Progetto Big Data

Polosa Sebastiano[1] Ungaro Riccardo[2]

Maggio 2021

Sommario

1 Presentazione del dataset

Per l'esecuzione del progetto è stato messo a disposizione un dataset comprendente lo storico dell'andamento giornaliero di un'ampia selezione di azioni sulla borsa di New York[1] e sul NASDAQ in un intervallo temporale che va dal 1970 al 2018. Il dataset è formato da due file in formato CSV[2] intitolati *historical_stock_prices.csv* e *historical_stocks.csv* i cui campi sono riassunti nelle tabelle 1.
Per permettere il confronto tra i tempi di esecuzione con i vari framework all'aumentare del numero di campioni in input sono stati prodotti tre file denominati *20973889_historical_stock_prices.csv*, *40964626_historical_stock_prices.csv* e *80009033_historical_stock_prices.csv* con dimensioni di 20 973 889 (2 GB), 40 964 626 (4 GB) e 80 009 033 (9 GB) record ciascuno. La creazione di questi file è stata effettuata utilizzando la libreria Pandas in uno script scritto in Python.

2 Job uno

Un job che sia in grado di generare un report contenente, per ciascuna azione:

a la data della prima quotazione;

b la data dell'ultima quotazione;

c la variazione percentuale della quotazione (differenza percentuale tra il primo e l'ultimo prezzo di chiusura presente nell'archivio);

d il prezzo massimo e quello minimo;

e *(facoltativo)* il massimo numero di giorni consecutivi in cui l'azione è cresciuta (chiusura maggiore dell'apertura) con indicazione dell'anno in cui questo è avvenuto.

Il report deve essere ordinato per valori decrescenti del punto b.

[1] matr. 498626, e-mail: seb.polosa@stud.uniroma3.it
[2] matr. 499606, e-mail: ric.ungaro@stud.uniroma3.it
[1] NYSE.
[2] Comma-Separated Values, trattasi di un file di testo con la stessa impostazione di una tabella di dati.

historical_stock_prices.csv	
ticker	simbolo univoco dell'azione
open	prezzo di apertura
close	prezzo di chiusura
adj_close	prezzo di chiusura "modificato"
low	prezzo minimo
high	prezzo massimo
volume	numero di transazioni
date	data nel formato aaaa-mm-gg

historical_stocks.csv	
ticker	simbolo univoco dell'azione
exchange	NYSE o NASDAQ
name	nome dell'azienda
sector	settore dell'azienda
industry	industria di riferimento per l'azienda

Tabella 1: Descrizione dei campi dei due file del dataset.

2.1 Map-Reduce

Per l'esecuzione del job uno tramite il framework Map-Reduce è stato necessario scrivere due file: *mapper.py* e *reducer.py*.

mapper.py

Il file *mapper.py* (algoritmo 1) in questa applicazione, svolge un compito molto semplice: deve leggere le righe che riceve tramite il canale di standard input e restituisce solo i campi necessari per lo svolgimento del job; tali campi sono: *ticker*, *open*, *close*, *low*, *high* e *date*. Si noti come il map avvenga sulla chiave primaria *ticker*.

Algorithm 1: mapper.py in Job 1

```
1 Inizializzazione;
2 while non sono terminati i record di historical_stock_prices.csv do
3 |   rimuovi gli spazi iniziali e finali della riga;
4 |   preleva i campi di interesse;
5 |   trasmetti i dati tramite Standard Output;
6 end
```

reducer.py

Il file *reducer.py* è quello che si occupa della reale esecuzione del job; questo infatti calcola tutte le informazioni richieste per poi trasmetterle sullo Standard Output.
Per lo svolgimento del compito si è deciso di utilizzare i *dizionari* di Python come struttura per raccogliere le informazioni elaborate, in particolare:

action_map è un dizionario che si occupa di raccogliere i dati per ogni ticker; le chiavi sono formate da tutti i ticker presenti nell'archivio mentre i valori sono altre mappe con i campi first_date, last_date, var, max_price, min_price, first_close, last_close, days_of_growth e year_of_growth;

meta_growth_days è un dizionario utilizzato per tenere traccia di tutte le date in cui l'azione ha avuto una crescita o una regressione;

growth_days è la struttura dati che tiene traccia del numero di giorni di crescita consecutivi massimo e l'anno in cui questo è avvenuto per ogni ticker in action_map.

L'esecuzione dello script si può dividere in tre parti:

1. una fase iniziale (algoritmo 2) in cui si leggono i dati provenienti da *mapper.py* e si effettuano i calcoli delle date di prima e ultima quotazione, della variazione percentuale tra il primo e l'ultimo prezzo di chiusura in archivio, dei prezzi massimi e minimi e dei giorni di crescita per ogni ticker nel dataset.

2. una fase successiva (algoritmo 3) in cui si calcola il numero massimo di giorni continui in cui è avvenuta una crescita del ticker.

3. una fase finale (algoritmo 4) in cui viene effettuata la stampa dei risultati.

Algorithm 2: reducer.py in Job 1 - Lettura dallo Standard Input.

```
Inizializzazione dei dizionari action_map, meta_growth_days e growth_days;
while mapper.py continua a trasmettere do
    rimuovi gli spazi iniziali e finali della riga;
    preleva i campi di interesse;
    cerco il ticker corrente nel dizionario action_map;
    if ticker corrente non è nel dizionario action_map then
        creo un nuovo campo nel dizionario associato al nuovo ticker;
    end
    if data corrente è minore della prima data associata al ticker nel dizionario action_map then
        imposto la prima data uguale alla data corrente;
    end
    if data corrente è maggiore dell'ultima data associata al ticker nel dizionario action_map then
        imposto l'ultima data uguale alla data corrente;
    end
    if il campo high è maggiore del prezzo massimo associato al ticker nel dizionario action_map
     then
        imposto il prezzo massimo uguale al valore di high;
    end
    if il campo low è minore del prezzo minimo associato al ticker nel dizionario action_map then
        imposto il prezzo minimo uguale al valore di low;
    end
    if il ticker nella data corrente ha avuto una chiusura maggiore dell'apertura then
        aggiungo nella lista associata al ticker nel dizionario meta_growth_days la data corrente e
         True;
    else
        aggiungo nella lista associata al ticker nel dizionario meta_growth_days la data corrente e
         False;
    end
end
```

Algorithm 3: reducer.py in Job 1 - Ricerca del numero massimo di giorni di crescita consecutivi per ogni ticker.

```
1  sort dei valori di meta_growth_days;          // Ordino le date associate ai valori di
     crescita in ordine crescente
2  inizializzo growth_days;
3  while non ho visitato tutti i ticker in meta_growth_days do
4      while non ho visitato tutte le date per il ticker corrente in meta_growth_days do
5          current_year = anno della data corrente;
6          if nella data corrente è avvenuta una crescita then
7              if current_year è uguale all'anno che stiamo esaminando then
8                  aggiungo 1 al contatore di giorni corrente di crescita;
9              else
10                 imposto come anno da esaminare uguale a current_year;
11                 imposto il contatore di giorni corrente di crescita uguale a 1;
12             end
13         else
14             imposto come anno da esaminare uguale a current_year;
15             imposto il contatore di giorni corrente di crescita uguale a 0;
16         end
17     end
18 end
19 prelevo il valore calcolato di massimo numero di giorni di crescita consecutivi;
20 prelevo l'anno in cui avviene il massimo numero di giorni di crescita consecutivi;
```

Algorithm 4: reducer.py in Job 1 - Stampa dei risultati.

```
1  ordino i risultati ottenuti per ordine decrescente di data dell'ultima quotazione;
2  stampa dei risultati ottenuti;
```

2.2 Hive

I dati del file *historical_stock_prices.csv*, vengono caricati nella tabella historical_stock_prices:

```
CREATE TABLE historical_stock_prices (ticker STRING, open float,
    close float, adj_close float,lowThe float, highThe float, volume
    float, dates date)
```

Si definisce una tabella firstAndLastData con la quale si calcola, per ciascuna azione, la prima e l'ultima data in archivio e il prezzo massimo e minimo:

```
SELECT ticker, min(dates) AS min_data , max(dates) AS max_data , min
    (lowThe) AS min_price , max(highThe) AS max_price
FROM historical_stock_prices
GROUP BY ticker;
```

Si definiscono due tabelle per selezionare, per ciascuna azione, il primo e l'ultimo prezzo di chiusura

```
CREATE TABLE firstPriceClose AS
SELECT data.ticker, data.min_data, hsp.close AS close
FROM firstAndLastData AS data, historical_stock_prices AS hsp
WHERE data.min_data=hsp.dates AND data.ticker=hsp.ticker;

CREATE TABLE lastPriceClose AS
SELECT data.ticker, data.max_data, hsp.close AS close
FROM firstAndLastData AS data, historical_stock_prices AS hsp
WHERE data.max_data=hsp.dates AND data.ticker=hsp.ticker;
```

Si definisce una tabella che, per ogni azione, calcola la variazione percentuale della quotazione come differenza tra il primo prezzo di chiusura e l'ultimo ((Valore finale − Valore iniziale)/Valore iniziale · 100):

```
CREATE TABLE variazione AS
SELECT vi.ticker, (((vf.close-vi.close)/vi.close) * 100) AS var
FROM   firstPriceClose AS vi join lastPriceClose AS vf ON vi.ticker=
    vf.ticker;
```

Infine, si effettua una query per ottenere il risultato voluto, ordinandolo per data dell'ultima quotazione:

```
SELECT data.ticker,
       data.min_data,
       data.max_data,
       variazione.var,
        data.min_price,
        data.max_price
FROM firstAndLastData AS data, variazione
WHERE data.ticker=variazione.ticker
SORT BY data.max_data DESC;
```

2.3 Spark

Per l'implementazione del job 1 in Spark (algoritmo 5) è stato necessario leggere l'input file *historical_stock_prices.csv* e creare un RDD con un record per ogni linea del file; inoltre, è stata filtrata la prima linea del file contenente

i nomi dei campi.
Per caricare il file di input è stato utilizzato un RDD denominato input_RDD ed avente i seguenti campi: ticker, open, close, adj_close, low, high, volume, date.
Successivamente è stato creato l'RDD first_date a partire da input_RDD e contenente il ticker, il prezzo di chiusura e la relativa data della prima quotazione di ogni azione. Sempre a partire dall'RDD di input è stato creato l'RDD last_date contenente per ogni azione, il suo ticker, il prezzo di chiusura e la relativa data dell'ultima quotazione.
Dall'RDD di input sono stati poi calcolati due RDD:

- min_price con campi ticker azione e prezzo minimo,
- max_price con campi ticker azione e prezzo massimo,

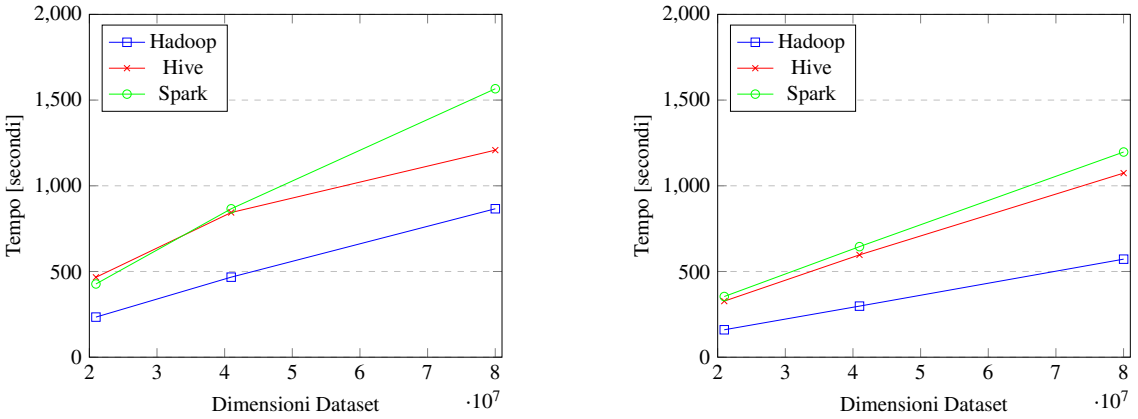
ed è stato eseguito un join tra min_price e max_price per avere un RDD min_max_price con campi ticker azione, prezzo minimo e prezzo massimo.
Il calcolo della variazione percentuale della quotazione dell'azione viene effettuato con un join tra l'RDD first_date e l'RDD last_date così da avere un RDD join_variazione_percentuale con i campi ticker, prezzo di chiusura e relativa data della prima quotazione, prezzo di chiusura e relativa data dell'ultima quotazione sul quale sarà calcolata la variazione percentuale con la formula (Valore finale − Valore iniziale)/Valore iniziale · 100.
Per ottenere l'output finale con i campi richiesti si effettua un join tra min_max_price e join_variazione_percentuale.

Algorithm 5: Spark Job1

```
1  data_min(x, y);
   /* funzione che calcola il minimo tra due valori                    */
2  data_max(x, y);
   /* funzione che calcola il massimo tra due valori                   */
3  input_RDD = spark carica File dall'input_filepath;
4  input_RDD = dall'input_RDD filtra la prima linea;
5  first_date = dall'input_RDD.map((ticker,(close,first_date)) .reduceByKey(data_min);
6  last_date = dall'input_RDD.map((ticker,(close,last_date))) .reduceByKey(data_max);
7  min_price = dall'input_RDD.map(ticker,min_price) .reduceByKey(min_price);
8  max_price = dall'input_RDD.map(ticker,max_price) .reduceByKey(max_price);
9  min_max_price=min_price.join(max_price);
   /* l'RDD prodotto avrà come campi (ticker,(min_price,max_price))    */
10 join_variazione_percentuale = first_date.join(last_date);
   /* (ticker,(closePrice,primaData),(closePrice,ultimaData))          */
11 variazione_percentuale = join_variazione_percentuale.map ((ticker,(( variazione percentuale della
     quotazione));
12 output=min_max_price.join(join_variazione_percentuale).join(variazione_percentuale);
```

2.4 Risultati Job 1

Nella figura 1 sono mostrati due grafici che confrontano i tempi (in secondi) di esecuzione del Job 1 al crescere delle dimensioni del dataset con i tre framework Hadoop, Hive e Spark; in particolare la figura 1a mostra i



(a) Esecuzione in locale

(b) Esecuzione su AWS

Figura 1: Comparazione dei tempi di esecuzione del Job 1 con i tre framework Hadoop, Hive e Spark

risultati ottenuti in locale mentre la figura 1b rappresenta quelli ottenuti con l'utilizzo di un cluster su AWS. Si può osservare che nel confronto il framework che risulta aver ottenuto i tempi migliori è Hadoop Map-Reduce, sia in locale che su AWS.
Tutti i tempi riportati nei grafici sono stati calcolati tramite il comando time ed è stato utilizzato il tempo di esecuzione "reale" (il tempo trascorso dall'avvio al termine del programma).
In Listing 1 a pagina 20 è possibile osservare i primi dieci record ottenuti tramite l'esecuzione degli script sui file del dataset.

3 Job due

Un job che sia in grado di generare un report contenente, per ciascun settore e per ciascun anno del periodo 2009- 2018:

a la variazione percentuale della quotazione del settore nell'anno;

b l'azione del settore che ha avuto il maggior incremento percentuale nell'anno (con indicazione dell'incremento);

c l'azione del settore che ha avuto il maggior volume di transazioni nell'anno (con indicazione del volume).

Il report deve essere ordinato per nome del settore.

3.1 Map-Reduce

Per l'esecuzione del job uno tramite il framework Map-Reduce è stato necessario scrivere due file: *mapper.py* e *reducer.py*.

mapper.py

In questa implementazione (algoritmo 6) il file *mapper.py* si occupa di due operazione: effettuare il join dei dati dei file *historical_stock_prices.csv* e *historical_stocks.csv* associando il settore dell'azienda al rispettivo ticker e della trasmissione dell'output tramite lo Standard Output avendo cura di comunicare solo i dati azionari compresi nell'intervallo temporale che va dal 2009 al 2018.

Algorithm 6: mapper.py in Job 2

```
1  Inizializzazione;
2  lettura del file historical_stocks.csv ed estrazione delle coppie ticker-sector;
3  while non sono terminati i record di historical_stock_prices.csv do
4  |   rimuovi gli spazi iniziali e finali della riga;
5  |   preleva i campi di interesse;
6  |   associa al ticker corrente con il rispettivo settore;
7  |   if anno corrente è compreso tra 2009 e il 2018 then
8  |   |   trasmetti i dati tramite Standard Output;
9  |   end
10 end
```

reducer.py

Il file di reducer.py utilizza le due strutture dati *years_map* e *return_map* per effettuare il calcolo degli elementi richiesti, in particolare la prima è un dizionario nidificato che si occupa di tenere traccia di tutti i valori calcolati per anno, settore e ticker mentre la seconda si occupa di raccogliere solo i valori da restituire per la soluzione del task.
Il codice di questo script può essere diviso nelle seguenti operazioni sulla base dell'informazione che si sta calcolando:

1. una primo blocco che calcola la variazione percentuale del settore in esame relativo all'anno del record che si sta osservando (algoritmo 7, righe 6-19);

2. un secondo blocco che si occupa di calcolare la variazione percentuale relativa all'anno corrente del ticker in osservazione. In questa fase viene anche tenuta traccia del volume del ticker (algoritmo 7, righe 20-34);

3. un ultimo blocco che raccoglie i risultati dell'analisi da restituire (algoritmo 7, riga 35).

4. infine avviene la stampa dei risultati così da poter essere facilmente consultabili.

Lo pseudo-codice 7 è l'implementazione di quanto appena detto nei punti 1, 2 e 3 mentre lo pseudo-codice 8 è relativo al punto 4.

Algorithm 7: reducer.py in Job 2 - Calcolo dei valori.

```
years_map = { } return_map = { }
while mapper.py continua a trasmettere do
    rimuovi gli spazi iniziali e finali della riga;
    preleva i campi di interesse;
    if l'anno corrente non è in years_map then years_map.curr_year uguale a { } ;
    /* Operazioni sui settori                                                    */
    if il settore non è nell'anno corrente in years_map then
        inizializzo first_date, last_date, first_date_count_close, last_date_count_close e
         var_count_close relativi al settore e all'anno corrente;
    else
        if la prima data di comparsa del settore nell'anno corrente è diversa da quella memorizzata
         then
            aggiorno la prima data;
            aggiorno il valore di chiusura cumulativo relativo alla prima data;
            aggiorno la variazione percentuale del settore relativo all'anno corrente;
        end
        if l'ultima data di comparsa del settore nell'anno corrente è diversa da quella memorizzata
         then
            aggiorno l'ultima data;
            aggiorno il valore di chiusura cumulativo relativo all'ultima data;
            aggiorno la variazione percentuale del settore relativo all'anno corrente;
        end
    end
    /* Operazioni sui ticker di ogni settore                                     */
    if il ticker non è nel settore dell'anno corrente in years_map then
        inizializzo i valori associati al ticker;
    else
        if la prima data di comparsa del ticker nell'anno corrente è diversa da quella memorizzata
         then
            aggiorno la prima data;
            aggiorno il valore di chiusura relativo alla prima data;
            aggiorno la variazione percentuale del ticker relativo all'anno corrente;
        end
        if l'ultima data di comparsa del ticker nell'anno corrente è diversa da quella memorizzata
         then
            aggiorno l'ultima data;
            aggiorno il valore di chiusura relativo all'ultima data;
            aggiorno la variazione percentuale del ticker relativo all'anno corrente;
        end
        aggiunta del volume corrente al volume totale di quell'azione nell'anno corrente;
    end
    /* aggiornamento dei valori di ritorno (se sono cambiati)                    */
    if i valori calcolati sono migliori di quelli memorizzati in return_map then aggiorno i valori in
     return_map ;
end
```

Algorithm 8: reducer.py in Job 2 - Stampa dei risultati.

1 ordino i risultati ottenuti per ordine crescente di nome del settore;
2 stampa dei risultati ottenuti;

3.2 Hive

I dati dei file historical_stock_prices.csv e historical_stocks.csv vengono caricati nelle tabelle historical_stock_prices e sectors:

```
CREATE TABLE historical_stock_prices (ticker STRING, open float,
    close float, adj_close float,lowThe float, highThe float, volume
    float, dates date)
CREATE TABLE sectors (ticker STRING, exchage STRING, name STRING,
    sector STRING,industry STRING)
```

Si effettua un join tra le due tabelle historical_stock_prices e sectors sul campo ticker sectors.ticker=hsp.ticker e di conseguenza vengono scartate tutte quelle azioni che non hanno un settore e si selezionano i ticker nel periodo di tempo di interesse (2009-2018).

```
CREATE TABLE sectorYears AS
SELECT hsp.ticker AS ticker, sectors.sector AS sector, hsp.dates AS
    data, hsp.close AS close,hsp.volume AS volume
FROM sectors join historical_stock_prices AS hsp ON sectors.ticker=
    hsp.ticker
WHERE YEAR(hsp.dates) >= '2009';
```

Si crea una tabella che per ciascun settore e per ogni anno,seleziona la data minima e la data massima di ogni azione:

```
CREATE TABLE firstAndLastData AS
SELECT
    sector,
    ticker,
    min(data) AS min_data,
    max(data) AS max_data
FROM sectorYears
GROUP BY sector, ticker, YEAR(data);
```

Per il calcolo della variazione percentuale della quotazione del settore1 nell'anno, per ogni settore, sommo tutti i prezzi di chiusura di tutte le azioni del settore nella data minima dell'anno precedentemente calcolata (valore iniziale) e sommo tutti i prezzi di chiusura di tutte le azioni del settore nella data massima dell'anno precedentemente calcolata (Valore finale) e utilizzo la formula: ((Valore finale-Valore iniziale)/Valore iniziale) x 100

```
CREATE TABLE variazioneSettore AS
SELECT a.sector AS sector, (((b.max_close-a.min_close)/a.min_close)
    * 100) AS varSettore , a.anno AS anno
FROM minClose AS a, maxClose AS b
WHERE a.anno=b.anno AND a.sector=b.sector
```

```
ORDER BY a.sector, anno;
```

Per il calcolo dell'azione del settore che ha avuto il maggior incremento percentuale nell'anno, seleziono per ogni anno,il primo prezzo di chiusura per ogni azione e l'ultimo prezzo di chiusura per ogni azione e calcolo la variazione percentuale con la formula sopra riportata così da avere per ogni azione la propria variazione percentuale per ogni anno dal 2009 al 2018. Seleziono l'azione del settore che ha avuto il maggior incremento percentuale nell'anno:

```
CREATE TABLE variazioneAzione AS
SELECT a.sector AS sector, a.ticker AS ticker, (((b.close-a.close)/a
    .close) * 100) AS varAzione, a.data AS anno
FROM firstAzioneClose AS a, lastAzioneClose AS b
WHERE a.data=b.data AND a.ticker=b.ticker;

CREATE TABLE maxVarAzione AS
SELECT sector,anno,max(varAzione) AS varAzione
FROM variazioneAzione
GROUP BY sector,anno;
```

Per il calcolo dell'azione del settore che ha avuto il maggior volume di transazioni nell'anno, per ogni azione di un settore sommo il volume di transazioni nell'anno e seleziono l'azione del settore che ha avuto il maggior volume di transazioni nell'anno:

```
CREATE TABLE sumVolume AS
SELECT sector, ticker, YEAR(data) AS anno, SUM(volume) AS volume
FROM sectorYears
GROUP BY sector,ticker,YEAR(data);

CREATE TABLE maxVolume AS
SELECT sector, anno, max(volume) AS volume
FROM sumVolume
GROUP BY sector,anno;
```

si effettua la query finale:

```
SELECT a.sector,c.varSettore,a.anno,a.ticker,a.varAzione,b.ticker,b.
    volume
FROM maxVarAzione2 AS a, maxVolume2 AS b , variazioneSettore AS c
WHERE a.sector=b.sector AND b.sector=c.sector AND a.anno=b.anno AND
    b.anno=c.anno
```

3.3 Spark

Per l'implementazione del job due in Spark (algoritmo 9) è stato necessario leggere l'input file *historical_stock_prices.csv* e creare un RDD input_RDD_prices con un record per ogni linea del file historical_stock_prices.csv dal quale sono state filtrate la prima linea[3] e tutti i record nel periodo di tempo di interesse (2009-2018). Inoltre si è letto l'input file historical_stocks.csv e creato un RDD input_RDD_sector con un record per ogni linea

[3] poiché contenente i nomi dei campi

del file historical_stocks.csv dove è stata filtrata la prima linea del file contenente i nomi dei campi. Come input sono stati utilizzati due RDD:

input_RDD_prices avente come campi: ticker, open, close, adj_close, low, high, volume, date

input_RDD_sector avente come campi: ticker,exchange,name,INC.,sector,industry.

Con un join tra questi due RDD è stato creato l'RDD input_RDD contenente, per ogni azione, il ticker, il prezzo di chiusura, il numero di transazioni, la data, il nome dell'azienda ed il settore dell'azienda. A partire dall'RDD input_RDD sono stati creati:

sector_close_min con campi (settore, anno)(data minima nell'anno,la somma dei prezzi di chiusura per settore nella data minima)

sector_close_max con campi (settore, anno)(data max nell'anno,la somma dei prezzi di chiusura per settore nella data max)

ed è stato effettuato un join tra questi per il calcolo della variazione percentuale del settore nell'anno.
Per il calcolo dell'azione del settore che ha avuto il maggior incremento percentuale nell'anno, a partire dall'input_RDD sono stati creati due RDD:

sector_azione_min con campi ((sector, anno)(ticker,min_data nell'anno,close nella data min);

sector_azione_max con campi (sector, anno)(ticker,max_data nell'anno,close nella data max)

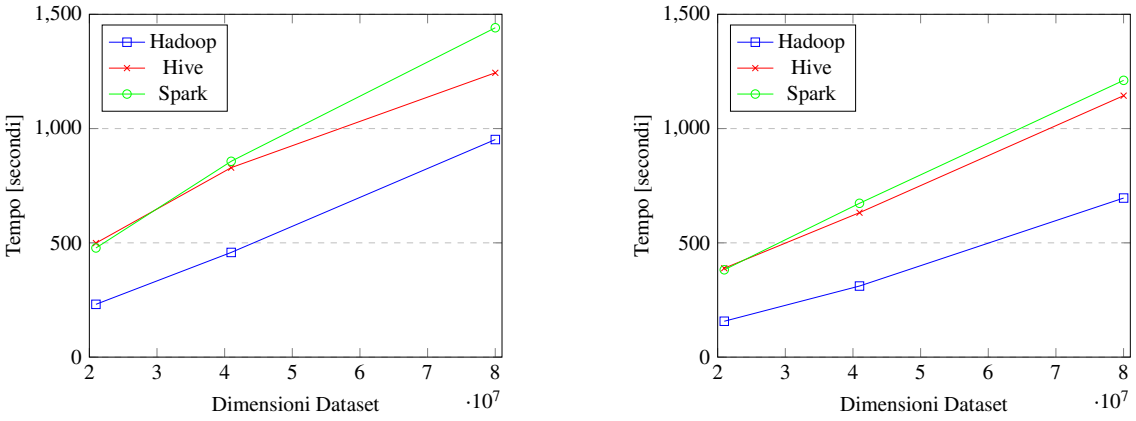
ed effettuo un join tra questi ottenendo un RDD variazione_azione contenente il settore,l'anno, l'azione con la massima variazione in quel settore nell'anno ed il relativo ticker. Per il calcolo dell'azione del settore che ha avuto il maggior volume di transazioni nell'anno è stato utilizzato l'RDD azione_max_volume che a partire dall'input_RDD mappa i campi (sector, anno),(ticker,somma_volume)) e seleziona l'azione con la somma_volume massima utilizzando la funzione max.
Si effettua un join per ottenere l'output finale con i campi richiesti: la variazione percentuale della quotazione del settore nell'anno, l'azione del settore che ha avuto il maggior incremento percentuale nell'anno e l'azione del settore che ha avuto il maggior volume di transazioni nell'anno (con indicazione del volume).

3.4 Risultati Job 2

Nella figura 2 sono mostrati due grafici che confrontano i tempi (in secondi) di esecuzione del Job 1 al crescere delle dimensioni del dataset con i tre framework Hadoop, Hive e Spark; in particolare la figura 2a mostra i risultati ottenuti in locale mentre la figura 2b rappresenta quelli ottenuti con l'utilizzo di un cluster su AWS. Si può osservare che nel confronto il framework che risulta aver ottenuto i tempi migliori è Hadoop Map-Reduce, sia in locale che su AWS; inoltre si osserva che, sebbene utilizzando il servizio di AWS i tempi di esecuzione si riducano, il risparmio temporale non è eccessivo.
Tutti i tempi riportati nei grafici sono stati calcolati tramite il comando time ed è stato utilizzato il tempo di esecuzione "reale" (il tempo trascorso dall'avvio al termine del programma).
In Listing 2 a pagina 21 è possibile osservare i primi dieci record ottenuti tramite l'esecuzione degli script sui file del dataset.

Algorithm 9: Spark Job2

```
1  data_min(x, y);
   /* funzione che calcola il minimo tra due valori                    */
2  data_max(x, y);
   /* funzione che calcola il massimo tra due valori                   */
3  input_RDD_prices = spark carica File dall'input_filepath_prices;
4  input_RDD_prices = dall'input_RDD_prices filtra la prima linea;
5  input_RDD_prices = dall'input_RDD_prices filtra tutti i record nel periodo di tempo di interesse
     (2009-2018);
6  input_RDD_sector = spark carica File dall'input_filepath_sector;
7  input_RDD_sector = dall'input_RDD_sector filtra la prima linea;
8  input_RDD= input_RDD_prices.join(input_RDD_sector);
   /* ticker,((close,volume,date),(name,sector))                       */
9  sector_close_min=input_RDD.map((sector, anno)(min_data nell'anno,somma close per settore
     nella data min));
10 sector_close_max=input_RDD.map((sector, anno)(max_data nell'anno,somma close per settore
     nella data max));
11 variazione_sector=sector_close_min.join(sector_close_max)
12 .map((sector,anno)variazione_settore_nell'anno));
13 sector_azione_min=input_RDD.map(((sector, anno)(ticker,min_data nell'anno,close nella data
     min)));
14 sector_azione_max=input_RDD.map(((sector, anno)(ticker,max_data nell'anno,close nella data
     max)));
15 variazione_azione=sector_azione_min.join(sector_azione_max)
16 .map(((sector,anno)variazione_settore_nell'anno));
17 azione_max_volume = input_RDD.map(((sector, anno),(ticker,somma_volume)));
18 output=variazione_sector.join(variazione_azione).join(azione_max_volume);
```



(a) Esecuzione in locale

(b) Esecuzione su aws.

Figura 2: Comparazione dei tempi di esecuzione del Job 2 con i tre framework Hadoop, Hive e Spark

4 Job tre

Il terzo lavoro richiede di generare le coppie di aziende che si somigliano (sulla base di una soglia scelta a piacere) in termini di variazione percentuale mensile nell'anno 2017 mostrando l'andamento mensile delle due aziende
(es. Soglia=1%, coppie: 1:Apple, Intel: GEN: Apple +2%, Intel +2,5%, FEB: Apple +3%, Intel +2,7%, MAR: Apple +0,5%, Intel +1,2%, ...; 2:Amazon, IBM: GEN: Amazon +1%, IBM +0,5%, FEB: Amazon +0,7%, IBM +0,5%, MAR: Amazon +1,4%, IBM +0,7%, ..)

4.1 Map-Reduce

Per l'esecuzione del job uno tramite il framework Map-Reduce è stato necessario scrivere due file: *mapper.py* e *reducer.py*.

mapper.py

In questa implementazione (algoritmo 10) il file *mapper.py* si occupa di effettuare il join tra i dati dei due file *historical_stock_prices.csv* e *historical_stocks.csv*, in particolare associa il nome dell'azienda al rispettivo ticker del file *historical_stock_prices.csv*. Successivamente trasmette l'output tramite lo Standard Output avendo cura di comunicare solo i dati azionari dell'anno 2017. Il codice seguente può essere letto dividendolo in due parti:

- una prima in cui si effettua la lettura del file *historical_stocks.csv*;
- una seconda in cui si effettua il join dei dati e la trasmissione sullo Standard Output.

Algorithm 10: mapper.py in Job 3

```
1  Inizializzazione;
2  lettura del file historical_stocks.csv ed estrazione delle coppie ticker-sector;
3  while non sono terminati i record di historical_stock_prices.csv do
4  |   rimuovi gli spazi iniziali e finali della riga;
5  |   preleva i campi di interesse;
6  |   associa al ticker corrente con il rispettivo nome;
7  |   if anno corrente è uguale a 2017 then
8  |   |   trasmetti i dati tramite Standard Output;
9  |   end
10 end
```

reducer.py

L'algoritmo utilizzato per la risoluzione del terzo job può essere suddiviso in quattro parti per essere compreso meglio:

1. i dati vengono ricevuti dallo script tramite lo Standard Input e vengono processati calcolando la prima e ultima data di apparizione del ticker per ogni mese e la relativa variazione percentuale. Questi dati vengono salvati nel dizionario nidificato *actions_map*. (Algoritmo 11)

2. si crea un nuovo dizionario di similarità *simil_map* che si inizializza ponendo ogni ticker simile a tutti gli altri ticker presenti in catalogo. Segue un ciclo iterativo su *action_map* in cui si effettuano le verifiche sull'effettiva similarità tra ticker basata su variazione percentuale mensile. Qualora dovesse risultare che due ticker non siano simili in un determinato mese, tale ticker verrebbe rimosso dal dizionario di similarità *simil_map*. (Algoritmo 12)

3. successivamente si esegue un ulteriore ciclo iterativo su *simil_map* andando a rimuovere tutte quelle coppie di elementi considerati simili che non risultano essere presenti in tutti e 12 i mesi; in questa fase si opera anche l'operazione di pulizia delle coppie simmetriche (ad esempio {Apple, Intel} e {Intel, Apple}) che risulterebbero essere solo un'informazione ridondante. (Algoritmo 15)

4. l'ultimo step prevede la stampa dei risultati così da poter essere facilmente consultabili. (Algoritmo 14)

Algorithm 11: reducer.py in Job 3 - processamento dei dati provenienti da mapper.py

```
1  while mapper.py continua a trasmettere do
2      rimuovi gli spazi iniziali e finali della riga;
3      preleva i campi di interesse;
4      if mese corrente non è una chiave di actions_map then
5          aggiungo il mese corrente in actions_map;
6      end
7      if ticker corrente non è una chiave di actions_map[mese_corrente] then
8          aggiungo il ticker corrente in actions_map[mese_corrente];
9      else
10         if la prima data di comparsa del ticker nel mese corrente è diversa da quella memorizzata in
             actions_map[mese_corrente] then
11             aggiorno la prima data;
12             aggiorno il valore di chiusura relativo alla prima data;
13             aggiorno la variazione percentuale del ticker relativo al mese corrente;
14         end
15         if l'ultima data di comparsa del ticker nel mese corrente è diversa da quella memorizzata in
             actions_map[mese_corrente] then
16             aggiorno l'ultima data;
17             aggiorno il valore di chiusura relativo all'ultima data;
18             aggiorno la variazione percentuale del ticker relativo al mese corrente;
19         end
20     end
21 end
```

Algorithm 12: reducer.py in Job 3 - ricerca dei ticker simili sulla base della variazione percentuale mensile nell'anno 2017.

```
1  ticker_list = lista dei ticker presenti in action_map;
2  inizializza ogni chiave ticker in simil_map uguale a ticker_list;
3  foreach mese in action_map do
4      foreach ticker T1 nel mese corrente in action_map do
5          foreach ticker T2 nel mese corrente in action_map do
6              if T1 e T2 sono lo stesso ticker then passa al prossimo ticker;
7              if T2 non è nella lista dei ticker simili di T1 then passa al prossimo ticker; // Elemento
                già rimosso dalla lista perché già considerato non simile.
8              ;
9              if T1 e T2 non hanno variazione percentuale simile sulla base di una soglia preposta
                then
10                 elimino T2 dalla lista delle similarità di T1 in simil_map
11             end
12         end
13     end
14 end
```

Algorithm 13: reducer.py in Job 3 - rimozione dei falsi-simili, delle ridondanze e degli elementi che non hanno similarità.

```
1  foreach ticker T1 in simil_map do
2      foreach ticker T2 in simil_map do
3          if in simil_map sono presenti entrambe le coppie (T1,T2) e (T2,T1) then
4              elimino T1 dalla lista delle similarità di T2 in simil_map
5          end
6          if T2 non è presente in tutti e 12 i mesi dell'anno 2017 then
7              rimuovo la chiave T2 da simil_map ed elimino T2 dalla lista delle similarità di T1 in
                simil_map
8          end
9          if la lista delle somiglianze associata a T1 in simil_map è vuota then
10             rimuovo la chiave T1 da simil_map
11         end
12     end
13 end
```

Algorithm 14: reducer.py in Job 3

```
1  ordino i risultati ottenuti per ordine crescente di nome del ticker;
2  stampa dei risultati ottenuti;
```

4.2 Hive

I dati dei file historical_stock_prices.csv e historical_stocks.csv vengono caricati nelle tabelle historical_stock_prices e sectors:

```
CREATE TABLE historical_stock_prices (ticker STRING, open float,
    close float, adj_close float,lowThe float, highThe float, volume
    float, dates date)
CREATE TABLE sectors (ticker STRING, exchage STRING, name STRING,
    sector STRING,industry STRING)
```

Si effettua un Join tra le due tabelle historical_stock_prices e sectors sul campo ticker sectors.ticker=hsp.ticker e di conseguenza vengono scartate tutte quelle azioni che non hanno un settore. Si selezionano solo i ticker nel periodo di tempo di interesse (2017); quindi si seleziona l'anno 2017 e per ogni azienda, prendo le azioni in quell'anno.

```
SELECT hsp.ticker AS ticker, hsp.close AS close, hsp.dates, sectors.
    name AS name
FROM historical_stock_prices AS hsp JOIN sectors ON hsp.ticker=
    sectors.ticker
WHERE YEAR(hsp.dates) = '2017';
```

per ogni mese seleziono la prima e l'ultima data di chiusura dell'azione; ad esempio per l'azione con ticker ZYNE: la prima data di chiusura per il mese di novembre è 2017-11-01; l'ultima data di chiusura per il mese di novembre è 2017-11-30; la prima data di chiusura per il mese di ottobre è 2017-10-02; l'ultima data di chiusura per il mese di ottobre è 2017-10-31.

```
SELECT ticker, min((dates)) AS min_data , max((dates)) AS max_data
FROM azioniAzienda
GROUP BY ticker,MONTH(dates);
```

per ogni mese prendo il primo prezzo di chiusura dell'azione; ad esempio per l'azienda NUVEEN MUNICIPAL CREDIT INCOME FUND il primo prezzo di chiusura nel mese di agosto è 15.26 mentre il primo prezzo di chiusura nel mese di settembre è 15.34:
–NZF 8 15.26 NUVEEN MUNICIPAL CREDIT INCOME FUND
–NZF 9 15.34 NUVEEN MUNICIPAL CREDIT INCOME FUND

```
SELECT b.ticker,MONTH(b.min_data) AS mese ,a.close AS close ,a.name
FROM azioniAzienda AS a , firstAndLastMese AS b
WHERE a.dates=b.min_data AND a.ticker=b.ticker;
```

per ogni mese prendo l'ultimo prezzo di chiusura dell'azione; ad esempio sempre per l'azienda NUVEEN MUNICIPAL CREDIT INCOME FUND l'ultimo prezzo di chiusura nel mese di agosto è 15.3 mentre l'ultimo prezzo di chiusura nel mese di settembre è 15.21:
–NZF 8 15.3 NUVEEN MUNICIPAL CREDIT INCOME FUND
–NZF 9 15.21 NUVEEN MUNICIPAL CREDIT INCOME FUND

```
SELECT b.ticker,MONTH(b.max_data) AS mese,a.close AS close, a.name
FROM azioniAzienda AS a , firstAndLastMese AS b
WHERE a.dates=b.max_data AND a.ticker=b.ticker;
```

per ogni mese del 2017 calcolo la variazione percentuale mensile con la formula (Prezzo chiusura finale mese-Prezzo chiusura iniziale mese)/Prezzo chiusura iniziale mese) x 100. Per l'azienda AKORN, INC. la variazione percentuale del mese di gennaio è stata del -13.45:
–AKRX 1 -13.45 AKORN, INC.

```
SELECT a.ticker,a.mese,(((b.close-a.close)/a.close) * 100) AS
    variazione , a.name
FROM firstCLose AS a , lastCLose AS b
WHERE a.ticker=b.ticker AND a.mese=b.mese;
```

creo la tabella per la query finale dove selezione,i nomi delle 2 aziende che hanno come differenza delle loro variazioni percentuali per ogni mese minore dell'1%. Per generare le coppie di aziende che si somigliano sulla base di una soglia dell'1%, seleziono tutte le aziende che per 12 mesi hanno la variazione percentuale simile:

```
CREATE TABLE finale AS
SELECT
    a1.name AS name1,
    a2.name AS name2,
    a1.mese,
    a1.variazione as variazione1,
        a2.variazione as variazione2
FROM variazionePercentuale AS a1 JOIN  variazionePercentuale AS a2
    ON
        (a1.mese=a2.mese)
WHERE a1.name!=a2.name AND (a1.variazione-a2.variazione<1) AND (a1.
    variazione-a2.variazione>0)
SORT BY name1,name2 DESC;
```

4.3 Spark

Per lo svolgimento del job è stato necessario creare i seguenti RDD:

hsp_RDD trattasi del RDD contenete i record provenienti dal file *historical_stock_prices.csv*;

hs_RDD è il RDD contenente i record provenienti dal file *historical_stocks.csv*;

input_RDD RDD generato tramite l'operazione di join di hsp_RDD e hs_RDD

fisrt_month_date_RDD questo RDD contiene tutte le informazioni relative alla prima data di apparizione di un ticker per ogni mese;

last_month_date_RDD questo RDD contiene tutte le informazioni relative all'ultima data di apparizione di un ticker per ogni mese;

percentage_month_RDD RDD contenete le variazioni percentuali di ogni ticker per ogni mese e generato tramite l'operazione di join tra fisrt_month_date_RDD con last_month_date_RDD;

similar_RDD questo RDD è un raccoglitore che contiene tutte le possibili combinazioni di coppie di aziende;

grouped_similar_RDD si tratta del RDD finale contenete le effettive coppie di aziende simili sulla base della variazione percentuale mensile nell'anno 2017.

L'algoritmo 15 utilizzato per la generazione dei risultati segue questi passi:

1. carico i file *historical_stock_prices.csv* e *historical_stocks.csv* dentro i rispettivi RDD *hsp_RDD* e *hs_RDD* facendo attenzione a prendere solo i record dei dati relativi all'anno 2017. Segue il join di questi due RDD generando *input_RDD*.

2. a partire da *input_RDD*, tramite l'utilizzo della funzione *map*, genero *fisrt_month_date_RDD* e *last_month_date_RDD* e ne effettuo il join creando *percentage_month_RDD*; su quest'ultimo si esegue la funzione *map* per calcolare la variazione percentuale mensile per ogni ticker.

3. utilizzando la funzione *cartesian* su *percentage_month_RDD* con se stesso è possibile creare tutte le possibili combinazioni di ticker; effettuando una scrematura basata sulla similarità di variazione percentuale e sulla disuguaglianza del ticker si genera *similar_RDD* contenete solo le coppie simili;

4. tramite l'utilizzo della funzione *reduceByKey* è possibile raggruppare tutti gli elementi con la stessa coppia di ticker come chiave generando *grouped_similar_RDD*. Effettuando l'operazione di filtraggio un'ultima volta è possibile eliminare gli elementi che sono simili solo in maniera parziale;

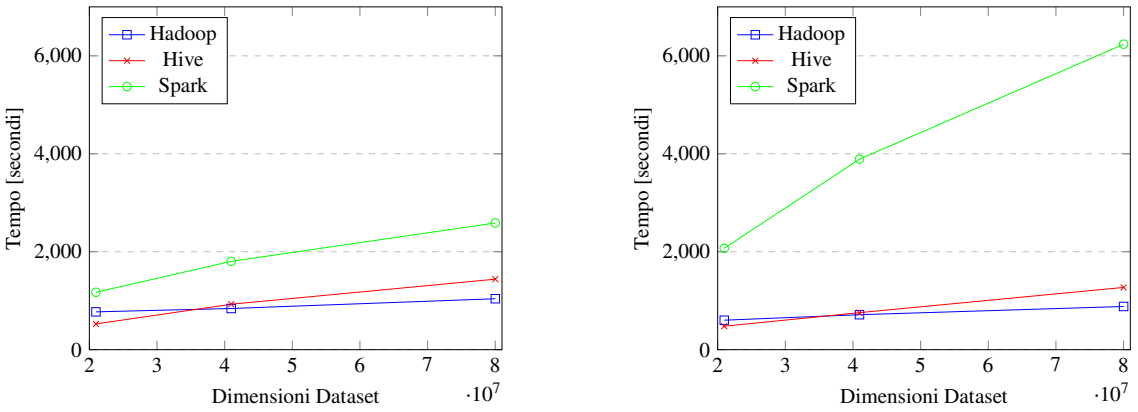
5. si stampano i record di *grouped_similar_RDD* per visualizzare i risultati.

Algorithm 15: Spark in Job 3.

```
1  def input = legge i file in ingresso generando un RDD;
2  hsp_RDD = input(historical_stock_prices.csv).filter(l'anno deve essere 2017).map((ticker, (close,
     date)));
3  hs_RDD = input(historical_stocks.csv).map((ticker, name));
4  input_RDD = hsp_RDD.join(hs_RDD).map((ticker, name, close, date));
5  input_RDD.persiste(memoria e disco);
6  fisrt_month_date_RDD = input_RDD.map(((ticker, name, month),(first_day, close)));
7  last_month_date_RDD = input_RDD.map(((ticker, name, month),(last_day, close)));
8  percentage_month_RDD = fisrt_month_date_RDD.join(last_month_date_RDD).map(((ticker,
     name, month), change_perc));
9  similar_RDD = percentage_month_RDD.cartesian(percentage_month_RDD).filter(stesso mese e
     ticker diverso).map((ticker_1, name_1, ticker_1, name_2, month), change_perc_1,
     change_perc_2);
10 grouped_similar_RDD = similar_RDD.reduceByKey(accorpa tutti i mesi di tutti i ticker
     simili).filter(elementi con esattamente 12 mesi in comune).map(((name_1, name_2), [lista dei mesi
     con le relative variazioni percentuali]));
11 salva come file di testo di grouped_similar_RDD;
```

4.4 Risultati Job 3

Nella figura 3 sono mostrati due grafici che confrontano i tempi (in secondi) di esecuzione del Job 1 al crescere delle dimensioni del dataset con i tre framework Hadoop, Hive e Spark; in particolare la figura 3a mostra i



(a) Esecuzione in locale.

(b) Esecuzione su aws.

Figura 3: Comparazione dei tempi di esecuzione del Job 3 con i tre framework Hadoop, Hive e Spark

risultati ottenuti in locale mentre la figura 3b rappresenta quelli ottenuti con l'utilizzo di un cluster su AWS. Si può osservare che nel confronto i frameworks che risultano aver ottenuto i tempi migliori sono Hadoop Map-Reduce e Hive, sia in locale che su AWS. Si osservi, invece, come Spark abbia avuto un andamento anomali in AWS arrivando a duplicare i tempi di esecuzione.
Tutti i tempi riportati nei grafici sono stati calcolati tramite il comando time ed è stato utilizzato il tempo di esecuzione "reale" (il tempo trascorso dall'avvio al termine del programma).
In Listing 3 a pagina 22 è possibile osservare i primi dieci record ottenuti tramite l'esecuzione degli script sui file del dataset.

5 Prime dieci stampe dei vari job

Job 1

Listing 1: Prime 10 righe del risultato di Job 1 ordinate per valori decrescenti di data dell'ultima quotazione.

```
Ticker: A, first_date: 1999-11-18 00:00:00, last_date: 2018-08-24
    00:00:00, var: 109.63646513864809, max_price: 115.879829406738,
    min_price: 7.51072978973389, first_close: 31.4735336303711,
    last_close: 65.9800033569336, days_of_growth: 12, year_of_growth:
     2001
Ticker: AA, first_date: 1970-01-02 00:00:00, last_date: 2018-08-24
    00:00:00, var: 508.32539151728577, max_price: 117.194313049316,
    min_price: 3.60450005531311, first_close: 7.14091491699219,
    last_close: 43.439998626709, days_of_growth: 13, year_of_growth:
    1989
Ticker: AABA, first_date: 1996-04-12 00:00:00, last_date: 2018-08-24
     00:00:00, var: 4910.909201882102, max_price: 125.03125,
    min_price: 0.645833313465118, first_close: 1.375, last_close:
    68.9000015258789, days_of_growth: 11, year_of_growth: 2003
```

```
Ticker: AAC, first_date: 2018-01-16 00:00:00, last_date: 2018-08-24
   00:00:00, var: 4.8565072848188215, max_price: 12.960000038147,
   min_price: 7.78999996185303, first_close: 9.0600004196167,
   last_close: 9.5, days_of_growth: 5, year_of_growth: 2018
Ticker: AAL, first_date: 2005-09-27 00:00:00, last_date: 2018-08-24
   00:00:00, var: 101.139902743005, max_price: 63.2700004577637,
   min_price: 1.45000004768372, first_close: 19.2999992370605,
   last_close: 38.8199996948242, days_of_growth: 9, year_of_growth:
   2016
Ticker: AAME, first_date: 1980-03-17 00:00:00, last_date: 2018-08-24
    00:00:00, var: -29.870126894421368, max_price: 15.8000001907349,
    min_price: 0.375, first_close: 3.84999990463257, last_close:
   2.70000004768372, days_of_growth: 31, year_of_growth: 1990
Ticker: AAN, first_date: 1987-01-20 00:00:00, last_date: 2018-08-24
   00:00:00, var: 4683.26319686329, max_price: 51.5299987792969,
   min_price: 0.481481492519379, first_close: 1.05555558204651,
   last_close: 50.4900016784668, days_of_growth: 24, year_of_growth:
    1995
Ticker: AAOI, first_date: 2013-09-26 00:00:00, last_date: 2018-08-24
    00:00:00, var: 330.42167437441736, max_price: 103.410003662109,
   min_price: 8.07999992370605, first_close: 9.96000003814697,
   last_close: 42.8699989318848, days_of_growth: 7, year_of_growth:
   2014
Ticker: AAON, first_date: 1992-12-16 00:00:00, last_date: 2018-08-24
    00:00:00, var: 41348.203536198605, max_price: 43.2999992370605,
   min_price: 0.0897707939147949, first_close: 0.0997630655765533,
   last_close: 41.3499984741211, days_of_growth: 12, year_of_growth:
    1999
Ticker: AAP, first_date: 2001-11-29 00:00:00, last_date: 2018-08-24
   00:00:00, var: 1084.1498505417885, max_price: 201.240005493164,
   min_price: 12.3299999237061, first_close: 13.8800001144409,
   last_close: 164.360000610352, days_of_growth: 10, year_of_growth:
    2002
```

Job 2

Listing 2: Prime 10 righe del risultato di Job 2 ordinate per nome del settore.

```
Settore: BASIC INDUSTRIES
        anno: 2009,     variazione quotazione del settore: 36.02%,
                azione con il maggior incremento percentuale: FOE
           (1005.49%),   azione con il maggior volume: FCX
           (9141685400)
        anno: 2016,     variazione quotazione del settore: 20.35%,
                azione con il maggior incremento percentuale: CLF
```

```
   (723.02%),    azione con il maggior volume: FCX
   (10464699500)
anno: 2013,     variazione quotazione del settore: 201.54%,
        azione con il maggior incremento percentuale: XRM
   (416.93%),    azione con il maggior volume: VALE
   (4428233700)
anno: 2015,     variazione quotazione del settore: -2.05%,
         azione con il maggior incremento percentuale: SUM
   (35191.63%),  azione con il maggior volume: FCX
   (7286761300)
anno: 2012,     variazione quotazione del settore: 2.35%,
          azione con il maggior incremento percentuale: PATK
    (261.86%),   azione con il maggior volume: VALE
   (4659766700)
anno: 2011,     variazione quotazione del settore: -24.47%,
        azione con il maggior incremento percentuale: BLD
   (360.42%),    azione con il maggior volume: FCX
   (5150807800)
anno: 2010,     variazione quotazione del settore: 28.57%,
         azione con il maggior incremento percentuale: BLD
   (519.80%),    azione con il maggior volume: FCX
   (6891808600)
anno: 2014,     variazione quotazione del settore: -82.93%,
        azione con il maggior incremento percentuale: BLD
   (901.40%),    azione con il maggior volume: VALE
   (5660183200)
anno: 2017,     variazione quotazione del settore: 16.81%,
         azione con il maggior incremento percentuale: VRS
   (421.36%),    azione con il maggior volume: VALE
   (7023267600)
```

Job 3

Listing 3: Prime 10 righe del risultato di Job 3.

```
Soglia=1%, coppie:
(ISHARES ASIA 50 ETF,ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX
   FUND):        1: ISHARES ASIA 50 ETF 6.454371444052506%, ISHARES
   MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND 6.345773175231027% 2:
   ISHARES ASIA 50 ETF 1.925783206730231%, ISHARES MSCI ALL COUNTRY
   ASIA EX JAPAN INDEX FUND 2.58926853163986%  3: ISHARES ASIA 50
   ETF 2.61872229927329%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN
   INDEX FUND 2.843133728885793%  4: ISHARES ASIA 50 ETF
   1.2050449676019408%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX
    FUND 1.2775986421122896%       5: ISHARES ASIA 50 ETF
   3.928841331637542%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX
```

FUND 3.6324185560334827% 8: ISHARES ASIA 50 ETF 0.543654056358565%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND 1.1508767646655502% 6: ISHARES ASIA 50 ETF 0.8111428176667842%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND 0.10388794549951344% 7: ISHARES ASIA 50 ETF 5.207968330413449%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND 4.7464131522057205% 9: ISHARES ASIA 50 ETF 0.39260242037650284%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND -0.2899338445228372% 10: ISHARES ASIA 50 ETF 4.89782754660588%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND 3.9045172055576205% 12: ISHARES ASIA 50 ETF 1.4127735413733329%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND 1.5306822413421393% 11: ISHARES ASIA 50 ETF 0.2290682480066195%, ISHARES MSCI ALL COUNTRY ASIA EX JAPAN INDEX FUND -0.4344415943024396%

(BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST,BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST): 1: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -0.39603608879972446%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST 0.5347588469385491% 2: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST 0.13244704360367243%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST -0.464188960069286% 3: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST 0.6640068061694412%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST 0.5329774982425756% 4: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST 0.3317862756962973%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST 0.13262903093360015% 5: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -0.5928863687651881%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST 0.33134652239433987% 8: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -0.1330639018278032%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST -0.13262270683411598% 6: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -0.06631451546747741%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST -0.19854224786723917% 7: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -0.661371819595201%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST 0.0% 9: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST 0.13315248654977796%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST -0.06644670269211185% 10: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -0.1331588359130991%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST -0.0664908836912545 7% 12: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -2.738809887098793%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST -3.2214735203043374% 11: BLACKROCK CALIFORNIA MUNICIPAL 2018 TERM TRUST -0.2664887491209624%, BLACKROCK FLORIDA MUNICIPAL 2020 TERM TRUST -0.9327137987201097%

(BLACKROCK MUNICIPAL 2018 TERM TRUST,BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST): 1: BLACKROCK MUNICIPAL 2018 TERM TRUST

0.7343165134381459%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST
0.33783912223258256% 2: BLACKROCK MUNICIPAL 2018 TERM TRUST
-0.39682186770189876%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM
TRUST 0.26791668476478253% 3: BLACKROCK MUNICIPAL 2018 TERM TRUST
0.13262903093360015%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM
TRUST -0.6675529429320063% 4: BLACKROCK MUNICIPAL 2018 TERM
TRUST -0.06640258036628366%, BLACKROCK NEW YORK MUNICIPAL 2018
TERM TRUST 0.5387200111181634% 5: BLACKROCK MUNICIPAL 2018 TERM
TRUST 0.1330702469663086%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM
TRUST 0.0% 8: BLACKROCK MUNICIPAL 2018 TERM TRUST
-0.06652877834431796%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM
TRUST 0.13423126359497076% 6: BLACKROCK MUNICIPAL 2018 TERM TRUST
0.0%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST
0.40322221425369253% 7: BLACKROCK MUNICIPAL 2018 TERM TRUST
0.13324119776439058%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM
TRUST 0.06693593817050207% 9: BLACKROCK MUNICIPAL 2018 TERM
TRUST -0.06649088369125457%, BLACKROCK NEW YORK MUNICIPAL 2018
TERM TRUST 0.33624876341362103% 10: BLACKROCK MUNICIPAL 2018 TERM
TRUST -0.19973825386931748%, BLACKROCK NEW YORK MUNICIPAL 2018
TERM TRUST -0.5340449008722518% 12: BLACKROCK MUNICIPAL
2018 TERM TRUST -1.3360040460760443%, BLACKROCK NEW YORK
MUNICIPAL 2018 TERM TRUST -0.8097158436582766% 11: BLACKROCK
MUNICIPAL 2018 TERM TRUST -0.2668443126341444%, BLACKROCK NEW
YORK MUNICIPAL 2018 TERM TRUST -0.40214758250593263%
(VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF,
VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF):
1: VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF
-0.13588710870728915%, VICTORYSHARES US EQ INCOME ENHANCED
VOLATILITY WTD ETF 0.15686888995953238% 2: VICTORYSHARES US
LARGE CAP HIGH DIV VOLATILITY WTD ETF 4.125840424028095%,
VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF
4.130334015198183% 3: VICTORYSHARES US LARGE CAP HIGH DIV
VOLATILITY WTD ETF -1.0424037816927028%, VICTORYSHARES US EQ
INCOME ENHANCED VOLATILITY WTD ETF -0.9921557537191644% 4:
VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF
0.6496799009988883%, VICTORYSHARES US EQ INCOME ENHANCED
VOLATILITY WTD ETF 0.5620558770055021% 5: VICTORYSHARES US
LARGE CAP HIGH DIV VOLATILITY WTD ETF 1.198178781415834%,
VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF
1.1485680190931131% 8: VICTORYSHARES US LARGE CAP HIGH DIV
VOLATILITY WTD ETF -0.373918775388581%, VICTORYSHARES US EQ
INCOME ENHANCED VOLATILITY WTD ETF -0.318832589694607% 6:
VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF
-0.9154915577880152%, VICTORYSHARES US EQ INCOME ENHANCED
VOLATILITY WTD ETF -0.7637768975195334% 7: VICTORYSHARES US
LARGE CAP HIGH DIV VOLATILITY WTD ETF 0.636784512617544%,

VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF 0.8289215008561609% 9: VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF 1.1338678404527491%, VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF 1.0816314794909925% 10: VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF 0.5613054390368701%, VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF 0.49283428250720873% 12: VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF 0.09278036460738819%, VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF -0.25778503512661316% 11: VICTORYSHARES US LARGE CAP HIGH DIV VOLATILITY WTD ETF 3.728264427966525%, VICTORYSHARES US EQ INCOME ENHANCED VOLATILITY WTD ETF 3.78089928154419%

(VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF,GLOBAL X S&P 500 CATHOLIC VALUES ETF): 1: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.404632008636316%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 1.388378330708319% 2: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 3.8704875049168774%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 3.753161132762598% 3: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF -1.2754303951748462%, GLOBAL X S&P 500 CATHOLIC VALUES ETF -1.0638279767858223% 4: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.6935215467819682%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 1.254350848198478% 5: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.1126554219460114%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 1.2324567669804185% 8: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF -0.4138270119977686%, GLOBAL X S&P 500 CATHOLIC VALUES ETF -0.1675942147874333% 6: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 0.022682216692087625%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 0.26899772804871636% 7: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.071346673627285%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 1.5554445407848831% 9: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 2.219228536705685%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 1.8985268025882545% 10: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.6572139368041243%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 2.2122430743647% 12: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 0.8889814237565469%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 0.335253858732965% 11: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 4.17337134197699%, GLOBAL X S&P 500 CATHOLIC VALUES ETF 3.2987783440963385%

(VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF,VICTORYSHARES US 500 VOLATILITY WTD ETF): 1: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.404632008636316%, VICTORYSHARES US 500 VOLATILITY WTD ETF 1.4261142909652165% 2: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 3.8704875049168774%, VICTORYSHARES US 500 VOLATILITY WTD ETF 3.993180915107678 3%: 3:

VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF -1.2754303951748462%, VICTORYSHARES US 500 VOLATILITY WTD ETF -1.340106360249816% 4: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.6935215467819682%, VICTORYSHARES US 500 VOLATILITY WTD ETF 1.5772174926142388% 5: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.1126554219460114%, VICTORYSHARES US 500 VOLATILITY WTD ETF 1.0884697021337435% 8: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF -0.4138270119977686%, VICTORYSHARES US 500 VOLATILITY WTD ETF -0.4689573251390857% 6: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 0.022682216692087625%, VICTORYSHARES US 500 VOLATILITY WTD ETF 0.24965227468416853% 7: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.071346673627285%, VICTORYSHARES US 500 VOLATILITY WTD ETF 0.881952462420806% 9: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 2.219228536705685%, VICTORYSHARES US 500 VOLATILITY WTD ETF 2.2396415807968237% 10: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 1.6572139368041243%, VICTORYSHARES US 500 VOLATILITY WTD ETF 1.655409113611740 1% 12: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 0.8889814237565469%, VICTORYSHARES US 500 VOLATILITY WTD ETF 0.7018995917155837% 11: VICTORYSHARES US 500 ENHANCED VOLATILITY WTD ETF 4.17337134197699%, VICTORYSHARES US 500 VOLATILITY WTD ETF 4.33355011905479%

(ISHARES INTERMEDIATE CREDIT BOND ETF,WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F): 1: ISHARES INTERMEDIATE CREDIT BOND ETF 0.4621499110056792%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F -0.1245020965004686% 2: ISHARES INTERMEDIATE CREDIT BOND ETF 0.775405080487313%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F 0.2502142386838668% 3: ISHARES INTERMEDIATE CREDIT BOND ETF 0.3222525293749374%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F -0.16597889997589857% 4: ISHARES INTERMEDIATE CREDIT BOND ETF 0.5135707574212331%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F -0.08327524848355213% 5: ISHARES INTERMEDIATE CREDIT BOND ETF 0.7048024192823745%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F -0.08354409907049686% 8: ISHARES INTERMEDIATE CREDIT BOND ETF 0.4083526225943165%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F -0.12505496845239197% 6: ISHARES INTERMEDIATE CREDIT BOND ETF 0.009111903559159472%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F 0.02086025760387139 7% 7: ISHARES INTERMEDIATE CREDIT BOND ETF 0.8137509072850516%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F 0.2714576524788984% 9: ISHARES INTERMEDIATE CREDIT BOND ETF -0.05443436574987267%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U

.S. AGGREGATE BOND F 0.355503941295507% 10: ISHARES INTERMEDIATE CREDIT BOND ETF 0.018195307359741363%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F 0.3538679552573689% 12: ISHARES INTERMEDIATE CREDIT BOND ETF -0.12802870745455888%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F 0.33160590137223267% 11: ISHARES INTERMEDIATE CREDIT BOND ETF -0.21875659992049581%, WISDOMTREE BARCLAYS INTEREST RATE HEDGED U.S. AGGREGATE BOND F -0.10386214726719675%

(ISHARES INTERMEDIATE CREDIT BOND ETF,BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST): 1: ISHARES INTERMEDIATE CREDIT BOND ETF 0.4621499110056792%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.33783912223258256% 2: ISHARES INTERMEDIATE CREDIT BOND ETF 0.775405080487313%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.26791668476478253% 3: ISHARES INTERMEDIATE CREDIT BOND ETF 0.3222525293749374%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST -0.6675529429320063% 4: ISHARES INTERMEDIATE CREDIT BOND ETF 0.5135707574212331%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.5387200111181634% 5: ISHARES INTERMEDIATE CREDIT BOND ETF 0.7048024192823745%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.0% 8: ISHARES INTERMEDIATE CREDIT BOND ETF 0.4083526225943165%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.13423126359497076% 6: ISHARES INTERMEDIATE CREDIT BOND ETF 0.009111903559159472%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.40322221425369253% 7: ISHARES INTERMEDIATE CREDIT BOND ETF 0.8137509072850516%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.06693593817050207% 9: ISHARES INTERMEDIATE CREDIT BOND ETF -0.05443436574987267%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST 0.33624876341362103% 10: ISHARES INTERMEDIATE CREDIT BOND ETF 0.018195307359741363%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST -0.5340449008722518% 12: ISHARES INTERMEDIATE CREDIT BOND ETF -0.12802870745455888%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST -0.8097158436582766% 11: ISHARES INTERMEDIATE CREDIT BOND ETF -0.21875659992049581%, BLACKROCK NEW YORK MUNICIPAL 2018 TERM TRUST -0.40214758250593263%

(ISHARES U.S. CREDIT BOND ETF,ISHARES INTERMEDIATE CREDIT BOND ETF): 1: ISHARES U.S. CREDIT BOND ETF -0.04573728248160165%, ISHARES INTERMEDIATE CREDIT BOND ETF 0.4621499110056792% 2: ISHARES U.S. CREDIT BOND ETF 0.943481539408875%, ISHARES INTERMEDIATE CREDIT BOND ETF 0.775405080487313% 3: ISHARES U.S. CREDIT BOND ETF 0.42028938025876433%, ISHARES INTERMEDIATE CREDIT BOND ETF 0.3222525293749374% 4: ISHARES U.S. CREDIT BOND ETF 0.6911623704599653%, ISHARES INTERMEDIATE CREDIT BOND ETF 0.5135707574212331% 5: ISHARES U.S. CREDIT BOND ETF 1.2336725659516306%, ISHARES INTERMEDIATE CREDIT BOND ETF

```
0.7048024192823745%   8: ISHARES U.S. CREDIT BOND ETF
0.606655658302606%, ISHARES INTERMEDIATE CREDIT BOND ETF
0.4083526225943165%    6: ISHARES U.S. CREDIT BOND ETF
0.2603473535479242%, ISHARES INTERMEDIATE CREDIT BOND ETF
0.009111903559159472% 7: ISHARES U.S. CREDIT BOND ETF
0.972186395406005%, ISHARES INTERMEDIATE CREDIT BOND ETF
0.8137509072850516%    9: ISHARES U.S. CREDIT BOND ETF
-0.07122625180208522%, ISHARES INTERMEDIATE CREDIT BOND ETF
-0.05443436574987267%       10: ISHARES U.S. CREDIT BOND ETF
0.09832017612839408%, ISHARES INTERMEDIATE CREDIT BOND ETF
0.018195307359741363%       12: ISHARES U.S. CREDIT BOND ETF
0.18780100612244757%, ISHARES INTERMEDIATE CREDIT BOND ETF
-0.12802870745455888%       11: ISHARES U.S. CREDIT BOND ETF
-0.20562872657723472%, ISHARES INTERMEDIATE CREDIT BOND ETF
-0.21875659992049581%
```