



# PRIMO PROGETTO - BIG DATA

24 maggio 2023

The worker nodes

GIANLUCA DI LORENZO   RAFFAELE SCARANO

MATRICOLA 583630

MATRICOLA 576304

Github repo: [https://github.com/not-Karot/amz\\_review\\_analysis](https://github.com/not-Karot/amz_review_analysis)

## Sommario

<b>Introduzione.....</b>	<b>2</b>
<b>Analisi del dataset / Data Cleaning .....</b>	<b>3</b>
<b>Job 1.....</b>	<b>4</b>
MapReduce .....	5
Pseudocodice .....	5
Execution Time .....	6
Hive .....	7
Execution Time .....	7
Spark Core .....	8
Pseudocodice .....	8
Output .....	8
Execution Time .....	9
SparkSQL .....	10
Output .....	10
Execution Time .....	11
Execution time overall .....	12
<b>Job2 .....</b>	<b>13</b>
MapReduce .....	14
Pseudocodice .....	14
Execution Time .....	15
Hive .....	16
Execution Time .....	17
Spark Core .....	18
Pseudocodice .....	18
Output .....	18
Execution Time .....	19
SparkSQL .....	20
Output .....	20
Execution Time .....	20
Execution time overall .....	21

# Introduzione

In questo rapporto verranno illustrate le procedure di analisi e manipolazione del dataset "Amazon Fine Food Reviews" estratto da Kaggle. Il processo si avvale di quattro tecnologie distinte per l'elaborazione parallela:

- MapReduce
- Hive
- SparkCore
- SparkSQL

Il dataset contiene circa 500.000 recensioni di prodotti gastronomici rilasciati su Amazon dal 1999 al 2012. Il dataset è in formato CSV e ogni riga ha i seguenti campi:

- Id,
- ProductId (unique identifier for the product),
- UserId (unique identifier for the user),
- ProfileName,
- HelpfulnessNumerator (number of users who found the review helpful),
- HelpfulnessDenominator (number of users who graded the review),
- Score (rating between 1 and 5),
- Time (timestamp of the review expressed in Unix time),
- Summary (summary of the review),
- Text (text of the review).

Sono stati progettati e realizzati per ciascuna tecnologia due diversi Job:

1. Un job che sia in grado di generare, per ciascun anno, i 10 prodotti che hanno ricevuto il maggior numero di recensioni e, per ciascuno di essi, le 5 parole con almeno 4 caratteri più frequentemente usate nelle recensioni (campo text), indicando, per ogni parola, il numero di occorrenze della parola.
2. Un job che sia in grado di generare una lista di utenti ordinata sulla base del loro apprezzamento, dove l'apprezzamento di ogni utente è ottenuto dalla media dell'utilità (rapporto tra HelpfulnessNumerator e HelpfulnessDenominator) delle recensioni che hanno scritto, indicando per ogni utente il loro apprezzamento.

L'implementazione delle procedure e il codice sorgente completo, per ogni tecnologia, è consultabile alla repository [amz\\_review\\_analysis](#).

Nel presente resoconto, per maggiore chiarezza e pulizia espositiva, si eviterà il riferimento diretto al codice sorgente, nonostante quest'ultimo sia disponibile.

# Analisi del dataset / Data Cleaning

L'analisi del dataset è stata svolta all'interno del notebook *Data analysis.ipynb* utilizzando Python. Il file csv del dataset è stato caricato in un DataFrame all'interno del notebook, utilizzando la libreria Pandas.

La prima operazione effettuata consiste nel verificare la presenza di valori nulli.

```
In [4]: reviews.isnull().sum()
```

```
Out[4]: Id                0
ProductId               0
UserId                 0
ProfileName             16
HelpfulnessNumerator    0
HelpfulnessDenominator  0
Score                  0
Time                   0
Summary                27
Text                   0
dtype: int64
```

Le colonne *"ProfileName"* e *"Summary"* contenenti valori nulli non sono utili al fine dello svolgimento dei Jobs assegnati, quindi possono essere ignorate.

Analizzando nel dettaglio la richiesta dei Jobs assegnati è possibile osservare che le colonne necessarie per lo svolgimento sono le seguenti:

- Job 1: *ProductID*, *Time* e *Text*
- Job 2: *UserId*, *HelpfulnessNumerator* e *HelpfulnessDenominator*
- Job 3: *ProductId*, *UserId* e *Score*

Per tale motivo sono stati generati tre diversi DataFrame.

Inoltre, analizzando nel dettaglio la colonna *Text* è possibile osservare che all'interno dei singoli record sono presenti segni di punteggiatura e tag HTML che necessitano di essere rimossi. Tale operazione, insieme a quella di conversione del timestamp in anno, verrà effettuata durante la progettazione del Job 1.

```
In [14]: reviews_job1.Text[30]
```

```
Out[14]: "I have never been a huge coffee fan. However, my mother purchased this little machine and talked me into trying the Latte Macciato. No Coffee Shop has a better one and I like most of the other products, too (as a usually non-coffee drinker!).<br />The little Dolche Guesto Machine is super easy to use and prepares a really good Coffee/Latte/Cappuccino/etc in less than a minute (if water is heated up). I would recommend the Dolce Gusto to anyone. To o good for the price and I'am getting one myself! :)"
```

Infine, per la generazione dei dataset di diverse dimensioni, all'interno del notebook *Data Analysis.ipynb* è presente una funzione che genera, a partire dai due DataFrame, tre dataset con dimensioni diverse per ciascun DataFrame.

Nel dettaglio le dimensioni dei dataset generati sono le seguenti:

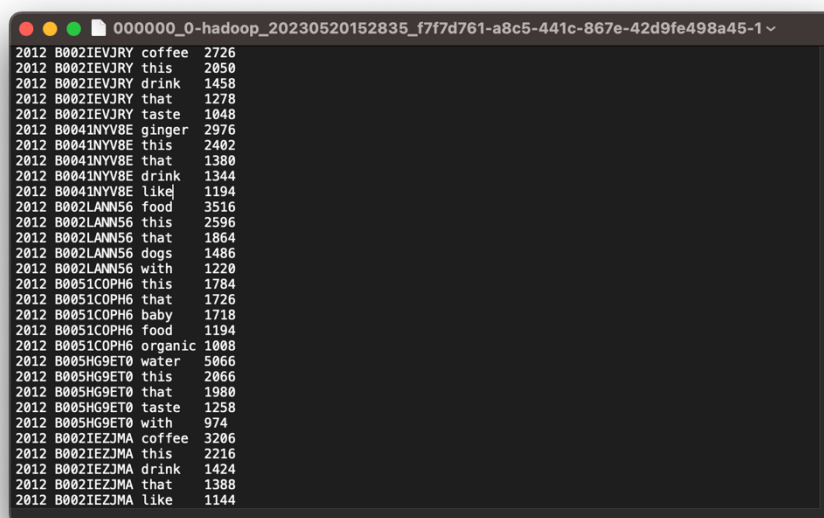
- Metà dataset
- Dimensione reale
- Cinque volte il dataset stesso

Questi tre file csv verranno utilizzati in ogni singolo Job implementato per avere un benchmark dei tempi di esecuzione su dimensione variabile del dataset di input.

## Job 1

L'obiettivo del Job 1 consiste nel generare, per ciascun anno, i 10 prodotti che hanno ricevuto il maggior numero di recensioni e, per ciascuno di essi, le 5 parole con almeno 4 caratteri più frequentemente usate nelle recensioni (campo text), indicando, per ogni parola, il numero di occorrenze della parola. Come anticipato precedentemente, per lo svolgimento del seguente Job sono stati utilizzati i tre dataset generati aventi diverse dimensioni e con i seguenti campi: *ProductID*, *Time* e *Text*.

Le prime righe del file parziale di output del job implementato utilizzando *MapReduce* nativo e *Hive* vengono mostrate di seguito, mentre per i risultati di *SparkCore* e *SparkSQL* si rimanda alla sezione apposita.



```
2012 B002IEVJRY coffee 2726
2012 B002IEVJRY this 2050
2012 B002IEVJRY drink 1458
2012 B002IEVJRY that 1278
2012 B002IEVJRY taste 1048
2012 B0041NYV8E ginger 2076
2012 B0041NYV8E this 2402
2012 B0041NYV8E that 1380
2012 B0041NYV8E drink 1344
2012 B0041NYV8E like 1194
2012 B002LANN56 food 3516
2012 B002LANN56 this 2596
2012 B002LANN56 that 1864
2012 B002LANN56 dogs 1486
2012 B002LANN56 with 1220
2012 B0051COPH6 this 1784
2012 B0051COPH6 that 1726
2012 B0051COPH6 baby 1718
2012 B0051COPH6 food 1194
2012 B0051COPH6 organic 1008
2012 B005HG9ET0 water 5066
2012 B005HG9ET0 this 2066
2012 B005HG9ET0 that 1980
2012 B005HG9ET0 taste 1258
2012 B005HG9ET0 with 974
2012 B002IEZJMA coffee 3206
2012 B002IEZJMA this 2216
2012 B002IEZJMA drink 1424
2012 B002IEZJMA that 1388
2012 B002IEZJMA like 1144
```

# MapReduce

L'implementazione del Job con *MapReduce* è stata svolta creando due script, uno script *Mapper* ed uno script *Reducer*.

## Pseudocodice

Di seguito lo pseudocodice dei due script:

### *Mapper*

*Per ogni riga nell'input letto da stdin:*

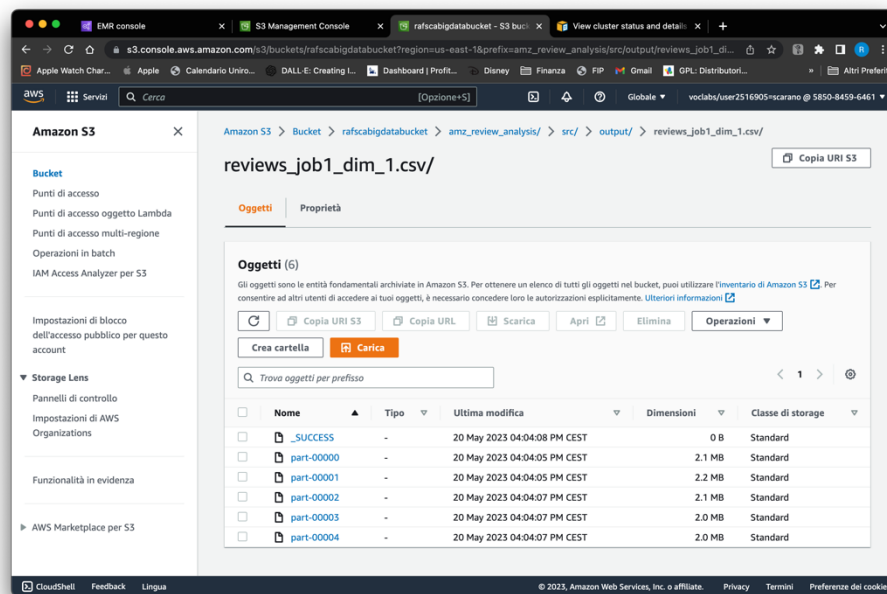
1. *Rimuovere gli spazi bianchi iniziali e finali dalla riga*
2. *Dividere la riga in product\_id, time e text*
3. *Convertire il timestamp time in year*
4. *Ripulire il testo da tag e punteggiatura*
5. *Estrarre le parole di almeno 4 caratteri dal campo text*
6. *Per ogni parola estratta stampare in output (year, product\_id) come chiave e (word,1) come valore*

### *Reducer*

*Per ogni riga nell'input letto da stdin:*

1. *Estrarre i valori e aggiornare il conteggio delle parole e delle recensioni nei rispettivi dizionari.*
2. *Prendere i 10 prodotti più recensiti per ogni anno sommando le occorrenze.*
3. *Per ogni prodotto tra i 10 con il maggior numero di recensioni per ogni anno, ordinare le parole in base al conteggio in ordine decrescente e prendere le prime 5.*
4. *Se il prodotto fa parte dei 10 prodotti più recensiti per quell'anno stampare l'anno, l'id del prodotto, la parola e il conteggio.*

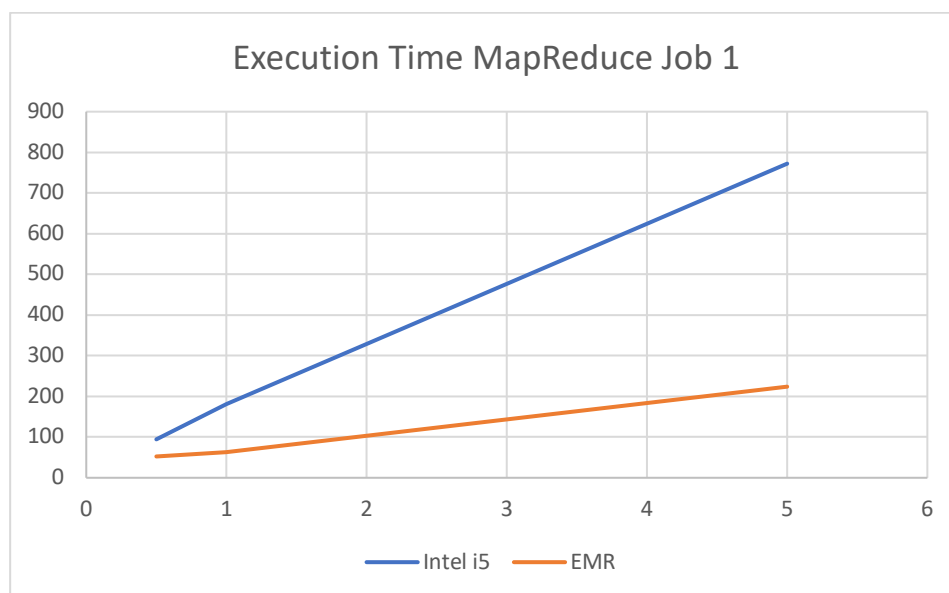
Il job *MapReduce* è stato anche eseguito su un cluster *EMR* di *AWS*. Di seguito uno screenshot mostrante la cartella di output.



## Execution Time

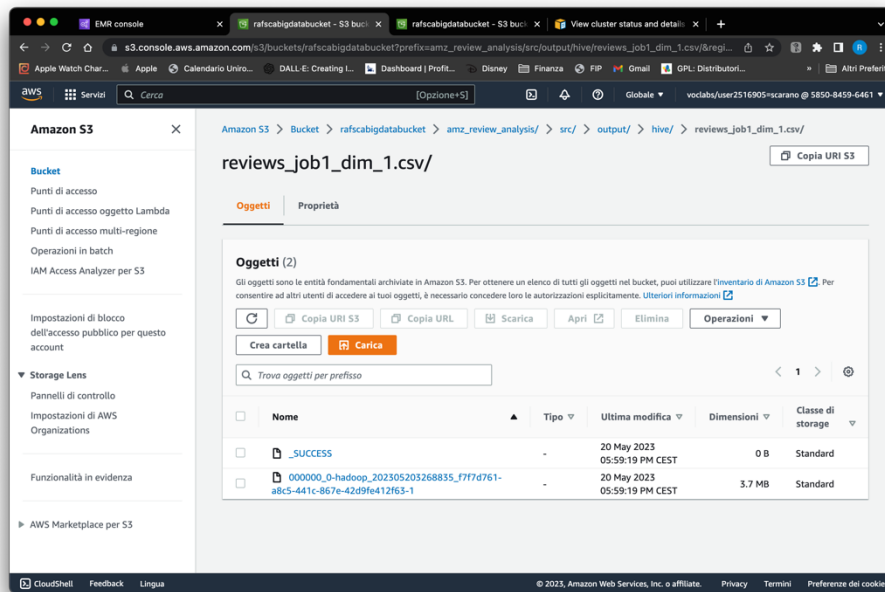
Di seguito un grafico che mostra i tempi di esecuzione del Job 1 utilizzando *MapReduce* su diverse infrastrutture hardware.

Per i tempi precisi si rimanda al grafico nella sezione conclusiva del job.



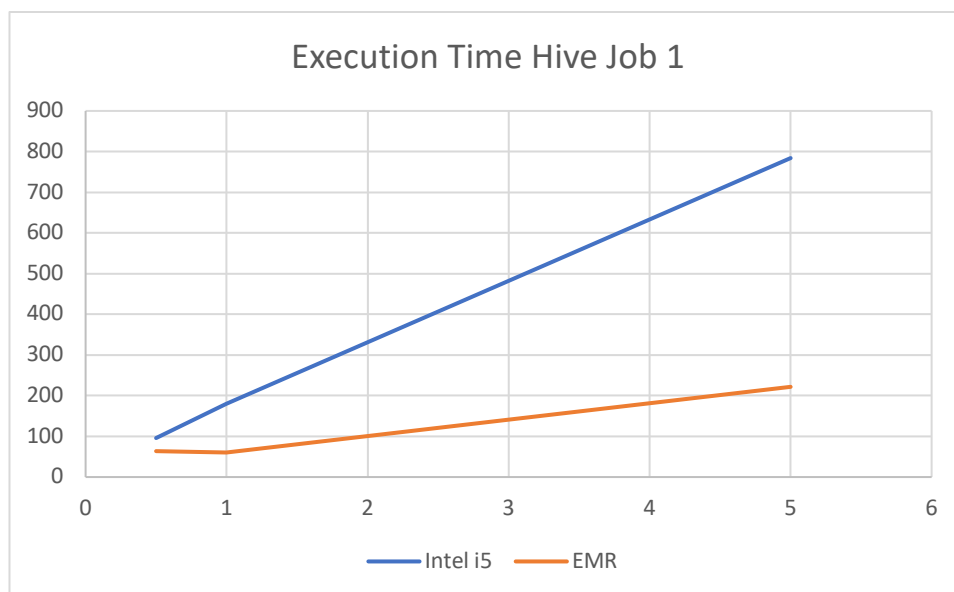
# Hive

Il job, nella sua implementazione *Hive*, è stato eseguito anche sul cluster di *EMR*. Di seguito lo screenshot mostrante la cartella di output generata.



## Execution Time

Di seguito un grafico che mostra i tempi di esecuzione del Job 1 utilizzando MapReduce su diverse infrastrutture hardware.





# Spark Core

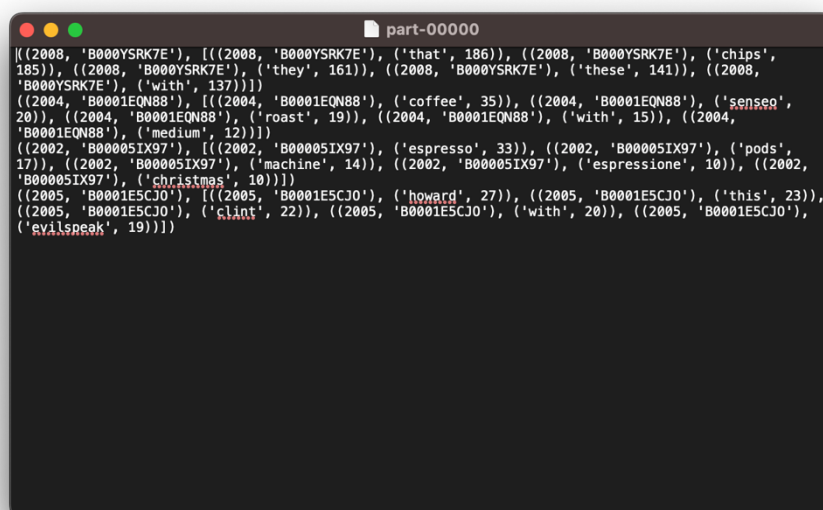
L'implementazione del Job con Spark Core è stata svolta creando uno script chiamato "core.py".

## Pseudocodice

Di seguito lo pseudocodice dello script:

1. *Caricare i dati in un RDD*
2. *Estrarre i campi*
3. *Convertire timestamp in anno*
4. *Contare le parole con lenght()>=4 per prodotto per anno*
5. *Contare recensioni per prodotto per anno*
6. *Ordinare e prendere i top10 prodotti per anno*
7. *Prendere le parole solo per i top prodotti*
8. *Ordinare e prendere le top 5 parole per ogni prodotto e per anno*
9. *Salvare i risultati su un txt*

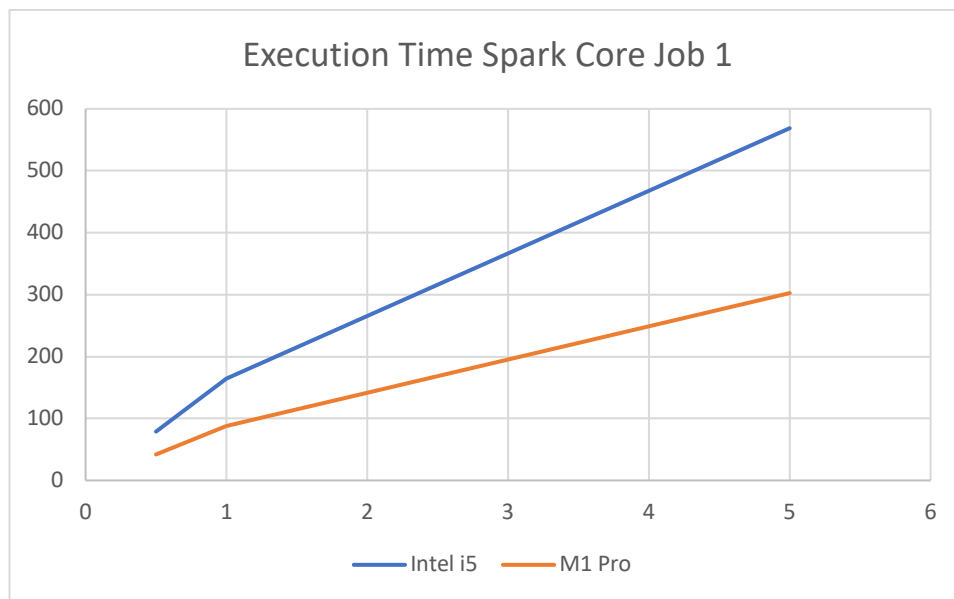
## Output



```
part-00000
[(2008, 'B000YSRK7E'), [(2008, 'B000YSRK7E'), ('that', 186)), (2008, 'B000YSRK7E'), ('chips',
185)), (2008, 'B000YSRK7E'), ('they', 161)), (2008, 'B000YSRK7E'), ('these', 141)), (2008,
'B000YSRK7E'), ('with', 137)]]
((2004, 'B0001EQN88'), [(2004, 'B0001EQN88'), ('coffee', 35)), (2004, 'B0001EQN88'), ('senseo',
20)), (2004, 'B0001EQN88'), ('roast', 19)), (2004, 'B0001EQN88'), ('with', 15)), (2004,
'B0001EQN88'), ('medium', 12)]]
((2002, 'B00005IX97'), [(2002, 'B00005IX97'), ('espresso', 33)), (2002, 'B00005IX97'), ('pods',
17)), (2002, 'B00005IX97'), ('machine', 14)), (2002, 'B00005IX97'), ('espressione', 10)), (2002,
'B00005IX97'), ('christmas', 10)]]
((2005, 'B0001E5CJ0'), [(2005, 'B0001E5CJ0'), ('howard', 27)), (2005, 'B0001E5CJ0'), ('this', 23)),
(2005, 'B0001E5CJ0'), ('clint', 22)), (2005, 'B0001E5CJ0'), ('with', 20)), (2005, 'B0001E5CJ0'),
('evilspeak', 19)]]
```

## Execution Time

Di seguito un grafico che mostra i tempi di esecuzione del Job 1 utilizzando SparkCore su diverse infrastrutture hardware, nello specifico due diversi laptop Apple con processori di natura diversa, non sono stati eseguiti calcoli sul cluster a causa di problemi di compatibilità riscontrati con le procedure sviluppate.

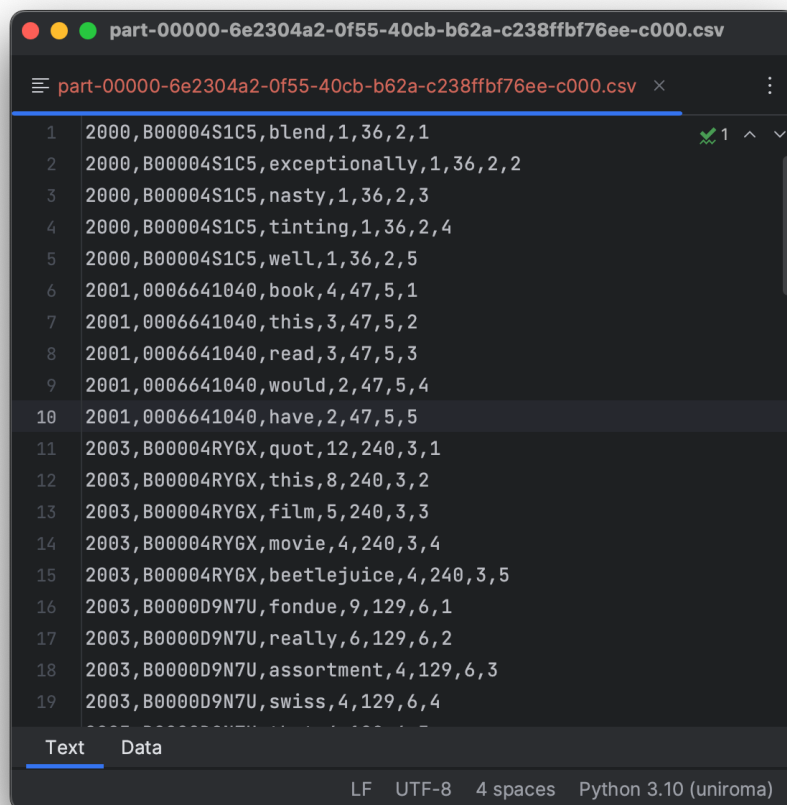


## SparkSQL

L'implementazione del codice Spark è stata riformulata sfruttando l'interfaccia del dataframe fornita direttamente da Spark.

## Output

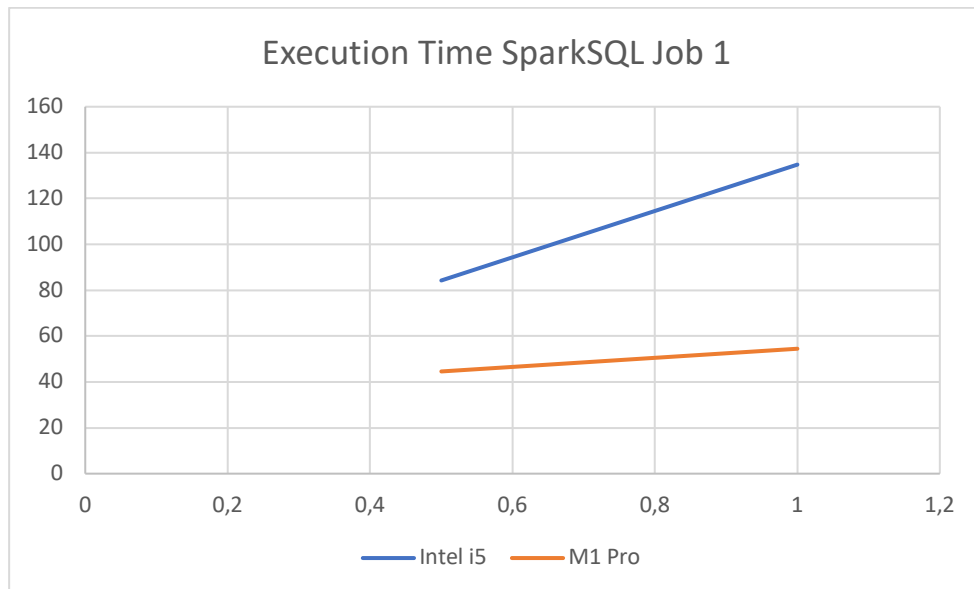
Di seguito la parte 0000 dell'output prodotto.



```
part-00000-6e2304a2-0f55-40cb-b62a-c238ffbf76ee-c000.csv
1 2000,B00004S1C5,blend,1,36,2,1
2 2000,B00004S1C5,exceptionally,1,36,2,2
3 2000,B00004S1C5,nasty,1,36,2,3
4 2000,B00004S1C5,tinting,1,36,2,4
5 2000,B00004S1C5,well,1,36,2,5
6 2001,0006641040,book,4,47,5,1
7 2001,0006641040,this,3,47,5,2
8 2001,0006641040,read,3,47,5,3
9 2001,0006641040,would,2,47,5,4
10 2001,0006641040,have,2,47,5,5
11 2003,B00004RYGX,quot,12,240,3,1
12 2003,B00004RYGX,this,8,240,3,2
13 2003,B00004RYGX,film,5,240,3,3
14 2003,B00004RYGX,movie,4,240,3,4
15 2003,B00004RYGX,beetlejuice,4,240,3,5
16 2003,B0000D9N7U,fondue,9,129,6,1
17 2003,B0000D9N7U,really,6,129,6,2
18 2003,B0000D9N7U,assortment,4,129,6,3
19 2003,B0000D9N7U,swiss,4,129,6,4
Text Data
LF UTF-8 4 spaces Python 3.10 (uniroma)
```

## Execution Time

Di seguito un grafico che mostra i tempi di esecuzione del Job 1 utilizzando SparkSQL su diverse infrastrutture hardware, nello specifico due diversi laptop Apple con processori di natura diversa, non sono stati eseguiti calcoli sul cluster a causa di problemi di compatibilità riscontrati con le procedure sviluppate.



Entrambi i PC non sono riusciti a completare l'esecuzione del dataset di dimensione 5. L'errore sembra legato al tempo di vita del contesto Spark, come mostrato nelle eccezioni riportate di seguito.

: org.apache.spark.SparkException: Job aborted.

Caused by: org.apache.spark.SparkException: Job 2 cancelled because SparkContext was shut down

## Execution time overall

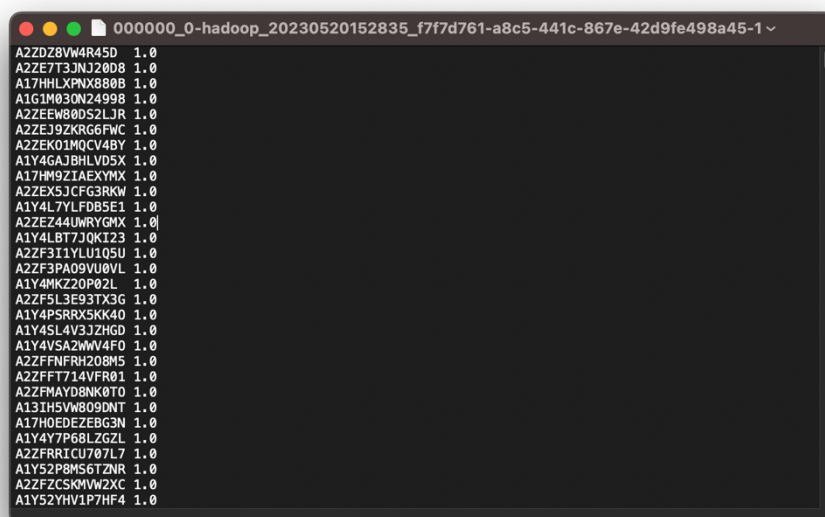


## Job2

L'obiettivo del Job 2 consiste nel generare una lista di utenti ordinata sulla base del loro apprezzamento, dove l'apprezzamento di ogni utente è ottenuto dalla media dell'utilità (rapporto tra *HelpfulnessNumerator* e *HelpfulnessDenominator*) delle recensioni che hanno scritto, indicando per ogni utente il loro apprezzamento.

Come anticipato precedentemente, per lo svolgimento del seguente Job sono stati utilizzati i tre dataset generati aventi diverse dimensioni e con i seguenti campi: *UserId*, *HelpfulnessNumerator* e *HelpfulnessDenominator*.

Le prime righe del file parziale di output del job implementato utilizzando *MapReduce* nativo e *Hive* vengono mostrate di seguito, mentre per i risultati di *Spark* e *SparkSQL* si rimanda alla sezione apposita.



```
000000_0-hadoop_20230520152835_f7f7d761-a8c5-441c-867e-42d9fe498a45-1
A2ZDZ8VW4R45D 1.0
A2ZE7T3JNJ2008 1.0
A17HHLXPNX880B 1.0
A1G1M030N24998 1.0
A2ZEEW80S2LJR 1.0
A2ZE30ZKR6FWC 1.0
A2ZEK01M0CV4BY 1.0
A1Y4GAJBHLVD5X 1.0
A17HM9ZIAEXYMX 1.0
A2ZEX5JCFG3RKW 1.0
A1Y4L7YLFDB5E1 1.0
A2ZEZ44UWRYGMX 1.0
A1Y4LBT7JQKI23 1.0
A2ZF3I1YLU105U 1.0
A2ZF3PA09VU0VL 1.0
A1Y4MKZ20P02L 1.0
A2ZF5L3E93TX3G 1.0
A1Y4PSRRXS5KK40 1.0
A1Y4SL4V3JZHGD 1.0
A1Y4VSA2WV4F0 1.0
A2ZFFNFRH208M5 1.0
A2ZFFT714VFR01 1.0
A2ZFMAYD8NK0T0 1.0
A13IH5VW809DNT 1.0
A17H0EDEZE8G3N 1.0
A1Y4Y7P68LZGZL 1.0
A2ZFRRI CU707L7 1.0
A1Y52P8MS6T2NR 1.0
A2ZFC5K0VW2XC 1.0
A1Y52YHV1P7HF4 1.0
```

# MapReduce

L'implementazione del Job con *MapReduce* è stata svolta creando due script, uno script *Mapper* ed uno script *Reducer*.

## Pseudocodice

Di seguito lo pseudocodice dei due script:

### *Mapper*

*Per ogni riga nell'input letto da stdin:*

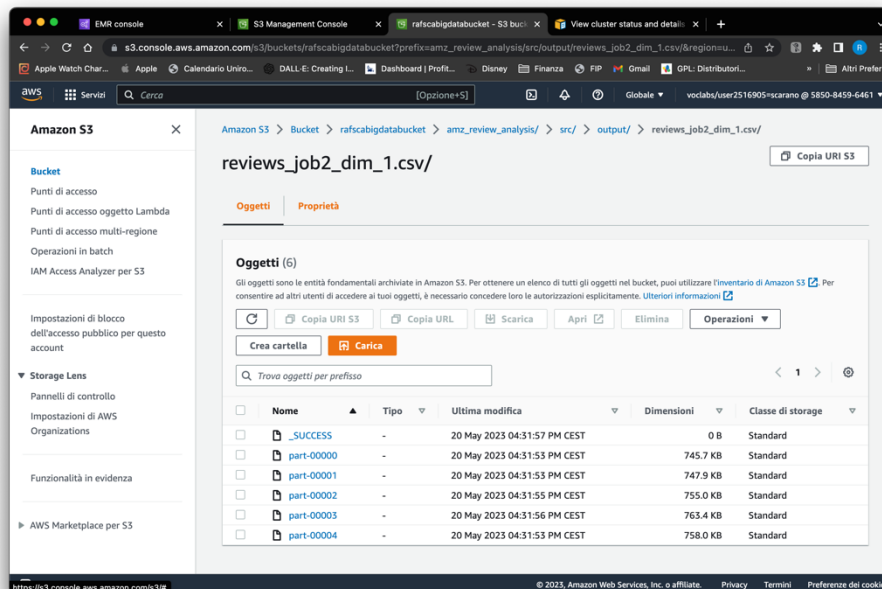
1. *Estrarre i campi*
2. *Calcolare l'utilità dividendo il numeratore per il denominatore*
3. *Stampare user\_id come chiave e utility come valore*

### *Reducer*

*Per ogni riga nell'input letto da stdin:*

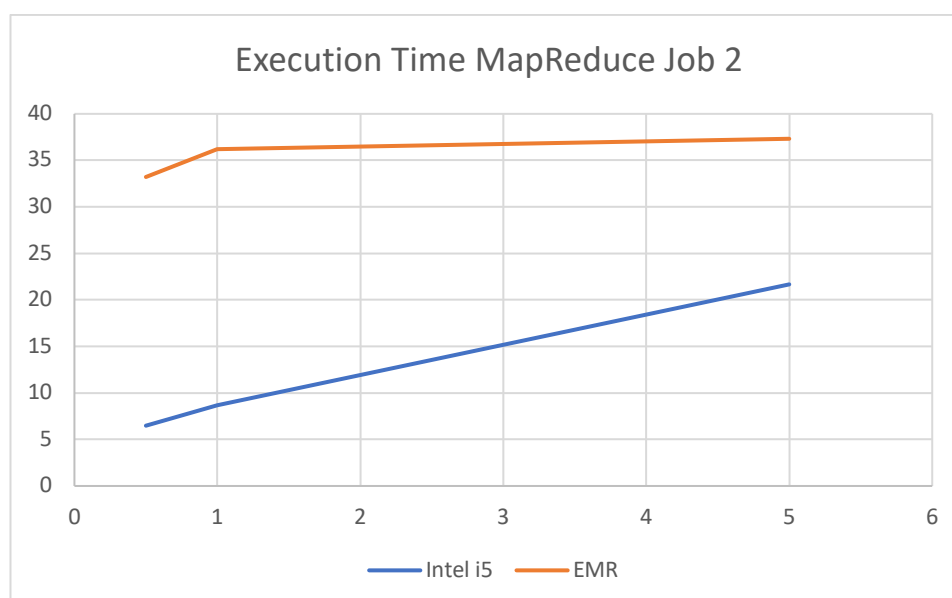
1. *Estrarre user\_id e utility dall'input*
2. *Salvare in un dizionario il conteggio delle recensioni e la relativa somma di utility*
3. *Calcolare la media dell'utility (apprezzamento) per ogni utente*
4. *Ordinare gli utenti con media più alta in ordine decrescente*
5. *Stampare i risultati*

Il Job *MapReduce* è stato anche eseguito su un cluster *EMR* di *AWS*. Di seguito uno screenshot mostrante la cartella di output.



## Execution Time

Di seguito un grafico che mostra i tempi di esecuzione del Job 2 utilizzando *MapReduce* su diverse infrastrutture hardware.



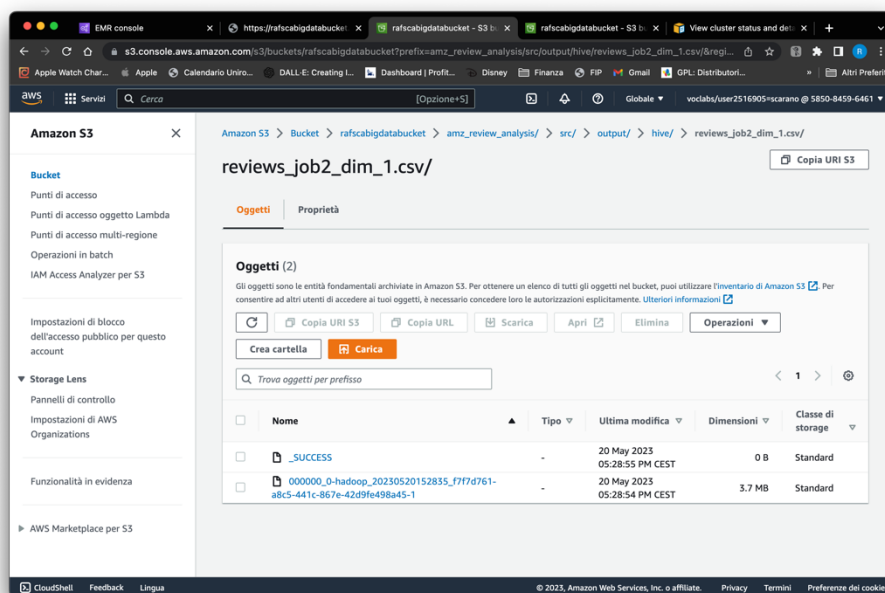


Il laptop locale risulta più veloce del cluster *EMR*, una possibile spiegazione può essere la latenza di trasferimento rete causata dalla posizione fisica del dataset, che, vista la semplicità computazionale del job, risulta prevalente rispetto al resto. Sul cluster il dataset è stato caricato su un object storage S3, mentre sulla macchina locale il dataset era sul SSD interno; non ci è chiaro se questa scelta architetturale può essere la causa dei tempi ottenuti.

## Hive

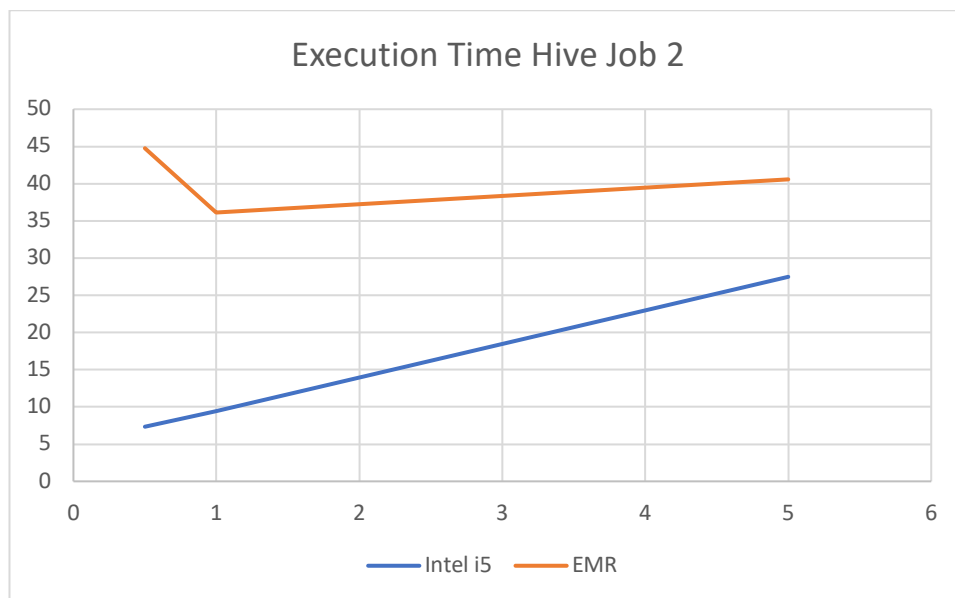
Il job, nella sua implementazione Hive, è stato eseguito sul cluster di EMR.

Di seguito lo screenshot mostrante la cartella di output generata.



## Execution Time

Di seguito un grafico che mostra i tempi di esecuzione del Job 2 utilizzando *Hive* su diverse infrastrutture hardware.



Anche in questo caso, probabilmente per gli stessi motivi individuati nell'esecuzione di MapReduce, il laptop locale risulta più veloce del cluster *EMR*.

## Spark Core

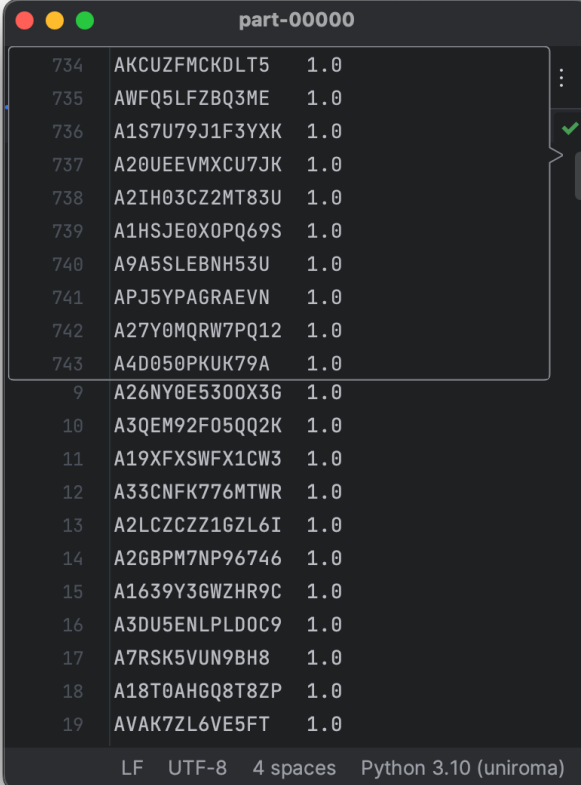
L'implementazione del Job con Spark Core è stata svolta creando uno script chiamato "core.py".

## Pseudocodice

Di seguito lo pseudocodice dello script:

1. *Caricare i dati in un RDD*
2. *Estrarre i campi*
3. *Calcolare l'utilità per ogni recensione con numeratore/denominatore*
4. *Raggruppare utilità per utente*
5. *Calcolare l'apprezzamento (media delle utilità)*
6. *Ordinare gli utenti per apprezzamento decrescente*
7. *Salvare i risultati su un txt*

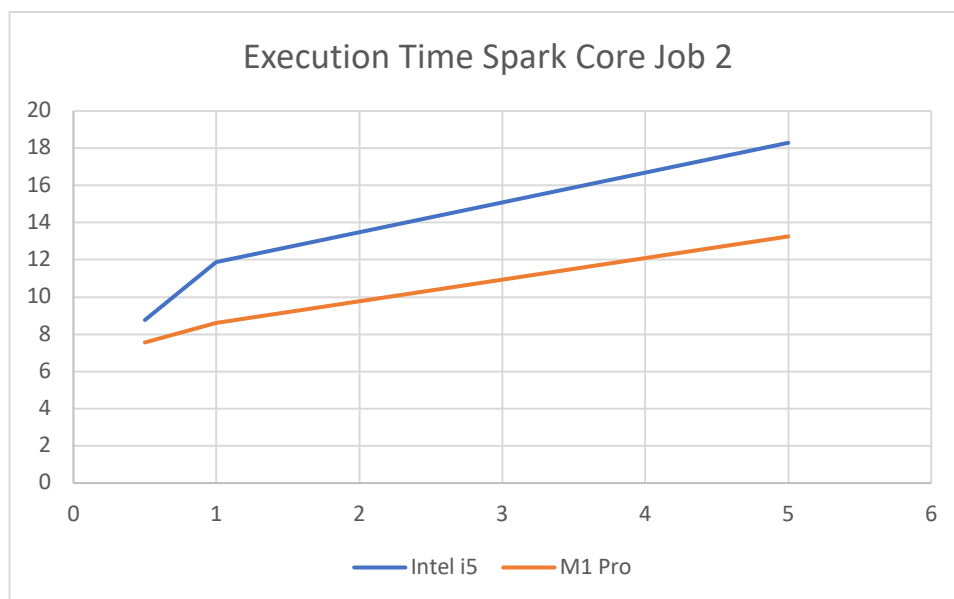
## Output



```
part-00000
734 AKCUZFMCKDLT5 1.0
735 AWFQ5LFZBQ3ME 1.0
736 A1S7U79J1F3YXK 1.0
737 A20UEEVVMXCU7JK 1.0
738 A2IH03CZ2MT83U 1.0
739 A1HSJE0X0PQ69S 1.0
740 A9A5SLEBNH53U 1.0
741 APJ5YPAGRAEVN 1.0
742 A27Y0MRW7PQ12 1.0
743 A4D050PKUK79A 1.0
9 A26NY0E5300X3G 1.0
10 A3QEM92F05QQ2K 1.0
11 A19XFXSWFX1CW3 1.0
12 A33CNFK776MTWR 1.0
13 A2LCZCZZ16ZL6I 1.0
14 A2GBPM7NP96746 1.0
15 A1639Y3GWZHR9C 1.0
16 A3DU5ENLPLDOC9 1.0
17 A7RSK5VUN9BH8 1.0
18 A18T0AHGQ8T8ZP 1.0
19 AVAK7ZL6VE5FT 1.0
LF UTF-8 4 spaces Python 3.10 (uniroma)
```

## Execution Time

Di seguito un grafico che mostra i tempi di esecuzione del Job 2 utilizzando *SparkCore* su diverse infrastrutture hardware, nello specifico due diversi laptop Apple con processori di natura diversa, non sono stati eseguiti calcoli sul cluster a causa di problemi di compatibilità riscontrati con le procedure sviluppate.

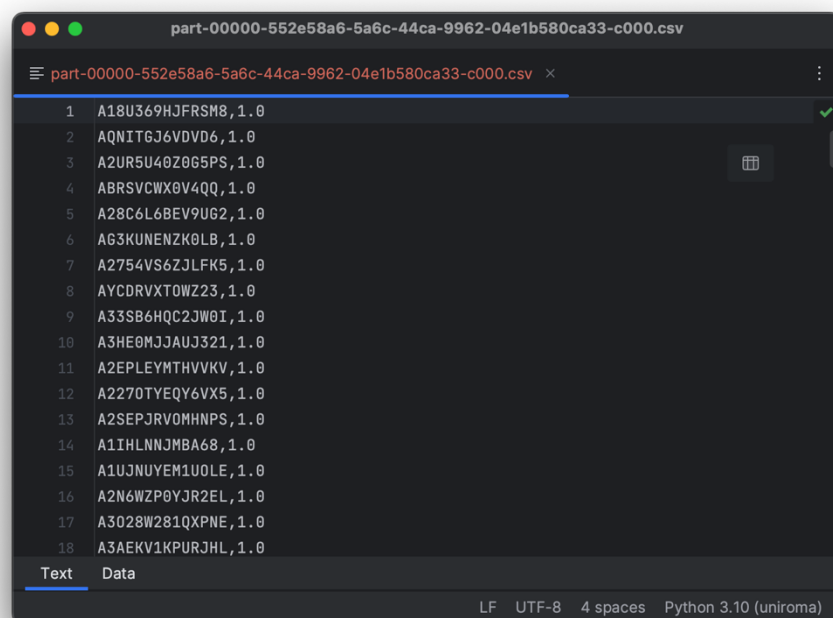


## SparkSQL

L'implementazione del codice Spark è stata riformulata sfruttando l'interfaccia del dataframe fornita direttamente da Spark.

## Output

Di seguito la parte 0000 dell'output prodotto

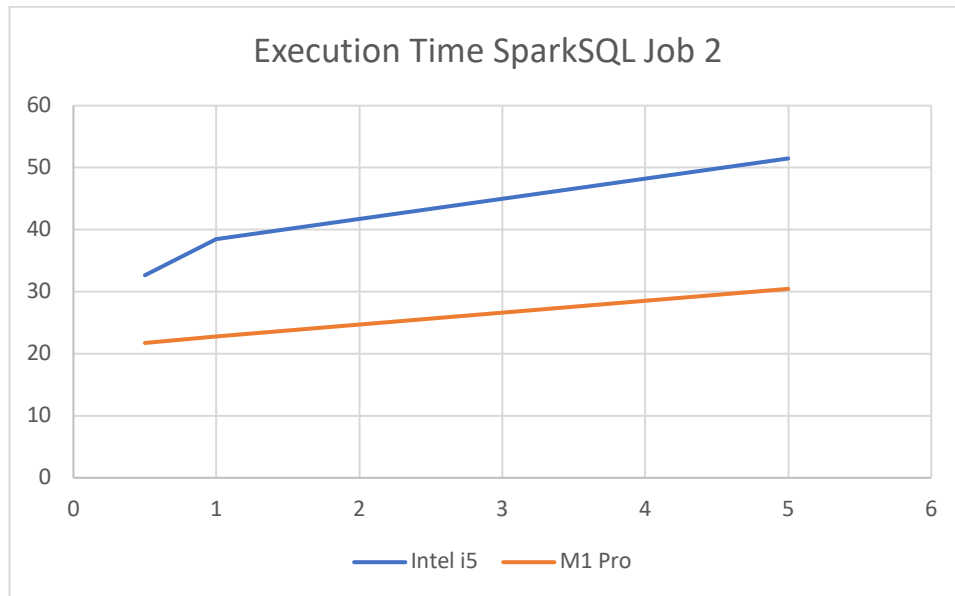


The screenshot shows a window titled 'part-00000-552e58a6-5a6c-44ca-9962-04e1b580ca33-c000.csv'. The window displays a list of 18 rows, each containing an alphanumeric string followed by ',1.0'. The strings are: A18U369HJFRSM8, AQNITGJ6VDVD6, A2UR5U40Z0G5PS, ABRVVCWX0V4QQ, A28C6L6BEV9UG2, AG3KUNENZK0LB, A2754VS6ZJLFK5, AYCDRVXTOWZ23, A33SB6HQC2JW0I, A3HE0MJJAUI321, A2EPLEYMTHVVKV, A2270TYEQY6VX5, A2SEPJRVOHNP5, A1IHLNNJMB68, A1UJNUYEM1UOLE, A2N6WZP0YJR2EL, A3028W281QXPNE, and A3AEKV1KPURJHL. The window has a 'Text' tab selected and a status bar at the bottom indicating 'LF UTF-8 4 spaces Python 3.10 (uniroma)'.

Line	Content
1	A18U369HJFRSM8,1.0
2	AQNITGJ6VDVD6,1.0
3	A2UR5U40Z0G5PS,1.0
4	ABRSVCWX0V4QQ,1.0
5	A28C6L6BEV9UG2,1.0
6	AG3KUNENZK0LB,1.0
7	A2754VS6ZJLFK5,1.0
8	AYCDRVXTOWZ23,1.0
9	A33SB6HQC2JW0I,1.0
10	A3HE0MJJAUI321,1.0
11	A2EPLEYMTHVVKV,1.0
12	A2270TYEQY6VX5,1.0
13	A2SEPJRVOHNP5,1.0
14	A1IHLNNJMB68,1.0
15	A1UJNUYEM1UOLE,1.0
16	A2N6WZP0YJR2EL,1.0
17	A3028W281QXPNE,1.0
18	A3AEKV1KPURJHL,1.0

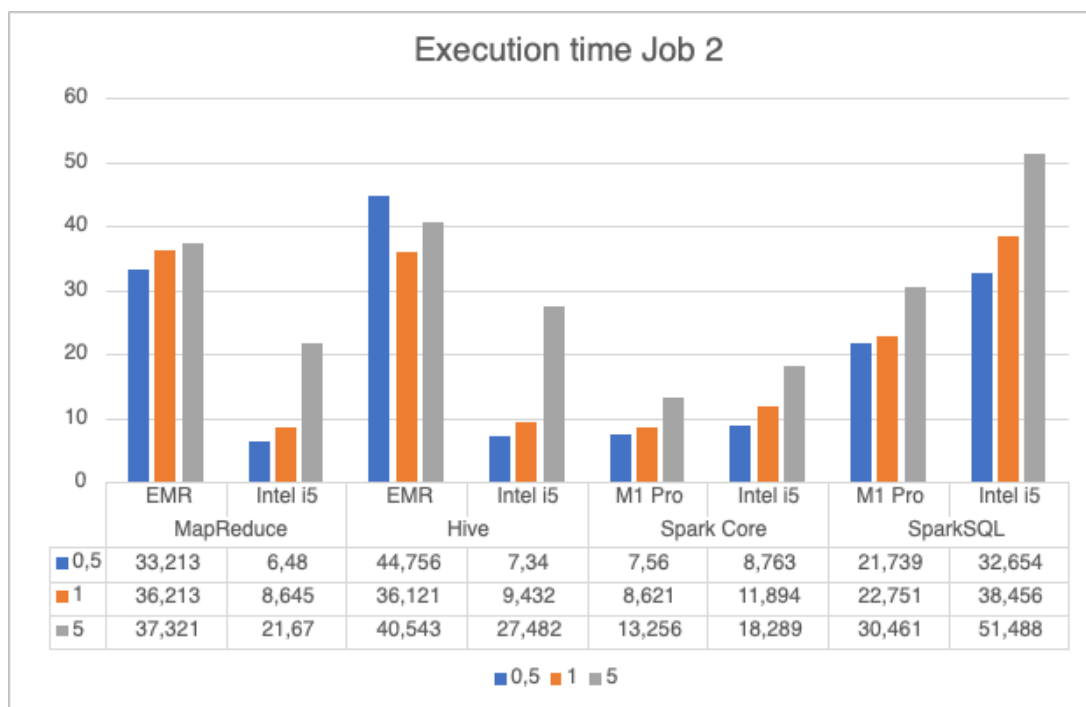
## Execution Time

Di seguito un grafico che mostra i tempi di esecuzione del Job 2 utilizzando SparkSQL su diverse infrastrutture hardware, nello specifico due diversi laptop Apple con processori di natura diversa, non sono stati eseguiti calcoli sul cluster a causa di problemi di compatibilità riscontrati con le procedure sviluppate.



## Execution time overall

Di seguito un grafico che mostra i tempi di esecuzione per ogni tecnologia al crescere dell'input e su diverse infrastrutture hardware.



Un risultato sorprendente del nostro test riguarda il tempo di esecuzione tra SparkCore e MapReduce. Contrariamente a quanto ci si aspetterebbe, SparkCore si è dimostrato più lento rispetto a MapReduce quando testato su un processore Intel i5 o addirittura rispetto al processore M1 Pro.

Ci vengono in mente alcune ipotesi che potrebbero spiegare questa discrepanza, soprattutto se si considera la semplicità del job testato. Una possibile spiegazione potrebbe essere l'overhead associato alla gestione di Spark, che deve coordinare molteplici componenti e può risultare più complesso rispetto a MapReduce. Un'altra possibile spiegazione è che l'algoritmo implementato in SparkCore potrebbe non essere stato ottimizzato come l'equivalente in MapReduce, risultando così in una minore efficienza.