Relazione del Progetto HistoricalStock

Alessio Marinucci & Riccardo Felici
Link repository GitHub

Indice

Introduzione

Negli ultimi decenni, l'analisi dei dati finanziari ha assunto un ruolo centrale nella gestione del rischio e nella formulazione di strategie di investimento. La disponibilità di grandi quantità di dati storici sui prezzi delle azioni offre opportunità uniche per l'analisi approfondita dei mercati finanziari, l'individuazione di trend e pattern, e la previsione delle dinamiche future. In questo contesto, il dataset "Daily Historical Stock Prices" rappresenta una risorsa preziosa per analizzare l'andamento giornaliero delle azioni su due delle principali borse valori degli Stati Uniti: la borsa di New York (NYSE) e il NASDAQ.

Il dataset, scaricabile dal sito Kaggle, copre un arco temporale che va dal 1970 al 2018, fornendo un'ampia gamma di dati storici per una vasta selezione di azioni. Il dataset è composto da due file CSV principali:

1. **historical_stocks.csv**: Questo file contiene informazioni statiche sulle aziende, come il simbolo del ticker, il nome dell'azienda, e altre informazioni identificative.
2. **historical_stock_prices.csv**: Questo file contiene i prezzi giornalieri delle azioni, inclusi i prezzi di apertura, chiusura, minimo, massimo e il volume di scambio.

L'obiettivo principale di questo lavoro è analizzare e pulire il dataset per rimuovere dati inconsistenti, valori mancanti, duplicati e outliers. La pulizia dei dati è un passo essenziale che precede qualsiasi analisi avanzata, poiché garantisce che i dati siano accurati e pronti per essere utilizzati in modelli predittivi o altre forme di analisi quantitativa.

Importanza della Pulizia dei Dati

La pulizia dei dati è una fase critica nel processo di data analysis. Dati non puliti possono portare a risultati fuorvianti e decisioni errate. Alcuni degli obiettivi specifici della pulizia dei dati includono:

- **Rimozione dei Valori Nulli o Mancanti**: Dati incompleti possono distorcere l'analisi e compromettere la qualità dei risultati.
- **Filtraggio dei Valori Negativi**: Prezzi e volumi di scambio negativi non sono realistici e devono essere eliminati.

- **Gestione degli Outliers**: Gli outliers possono influenzare negativamente le analisi statistiche e i modelli predittivi.
- **Rimozione dei Duplicati**: I dati duplicati possono causare un sovraccarico di informazioni e influenzare la precisione delle analisi.
- **Verifica delle Date**: Assicurarsi che le date rientrino nell'intervallo specificato garantisce che i dati siano rilevanti per l'analisi storica.

Struttura della Relazione

Il seguito dell'elaborato è suddiviso in due componenti principali:

1. **Pulizia del Dataset Iniziale**: Descrizione dettagliata delle operazioni di pulizia effettuate sui due file CSV utilizzando due script Python. Questa sezione include una spiegazione del codice e delle azioni intraprese per garantire la qualità dei dati.
2. **Analisi e Risultati**: Dopo la pulizia dei dati, verranno presentate le analisi effettuate e i risultati ottenuti, utilizzando le seguenti tecnologie: *MapReduce, SparkSQL e SparkCore* eseguite in locale. Successivamente verrà presentata l'analisi e i risultati ottenuti utilizzando un'infrastruttura cloud AWS per l'esecuzione MapReduce. Questa parte della relazione dimostrerà come i dati puliti possono essere utilizzati per ottenere insights significativi sui mercati finanziari. Si è deciso di progettare e implementare l'applicazione per il job1 e per il job3, svolti entrambi usando le tecnologie citate sopra. Verranno quindi mostrate le tuple ottenute dall'esecuzione e i risultati in termini di performance (tempi di esecuzione) per ogni job con la relativa tecnologia.

Nella sezione successiva, viene descritta in dettaglio la prima fase della relazione, che comprende la pulizia del dataset iniziale utilizzando due script Python.

Pulizia del Dataset

Prima Fase: Pulizia del File "*historical_stock_prices.csv*"

Di seguito è riportato il codice *Python* utilizzato per la pulizia del dataset *`historical_stock_prices.csv`*:

```python
import pandas as pd
df = pd.read_csv('historical_stock_prices.csv')

# Rimuovere valori nulli o mancanti
df = df.dropna()

# Filtra i valori con prezzi negativi e con volumi di scambio negativi
df = df[df['close'] > 0]
df = df[df['volume'] > 0]

# gestione outliers
from scipy import stats
import numpy as np
df = df[(np.abs(stats.zscore(df['close'])) < 3)]

# Rimuove duplicati
df = df.drop_duplicates()

# Rimuove righe con date fuori intervallo
df['date'] = pd.to_datetime(df['date'])
df = df[(df['date'] >= '1970-01-01') & (df['date'] <= '2018-12-31')]

# Salva il DataFrame pulito in un nuovo file CSV
df.to_csv('historical_stock_prices_pulito.csv', index=False)
```

Spiegazione dello Script 1

1. **Caricamento dei Dati**: Si usa *`pandas`* per leggere il file CSV contenente i prezzi storici delle azioni.
2. **Rimozione dei Valori Nulli o Mancanti**: Vengono eliminate le righe che contengono valori nulli per assicurare che tutte le informazioni siano complete.
3. **Filtraggio dei Valori Negativi**: Si rimuovono le righe con prezzi di chiusura e volumi di scambio negativi, poiché non sono valori realistici.
4. **Gestione degli Outliers**: Si usa lo Z-score per identificare e rimuovere gli outliers nei prezzi di chiusura.

5. **Rimozione dei Duplicati**: Sono eliminate le righe duplicate per evitare ridondanze.
6. **Rimozione delle Righe con Date Fuori Intervallo**: Si filtrano le righe per mantenere solo le date comprese tra il 1970 e il 2018.
7. **Salvataggio del DataFrame Pulito**: Il DataFrame pulito è salvato in un nuovo file CSV per l'analisi successiva.

Seconda Fase: Pulizia del File "*historical_stocks.csv*"

Di seguito è riportato il codice Python utilizzato per la pulizia del dataset `historical_stocks.csv`:

```
import pandas as pd
df = pd.read_csv('historical_stocks.csv')

# Rimuove i valori nulli sulle colonne di interesse
df = df.dropna(subset=['ticker', 'name'])

# Rimuove i duplicati
df = df.drop_duplicates()

# verifica che i simboli e i nomi delle aziende siano validi
df = df[df['ticker'].str.match(r'^[A-Z]+$')]
df = df[df['name'].str.len() > 0]

# Verifica che le stringhe siano in maiuscolo
df['ticker'] = df['ticker'].str.upper()
df['name'] = df['name'].str.upper()

df.to_csv('historical_stock_pulito.csv', index=False)
```

Spiegazione dello Script 2

1. **Caricamento dei Dati**: Si utilizza `pandas` per leggere il file CSV contenente le informazioni delle aziende.
2. **Rimozione dei Valori Nulli**: Vengono eliminate le righe che contengono valori nulli nelle colonne 'ticker' e 'name'.
3. **Rimozione dei Duplicati**: Vengono cancellate le righe duplicate per evitare ridondanze.

4. **Verifica dei Simboli e dei Nomi**: Si filtrano i dati per mantenere solo i simboli azionari validi (solo lettere maiuscole) e i nomi delle aziende con una lunghezza maggiore di zero.
5. **Normalizzazione delle Stringhe**: I simboli e i nomi delle aziende vengono convertiti in maiuscolo per una maggiore coerenza.
6. **Salvataggio del DataFrame Pulito**: Il DataFrame pulito è salvato in un nuovo file CSV per l'analisi successiva.

La pulizia del dataset "Daily Historical Stock Prices" è stata fondamentale per preparare i dati a un'analisi accurata e affidabile. I passaggi descritti hanno contribuito a rimuovere dati inconsistenti e a garantire che le informazioni fossero pronte per l'analisi successiva. Questo processo ha migliorato la qualità del dataset, riducendo il rischio di errori nelle fasi successive del progetto.

MapReduce - Esecuzione Job 1

Introduzione a MapReduce

MapReduce è un modello di programmazione che consente di elaborare grandi volumi di dati in parallelo su un cluster di macchine. Il modello divide il lavoro in due fasi principali: **Map** e **Reduce**.

- **Mapper**: La funzione mapper elabora l'input e genera coppie chiave-valore.
- **Reducer**: La funzione reducer prende l'output dei mapper, raggruppato per chiave, e produce l'output finale.

Mapper

Il file mapper legge l'input dal dataset e genera coppie chiave-valore. Nel contesto del dataset sui prezzi storici delle azioni, il mapper elabora ogni riga del file CSV, estraendo il ticker, il nome dell'azienda, l'anno, la data, il prezzo di chiusura, il prezzo minimo, il prezzo massimo e il volume di scambio. Emette quindi una coppia chiave-valore del tipo (ticker, anno) e i relativi dettagli.

E’ riportato sotto un possibile pseudocodice:

```
initialize stocks dictionary from historical_stocks.csv

for each line in historical_stock_prices.csv:
    parse line to extract ticker, close_date, low, high, volume
    lookup company_name from stocks dictionary using ticker
    convert date to year
    emit (ticker, year), (company_name, date, close, low, high, volume)
```

Reducer

Il file reducer riceve le coppie chiave-valore generate dal mapper e le combina in base alla chiave. Per ogni chiave (ticker, anno), il reducer raccoglie i dati necessari per calcolare le statistiche annuali delle azioni: la variazione percentuale del prezzo di chiusura, il prezzo minimo e massimo dell'anno e il

volume medio di scambio. Il reducer calcola queste statistiche e le emette come output.

E’ riportato sotto un possibile pseudocodice:

```
initialize action_data dictionary
initialize date_data dictionary

for each line in mapper output:
    parse line to extract ticker, company name, year, date, close, low, high, volume
    add date to date_data for (ticker, year)
    update action_data for (ticker, year) with company_name, low, high, volume, and

for each (ticker, year) in action data:
    get first and last dates from date_data
    calculate first_close and last_close from action_data closes
    calculate pct_change as ((last_close - first_close) / first_close) * 100
    calculate min_price, max_price, and avg_volume from action_data prices
    emit (ticker, company name, year, pct_change, min price, max price, avg volume)
```

Risultati ottenuti

Dopo aver eseguito il job MapReduce, abbiamo ottenuto un output che riassume le statistiche annuali per ogni azione. Ogni riga del risultato rappresenta un'azione specifica in un anno specifico, con varie metriche calcolate come la variazione percentuale del prezzo di chiusura, il prezzo minimo e massimo dell'anno, e il volume medio di scambio.

Esempio di Output

L'immagine allegata mostra un esempio dell'output generato dal reducer. Di seguito, descriviamo brevemente ciascuna colonna del risultato:

1. **Ticker**: Il simbolo dell'azione.
2. **Company Name**: Il nome dell'azienda.
3. **Year**: L'anno di riferimento.
4. **Percent Change**: La variazione percentuale del prezzo di chiusura dall'inizio alla fine dell'anno.
5. **Min Price**: Il prezzo minimo registrato nell'anno.
6. **Max Price**: Il prezzo massimo registrato nell'anno.
7. **Average Volume**: Il volume medio di scambio nell'anno.

	A	B	C	D	E	F	G
1	A	AGILENT TECHNOLOGIES INC	2010	32,36	19,0844058990479	30,1001434326172	4927413,49
2	A	AGILENT TECHNOLOGIES INC	2007	7,11	21,6452083587646	28,9127330780029	3521088,84
3	A	AGILENT TECHNOLOGIES INC	2003	52,77	8,08297538757324	21,0443496704102	3244625,79
4	A	AGILENT TECHNOLOGIES INC	2012	12,23	25,2646636962891	33,104434967041	4596130,0
5	A	AGILENT TECHNOLOGIES INC	2013	36,56	28,7482109069824	41,4449195861816	4259520,24
6	A	AGILENT TECHNOLOGIES INC	2001	-43,96	12,8755369186401	48,6409149169922	3311097,18
7	A	AGILENT TECHNOLOGIES INC	2004	-16,32	13,9556512832642	27,7539348602295	4267519,05
8	A	AGILENT TECHNOLOGIES INC	2014	1,82	35,6223182678223	43,791130065918	3011421,83
9	A	AGILENT TECHNOLOGIES INC	2002	-38,6	7,51072978973389	27,1816883087158	3700309,13
10	A	AGILENT TECHNOLOGIES INC	2006	4,03	19,2846927642822	28,2832622528076	3877185,26
11	A	AGILENT TECHNOLOGIES INC	2005	39,41	14,3848352432251	25,822603225708	4200603,57
12	A	AGILENT TECHNOLOGIES INC	2011	-16,6	20,5078678131104	39,5779685974121	6060477,78
13	A	AGILENT TECHNOLOGIES INC	2008	-56,94	10,5579395294189	27,1816883087158	4467688,54
14	A	AGILENT TECHNOLOGIES INC	2018	-2,4	60,4199981689453	75,0	2605923,78
15	A	AGILENT TECHNOLOGIES INC	2000	-23,96	27,2263946533203	115,879829406738	4238207,54
16	A	AGILENT TECHNOLOGIES INC	2017	44,05	45,7400016784668	70,9300003051758	1850781,27
17	A	AGILENT TECHNOLOGIES INC	2016	11,97	34,1500015258789	48,6300010681152	2080817,86
18	A	AGILENT TECHNOLOGIES INC	2015	3,08	33,1199989318848	43,5900001525879	2558341,67
19	A	AGILENT TECHNOLOGIES INC	1999	75,71	28,4781837463379	57,2246055603027	5739950,0
20	A	AGILENT TECHNOLOGIES INC	2009	91,32	8,59799671173096	22,7253227233887	4986684,92

Questi risultati forniscono una panoramica significativa delle performance annuali delle azioni, permettendo di identificare le tendenze e le fluttuazioni dei prezzi e dei volumi di scambio. Questi dati possono essere utilizzati per ulteriori analisi finanziarie, come la valutazione delle performance aziendali nel tempo e l'analisi dei trend di mercato.

MapReduce - Esecuzione Job 3

Il job MapReduce 3 è stato progettato per identificare gruppi di aziende che hanno avuto lo stesso trend percentuale di chiusura dei prezzi delle azioni per almeno tre anni consecutivi. Questo permette di analizzare i pattern di comportamento del mercato azionario e scoprire quali aziende seguono trend simili nel tempo.

In questa sezione, viene spiegato il funzionamento generale dei file Mapper e Reducer utilizzati nel job MapReduce 3 e si presentano i risultati ottenuti.

Mapper

Il Mapper ha il compito di elaborare i dati del file `historical_stock_prices.csv`, associando ogni riga al ticker dell'azione e al nome dell'azienda. Successivamente, estrae e calcola il prezzo di chiusura per ciascun anno, organizzando questi dati in un formato strutturato che può essere facilmente utilizzato dal Reducer.

E’ riportato sotto un possibile pseudocodice:

```
1. Initialize the stocks dictionary from 'historical_stocks.csv'.
  2. For each line in 'historical_stock_prices.csv':
  a. Parse the line to extract ticker, close, date, low, high, volume.
  b. Look up the company_name corresponding to ticker in stocks.
  c. Convert the date to year.
  d. Emit (ticker, year), (company_name, date, close, low, high, volume).
```

Reducer

Il Reducer elabora i dati ricevuti dal Mapper per calcolare il cambiamento percentuale del prezzo di chiusura tra l'inizio e la fine di ogni anno per ciascun ticker. Successivamente, identifica gruppi di aziende che hanno avuto lo stesso trend percentuale di chiusura per almeno tre anni consecutivi.

E’ riportato sotto un possibile pseudocodice:

```
1. Initialize action_data and date_data dictionaries.
2. For each input line:
  a. Extract ticker, company name, year, date, and closing price.
  b. Add date to the corresponding year-ticker date list.
  c. Store closing price for the date.
3. Calculate min and max dates for each ticker and year.
4. For each ticker and year:
  a. Find start and end dates.
  b. Calculate percentage change in closing price.
5. Identify groups with the same trend for at least three consecutive years.
6. Print the identified groups.
```

Risultati ottenuti

L'immagine allegata mostra i risultati ottenuti dal Reducer, che riassumono i trend percentuali di chiusura delle azioni per vari anni. Di seguito, viene descritta ciascuna colonna del risultato:

1. **Years:** sequenza di anni consecutivi
2. **Trend:** percentuale di variazione.
3. **Companies:** lista di aziende che hanno la stessa variazione per quei 3 anni.

	A	B	C	D	E
1	year	year2	year3	pct_chang	company_names
2	2011	2012	2013	0%	ISHARES SHORT TREASURY BOND ETF
3	2012	2013	2014	0%	ISHARES SHORT TREASURY BOND ETF
4	2013	2014	2015	0%	ISHARES SHORT TREASURY BOND ETF
5	2014	2015	2016	0%	ISHARES SHORT TREASURY BOND ETF, ISHARES 1-3 YEAR TREASURY BOND ETF, VANGUARD SHORT-TERM TREASURY ETF
6	2015	2016	2017	0%	ISHARES SHORT TREASURY BOND ETF, ISHARES 0-5 YEAR INVESTMENT GRADE CORPORATE BOND ETF
7	2016	2017	2018	0%	ISHARES SHORT TREASURY BOND ETF

SparkSQL - Esecuzione Job 1

In questa sezione, si descrive l'esecuzione del primo job utilizzando SparkSQL per analizzare i dati storici delle azioni. L'obiettivo è unire i dati dei prezzi con i nomi delle aziende, calcolare le aggregazioni richieste per ciascun ticker e anno, e determinare la variazione percentuale del prezzo di chiusura nel tempo.

Funzionamento del Codice

Il codice inizia con l'inizializzazione di una sessione Spark, necessaria per eseguire operazioni di analisi sui dati. I dati storici delle azioni e dei prezzi vengono caricati nei rispettivi DataFrame di Spark dai file CSV. L'uso dell'opzione `inferSchema=True` consente a Spark di determinare automaticamente il tipo di dati di ciascuna colonna.

Successivamente, vengono create viste temporanee dai DataFrame caricati. Queste viste permettono di eseguire query SQL sui dati, facilitando l'unione e l'aggregazione delle informazioni.

Una query SQL viene utilizzata per unire i dati dei prezzi delle azioni con i nomi delle aziende basandosi sul ticker. Il risultato è un DataFrame contenente tutte le informazioni necessarie per le analisi successive.

Si esegue quindi un'altra query SQL per calcolare le aggregazioni richieste (prezzo minimo, prezzo massimo, volume medio, primo e ultimo prezzo di chiusura) per ciascun ticker e anno.

Utilizzando le funzioni di finestra, si calcola la variazione percentuale del prezzo di chiusura per ogni ticker e anno. Questo viene fatto considerando il primo e l'ultimo prezzo di chiusura dell'anno.

Infine, si riduce il numero di partizioni a una per facilitare la scrittura dei risultati su un unico file CSV, si visualizzano i risultati e si salvano su un file CSV. Questo file contiene tutte le informazioni calcolate, pronte per ulteriori analisi o visualizzazioni.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, min as spark_min, max as spark_max, avg, first, last, round as spark_round
from pyspark.sql.window import Window

# Inizializza una sessione Spark
spark = SparkSession.builder.appName("Stock Analysis").getOrCreate()

# Carica i dati
historical_stocks_df = spark.read.csv('/user/hadoop/historical_stock_pulito.csv', header=True, inferSchema=True)
historical_prices_df = spark.read.csv('/user/hadoop/historical_stock_prices_pulito.csv', header=True, inferSchema=True)

# Crea una vista temporanea per eseguire query SQL
historical_stocks_df.createOrReplaceTempView("historical_stocks")
historical_prices_df.createOrReplaceTempView("historical_prices")

# Unisci i dati dei prezzi con i nomi delle aziende
joined_df = spark.sql("""
    SELECT
        p.ticker,
        s.name as company_name,
        YEAR(p.date) as year,
        p.date,
        p.close,
        p.low,
        p.high,
        p.volume
    FROM historical_prices p
    LEFT JOIN historical_stocks s ON p.ticker = s.ticker
""")

# Crea una vista temporanea per i dati uniti
joined_df.createOrReplaceTempView("joined_data")

# Calcola le aggregazioni richieste per ciascun ticker e anno
aggregated_df = spark.sql("""
    SELECT
        ticker,
        company_name,
        year,
        MIN(low) as min_price,
        MAX(high) as max_price,
        ROUND(AVG(volume), 2) as avg_volume,
        FIRST(close) as first_close,
        LAST(close) as last_close
    FROM joined_data
    GROUP BY ticker, company_name, year
""")

# Calcola la variazione percentuale utilizzando le funzioni di finestra
window_spec = Window.partitionBy("ticker", "year").orderBy("date")

result_df = joined_df \
    .withColumn("first_close", first("close").over(window_spec)) \
    .withColumn("last_close", last("close").over(window_spec)) \
    .groupBy("ticker", "company_name", "year") \
    .agg(
        spark_min("low").alias("min_price"),
        spark_max("high").alias("max_price"),
        spark_round(avg("volume"), 2).alias("avg_volume"),
        first("first_close").alias("first_close"),
        last("last_close").alias("last_close")
    ) \
    .withColumn("pct_change", spark_round((col("last_close") - col("first_close")) / col("first_close") * 100, 2)) \
    .select("ticker", "company_name", "year", "pct_change", "min_price", "max_price", "avg_volume")

# Riduci il numero di partizioni a una
single_partition_df = result_df.coalesce(1)

# Mostra il risultato
single_partition_df.show()

# Salva il risultato su un file CSV in una singola partizione
single_partition_df.write.csv('/user/hadoop/SQL1_result', header=True, mode='overwrite')
```

Risultati ottenuti

L'analisi ha prodotto un DataFrame contenente i seguenti campi per ciascun ticker e anno:

- Ticker
- Nome dell'azienda
- Anno
- Variazione percentuale del prezzo di chiusura
- Prezzo minimo
- Prezzo massimo
- Volume medio

	A	B	C	D	E	F	G
1	ticker	company_name	year	pct_change	min_price	max_price	avg_volume
2	A	AGILENT TECHNOLOGIES, INC.	2002	-38,6	7,51073	27,18168831	3700309,13
3	A	AGILENT TECHNOLOGIES, INC.	2006	4,03	19,28469	28,28326225	3877185,26
4	AA	ALCOA CORPORATION	1976	46,56	5,755185	9,18546772	296342,69
5	AA	ALCOA CORPORATION	1987	32,5	10,05655	19,36817932	1673938,34
6	AA	ALCOA CORPORATION	1990	-23,87	14,83852	23,1769352	1447731,62
7	AA	ALCOA CORPORATION	1992	11,56	18,32287	24,1501503	1097310,63
8	AA	ALCOA CORPORATION	1999	125,91	42,7734	100,0128632	1630363,1
9	AA	ALCOA CORPORATION	2002	-36,15	42,34086	95,51924896	1623076,19
10	AA	ALCOA CORPORATION	2003	61,36	44,33535	93,52475739	1645092,86
11	AABA	ALTABA INC.	2001	-37,06	4,01	21,6875	22801817,74
12	AABA	ALTABA INC.	2003	155,85	8,25	22,73999977	24458893,65
13	AAL	AMERICAN AIRLINES GROUP, INC.	2011	-52,39	3,96	11,56000042	7243381,35
14	AAME	ATLANTIC AMERICAN CORPORATION	1990	-40	1,25	2,75	6511,84
15	AAME	ATLANTIC AMERICAN CORPORATION	2011	-4,37	1,26	2,25	4875,3
16	AAME	ATLANTIC AMERICAN CORPORATION	2012	56,85	1,91	3,349999905	6052,03
17	AAME	ATLANTIC AMERICAN CORPORATION	2014	1,26	3,23	4,380000114	8384,27
18	AAME	ATLANTIC AMERICAN CORPORATION	2018	-16,92	2,2	3,900000095	3695,74
19	AAN	AARON'S, INC.	1988	7,14	0,481481	0,555555582	95822,22
20	AAN	AARON'S, INC.	2006	33,91	12,8	17,58666611	4249,32
21	AAN	AARON'S, INC.	2007	-32,31	10,84	18,34666634	2352,68

Questi risultati forniscono una visione completa delle performance delle azioni nel tempo, facilitando l'identificazione di trend significativi e le decisioni di investimento.

SparkSQL - Esecuzione Job 3

In questa sezione, viene descritto il funzionamento del Job3 SparkSQL per analizzare i dati storici delle azioni. L'obiettivo principale di questo job è individuare le aziende che hanno mantenuto una variazione percentuale del prezzo di chiusura costante per tre anni consecutivi. Questo tipo di analisi è utile per identificare aziende con performance stabili nel tempo, facilitando decisioni di investimento più accurate.

Funzionamento del Codice

Il codice inizia con l'inizializzazione di una sessione Spark, necessaria per eseguire operazioni di analisi sui dati. Successivamente, i dati storici delle azioni e dei prezzi vengono caricati nei rispettivi DataFrame di Spark dai file CSV.

Successivamente, i dati dei prezzi delle azioni vengono uniti con i nomi delle aziende utilizzando il ticker come chiave di unione. Viene anche estratto l'anno dalla data e si filtra per considerare solo gli anni successivi al 2000.

Una volta ottenuti i dati uniti, si calcolano le aggregazioni richieste per ciascun ticker e anno. In particolare, vengono calcolati il primo e l'ultimo prezzo di chiusura dell'anno e la variazione percentuale tra questi due valori. La variazione percentuale viene arrotondata e convertita in formato percentuale.

Per identificare le aziende con variazioni percentuali costanti per tre anni consecutivi, si utilizza una finestra di partizione per ticker, ordinando i dati per anno. Si calcolano le variazioni percentuali dei tre anni consecutivi utilizzando funzioni di finestra, e si filtrano solo i record che soddisfano questa condizione.

Il risultato viene ulteriormente filtrato per assicurarsi che i tre anni consecutivi siano validi, e successivamente le aziende con la stessa variazione percentuale vengono raggruppate insieme. I risultati vengono ordinati per anno e percentuale di cambiamento.

Infine, il risultato viene mostrato e salvato su un file CSV in una singola partizione per facilitare l'accesso e l'analisi futura.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, first, last, round as spark_round, lag, lead, concat, lit, collect_list, array_join
from pyspark.sql.window import Window

# Inizializza una sessione Spark
spark = SparkSession.builder.appName("Stock Analysis").getOrCreate()

# Carica i dati
historical_stocks_df = spark.read.csv('/user/hadoop/historical_stock_pulito.csv', header=True, inferSchema=True)
historical_prices_df = spark.read.csv('/user/hadoop/historical_stock_prices_pulito.csv', header=True, inferSchema=True)

# Unisci i dati dei prezzi con i nomi delle aziende
joined_df = historical_prices_df.join(historical_stocks_df, "ticker") \
    .withColumn("year", col("date").substr(1, 4).cast("int")) \
    .filter(col("year") > 2000)

# Calcola le aggregazioni richieste per ciascun ticker e anno
aggregated_df = joined_df.groupBy("ticker", "name", "year") \
    .agg(
        first("close").alias("first_close"),
        last("close").alias("last_close")
    ) \
    .withColumn("pct_change", spark_round((col("last_close") - col("first_close")) / col("first_close") * 100, 0)) \
    .withColumn("pct_change", concat(col("pct_change").cast("int"), lit("%"))) \
    .select("ticker", "name", "year", "pct_change")

# Definisci la finestra di partizione per ticker e ordina per anno
window_spec = Window.partitionBy("ticker").orderBy("year")

# Calcola la variazione percentuale dei tre anni consecutivi
consecutive_df = aggregated_df \
    .withColumn("pct_change_lag1", lag("pct_change", 1).over(window_spec)) \
    .withColumn("pct_change_lag2", lag("pct_change", 2).over(window_spec)) \
    .filter((col("pct_change") == col("pct_change_lag1")) & (col("pct_change") == col("pct_change_lag2"))) \
    .select("year", lead("year", 1).over(window_spec).alias("year2"), lead("year", 2).over(window_spec).alias("year3"), "pct_change", "name")

# Filtra per avere solo anni validi
filtered_df = consecutive_df.filter(col("year3").isNotNull())

# Raggruppa le aziende che hanno la stessa percentuale per tre anni consecutivi
grouped_df = filtered_df.groupBy("year", "year2", "year3", "pct_change") \
    .agg(array_join(collect_list("name"), ", ").alias("company_names"))

# Ordina per anno e percentuale di cambiamento
final_df = grouped_df.orderBy("year", "pct_change")

# Mostra il risultato
final_df.show(truncate=False)

# Salva il risultato su un file CSV in una singola partizione
final_df.coalesce(1).write.csv('/user/hadoop3/SQL3_result', header=True, mode='overwrite')
```

Risultati Ottenuti

L'analisi ha prodotto un DataFrame contenente i seguenti campi:

- Anno (primo anno del periodo di tre anni)
- Anno 2 (secondo anno del periodo di tre anni)
- Anno 3 (terzo anno del periodo di tre anni)

- Variazione percentuale del prezzo di chiusura (costante per i tre anni)
- Nomi delle aziende che hanno mantenuto questa variazione percentuale per i tre anni consecutivi

	A	B	C	D	E
1	year	year2	year3	pct_chang	company_names
2	2011	2012	2013	0%	ISHARES SHORT TREASURY BOND ETF
3	2012	2013	2014	0%	ISHARES SHORT TREASURY BOND ETF
4	2013	2014	2015	0%	ISHARES SHORT TREASURY BOND ETF
5	2014	2015	2016	0%	ISHARES SHORT TREASURY BOND ETF, ISHARES 1-3 YEAR TREASURY BOND ETF, VANGUARD SHORT-TERM TREASURY ETF
6	2015	2016	2017	0%	ISHARES SHORT TREASURY BOND ETF, ISHARES 0-5 YEAR INVESTMENT GRADE CORPORATE BOND ETF
7	2016	2017	2018	0%	ISHARES SHORT TREASURY BOND ETF

Questi risultati offrono una visione chiara delle aziende che mostrano una performance stabile nel tempo, rendendoli un punto di riferimento utile per decisioni di investimento strategiche.

SparkCore - Esecuzione Job 1

In questa sezione, viene descritta l'esecuzione del primo job utilizzando SparkCore per analizzare i dati storici delle azioni. L'obiettivo è unire i dati dei prezzi con i nomi delle aziende, calcolare le aggregazioni richieste per ciascun ticker e anno, e determinare la variazione percentuale del prezzo di chiusura nel tempo. Dopo aver eseguito il join preliminare tra i due dataset, si effettuano tutte le operazioni usando il costrutto *Resilient Distributed Dataset - RDD*. Infine, l'aggregato è convertito in dataframe per poterlo correttamente presentare all'utente finale.

Funzionamento del Codice

Il codice inizia con l'inizializzazione di una sessione Spark, necessaria per eseguire operazioni di analisi sui dati. I dati storici delle azioni e dei prezzi vengono caricati nei rispettivi DataFrame di Spark dai file CSV. L'uso dell'opzione `inferSchema=True` consente a Spark di determinare automaticamente il tipo di dati di ciascuna colonna.

Successivamente si esegue il primo join tra i due dataframe appena caricati, al fine di associare il *company_name* al relativo *ticker*. Questa operazione viene eseguita mantenendo alcune funzioni provenienti dal modulo Spark.SQL in quanto eseguire tali operazioni usando il costrutto RDD aumenterebbe il costo computazionale necessario a completare l'esecuzione.

In seguito viene convertito il dataframe in RDD al fine di eseguire la successiva aggregazione. La funzione di aggregazione dei dati calcola tutti i valori richiesti dalla task, ovvero il valore minimo, massimo, medio e procede a calcolare la percentuale di cambiamento dopo aver estratto la prima e ultima data di chiusura usando come chiave il ticker e l'anno.

Si aggiorna l'RDD raggruppando per *ticker, company_name* e *year* e si applica l'aggregazione a tali chiavi.

Viene convertito l'RDD aggregato in DataFrame al fine di utilizzare le funzioni proprie dei DataFrame per ridurre il numero di partizioni ad una, mostrare il risultato a schermo e salvarlo su un file CSV in una singola partizione.

Nel seguito è riportato un possibile pseudocodice:

```
1. Initialize a Spark session
   - Set the application name to "Stock Analysis"
   - Create a Spark session

2. Load the data
   - Read the CSV file 'historical_stock_pulito.csv' into a DataFrame called `historical_stocks_df`
   - Read the CSV file 'historical_stock_prices_pulito.csv' into a DataFrame called `historical_prices_df`

3. Join the price data with the company names
   - Join `historical_prices_df` with `historical_stocks_df` using the `ticker` column as the key

4. Convert the joined DataFrame to an RDD for aggregation
   - Map each row of the joined DataFrame to a key composed of (`ticker`, `company_name`, `year`)
   - Values composed of (`date`, `close`, `low`, `high`, `volume`)

5. Define a function to aggregate the data
   - Sort the values by `date`
   - Calculate `min_price` as the minimum of `low`
   - Calculate `max_price` as the maximum of `high`
   - Calculate `avg_volume` as the average of `volume`, rounded to 2 decimal places
   - Take the `close` of the first and last date to calculate `pct_change` (percentage change)

6. Group the data by `ticker`, `company_name`, and `year` and apply the aggregation function

7. Convert the aggregated RDD back to a DataFrame
   - Map the aggregated data to a new structure
   - Convert the RDD to a DataFrame with the columns:
     - `ticker`, `company_name`, `year`, `pct_change`, `min_price`, `max_price`, `avg_volume`

8. Reduce the number of partitions to one

9. Display the result in the single-partitioned DataFrame

10. Save the result to a CSV file in a single partition
```

Risultati Ottenuti

L’analisi ha prodotto un DataFrame contenente i seguenti campi per ciascun ticker e anno:

- Ticker
- Nome dell'azienda
- Anno
- Variazione percentuale del prezzo di chiusura
- Prezzo minimo
- Prezzo massimo
- Volume medio

	A	B	C	D	E	F	G
1	ticker	company_name	year	pct_change	min_price	max_price	avg_volume
2	A	AGILENT TECHNOLOGIES, INC.	2002	-38,6	7,5107298	27,1816883	3700309,13
3	A	AGILENT TECHNOLOGIES, INC.	2006	4,03	19,284693	28,2832623	3877185,26
4	AA	ALCOA CORPORATION	1976	46,56	5,7551851	9,18546772	296342,69
5	AA	ALCOA CORPORATION	1987	32,5	10,056555	19,3681793	1673938,34
6	AA	ALCOA CORPORATION	1990	-23,87	14,838525	23,1769352	1447731,62
7	AA	ALCOA CORPORATION	1992	11,56	18,322874	24,1501503	1097310,63
8	AA	ALCOA CORPORATION	1999	125,91	42,773399	100,012863	1630363,1
9	AA	ALCOA CORPORATION	2002	-36,15	42,340858	95,519249	1623076,19
10	AA	ALCOA CORPORATION	2003	61,36	44,33535	93,5247574	1645092,86
11	AABA	ALTABA INC.	2001	-37,06	4,0100002	21,6875	22801817,74
12	AABA	ALTABA INC.	2003	155,85	8,25	22,7399998	24458893,65
13	AAL	AMERICAN AIRLINES GROUP, INC.	2011	-52,39	3,96	11,5600004	7243381,35
14	AAME	ATLANTIC AMERICAN CORPORATION	1990	-40	1,25	2,75	6511,84
15	AAME	ATLANTIC AMERICAN CORPORATION	2011	-4,37	1,26	2,25	4875,3
16	AAME	ATLANTIC AMERICAN CORPORATION	2012	56,85	1,91	3,3499999	6052,03
17	AAME	ATLANTIC AMERICAN CORPORATION	2014	1,26	3,23	4,38000011	8384,27
18	AAME	ATLANTIC AMERICAN CORPORATION	2018	-16,92	2,2	3,9000001	3695,74
19	AAN	AARON'S, INC.	1988	7,14	0,4814815	0,55555558	95822,22
20	AAN	AARON'S, INC.	2006	33,91	12,8	17,5866661	4249,32
21	AAN	AARON'S, INC.	2007	-32,31	10,84	18,3466663	2352,68

SparkCore - Esecuzione Job 3

In questa sezione, viene descritto il funzionamento del Job3 tramite SparkCore per analizzare i dati storici delle azioni. L'obiettivo principale di questo job è individuare le aziende che hanno mantenuto una variazione percentuale del prezzo di chiusura costante per tre anni consecutivi.

Funzionamento del Codice

Il codice inizia con l'inizializzazione di una sessione Spark, necessaria per eseguire operazioni di analisi sui dati. Successivamente, i dati storici delle azioni e dei prezzi vengono caricati nei rispettivi DataFrame di Spark dai file CSV.

Successivamente si esegue il primo join tra i due dataframe appena caricati, al fine di associare il *company_name* al relativo *ticker*. Questa operazione viene eseguita mantenendo alcune funzioni provenienti dal modulo Spark.SQL in quanto eseguire tali operazioni usando il costrutto RDD aumenterebbe il costo computazionale necessario a completare l'esecuzione.

Viene applicato un filtro che permette di eliminare tutte le ennuple che hanno una data precedente all'anno 2000 e si procede a convertire il DataFrame in RDD per eseguire l'aggregazione. La funzione aggregativa si occupa di calcolare i valori di cambiamento percentuale con chiave ticker e anno, esclusivamente dopo aver calcolato la prima e l'ultima chiusura di ciascun anno per singola azione.

Si procede ad ordinare i dati per anno e si verifica se ci sono tre anni consecutivi per i quali la variazione percentuale rimane invariata, in tal caso si aggiungono tali ennuple all'interno del risultato.

Infine si raggruppano i valori trovati eseguendo una *map* sulla chiave *ticker*. Si preparano i dati per poter essere facilmente visualizzati e si salva il risultato all'interno di un'unica partizione in formato CSV.

Nel seguito è riportato un possibile pseudocodice:

```
1. Initialize a Spark session
   - Set the application name to "Stock Analysis"
   - Create a Spark session

2. Load the data
   - Read the CSV file 'historical_stock_pulito.csv' into a DataFrame called `historical_stocks_df`
   - Read the CSV file 'historical_stock_prices_pulito.csv' into a DataFrame called `historical_prices_df`

3. Join the price data with the company names
   - Join `historical_prices_df` with `historical_stocks_df` using the `ticker` column as the key

4. Filter the data from the year 2000 onwards
   - Filter the joined DataFrame to include only rows where the year is greater than or equal to 2000

5. Convert the filtered DataFrame to RDD for aggregation

6. Define a function to aggregate the data
   - Sort the values by `date`
   - Take the `close` of the first and last date to calculate `pct_change` (percentage change)

7. Group the data by `ticker`, `company_name`, and `year` and apply the aggregation function
   - Use `groupByKey` to group the data
   - Apply the `aggregate_data` function to the grouped values

9. Define a function to find sequences of similar percentage changes for at least three consecutive years
   - Sort the data by year
   - Iterate through the sorted data to find sequences where the percentage change is the same for three consecutive years
   - Store the results in a list

10. Group by `ticker` and apply the find_sequences function
    - Map each row to a key of `ticker` and a value of the relevant data
    - Group the data by `ticker`
    - Apply the `find_sequences` function to the grouped values

11. Group by common trend and collect the companies
    - Map each sequence to a key of the trend and a value of the company name
    - Group the data by the trend
    - Filter the groups to include only those with more than one company

12. Convert to DataFrame
    - Map the grouped data to a new structure
    - Convert the RDD to a DataFrame with columns:
      - `trend`, `companies`

14. Show the result and save to a CSV file in a single partition
```

Risultati Ottenuti

L'analisi ha prodotto un DataFrame contenente i seguenti campi:

- Anno (primo anno del periodo di tre anni)
- Anno 2 (secondo anno del periodo di tre anni)
- Anno 3 (terzo anno del periodo di tre anni)

- Variazione percentuale del prezzo di chiusura (costante per i tre anni)
- Nomi delle aziende che hanno mantenuto questa variazione percentuale per i tre anni consecutivi

	A	B	C	D	E
1	year	year2	year3	pct_chang	company_names
2	2011	2012	2013	0%	ISHARES SHORT TREASURY BOND ETF
3	2012	2013	2014	0%	ISHARES SHORT TREASURY BOND ETF
4	2013	2014	2015	0%	ISHARES SHORT TREASURY BOND ETF
5	2014	2015	2016	0%	ISHARES SHORT TREASURY BOND ETF, ISHARES 1-3 YEAR TREASURY BOND ETF, VANGUARD SHORT-TERM TREASURY ETF
6	2015	2016	2017	0%	ISHARES SHORT TREASURY BOND ETF, ISHARES 0-5 YEAR INVESTMENT GRADE CORPORATE BOND ETF
7	2016	2017	2018	0%	ISHARES SHORT TREASURY BOND ETF

Analisi a dimensioni variabili

Si valutano le performance avendo come input un dataset a dimensioni variabili e alterando i valori dei dati a disposizione. Nel seguito sono presentati metodi e risultati ottenuti tramite l'analisi del dataset duplicato. La stessa analisi è stata svolta effettuando anche una riduzione delle dimensioni del dataset, ma omessa nell'elaborato (codice nel repository GitHub).

Duplicazione del dataset

Si vuole analizzare come variano le prestazioni di ciascuna tecnologia utilizzata al raddoppio delle dimensioni del dataset.
Si riporta pertanto il codice Python usato per raddoppiare e alterare i valori dei dati di entrambi i dataset di riferimento, si sottolinea come all'interno del dataset historical_stocks.csv vengono solamente raddoppiate le ennuple in quanto il dataset non ha valori di tipo *float* o *int*.

```python
import random
import pandas as pd

def modify_value(value):
    if isinstance(value, (int, float)):
        return value * random.random()
    else:
        return value

def process_csv(file_path):
    # Legge il file CSV in un DataFrame pandas
    df = pd.read_csv(file_path)

    # Raddoppia il DataFrame concatenando il DataFrame con una sua copia
    df_doubled = pd.concat([df, df], ignore_index=True)

    # Modifica i valori nel DataFrame raddoppiato
    df_modified = df_doubled.map(modify_value)

    # Salva il DataFrame modificato in un nuovo file CSV
    output_file_path = file_path.replace("_pulito.csv", "_double.csv")
    df_modified.to_csv(output_file_path, index=False)
    print(f"File salvato come {output_file_path}")

# Esempio di utilizzo
process_csv("historical_stock_prices_pulito.csv")
process_csv("historical_stock_pulito.csv")
```

Il codice prevede di leggere, usando la libreria *Python Pandas*, i file CSV, successivamente si occupa di raddoppiare il DataFrame concatenando

l'originale con una sua copia. Si procede quindi con l'alterazione dei valori andando a moltiplicare tutti i valori di tipo *int* o *float* per un numero generato in maniera randomica ad ogni istanza. Viene quindi salvato il DataFrame modificato in un nuovo file CSV. La funzione è chiamata due volte per processare entrambi i dataset di riferimento.

Risultati ottenuti

Sono stati eseguiti tutti i codici presentati in precedenza cambiando l'input con i nuovi dataset raddoppiati e modificati appena ottenuti. Si presentano quindi i seguenti risultati ottenuti eseguendo il Job 1.

	A	B	C	D	E	F	G
1	ticker	company_name	year	pct_change	min_price	max_price	avg_volume
2	A	AGILENT TECHNOLOGIES, INC.	2006	-55,26	0,008666992	27,69494492	2022748,73
3	AA	ALCOA CORPORATION	1987	12,46	0,152371416	18,60586238	818456,81
4	AA	ALCOA CORPORATION	1992	27,95	0,068051211	23,13746305	568439,18
5	AA	ALCOA CORPORATION	2002	2727,27	0,139098226	91,87859424	817023,04
6	AABA	ALTABA INC.	2001	-75,12	0,008219781	19,23037395	10622582,2
7	AAME	ATLANTIC AMERICAN CORPORATION	2012	124,99	0,003006574	2,923635312	3304,78
8	AAON	AAON, INC.	2009	-50,26	0,005414911	6,392571918	166514,36
9	AAP	ADVANCE AUTO PARTS INC	2012	465,27	0,126629147	89,77332399	593985,85
10	AAPL	APPLE INC.	1993	14,37	0,000310148	2,061317457	26657939,14
11	AAT	AMERICAN ASSETS TRUST, INC.	2018	8,31	0,004825279	38,39979861	137415,88
12	AB	ALLIANCEBERNSTEIN HOLDING L.P.	2018	5472,72	0,048654076	29,31625539	149386,81
13	ABAX	ABAXIS, INC.	1992	-46,83	0,004106997	14,40944367	25666,33
14	ABAX	ABAXIS, INC.	2000	1027,26	0,003217747	9,490550056	31135,81
15	ABCD	CAMBIUM LEARNING GROUP, INC.	2017	19,96	0,008861039	6,654707523	24506,23
16	ABEO	ABEONA THERAPEUTICS INC.	2009	524,16	0,068110874	230,5978833	365,33
17	ABEV	AMBEV S.A.	2018	-11,22	0,006683767	7,02310783	12808230,9
18	ABIL	ABILITY INC.	2014	-14,45	0,009672788	9,774514453	11673,05
19	ABMD	ABIOMED, INC.	2000	803,31	0,07045268	37,06482798	92545,63
20	ABMD	ABIOMED, INC.	2010	193,33	0,009020387	12,03487358	116266,11
21	ABMD	ABIOMED, INC.	2012	-59,06	0,009119568	23,86114876	318476,1

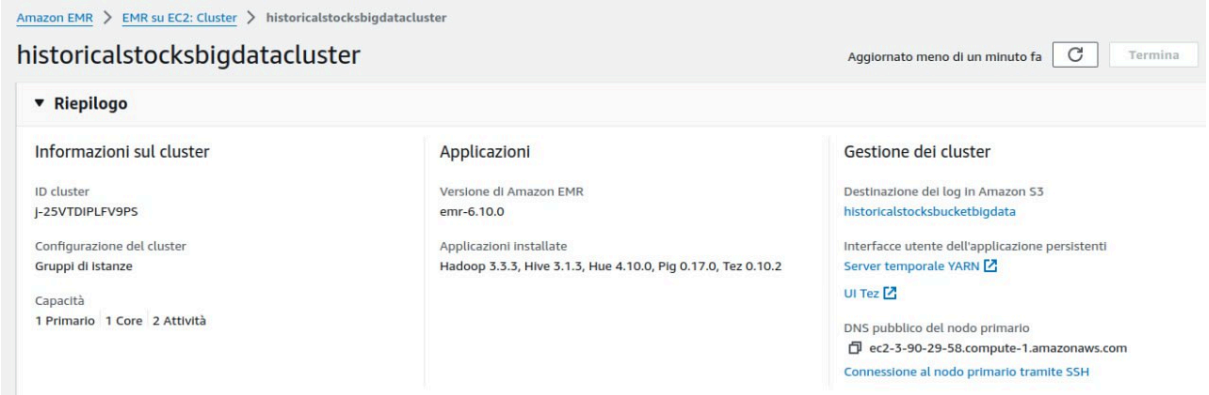
I risultati ottenuti presentano valori che risultano totalmente fuori scala rispetto a quelli calcolati usando il dataset originale, questo è dovuto proprio all'introduzione della componente stocastica.
L'esecuzione del Job 3 non restituisce alcun valore di output, in quanto l'alterazione dei valori non ha prodotto alcun andamento percentuale costante negli anni.

Esecuzione MapReduce su Cluster AWS

In questa sezione si analizzano le prestazioni ottenute dall'esecuzione dei job MapReduce su un'infrastruttura a cluster in cloud AWS. Nel seguito sono elencate le attività svolte per la creazione del Cluster tramite strumento ElasticMapReduce - EMR offerto dai servizi Amazon.

Creazione Cluster

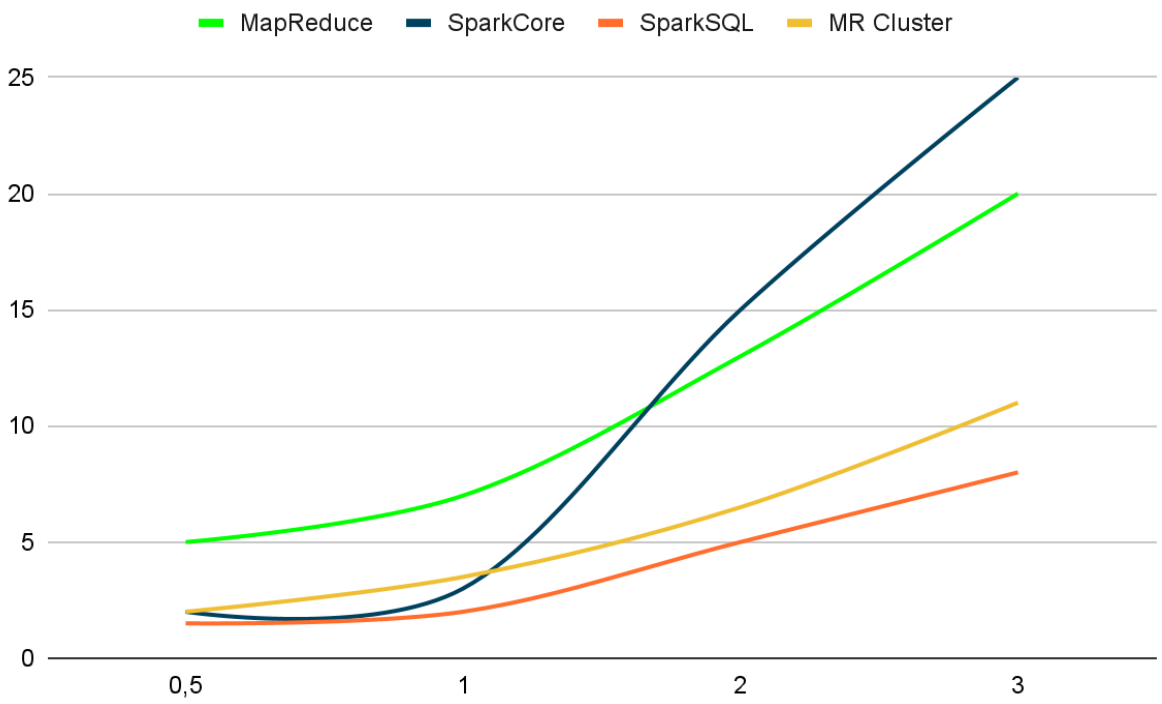
Per la creazione del cluster è stato necessario creare un bucket S3 di storage per memorizzare i log, oltre che una coppia di chiavi EC2 per poter accedere correttamente e in maniera sicura alle macchine remote. Infine è stato creato un cluster EMR basato sul Core di Hadoop. Le caratteristiche di configurazione dei nodi del cluster che sono state scelte sono di tipo *m5a.xlarge* e offrono 4 vCore e 16 GB di memoria.



Valutazione prestazioni

Si valutano le performance ottenute in locale e tramite esecuzione su cluster in cloud, usando il servizio di Amazon Web Service - AWS, di entrambi i job 1 e 3 usando tutte le tecnologie presentate e al variare delle dimensioni dei dati di input.
Nel seguito è riportato un line chart che rappresenta le prestazioni ottenute tramite esecuzione del Job 1 utilizzando le tre diverse tecnologie presentate. Sull'asse delle ascisse è presente il fattore di duplicazione del dataset di riferimento, sull'asse delle ordinate sono riportati i timestamp rilevati per poter terminare l'esecuzione.
Il grafico mostra come SparkSQL sia lo strumento più efficiente per poter eseguire questo tipo di analisi, anche se aumentano le dimensioni del dataset. MapReduce, a fronte di un tempo computazionale elevato anche per dataset

di dimensionalità limitata, non presenta importanti incrementi all'aumentare delle dimensioni. SparkCore, che per dataset a dimensione limitata presenta un andamento approssimabile a quello di SparkSQL, all'aumentare delle dimensioni del dataset presenta un notevole incremento del tempo necessario a terminare la computazione.
Infine, dall'esecuzione delle operazioni lanciate su infrastruttura a cluster in cloud AWS, si può notare che i tempi di esecuzione sono proporzionali con quanto si è rilevato in locale.



I tempi di esecuzione rilevati per il Job 3 presentano andamenti analoghi a quelli appena presentati, ma con tempi computazionali leggermente ridotti dettati dalla dimensione dell'output ridotta.