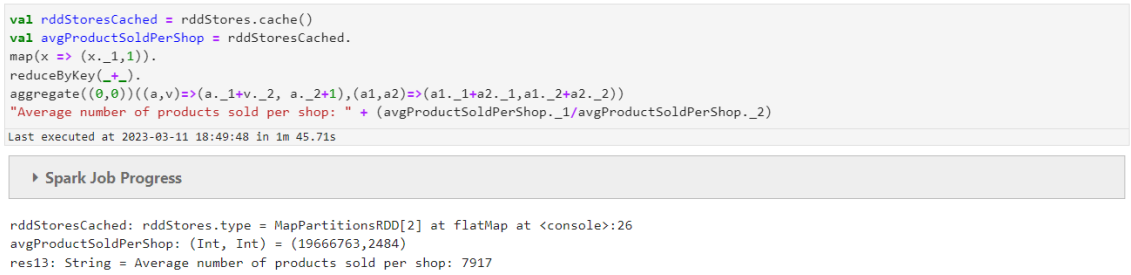
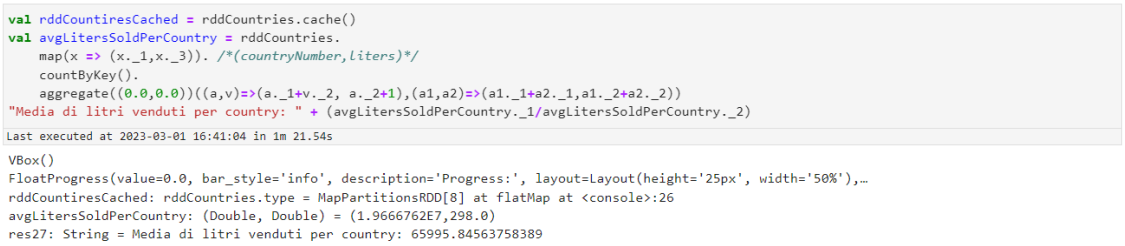


Figure 4: In Media, quanti prodotti sono stati venduti per negozio

2. **In Media, quanti litri sono stati vednuti per stato(country);** Inizialmente, la funzione "map" viene utilizzata per estrarre le informazioni relative al numero del paese e al numero di litri venduti. Successivamente, la funzione "countByKey" viene utilizzata per calcolare il numero di occorrenze di ogni paese nel dataset. Infine, la funzione "aggregate" viene utilizzata per calcolare la somma totale dei litri venduti e il numero totale di paesi presenti nel dataset. Infine, viene calcolata la media dei litri venduti per paese



```
val rddCountiresCached = rddCountries.cache()
val avgLitersSoldPerCountry = rddCountries.
    map(x => (x._1,x._3)). /*(countryNumber,liters)*/
    countByKey().
    aggregate((0.0,0.0))((a,v)=>(a._1+v._2, a._2+1),(a1,a2)=>(a1._1+a2._1,a1._2+a2._2))
"Media di litri venduti per country: " + (avgLitersSoldPerCountry._1/avgLitersSoldPerCountry._2)
```

```
Last executed at 2023-03-01 16:41:04 in 1m 21.54s
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
rddCountiresCached: rddCountries.type = MapPartitionsRDD[8] at flatMap at <console>:26
avgLitersSoldPerCountry: (Double, Double) = (1.9666762E7,298.0)
res27: String = Media di litri venduti per country: 65995.84563758389
```

Figure 5: In Media, quanti litri venduti per country

3. In Media, quante bottiglie sono state vednute per categoria;

```
val rddCategoriesCached = rddCategories.cache()
val avgBottlesPerCategory = rddCategories.
    map(x => (x._1,x._3)).
    countByKey().
    aggregate((0.0,0.0))((a,v)=>(a._1+v._2, a._2+1),(a1,a2)=>(a1._1+a2._1,a1._2+a2._2))
"Media di bottiglie vendute per categoria: " + (avgBottlesPerCategory._1/avgBottlesPerCategory._2)
```

```
Last executed at 2023-03-01 16:42:07 in 1m 3.48s
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
rddCategoriesCached: rddCategories.type = MapPartitionsRDD[14] at flatMap at <console>:26
avgBottlesPerCategory: (Double, Double) = (1559988.0,100.0)
res30: String = Media di bottiglie vendute per categoria: 15599.88
```

Figure 6: In Media, quante bottiglie sono state vendute per categoria

4. items più venduti in termini di fatturato e unità;

```
println("Risultato cosi' composto: nomeItem, fatturato, numero di vendite \n")
val rddItemsCached = rddItems.cache()
rddItemsCached.
        map(e => (e._2, (e._3, 1))).
        reduceByKey((t1, t2) => (t1._1 + t2._1, t1._2 + t2._2)).
        map(v => (v._1, v._2._1, v._2._2)).
        sortBy(_._2, false).
        collect().take(10).foreach(println(_))
```

```
Last executed at 2023-03-01 16:43:35 in 1m 27.54s
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
Risultato cosi' composto: nomeItem, fatturato, numero di vendite

rddItemsCached: rddItems.type = MapPartitionsRDD[5] at flatMap at <console>:26
(Hawkeye Vodka,7275708.29,411211)
(Black Velvet,5242031.529999999,220957)
(Captain Morgan Spiced Rum,4018092.1699999985,188778)
(Titos Handmade Vodka,3770229.4600000014,166741)
(Barton Vodka,3219200.8399999994,112894)
(Five O'clock Vodka,2997823.2799999984,226502)
(Phillips Vodka,2366347.33,107766)
(Jack Daniels Old #7 Black Lbl,2306382.9500000007,159353)
(Admiral Nelson Spiced Rum,1944860.9900000014,177337)
(Fireball Cinnamon Whiskey,1850195.199999996,122337)
```

Figure 7: items più venduti in termini di fatturato e unità

5. Categorie più vendute in termini di fatturato e unità;

```
println("Risultato cosi' composto: nomeCategor, fatturato, numero di vendite \n")
val rddCategoriesCached = rddCategories.cache()
rddCategoriesCached.
        map(e => (e._2, (e._4, 1))).
        reduceByKey((t1, t2) => (t1._1 + t2._1, t1._2 + t2._2)).
        map(v => (v._1, v._2._1, v._2._2)).
        sortBy(_._2, false).
        collect().take(10).foreach(println(_))
```

```
Last executed at 2023-03-03 12:28:18 in 1m 7.54s
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
Risultato cosi' composto: nomeCategor, fatturato, numero di vendite

rddCategoriesCached: rddCategories.type = MapPartitionsRDD[14] at flatMap at <console>:26
(American Vodkas,1.4152316429999966E7,140485)
(Canadian Whiskies,1.389965760000001E7,103166)
(VODKA 80 PROOF,9530538.619999986,109531)
(Imported Brandies,8827292.330000015,30078)
(Spiced Rum,7252489.699999989,49083)
(SPICED RUM,7068700.709999999,44743)
(Whiskey Liqueur,6887928.550000002,50702)
(CANADIAN WHISKIES,6197296.109999994,47384)
(TEQUILA,4861197.939999997,35800)
(Straight Bourbon Whiskies,4779353.42,47715)
```

Figure 8: Categorie più vendute in termini di fatturato e unità

5 Job

Dopo aver fatto le varie query esplorative, ci siamo focalizzati sui job da svolgere, nel dettaglio:

5.1 Job 1

il seguente job ha lo scopo di calcolare il prezzo medio di ogni prodotto per ogni anno utilizzando un RDD di oggetti item.

Inizialmente, abbiamo calcolato il prezzo medio di ogni prodotto per ogni anno, facendo:

1. map per ogni elemento dell'RDD "rddItems" in una coppia di chiave-valore, in cui la chiave è una tupla di (itemName,anno) e il valore è una coppia di (price, 1),
2. raggruppato le coppie di chiave-valore per chiave, ovvero per (itemName,anno) e somma tutti i prezzi e tutti i valori 1 associati a quella chiave,
3. map la coppia di chiave e valore in una tripla in cui la prima componente è il nome del prodotto, la seconda è il prezzo medio e la terza è l'anno.

Dopodichè, abbiamo raggruppato per nomeItem e fatto una flatmap con chiave prodotto e anno.Alla fine, abbiamo svolto una conversione in DataFrame con stringa e salvataggio su file CSV

```
val bucketname = "unibo-bigdata-project"
val path_itemsForYear_revenue = "s3a://"+bucketname+"/risultati/satates_revenue"

// calcolare il prezzo medio di ogni prodotto per ogni anno
val rddItemsCached = rddItems.cache()

val avgPricesByYear = rddItems.map(item => ((item._2, item._5), (item._3, 1))).//(itemName,anno), price
  reduceByKey((accum, value) => (accum._1 + value._1, accum._2 + value._2)).
  mapValues(sumCount => sumCount._1 / sumCount._2).
  map(item => (item._1._1, item._2, item._1._2))
  //collect() //(itemName,avg,year)

// Raggruppo per nomeItem
val avgPricesByProduct = avgPricesByYear.groupBy(_._1).
  mapValues(_.map(item => (item._2, item._3)))//.collect()

//flatmap con chiave prodotto e anno
val avgPricesByProductAndYear = avgPricesByProduct.flatMap { case (itemName, yearPrices) =>
  yearPrices.map { case (avgPrice, year) =>
    ((itemName, year), avgPrice)
  }
}.coalesce(1)

// Conversione in DataFrame con stringa
val df = avgPricesByProductAndYear.map{case ((itemName, year), avgPrice) => (itemName,year,avgPrice)}.toDF()
// Salvataggio su file CSV
df.write.format("csv").mode(SaveMode.Overwrite).save(path_itemsForYear_revenue)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
bucketname: String = unibo-bigdata-project
path_itemsForYear_revenue: String = s3a://unibo-bigdata-project/risultati/satates_revenue
rddItemsCached: rddItems.type = MapPartitionsRDD[5] at flatMap at <console>:26
avgPricesByYear: org.apache.spark.rdd.RDD[(String, Double, Int)] = MapPartitionsRDD[75] at map at <console>:28
avgPricesByProduct: org.apache.spark.rdd.RDD[(String, Iterable[(Double, Int)])] = MapPartitionsRDD[78] at mapValues at <console>:28
avgPricesByProductAndYear: org.apache.spark.rdd.RDD[((String, Int), Double)] = CoalescedRDD[80] at coalesce at <console>:30
df: org.apache.spark.sql.DataFrame = [_1: string, _2: int ... 1 more field]
```

Figure 9: job1

5.2 Job 2

Job 2 il seguente job ha lo scopo calcolare il fatturato e il numero di prodotti venduti per categoria e classificari con livello detto tier.

Inizialmente, abbiamo creato una variabile chiamata "v" che contiene una sequenza di tuple che rappresentano il nameItem, la categoria del prodotto, il fatturato e il numero di unità vendute, facendo:

1. fatto una map per ogni elemento dell'RDD "rddItems" crea una coppia di chiave-valore, dove la chiave è una tupla di (nameItem, categoryName) e il valore è una tupla di (price,1).

2. fatto una reduceByKey per sommare il fatturato e il numero di unità vendute per ogni coppia di chiavi.

3. fatto una map per creare una tupla di (nomeItem, categoryName, fatturato, unitàVendute) a partire dalla coppia di chiave-valore.

4. fatto un sortBy per ordinare gli elementi per fatturato decrescente.

Dopodichè, abbiamo classificato la categoria di appartenenza dei prodotti in base al fatturato: "Top Tier" se il fatturato supera 1 milione di euro, "Mid Tier" se il fatturato è compreso tra 100.000 e 1 milione di euro, "Low Tier" altrimenti. Per ottenere la classificazione,abbiamo:

1. fatto una map per estrarre il nomeItem, fatturato dal RDD v

2. fatto una reduceByKey e quindi aggregare i valori del fatturato per ciascun prodotto

3. fatto un match, assegna a ciascun prodotto una classificazione in base al suo fatturato. In particolare, i prodotti con un fatturato superiore a 1 milione vengono classificati come "Top Tier", quelli con un fatturato inferiore a 1 milione ma superiore a 100.000 vengono classificati come "Mid Tier", mentre i prodotti con un fatturato inferiore a 100.000 vengono classificati come "Low Tier". Il risultato è un RDD contenente il nomeItem, Tier.

Successivamente, abbiamo calcolato la distribuzione dei ricavi in base alle categorie di prodotto e ai livelli di fatturato dei prodotti. Per fare ciò abbiamo:

1. creato una variabile chiamata revenueByCategory,

2. fatto una map per creare una nuova struttura dati formata da nomeItem, (fatturato, categoryName)

3. fatto una join tra la struttura di dati appena creata e l'itemsClassification

4. fatto una map per creare una tupla che contiene il categoryName, Tier e fatturato dell'articolo.

5. fatto un aggregateByKey per aggregare i record per categoryName, tier, price e count per ogni categoria e tier.

6. fatto un map per creare una tupla che contiene categoryName, Tier, revenue, count

7. fatto il coalesce per semplificare l'output

Infine, revenueByCategory viene salvato in un file di testo in formato CSV con gli elementi separati da virgole mediante la funzione "mkString".

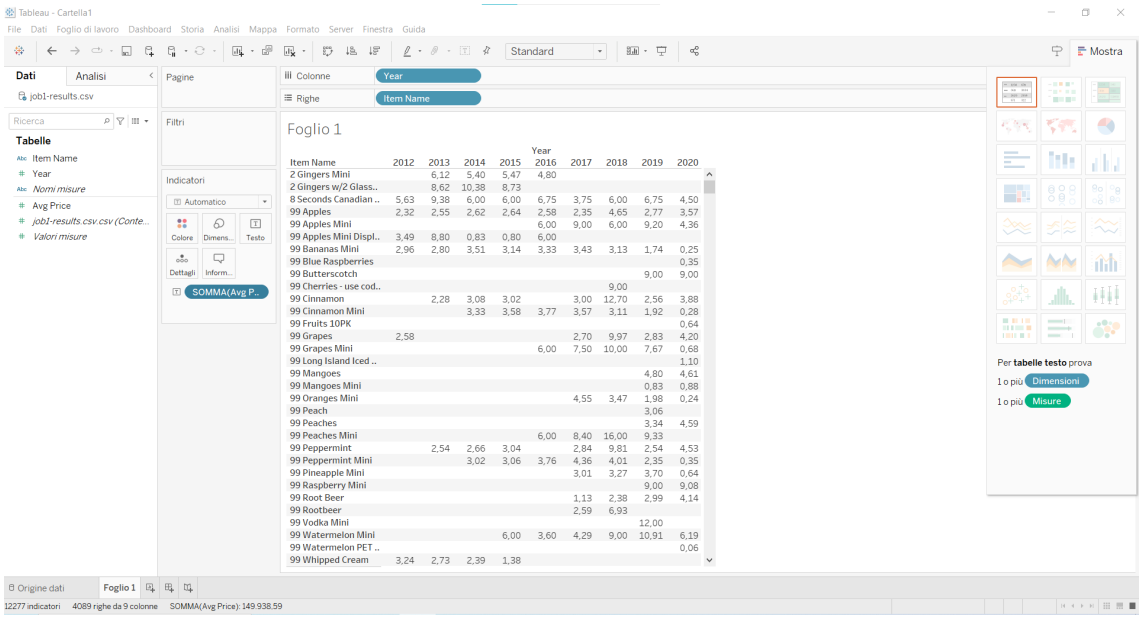
```
val bucketname = "unibo-bigdata-project"
val path_category_revenue = "s3a://"+bucketname+"/risultati/categories_revenue"

val rddItemsCached = rddItems.cache()
val v = rddItemsCached.
        map(e => ((e._2, e._4), (e._3, 1))). //(nameItem, categoryName),(price,1)
        reduceByKey((t1, t2) => (t1._1 + t2._1, t1._2 + t2._2)). //(nameItem,categoryName),(fatturato,unitàVendute)
        map(v => (v._1._1, v._1._2, v._2._1, v._2._2)). //nomeItem, categoryName, fatturato, unitàVendute
        sortBy(_._3, false)


val itemsClassification = v.
    map(e => (e._1, e._3)). // nomeItem, fatturato
    reduceByKey(_ + _).
    map(e => (e._1, e._2 match {
      case f if f >= 1000000 => "Top Tier"
      case f if f < 1000000 && f >= 100000 => "Mid Tier"
      case _ => "Low Tier"
      })) //nomeItem, Tier


val revenueByCategory = v.
    map(p => (p._1, (p._3, p._2))). // nomeItem, (fatturato, categoryName)
    join(itemsClassification). // nomeItem, ((fatturato,categoryName), Tier)
    map(p => ((p._2._1._2, p._2._2), p._2._1._1)). // (categoryName, Tier), fatturato
    aggregateByKey((0.0,0))((agg, v) => (agg._1 + v, agg._2 + 1),((agg1,agg2) => (agg1._1 + agg2._1, agg1._2 + agg2._2))). // (categoryName, Tier), price, count
    map(x => (x._1._1, x._1._2, x._2._1, x._2._2)).// categoryName, Tier, revenue, count
    coalesce(1)

revenueByCategory.map(x => x.productIterator.mkString(",")).saveAsTextFile(path_category_revenue)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
bucketname: String = unibo-bigdata-project
path_category_revenue: String = s3a://unibo-bigdata-project/risultati/categories_revenue
rddItemsCached: rddItems.type = MapPartitionsRDD[5] at flatMap at <console>:26
v: org.apache.spark.rdd.RDD[(String, String, Double, Int)] = MapPartitionsRDD[58] at sortBy at <console>:29
itemsClassification: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[61] at map at <console>:29
revenueByCategory: org.apache.spark.rdd.RDD[(String, String, Double, Int)] = CoalescedRDD[69] at coalesce at <console>:33
```

Figure 10: job2

6 Risultati ottenuti e tableau

Per concludere, abbiamo rappresentato i risultati ottenuti dai due job attraverso Tableau, per permettere una migliore lettura e chiarezza visiva dei dati.



Item Name	2012	2013	2014	2015	Year 2016	2017	2018	2019	2020
2 Gingers Mini		6,12	5,40	5,47	4,80				
2 Gingers w/2 Glass..		8,62	10,38	8,73					
8 Seconds Canadian ..	5,63	9,38	6,00	6,00	6,75	3,75	6,00	6,75	4,50
99 Apples	2,32	2,55	2,62	2,64	2,58	2,35	4,65	2,77	3,57
99 Apples Mini					6,00	9,00	6,00	9,20	4,36
99 Apples Mini Displ..	3,49	8,80	0,83	0,80	6,00				
99 Bananas Mini	2,96	2,80	3,51	3,14	3,33	3,43	3,13	1,74	0,25
99 Blue Raspberries									0,35
99 Butterscotch								9,00	9,00
99 Cherries - use cod..							9,00		
99 Cinnamon		2,28	3,08	3,02		3,00	12,70	2,56	3,88
99 Cinnamon Mini			3,33	3,58	3,77	3,57	3,11	1,92	0,28
99 Fruits 10PK									0,64
99 Grapes	2,58					2,70	9,97	2,83	4,20
99 Grapes Mini					6,00	7,50	10,00	7,67	0,68
99 Long Island Iced ..									1,10
99 Mangoes								4,80	4,61
99 Mangoes Mini								0,83	0,88
99 Oranges Mini						4,55	3,47	1,98	0,24
99 Peach								3,06	
99 Peaches								3,34	4,59
99 Peaches Mini					6,00	8,40	16,00	9,33	
99 Peppermint		2,54	2,66	3,04		2,84	9,81	2,54	4,53
99 Peppermint Mini			3,02	3,06	3,76	4,36	4,01	2,35	0,35
99 Pineapple Mini						3,01	3,27	3,70	0,64
99 Raspberry Mini								9,00	9,08
99 Root Beer						1,13	2,38	2,99	4,14
99 Rootbeer						2,59	6,93		
99 Vodka Mini								12,00	
99 Watermelon Mini				6,00	3,60	4,29	9,00	10,91	6,19
99 Watermelon PET ..									0,06
99 Whipped Cream	3,24	2,73	2,39	1,38					

Figure 11: job 1 results

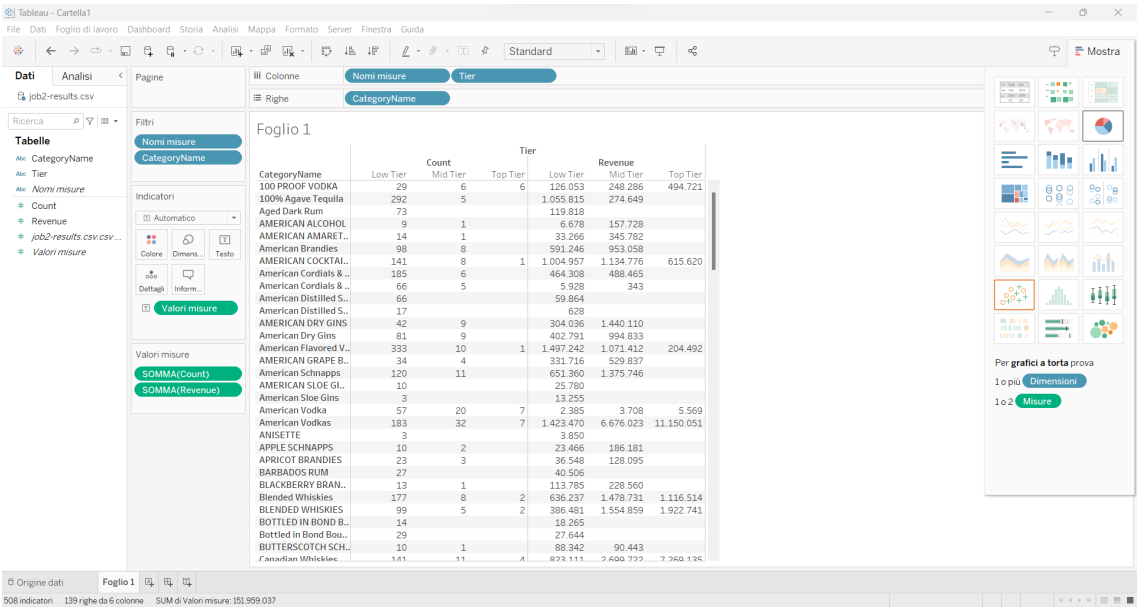


Figure 12: job 2 results