



# TP Session Partie 1 BD 6 2019

*Traitement de données  
massives*

Juan Hernandez, Jasmin Parent

---

## INTRODUCTION

Le traitement de mégadonnées exige l'utilisation d'outils spécialisés, tels que les plateformes MapReduce, Hive et Pig, ainsi que des langages appropriés au traitement de mégadonnées, tels que Python, HiveQL et Pig Latin. Dans le cadre de notre projet intégrateur de notre AEC, notre équipe a développé le code pour traiter les jeux de données qui y sont reliés. Le jeu de données principal pour ce travail pratique consiste en l'ensemble des meilleurs prix d'achat de 69 cryptomonnaies, récoltés à des intervalles de 10 secondes du 2 février 2019 au 16 février 2019, la période d'étude (voir fichier prices2.csv). Ces données ont été obtenues de l'échange de cryptomonnaies Binance. Ce fichier montre les prix des cryptomonnaies en termes de Bitcoin, à l'exception du prix de Bitcoin, lequel est montré en termes de dollars américains. Nous avons développé du code MapReduce en python pour exploiter ce premier jeu de données. De plus, pour les requêtes Hive nous avons utilisé un deuxième jeu de données qui consiste en l'ensemble de toutes les transactions achat-vente qui ont eu lieu pendant la période d'étude (voir fichier transactions.csv). Et, pour exécuter notre code Pig Latin nous avons utilisé un troisième jeu de données lequel, comme le fichier prices2.csv, contient des prix des cryptomonnaies analysées, mais sans en têtes pour faciliter son traitement.

## DESCRIPTION DU CODE

### Code MapReduce avec Python

1. Nous avons développé un premier script MapReduce en python que nous avons nommé « Find highest high ». Ce script contient du code pour trouver le prix maximal que chaque cryptomonnaie a atteint durant la période d'étude ainsi que le moment où elle a atteint ce prix. Nous considérons que d'un point de vue d'analyse d'investissements il est intéressant de mettre les prix de toutes les cryptomonnaies sur une même base. Nous avons donc décidé de convertir les prix de toutes les cryptomonnaies en termes de dollars américains. Notre code donc effectue cette conversion. Pour exécuter notre code nous avons utilisé le jeu de données prices2.csv. Ci-dessous nous montrons notre code avec des commentaires le décrivant.

```
from mrjob.job import MRJob
from mrjob.step import MRStep
from datetime import datetime

class HighestHigh(MRJob):
```

"""Find the highest price of each coin and the time at which it occurred"""

```
def configure_options(self):
    super(HighestHigh, self).configure_options()
    self.add_file_option('--names', help='Names of the coin')

def steps(self):
    return [
        MRStep(mapper=self.mapper_get_prices,
                reducer_init=self.reducer_init,
                reducer=self.reducer_get_high)
    ]

# A line is the price for 69 coins and the time
def mapper_get_prices(self, _, line):
    fields = line.split(',')
    time = fields[0]
    prices = fields[1:]
    btc_price = fields[1]

    if btc_price != 'BTCUSDT': # Skip header
        for idx, price in enumerate(prices):
            if idx != 0:
                # Convert alt coin btc price to US Dollar
                yield idx, (time, float(price) * float(btc_price))
            else:
                yield idx, (time, float(btc_price))

def reducer_init(self):
    with open('names.txt', 'U') as f:
```

```

self.names = f.read().split('\n')

def reducer_get_high(self, idx, values):
    _max = 0
    max_time = 0
    for time, price in values:
        if price > _max:
            _max = price
            max_time = time

    max_time_human = datetime.fromtimestamp(int(max_time)/1000.0)
    rounded_max = round(_max, 2)
    output = '{0} $ {1}'.format(rounded_max, max_time_human)

    yield self.names[idx], output

if __name__ == '__main__':
    HighestHigh.run()

```

Pour exécuter le code ci-dessus nous avons utilisé la commande suivante dans la console de Canopy :

```

!python
home/cloudera/BD6_projet_session/1_best_returns/find_highest_high.py --
names=/home/cloudera/BD6_projet_session/1_best_returns/names.txt
/home/cloudera/prices2.csv

```

Voici la capture d'écran de la sortie produite par notre code ci-dessus :

```

class HighestHigh(HRJob):
    """Find the highest price of each coins and the time at which it occurred"""
    usage:
    python find_highest_high.py --names=names.txt ../prices2.csv
    ...
    def configure_options(self):
        super(HighestHigh, self).configure_options()
        self.add_file_option('--names', help='Names of the coin')
    ...

Job output is in /tmp/find_highest_high.cloudera.20190306.194521.080915/output
Streaming final output from /tmp/find_highest_high.cloudera.20190306.194521.080915/output...
"BTCUSDT" "3728.88 $ 2019-02-08 08:55:00.256000"
"XRPBTC" "0.32 $ 2019-02-08 08:53:30.255000"
"ZECBTC" "55.53 $ 2019-02-12 20:27:21.458000"
"ONTBTC" "0.71 $ 2019-02-15 16:44:28.523000"
"BTGBTC" "10.67 $ 2019-02-08 08:54:30.256000"
"REPBTC" "16.01 $ 2019-02-02 16:29:57.309000"
"OMGBTC" "1.26 $ 2019-02-16 11:14:45.787000"
"ZILBTC" "0.02 $ 2019-02-02 15:45:37.204000"
"DCRBTC" "17.06 $ 2019-02-12 22:32:32.027000"
"ZRXBTC" "0.26 $ 2019-02-08 08:54:10.255000"
"LINKBTC" "0.48 $ 2019-02-09 07:55:47.175000"
"BATBTC" "0.14 $ 2019-02-06 14:02:07.903000"
"DOGBTC" "18.27 $ 2019-02-07 21:08:07.006000"
"POWRBTC" "0.09 $ 2019-02-15 03:04:26.646000"
"MCBTC" "3.01 $ 2019-02-15 16:12:40.369000"
"RVNBTC" "0.01 $ 2019-02-09 12:30:50.467000"
"BNBTBTC" "0.55 $ 2019-02-08 09:13:50.345000"
"MANBTC" "0.3 $ 2019-02-08 08:53:10.248000"
"POLYBTC" "0.11 $ 2019-02-07 06:37:22.687000"
"NLGBTC" "0.45 $ 2019-02-16 08:41:24.953000"
"LOMBTC" "0.05 $ 2019-02-03 19:02:51.592000"
"TRXBTC" "0.03 $ 2019-02-04 01:03:02.519000"
"NASBTC" "0.59 $ 2019-02-08 14:27:51.780000"
"QKCBTC" "0.03 $ 2019-02-05 16:44:01.634000"

```

Sur la capture d'écran ci-dessus, on peut apprécier trois colonnes. Les premières lettres de chaque élément de cette colonne constitue le symbol de chaque cryptomonnaie. Par exemple, BTCUSDT represent Bitcoin, XRPBTC représente Ripple, et ainsi de suite. La deuxième colonne montre le prix maximal atteint par chaque cryptomonnaie en termes de dollars américains, tel qu'indiqué par le symbole «\$». Par exemple, on peut voir que le prix maximal du Bitcoin a été de 3 728,88 dollars américains et que celui de Ripple a été de 0,32 dollars américains. La troisième colonne montre le moment où chaque cryptomonnaie a atteint son prix maximal en termes de dollars américains pendant la période d'étude.

2. Nous avons développé notre deuxième script MapReduce en python que nous avons nommé «Find best returns». Ce script contient du code pour trouver la cryptomonnaie avec le meilleur rendement pendant la période d'étude. Pour calculer le rendement des cryptomonnaies, notre code tient compte du prix au début et à la fin de la période d'étude et utilise la formule suivante:

$$\text{Rendement pour la période} = \frac{(\text{prix de fin de la période} - \text{prix du début de la période})}{\text{prix du début de la période}}$$

Pour exécuter notre code nous avons utilisé le jeux de données prices2.csv. Ci-dessous nous montrons notre code avec des commentaires le décrivant.

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class BestReturns(MRJob):
    """Find the currency that was the best investment for a given period

    usage:
    !python find_best_returns.py --names=names.txt --dates=dates.txt
    prices2.csv
    """

    def configure_options(self):
        super(BestReturns, self).configure_options()
        self.add_file_option('--names', help='Names of the coin')
        self.add_file_option('--dates', help='Dates for the start of the period and
        for the end')

    def steps(self):
        return [
            MRStep(mapper_init=self.mapper_init,
                    mapper=self.mapper_get_prices,
                    reducer_init=self.reducer_init,
                    reducer=self.reducer_get_returns),
            MRStep(reducer=self.reducer_get_best_return)
        ]

    def mapper_init(self):
```

```

with open('dates.txt', 'U') as f:
    self.dates = f.read().split('\n')

# A line is the price for 69 coins and the time
def mapper_get_prices(self, _, line):
    fields = line.split(',')
    prices = fields[1:]
    time = fields[0]

    # Skip header
    if fields[1] != 'BTCUSDT':
        for idx, price in enumerate(prices):
            # Date of price must be in dates.txt
            if time == self.dates[0]:
                yield idx, ('open', float(price))
            elif time == self.dates[1]:
                yield idx, ('close', float(price))

def reducer_init(self):
    with open('names.txt') as f:
        self.names = f.read().split('\n')

def reducer_get_returns(self, idx, values):
    lst = list(values)
    profit = round((((lst[1][1] / lst[0][1]) - 1) * 100, 1)
    yield None, (self.names[idx], profit)

def reducer_get_best_return(self, _, values):
    _max = 0
    coin_max = "

```

```
for coin, profit in values:
```

```
    if profit > _max:
```

```
        _max = profit
```

```
    coin_max = coin
```

```
yield coin_max, '{0} %'.format(_max)
```

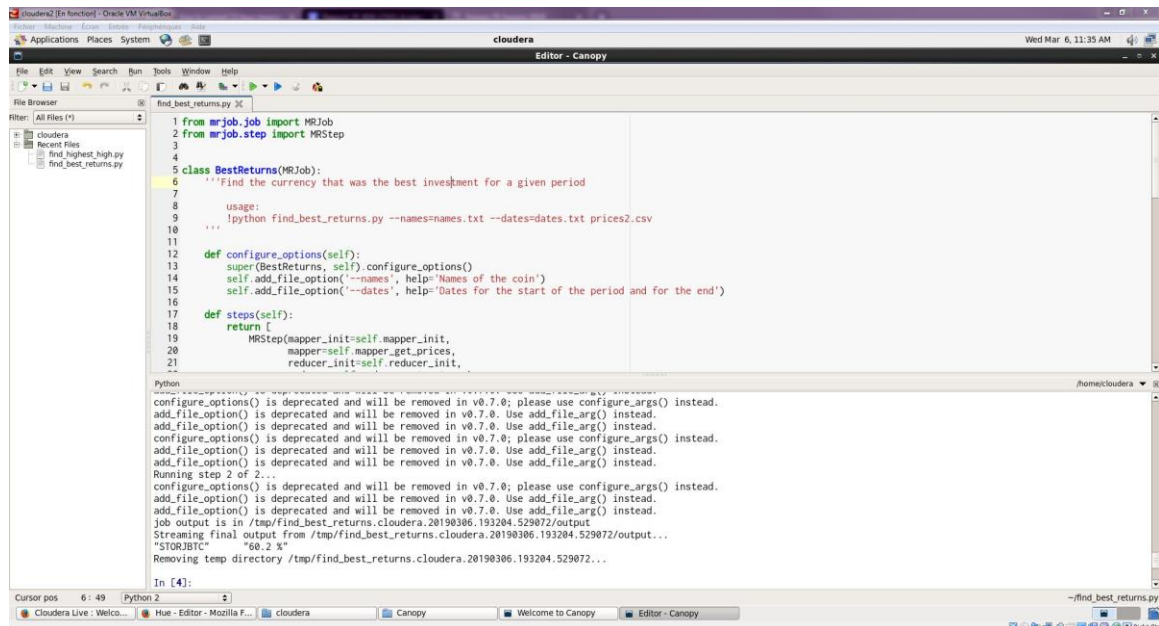
```
if __name__ == '__main__':
```

```
    BestReturns.run()
```

Pour exécuter le code ci-dessus nous avons utilisé la commande suivante dans la console de Canopy :

```
!python find_best_returns.py --names=names.txt --dates=dates.txt  
prices2.csv
```

Voici la capture d'écran de la sortie produite par le notre code ci-dessus:



```
1 from mrjob.job import MRJob
2 from mrjob.step import MRStep
3
4
5 class BestReturns(MRJob):
6     '''Find the currency that was the best investment for a given period
7
8     usage:
9     ... !python find_best_returns.py --names=names.txt --dates=dates.txt prices2.csv
10
11
12     def configure_options(self):
13         super(BestReturns, self).configure_options()
14         self.add_file_option('--names', help='Names of the coin')
15         self.add_file_option('--dates', help='Dates for the start of the period and for the end')
16
17     def steps(self):
18         return [
19             MRStep(mapper_init=self.mapper_init,
20                   mapper=self.mapper_get_prices,
21                   reducer_init=self.reducer_init,
22
23
24 Python
25 .../canopy/lib/python2.7/site-packages/mrjob/job.py:100: DeprecationWarning: configure_options() is deprecated and will be removed in v0.7.0; please use configure_args() instead.
26 add_file_option() is deprecated and will be removed in v0.7.0. Use add_file_arg() instead.
27 add_file_option() is deprecated and will be removed in v0.7.0. Use add_file_arg() instead.
28 configure_options() is deprecated and will be removed in v0.7.0; please use configure_args() instead.
29 add_file_option() is deprecated and will be removed in v0.7.0. Use add_file_arg() instead.
30 add_file_option() is deprecated and will be removed in v0.7.0. Use add_file_arg() instead.
31 Running step 2 of 2...
32 configure_options() is deprecated and will be removed in v0.7.0; please use configure_args() instead.
33 add_file_option() is deprecated and will be removed in v0.7.0. Use add_file_arg() instead.
34 add_file_option() is deprecated and will be removed in v0.7.0. Use add_file_arg() instead.
35 job output is in /tmp/find_best_returns.cloudera.20190306.193204.529072/output
36 Streaming final output from /tmp/find_best_returns.cloudera.20190306.193204.529072/output...
37 "STORBTC"      "60.2 %"
38 Removing temp directory /tmp/find_best_returns.cloudera.20190306.193204.529072...
39
40 In [4]:
```



Sur la capture d'écran on peut apprécier « "STORJBTC" "60.2%" », ce qui indique que la cryptomonnaie avec le rendement le plus élevé pour la période d'étude a été Storj, avec un rendement de 60,2 %. Ce rendement est en termes de Bitcoin. Pour corroborer l'exactitude de notre résultat, nous avons consulté le site web [www.tradingview.com](http://www.tradingview.com) et nous avons obtenu un graphique du prix de Storj en termes de Bitcoin.



Sur le graphique on peut apprécier une ligne droite verte que nous avons tracé, laquelle va du prix initial au prix final de Storj pour période d'étude. La différence entre ces prix donne un rendement de 60,2 %, ce qui confirme le résultat produit par notre code. Pour mettre les choses en perspective, ce rendement de 60,2 % en seulement deux semaines équivaut à un rendement annualisé de 1 565,2 %!

3. Nous avons créé un troisième script MapReduce en python que nous avons nommé «Find MA». Ce script contient du code qui calcule la moyenne mobile du prix d'une cryptomonnaie pour une période donnée. Le nombre d'intervalles contenues dans la période doit être précisé dans le paramètre de ligne de commande "--period". Pour exécuter notre code nous avons utilisé le fichier prices2.csv et nous avons choisi la cryptomonnaie OX. Notre code a utilisé son prix à la fin de chacun de vingt intervalles de 10 secondes, il a calculé la moyenne de ces 20 prix, et il a répété ce calcul pour des périodes consécutives de 20 intervalles de 10 secondes chacun, produisant ainsi la moyenne mobile désirée.

Une partie de code avec un mapper et un reducer ont été ajoutés pour regrouper les prix et ouvrir un graphique matplotlib du résultat, permettant ainsi sa visualisation.

Notre code est montré ci-dessous avec des commentaires le décrivant.

```
from mrjob.job import MRJob
from mrjob.step import MRStep
from plot import plot

class FindMA(MRJob):
    """Calculate a moving average series for a currency.

    usage:

    python find_ma.py --names=names.txt --period=20 --coin=ZRXBTC --
names=names.txt ../prices2.csv
    """

    def configure_options(self):
        super(FindMA, self).configure_options()
        self.add_passthrough_option(
            '--coin', type='str', default='BTCUSDT', help='Name of a specific coin
to calc the MA')
        self.add_passthrough_option(
            '--period', type='int', default=20, help='period of the MA in hours')
        self.add_file_option('--names', help='Names of the coin')

    def steps(self):
        return [
            MRStep(mapper_init=self.mapper_init,
                    mapper=self.mapper_get_prices,
                    reducer_init=self.reducer_init,
                    reducer=self.reducer_get_avgs),
            MRStep(mapper=self.mapper_regroup,
```

```

        reducer=self.show_results)
    ]

def mapper_init(self):
    with open('names.txt', 'U') as f:
        self.names = f.read().split('\n')

# A line is the price for 69 coins and the time
def mapper_get_prices(self, _, line):
    fields = line.split(',')
    time = fields[0]
    prices = fields[1:]
    btc_price = fields[1]

    if btc_price != 'BTCUSDT': # Skip header
        for idx, price in enumerate(prices):
            coin = self.names[idx]

            if coin == self.options.coin:
                if coin == 'BTUSDT':
                    yield coin, (int(time), float(btc_price))
                else: # Convert alt coin btc price to US Dollar
                    yield coin, (int(time), float(price) * float(btc_price))

def reducer_init(self):
    # Calculate bin size of prices based on period
    period_in_ms = self.options.period * 1000 * 60 * 60
    # Div by 10k since prices were sampled every 10s
    self.options.bin = round(period_in_ms / 10000.0)

```

```

def reducer_get_avgs(self, coin, values):
    prices = []
    for i, arr in enumerate(list(values)):
        time = arr[0]
        price = arr[1]
        prices.append(price)
        if i > self.options.bin:
            avg = sum(prices[i - self.options.bin:]) / self.options.bin
            yield coin, (time, price, avg)
        else:
            yield coin, (time, price, float('NaN'))

# Those mapper & reducer are just to show the results
def mapper_regroup(self, coin, values):
    yield coin, values # Just regroup by keys

def show_results(self, _, values):
    plot(list(values))

if __name__ == '__main__':
    FindMA.run()

```

Voici le module pour ouvrir le graphique:

```

import matplotlib.pyplot as plt
import pandas as pd

def plot(vals):
    time, price, avg = [], [], []
    for x in vals:

```

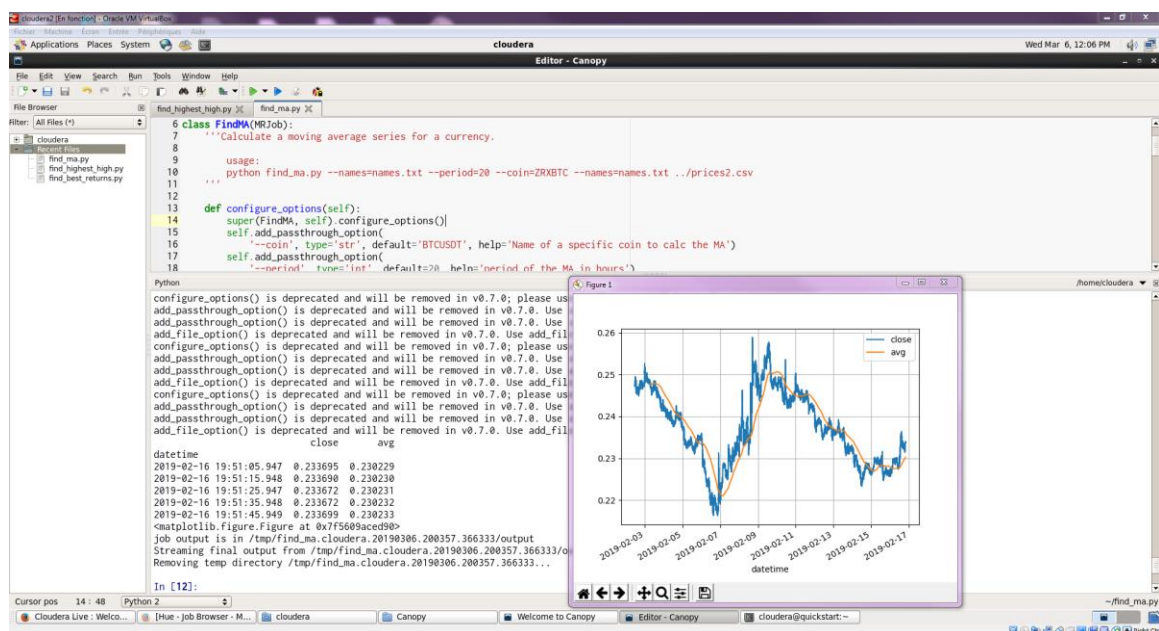
```

time.append(x[0])
price.append(x[1])
avg.append(x[2])

prices = pd.DataFrame()
prices['datetime'] = pd.to_datetime(time, unit='ms')
prices['close'] = price
prices['avg'] = avg
prices = prices.set_index('datetime')
print(prices.tail())
prices.plot(grid=True)
plt.show()

```

Voici la capture d'écran de la sortie générée par notre code:



Sur l'image on peut apprécier trois colonnes. La première et la deuxième colonnes montrent les prix de fermeture de la cryptomonnaie Ox a chaque dix secondes. La troisième colonne montre la moyenne mobile du prix de cette cryptomonnaie pour des

périodes comprenant 20 intervalles de 10 secondes. Le graphique montre une ligne bleue, laquelle représente le prix de cette cryptomonnaie se trouvant dans la deuxième colonne. Et il montre ligne jaune, laquelle est la représentation des valeurs de la moyenne mobile contenues dans la troisième colonne. On peut constater que la ligne jaune suit de près la ligne bleue, étant donné que c'est une moyenne mobile de courte durée, comprenant seulement 20 périodes.

## **Code HiveQL**

Les requêtes en Hive utilisent le jeu de données transactions.csv, lequel contient toutes les transactions achat-vente qui ont eu lieu pendant la même période d'étude couverte par le jeu de données prices2.csv. Le dataset contient 16 millions de lignes, confirmant le contexte mégadonnées de notre projet. Notre code est montré et décrit ci-dessous.

Création et chargement de la table:

```
CREATE TABLE transactions(tradetime STRING, eventime STRING, isMM  
STRING, price DOUBLE, quantity DOUBLE, symbol STRING)
```

```
row format delimited fields terminated by ','
```

```
lines terminated by '\n'
```

```
tblproperties ("skip.header.line.count"="1");
```

```
load data inpath '/user/cloudera/transactions.csv' into table transactions;
```

1. Notre premier code HiveQL détermine le nombre de transactions dont l'ordre d'achat était de type «limite» ou «marché» pour chaque paire de cryptomonnaies. Un ordre d'achat de type limite est celle où l'acheteur détermine le prix qu'il est disposé à payer. Par contre, un ordre d'achat de type marché est celui où l'acheteur est disposé à payer le meilleur prix sur le marché au moment où l'acheteur place son ordre d'achat. Pour exécuter notre code nous avons choisi la cryptomonnaie Bitcoin. Notre code est montré ci-dessous.

## SELECT

**round(sum(price \* quantity)) AS vol,**

**symbol,**

**CASE isMM**

**WHEN 'False' THEN 'Market'**

**EISE 'Limit'**

**END AS Type**

**FROM transactions**

**WHERE symbol != 'BTCUSDT'**

**GROUP BY symbol, isMM**

**ORDER BY symbol;**

The screenshot shows the Hue web interface running on a Cloudera VM. The query editor displays the following HiveQL code:

```
1 SELECT
2   round(sum(price * quantity)) AS vol,
3   symbol,
4   CASE isMM
5     WHEN 'False' THEN 'Market'
6     ELSE 'Limit'
7   END AS Type
8 FROM transactions
9 WHERE symbol != 'BTCUSDT'
10 GROUP BY symbol, isMM
11 ORDER BY symbol;
```

The query has been executed, and the results are displayed in a table with 11 rows. The table has three columns: 'vol', 'symbol', and 'type'. The results are as follows:

	vol	symbol	type
1	753	AEBTC	Market
2	1103	AEBTC	Limit
3	564	AGBTC	Market
4	659	AGBTC	Limit
5	1015	AGNBTC	Market
6	1318	AGNBTC	Limit
7	309	ARDBTC	Market
8	552	ARDBTC	Limit
9	2505	ARKBTC	Limit
10	2353	ARKBTC	Market
11	4096	BATBTC	Limit

Le résultat de notre requête montre que le type d'achat limite est plus utilisée mais il demeure tout de même que beaucoup de transactions soient faites avec le type d'achat marché pour permettre l'acquisition immédiate de la monnaie. Puisque les ordres de type limit exigent un certain niveau de sophistication de la part des investisseurs, ces résultats suggèrent que la plupart des investisseurs en cryptomonnaies ce sont des investisseurs d'un certain niveau de sophistication.

2. Notre deuxième code HiveQL détermine l'heure de la journée, en moyenne, avec la plus grande activité en termes du nombre de transactions, pour chaque paire de cryptomonnaies.

## SELECT

```
ROUND(AVG(HOUR(from_unixtime(CAST(eventime/1000 AS BIGINT),  
'MM-dd-yyyy H:mm:ss')))) AS `Hour`,
```

```
symbol
```

```
FROM transactions
```

```
GROUP BY symbol
```

```
ORDER BY `Hour` DESC;
```

The screenshot shows the Hue interface running a HiveQL query. The query is as follows:

```
1 SELECT  
2   ROUND(AVG(HOUR(from_unixtime(CAST(eventime/1000 AS BIGINT),  
3     'MM-dd-yyyy H:mm:ss')))) AS `Hour`,  
4   symbol  
5 FROM transactions  
6 GROUP BY symbol  
7 ORDER BY `Hour` DESC;
```

The results are displayed in a table with two columns: 'hour' and 'symbol'. The table shows the average hour of the day for each symbol, ordered by hour in descending order.

hour	symbol
12	OKBTC
12	DGBTC
12	BATBTC
12	REPBTC
12	SNTBTC
12	NASBTC
12	MCBTC
11	XRPBTC
11	WTCBTC
11	THETBTC
11	RVBTC
11	POWRBTC
11	NALBTC
11	LINKBTC
11	ENGBTC



Il est très clair que les monnaies les plus populaires se retrouvent vers 11-12 heures alors que les monnaies avec nettement moins de volume sont échangées plus tôt dans l'avant-midi.

## Code Pig Latin

Le jeu de données utilisé est prices.csv au lieu de prices2.csv (les en-têtes ont été enlevées pour faciliter le traitement).

Chargement initial et déclaration du bag:

```
>> prices = LOAD '/user/cloudera/prices.csv' USING PigStorage(',') AS
(time: long, BTCUSDT: float, XRPBTC: float, ETHBTC: float, EOSBTC: float,
LTCBTC: float, TRXBTC: float, XLMBTC: float, XMRBTC: float, NEOBTC: float,
XEMBTC: float, ZECBTC: float, ONTBTC: float, BTGBTC: float, REPBTC: float,
OMGBTC: float, ZILBTC: float, DCRBTC: float, ZRXBTC: float, LINKBTC: float,
BATBTC: float, LSKBTC: float, NANOBTC: float, BTSBTC: float, ICXBTC: float,
STEEMBTC: float, AEBTC: float, STRATBTC: float, KMDBTC: float, SNTBTC:
float, PPTBTC: float, GNTBTC: float, ARDRBTC: float, LRCBTC: float, ARKBTC:
float, THETABTC: float, HCBTC: float, PIVXBTC: float, WTCBTC: float,
MANABTC: float, XZCBTC: float, AIONBTC: float, DGDBTC: float, POWRBTC:
float, MCOBTC: float, RVNBTC: float, BNTBTC: float, WANBTC: float, POLYBTC:
float, NULSBTC: float, LOOMBTC: float, NASBTC: float, QKCBTC: float,
SYSBTC: float, ZENBTC: float, AGIBTC: float, ENJBTC: float, ENGBTC: float,
EDOBTC: float, RLCBTC: float, GASBTC: float, MITHBTC: float, NXSBTC: float,
KNCBTC: float, CMTBTC: float, STORJBTC: float, SALTBTC: float, SUBBTC:
float, BRDBTC: float, CVCBTC: float);
```

1. Notre premier code Pig Latin obtient les prix minimal et maximal de Bitcoin en termes de dollars américains, ainsi que les prix minimal et maximal de Ethereum en termes de Bitcoin :

```
>> grouped = group prices all;
```

```
>> describe grouped;
```

```
grouped: {group: chararray,prices: {(time: long, BTCUSDT: float...)}}
```

```
>> minmax = FOREACH grouped GENERATE MIN(prices.BTCUSDT) as minbtc,
MAX(prices.BTCUSDT) as maxbtc, MIN(prices.ETHBTC) as mineth,
MAX(prices.ETHBTC) as maxeth;
```

>> describe minmax;

**minmax: {minbtc: float,maxbtc: float,mineth: float,maxeth: float}**

>> dump minmax;

**(3379.39,3728.88,0.029869,0.034461)**

La dernière ligne montre les prix minimal et maximal de Bitcoin en termes de dollars américains (3379.39 et 3728.88, respectivement) et les prix minimal et maximal de Ethereum en termes de Bitcoin (0.029869 et 0.034461, respectivement).

Ci-bas nous montrons la capture d'écran de la sortie que notre code Pig a produite :

```
2019-02-24 07:49:09.787 [JobControl] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths (combined) to process : 1
2019-02-24 07:49:09.878 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - number of splits:1
2019-02-24 07:49:09.930 [JobControl] INFO org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job: job_1551014138910_0013
2019-02-24 07:49:10.944 [JobControl] INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application application_1551014138910_0013
2019-02-24 07:49:10.956 [JobControl] INFO org.apache.hadoop.mapreduce.Job - The url to track the job: http://quickstart.cloudera:8080/proxy/application_1551014138910_0013/
2019-02-24 07:49:10.109 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - HadoopJobId: job_1551014138910_0013
2019-02-24 07:49:10.110 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - Processing aliases: grouped,minmax,prices
2019-02-24 07:49:10.110 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - detailed locations: M: prices[1,9],prices[1,-1],minmax[6,9],grouped[2,10] C: minmax[6,9],grouped[2,10] R: minmax[6,9]
2019-02-24 07:49:10.132 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - More information at: http://localhost:50030/jobdetails.jsp?jobid=job_1551014138910_0013
2019-02-24 07:50:24.278 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - 8% complete
2019-02-24 07:50:45.711 [main] INFO org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2019-02-24 07:50:46.162 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - 100% complete
2019-02-24 07:50:46.162 [main] INFO org.apache.pig.tools.pigstats.SimplePigStats - Script Statistics:

HadoopVersion  PigVersion  Userid  StartedAt  FinishedAt  Features
2.6.0-cdh5.13.0 0.12.0-cdh5.13.0 cloudera 2019-02-24 07:49:03 2019-02-24 07:50:46 GROUP_BY

Success!

Job Stats (time in seconds):
JobId  Maps  Reduces  MaxMapTime  MinMapTime  AvgMapTime  MedianMapTime  MaxReduceTime  MinReduceTime  AvgReduceTime  MedianReduceTime  Alias  Feature Outputs
job_1551014138910_0013  1  1  57  57  57  11  11  11  grouped,minmax,prices  GROUP_BY,COMBINER  hdfs://quickstart.cloudera:8026/tmp/1333142064/tmp-1633948896

Input(s):
Successfully read 124064 records (82855746 bytes) from: "/user/cloudera/prices.csv"

Output(s):
Successfully stored 1 records (31 bytes) in: "hdfs://quickstart.cloudera:8026/tmp/1333142064/tmp-1633948896"

Counters:
Total records written : 1
Total bytes written : 31
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1551014138910_0013

2019-02-24 07:50:46.235 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - Success!
2019-02-24 07:50:46.373 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-02-24 07:50:46.373 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2019-02-24 07:50:46.373 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-24 07:50:46.398 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-24 07:50:46.398 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(3379.39,3728.88,0.029869,0.034461)
```

2. Notre deuxième code Pig Latin calcule le prix moyen pour des cryptomonnaies précisées.

>> grouped = group prices all;

>> describe grouped;

**grouped: {group: chararray,prices: {(time: long, BTCUSDT: float...)}}**

>> mean = FOREACH grouped GENERATE AVG(prices.BTCUSDT) as avgbtc,  
AVG(prices.ETHBTC) as avgeth;

>> describe mean;

**mmean: {avgbtc: double,avgeth: double}**

>> dump mean;

**(3546.094176567182,0.032297750510539486)**

La dernière ligne ci-dessus montre le prix moyen de Bitcoin en termes de dollars américains (3546.094176567182) et le prix moyen de Ethereum en termes de Bitcoin (0.032297750510539486) pendant la période d'étude.

Ci-dessous nous montrons la capture d'écran de la sortie que notre script Pig a produite :

```
Job Stats (time in seconds):
JobId  Maps  Reduces  MaxMapTime  MinMapTime  AvgMapTime  MedianMapTime  MaxReduceTime  MinReduceTime  AvgReduceTime  MedianReduceTime  Alias  Feature Outputs
job_1551534344743_0002  1  1  23  23  23  23  13  13  13  13  grouped,mean,prices  GROUP_BY,COMBINER  hdfs://quickstart.cloudera:8020/tmp/temp1234676408/tmp-998876048,

Input(s):
Successfully read 124064 records (82855746 bytes) from: "/user/cloudera/prices.csv"

Output(s):
Successfully stored 1 records (29 bytes) in: "hdfs://quickstart.cloudera:8020/tmp/temp1234676408/tmp-998876048"

Counters:
Total records written : 1
Total bytes written : 29
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1551534344743_0002

2019-03-02 07:06:21.249 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-03-02 07:06:21.250 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2019-03-02 07:06:21.250 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2019-03-02 07:06:21.250 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-03-02 07:06:21.255 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-03-02 07:06:21.255 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths to process : 1
(3546.094176567182,0.032297750510539486)
```

## CONCLUSION

Nous avons réussi à extraire des jeux de données différentes pour permettre d'analyser des cryptomonnaies, le sujet de notre projet intégrateur, depuis des angles différents. Nous avons réussi à créer et à exécuter du code MapReduce en Python, ainsi que du code HiveQL et Pig Latin pour traiter les mégadonnées reliées aux cryptomonnaies analysées. En traitant ces mégadonnées, nous avons obtenu des connaissances approfondies sur les cryptomonnaies. Notamment, nous avons trouvé que les cryptomonnaies offrent des rendements annualisés potentiels spectaculaires lesquels sont nettement supérieurs à ceux offerts par les marchés financiers traditionnels. Aussi, nous avons déniché que la plupart des investisseurs en Bitcoin sur l'échange Binance sont des investisseurs plutôt sophistiqués. De plus, nous avons découvert que les cryptomonnaies les plus populaires se transigent entre 11h et 12h, tandis que les moins populaires se transigent plus tôt dans l'avant midi. Nous estimons donc que nous avons réussi notre projet de session de ce cours et que nous avons nettement enrichi notre projet intégrateur du AEC.

## **RÉFÉRENCES**

Notes du cours BD6, Nesrine Zemirli, Ph.D.

Documentation mrjob, version 0.6.7, Steve Johnson

[www.tradingview.com](http://www.tradingview.com)