

Assignment 2: MongoDB and Cassandra

By

Rachael Joan Dias

0651897

Addressed to,

Professor Brian Srivastava

Trent University

AMOD-5410H-A-2019GW-PTBO Big Data

**Objective:** Data was gathered from 2 different sources Twitter and Coinmarketcap. The information from Twitter consisted of all tweets related to a cryptocurrency posted by a twitter user, while coinmarketcap keeps track of the latest value of a cryptocurrency. The data extracted from both the API were stored on MongoDB in the form of documents. Similarly, data was extracted from the same 2 sources and stored on tables in Cassandra.

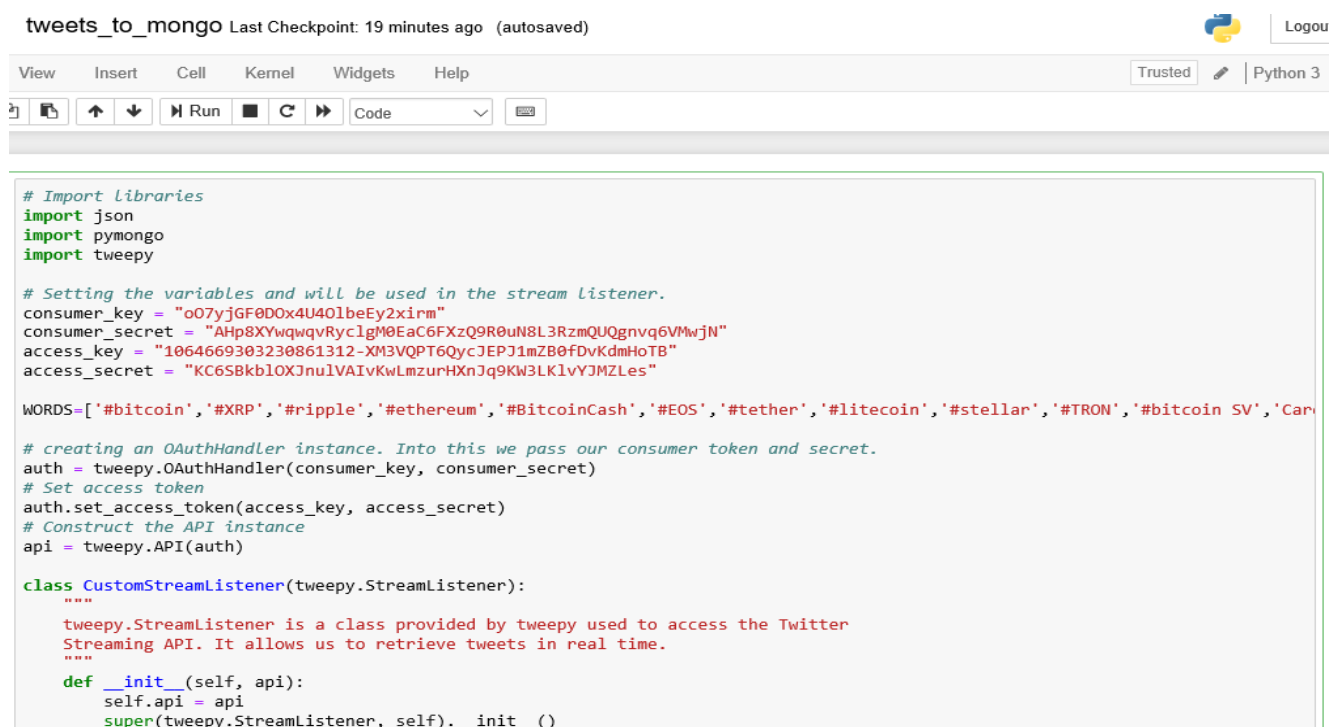
After gathering and storing the data on 2 different databases multiple data manipulation queries were run to search for keywords, determine the number of documents containing a particular keyword and order data based on the timestamp.

## Part 1: Data Gathering and Storage


### MongoDB Data Gathering and Storage


All tweets that contained names of cryptocurrencies such as “Bitcoin”, “TRON”, “Ethereum” etc. were stored on a database “mongodb” on MongoDB. A python script “tweets\_to\_mongo” was executed to load streaming tweets into MongoDB using twitter’s streaming API.


The 3 libraries used were json, pymongo and tweepy. Json library is used to read the tweets, pymongo is the python connector used to insert the tweets into MongoDB and tweepy is the library used to extract streaming tweets. First, the consumer and access keys are specified to set-up the connection, these keys are passed through the OAuthHandler() and set\_access\_token() functions of the tweepy library to initiate the authentication. Once auth is set-up it is passed through the API() function, using the CustomerStreamListener(tweepy.StreamListener) class streaming tweets are pulled from twitter. Please see below screenshots of the script.



The screenshot shows a Jupyter Notebook interface with a toolbar at the top containing buttons for View, Insert, Cell, Kernel, Widgets, and Help. Below the toolbar is a row of icons for file operations and a 'Run' button. The main area displays a Python script with the following content:

```
tweets_to_mongo Last Checkpoint: 19 minutes ago (autosaved)  Logou

View Insert Cell Kernel Widgets Help Trusted  Python 3

📄 📁 ⬆ ⬇ ▶ Run 🛑 ↺ ▶ Code 

# Import Libraries
import json
import pymongo
import tweepy

# Setting the variables and will be used in the stream listener.
consumer_key = "o07yjGF0D0x4U40lbeEy2xirm"
consumer_secret = "AHp8XYwqwqvRyc1gM0EaC6FXzQ9R0uN8L3RzmQUQgnvq6VMwjN"
access_key = "1064669303230861312-XM3VQPT6QycJEPJ1mZB0fDvkdmHoTB"
access_secret = "KC6SBkbl0XJnu1VAIvKwLmzurHXnJq9KW3LklvYJMZLes"

WORDS=['#bitcoin', '#XRP', '#ripple', '#ethereum', '#BitcoinCash', '#EOS', '#tether', '#litecoin', '#stellar', '#TRON', '#bitcoin SV', 'Car

# creating an OAuthHandler instance. Into this we pass our consumer token and secret.
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
# Set access token
auth.set_access_token(access_key, access_secret)
# Construct the API instance
api = tweepy.API(auth)

class CustomStreamListener(tweepy.StreamListener):
    """
    tweepy.StreamListener is a class provided by tweepy used to access the Twitter
    Streaming API. It allows us to retrieve tweets in real time.
    """
    def __init__(self, api):
        self.api = api
        super(tweepy.StreamListener, self).__init__()
```

Using the pymongo library a connection is established between python and MongoDB. The `self.db.tweets.insert_one(json.loads(tweet))` inserts streaming tweets into MongoDB. Only English tweets are filter by specifying `languages=["en"]` in the `sapi.filter()` function filter, tweets related to keywords mentioned in the list “WORDS” are inserted.

```
# Connecting to MongoDB and use the database twitter.
self.db = pymongo.MongoClient().mongodb

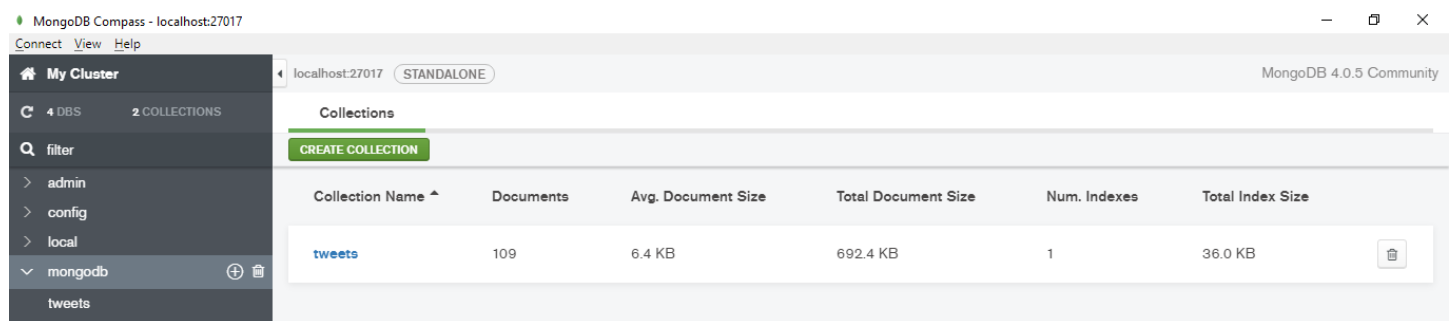
def on_data(self, tweet):
    self.db.tweets.insert_one(json.loads(tweet))

def on_error(self, status_code):
    # This is called when an error occurs
    print >> sys.stderr, 'Encountered error with status code:', status_code
    return True # Don't kill the stream

def on_timeout(self):
    # This is called if there is a timeout
    print >> sys.stderr, 'Timeout.....'
    return True # Don't kill the stream

# Create our stream object
sapi = tweepy.streaming.Stream(auth, CustomStreamListener(api))
sapi.filter(track=WORDS, languages=["en"])
```

Below is a screenshot which shows the database “mongodb”, the collection “tweets” and the 109 documents that are stored within the collection.



Below is an example depicting the way information related to a single tweet is stored on MongoDB.

localhost:27017 STANDALONE MongoDB 4.0.5 Community

**mongodb.tweets** DOCUMENTS 109 TOTAL SIZE 692.4KB AVG. SIZE 6.4KB INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } OPTIONS FIND

INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 20 of 109

```
> {
  "_id": ObjectId("5c82a90340ab6d1a2c0f0739"),
  "created_at": "Fri Mar 08 17:40:12 +0000 2019",
  "id": 1104074352167960578,
  "id_str": "1104074352167960578",
  "text": "RT @justinsuntron: Wanna know more information about #TRON Japan commu...",
  "source": "<a href='\"http://twitter.com\"' rel='\"nofollow\"'>Twitter Web Client</a>",
  "truncated": false,
  "in_reply_to_status_id": null,
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id": null,
  "in_reply_to_user_id_str": null,
  "in_reply_to_screen_name": null,
  "user": {
    "geo": null,
    "coordinates": null,
    "place": null,
    "contributors": null,
    "retweeted_status": {
      "quoted_status_id": 1103928030236299264,
      "quoted_status_id_str": "1103928030236299264",
      "quoted_status": {
        "quoted_status_permalink": {
          "is_quote_status": true,
          "quote_count": 0,
          "reply_count": 0
        }
      }
    }
  }
}
```

MongoDB is a NoSQL database, it stores information in the form of documents, unlike SQL databases that stores information as rows. Multiple documents form a collection, all the documents within a collection may not contain the same information. In order to search for patterns queries should be run within a field or column of a document.

Python script to load Twitter data to MongoDB: tweets\_to\_mongo.ipynb

In a similar manner data from coinmarketcap was loaded into MongoDB. Below is a screenshot the script used to load data into MongoDB.

First, a connection is established between MongoDB and Python by using the MongoClient() function from the pymongo library. Once the connection is made, the database “mongodb” is specified followed by the collection “coin”. In the case of coinmarketcap data is extracted from a url, the url link specifies the number of entries we would like to extract, the top 100 entries are selected. Using the json library the top 100 entries are loaded into a json object and then with the insert\_many() function of pymongo library they are loaded into the collection “coin” on MongoDB.

jupyter coin\_to\_mongo Last Checkpoint: a minute ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [3]: #import the required libraries
import urllib.request
import json
import requests
import pymongo
import datetime

#Connect to MySQL database
conn = pymongo.MongoClient()
db = conn.mongoddb
coin = db.coincoll
url='https://api.coinmarketcap.com/v1/ticker/?limit=100'; #API url

response=urllib.request.urlopen(url).read() #connecting to API url to get response

#create json object
json_obj=json.loads(response)
db.coin.insert_many(json_obj)

Out[3]: <pymongo.results.InsertManyResult at 0x27821a92ec8>
```

The collection “coin” is stored on the database “mongoddb”. There are 100 documents present under the collection coin.

My Cluster

4 DBS3 COLLECTIONS

filter

> admin

> config

> local

▼ mongoddb

coin

tweets

localhost:27017STANDALONE

MongoDB 4.0.5 Community

Collections

CREATE COLLECTION

Collection Name ^	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	
coin	100	402.9 B	39.3 KB	1	4.0 KB	
tweets	166	6.0 KB	996.4 KB	1	36.0 KB	

All information related to a cryptocurrency is stored as follows.

**mongodb.coin**

DOCUMENTS 100 TOTAL SIZE 39.3KB AVG. SIZE 403B INDEXES 1 TOTAL SIZE 16.0KB AVG. SIZE 16.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

**FILTER** { field: 'value' } **OPTIONS** **FIND**

**INSERT DOCUMENT** **VIEW** **LIST** **TABLE** Displaying documents 1 - 20 of 100

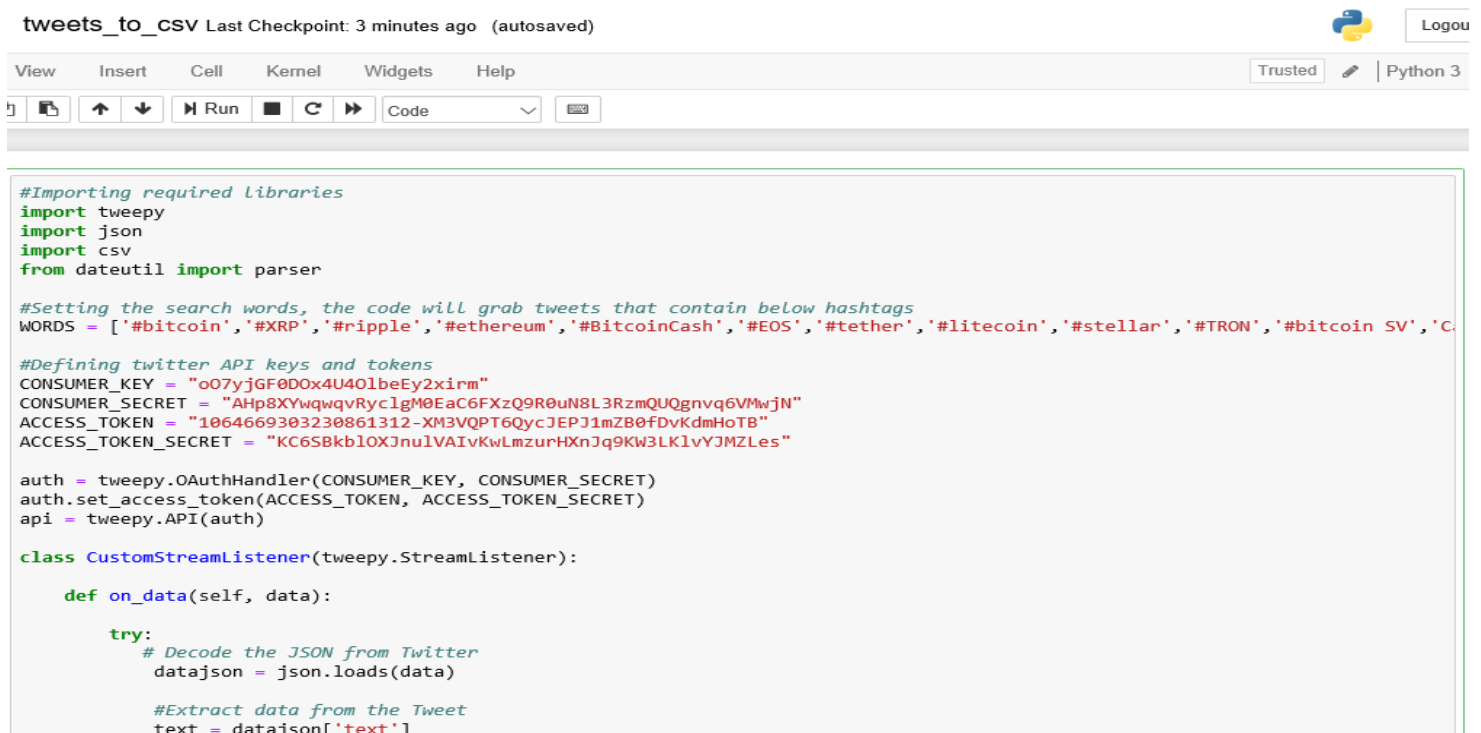
```
{
  "_id": ObjectId("5c86c00f40ab6d3f30406660"),
  "id": "bitcoin",
  "name": "Bitcoin",
  "symbol": "BTC",
  "rank": "1",
  "price_usd": "3894.170931",
  "price_btc": "1.0",
  "24h_volume_usd": "9953453908.61",
  "market_cap_usd": "68471304834.0",
  "available_supply": "17583025.0",
  "total_supply": "17583025.0",
  "max_supply": "21000000.0",
  "percent_change_1h": "-0.29",
  "percent_change_24h": "-1.19",
  "percent_change_7d": "3.45",
  "last_updated": "1552334728"
}
```

Python script to load coinmarketcap data to MongoDB: coin\_to\_mongo.ipynb

## Cassandra Data Gathering and Storage

Twitter and coinmarketcap data were extracted into a .csv files and then import into tables on Cassandra.

Streaming tweets were extracted using the tweepy library based on keywords related to cryptocurrencies. With the help of the csv library data was loaded into a csv file, since twitter data is in json format, only some fields for each tweet were extracted and loaded into the csv file. Below are screenshots of the script which loads extracts tweets into a csv file



```
tweets_to_CSV Last Checkpoint: 3 minutes ago (autosaved)

View Insert Cell Kernel Widgets Help Trusted Python 3

#Importing required Libraries
import tweepy
import json
import csv
from dateutil import parser

#Setting the search words, the code will grab tweets that contain below hashtags
WORDS = ['#bitcoin', '#XRP', '#ripple', '#ethereum', '#BitcoinCash', '#EOS', '#tether', '#litecoin', '#stellar', '#TRON', '#bitcoin SV', 'C

#Defining twitter API keys and tokens
CONSUMER_KEY = "o07yjGF0D0x4U40lbeEy2xirm"
CONSUMER_SECRET = "AHp8XYwqwqvRyc1gM0EaC6FXzQ9R0uN8L3RzmQUQgnvq6VMwJN"
ACCESS_TOKEN = "1064669303230861312-XM3VQPT6QycJEPJ1mZB0fDvKdmHoTB"
ACCESS_TOKEN_SECRET = "KC6SBkb10XJnu1VAIvKwLmzurHXnJq9KW3LKlvYJMZLes"

auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
api = tweepy.API(auth)

class CustomStreamListener(tweepy.StreamListener):

    def on_data(self, data):

        try:
            # Decode the JSON from Twitter
            datajson = json.loads(data)

            #Extract data from the Tweet
            text = datajson['text']
```

First json data is loaded into a json object using the json.loads(data) command. Then, certain fields of the json object text, name, screen\_name, tweet\_id, created\_at, followers and friends are extracted from the tweets.

With the help of the writer function from the csv library, a file tweets.csv is opened and encoding="utf-8" is specified so that tweets or usernames containing special characters can be inserted. The writerow() function appends the tweets one at a time to the csv file. Finally a header is hardcoded to csv file so that each column can be identified.

```

#Extract data from the Tweet
text = datajson['text']
name = datajson['user']['name']
screen_name = datajson['user']['screen_name']
tweet_id = datajson['id']
created_at = parser.parse(datajson['created_at'])
followers = datajson['user']['followers_count']
friends = datajson['user']['friends_count']

#write tweets to tweets.csv file
with open('tweets.csv', 'a', encoding="utf-8") as f:
    writer = csv.writer(f)
    writer.writerow([tweet_id,name,screen_name,created_at,text,followers,friends])

except Exception as e:
    print(e)

def on_error(self, status_code):
    print >> sys.stderr, 'Encountered error with status code:', status_code
    return True # Don't kill the stream

def on_timeout(self):
    print >> sys.stderr, 'Timeout...'
    return True # Don't kill the stream

# Writing csv titles
with open('tweets.csv', 'w') as f:
    writer = csv.writer(f)
    writer.writerow(['tweet_id', 'name', 'screen_name', 'created_at', 'text', 'followers', 'friends'])

streamingAPI=tweepy.streaming.Stream(auth, CustomStreamListener(api))
streamingAPI.filter(track=WORDS, languages=["en"])








```

Python script to load tweets to a csv file: tweets\_to\_csv.ipynb

Using the request library the data is read from the url specified, in this case top 100 cryptocurrencies are extracted and saved in a json object called response. Since the data is in json format using the read\_json() from the pandas library we can convert the data to a dataframe and then store it on a csv file.

coin\_to\_CSV
Last Checkpoint: a few seconds ago (autosaved)

View
Insert
Cell
Kernel
Widgets
Help








Code

```

#import the required libraries
import urllib.request
import json
import requests
import pandas
import datetime

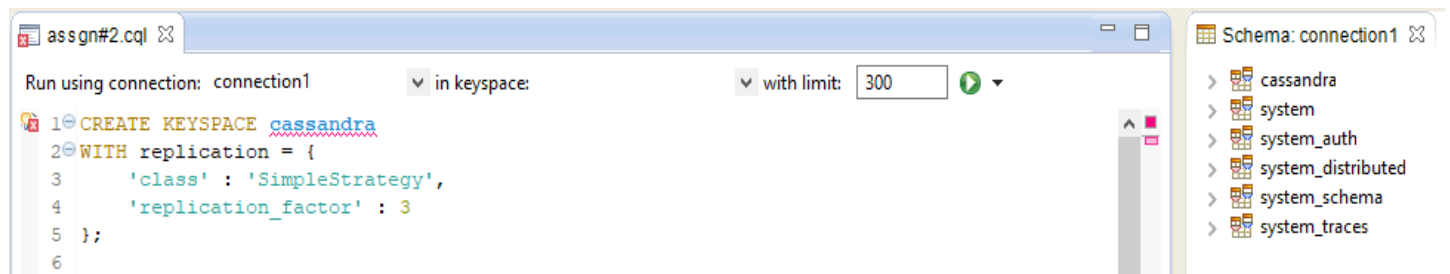
#Connect to MySQL database
url='https://api.coinmarketcap.com/v1/ticker/?limit=100'; #API url
response=urllib.request.urlopen(url).read()

df=pandas.read_json(response)
df.to_csv('coin_data.csv')

```

Python script to load tweets to a csv file: coin\_to\_csv.ipynb

A keyspace is created in Cassandra, keyspace is like a database which stores all the tables, in the query below a keyspace named “cassandra” is created the replication factor is set to 3 which means that 3 copies of the data are kept in a given data center, “SimpleStrategy” means that the same replication factor is assigned to the whole cluster.



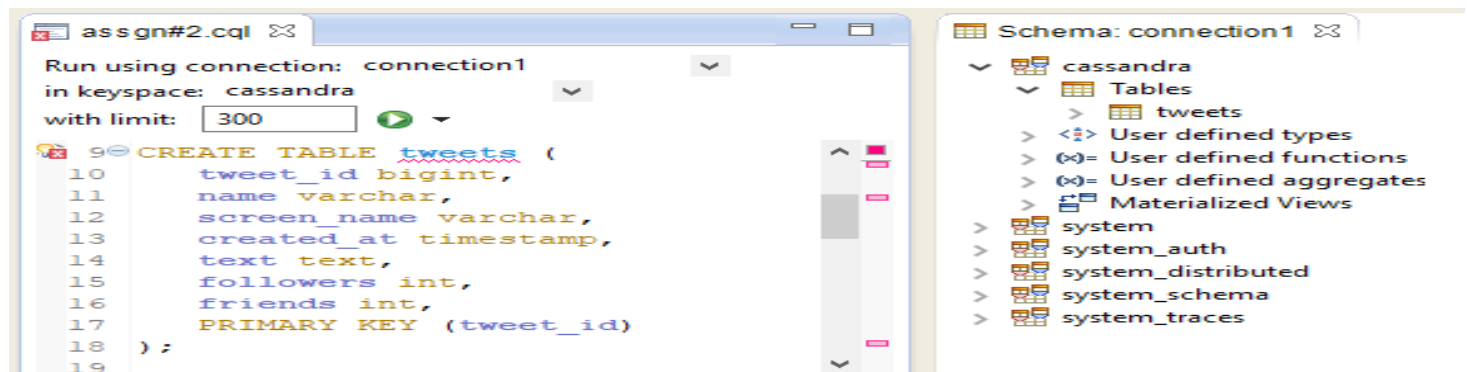
```
1 CREATE KEYSPACE cassandra
2 WITH replication = {
3   'class' : 'SimpleStrategy',
4   'replication_factor' : 3
5 };
6
```

Run using connection: connection1 in keyspace: with limit: 300

Schema: connection1

- cassandra
- system
- system\_auth
- system\_distributed
- system\_schema
- system\_traces

Now, that the database is created, tables need to be created to store data from the csv files. A table “tweets” is created to store twitter data and “coin” stores all the data from coinmarketcap. Below are the data definition commands used to create the tables.

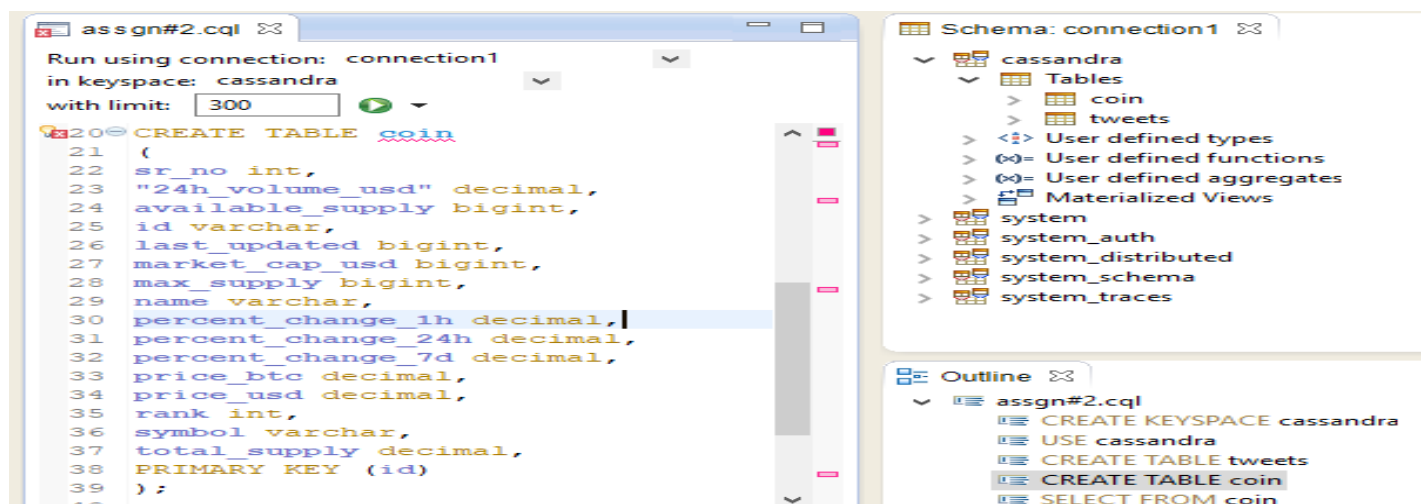


```
9 CREATE TABLE tweets (
10   tweet_id bigint,
11   name varchar,
12   screen_name varchar,
13   created_at timestamp,
14   text text,
15   followers int,
16   friends int,
17   PRIMARY KEY (tweet_id)
18 );
19
```

Run using connection: connection1 in keyspace: cassandra with limit: 300

Schema: connection1

- cassandra
  - Tables
    - tweets
  - User defined types
  - User defined functions
  - User defined aggregates
  - Materialized Views
  - system
  - system\_auth
  - system\_distributed
  - system\_schema
  - system\_traces



```
20 CREATE TABLE coin
21 (
22   sr_no int,
23   "24h_volume_usd" decimal,
24   available_supply bigint,
25   id varchar,
26   last_updated bigint,
27   market_cap_usd bigint,
28   max_supply bigint,
29   name varchar,
30   percent_change_1h decimal,
31   percent_change_24h decimal,
32   percent_change_7d decimal,
33   price_btc decimal,
34   price_usd decimal,
35   rank int,
36   symbol varchar,
37   total_supply decimal,
38   PRIMARY KEY (id)
39 );
40
```

Run using connection: connection1 in keyspace: cassandra with limit: 300

Schema: connection1

- cassandra
  - Tables
    - coin
    - tweets
  - User defined types
  - User defined functions
  - User defined aggregates
  - Materialized Views
  - system
  - system\_auth
  - system\_distributed
  - system\_schema
  - system\_traces

Outline

- assgn#2.cql
  - CREATE KEYSPACE cassandra
  - USE cassandra
  - CREATE TABLE tweets
  - CREATE TABLE coin
  - SELECT FROM coin

Now that we have tables data has to be imported from the csv files to the tables. Below commands were executed in cqlsh to import data into Cassandra tables “tweets” and “coin”.



WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.  
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.

[cqlsh 5.0.1 | Cassandra 3.9.0 | CQL spec 3.4.2 | Native protocol v4]

Use HELP for help.

WARNING: pyreadline dependency missing. Install to enable tab completion.

cqlsh> COPY cassandra.tweets (tweet\_id,name,screen\_name,created\_at,text,followers,friends) FROM 'C:\Users\17059\Desktop\BigData\tweets.csv' WITH HEADER=FALSE;

43 select \* from tweets;

tweet_id	created_at	followers	friends	name	screen_name	text
1106012813758062592	2019-03-13 10:02:58-0...	398	1181	Elena Butkovic (Digi4...	butkovic_elena	RT @CoinDeskMarket...
1106015127164182528	2019-03-13 10:12:09-0...	567	225	www.theEOSwriter.io	theeoswriter	Now that is a solid res...
1106020987978567680	2019-03-13 10:35:27-0...	184	155	Swolesome Dudmuffin	swolesome	Amazing to see how f...
1106015959507075072	2019-03-13 10:15:28-0...	1456	0	EOS Evangelist	iambliiss	RT @bytemaster7: A...
1106011636853288960	2019-03-13 09:58:17-0...	496	561	LTCSTRXSADABTCS	twizmenz	RT @Tronfoundation: ...
1106012556320137218	2019-03-13 10:01:56-0...	0	2	hung dung	hungdun39653182	good project
1106013732214210560	2019-03-13 10:06:37-0...	102	188	gadisav	gadisav1	RT @coindesk: JUST I...

1 selected statement successfully executed in 89 ms. R

[Feedback?](#)

cqlsh> COPY cassandra.coin (sr\_no,"24h\_volume\_usd",available\_supply,id,last\_updated,market\_cap\_usd,max\_supply,name,percent\_change\_1h,percent\_change\_24h,percent\_change\_7d,price\_btc,price\_usd,rank,symbol,total\_supply) FROM 'C:\Users\17059\Desktop\BigData\coin\_data.csv' WITH DELIMITER=';' AND HEADER=FALSE;

Using 3 child processes

Starting copy of cassandra.coin with columns [sr\_no, 24h\_volume\_usd, available\_supply, id, last\_updated, market\_cap\_usd, max\_supply, name, percent\_change\_1h, percent\_change\_24h, percent\_change\_7d, price\_btc, price\_usd, rank, symbol, total\_supply].

41 select \* from coin;

id	24h_volume_usd	available_supply	last_updated	market_cap_usd	max_supply	name	percent_chang...	percent_chang...	percent_chang...
ethereum	4207160939.52	105225799	1552527198	14028043288	<<null>	Ethereum	0.0	-0.58	-4.2
qash	459793.689225	350000000	1552527183	56302209	<<null>	QASH	-0.44	4.81	39.7
qubitica	80996.098482	2805342	1552527187	48578271	<<null>	Qubitica	-0.21	0.81	19.67
factom	115711.874255	9401050	1552527181	61323041	<<null>	Factom	0.13	0.66	3.39
digitex-futures	2236457.9376	737500000	1552527185	54030246	<<null>	Digitex Futures	-1.38	9.62	9.86
wanchain	7732257.24389	106152493	1552527185	46856077	<<null>	Wanchain	0.49	6.88	43.65
loopring	1733597.55462	828954240	1552527182	52312293	<<null>	Loopring	0.54	-1.19	10.93
icon	13127460.1345	473406688	1552527183	158128779	<<null>	ICON	-0.08	-3.39	16.42

1 selected statement successfully executed in 91 ms. R

[Feedback?](#)

## Data Gathering and Storage Challenges

The first challenge encountered was inserting data directly into Cassandra. I tried to use multiple libraries that were python connectors for Cassandra. But since they didn't work, I eventually settled on extracting the data to a csv file and importing it using commands in cqlsh.

It was harder to load data into Cassandra as compared to MongoDB, since data is stored in the form of tables in Cassandra. Data had to be extracted from the JSON object and inserted as rows. MongoDB on the other hand reads the data in JSON format and saves it in JSON format.

Later, I also faced issues with connecting to Cassandra because of multiple versions of Java, Cassandra abruptly stopped working. After uninstalling version 11 of Java and installing only version 8 Cassandra started working properly. Another issue encountered was while inserting data into the tables in Cassandra the .csv had to be stored with utf-8 encoding so that it could be imported to the tables.

## **Part 2: Data exploration**

After storing all the tweets and coinmarketcap currency data on MongoDB a few queries were executed.

### **Data Exploration in MongoDB**

#### **(a) Searching for keywords**

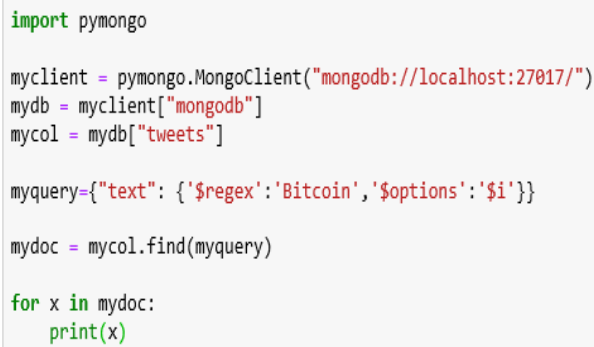
With the following query, all tweets that contain the keyword 'Bitcoin' in the text column were filtered out, '\$i' means that the keyword is case insensitive, so it will search for both lower- and upper-case words.

*myquery={"text": {'\$regex':'Bitcoin','\$options':'\$i'}}*

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mongodb"]
mycol = mydb["tweets"]

myquery={"text": {'$regex':'Bitcoin','$options':'$i'}}
```

The image shows a screenshot of a Jupyter Notebook with a light gray background. It contains Python code using the pymongo library to connect to a MongoDB instance at localhost:27017, access the 'mongodb' database and 'tweets' collection, and execute a query to find tweets containing the word 'Bitcoin' (case-insensitive). The code is as follows:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mongodb"]
mycol = mydb["tweets"]

myquery={"text": {'$regex':'Bitcoin','$options':'$i'}}
```

```
mydoc = mycol.find(myquery)

for x in mydoc:
    print(x)
```

The code is color-coded: 'import' is green, 'pymongo' is blue, and the rest of the code is black. The output of the code is not visible in the screenshot.

Below is the output of a tweet that contains the word bitcoin all the fields of the tweet are printed to the screen

```
{'_id': ObjectId('5c82a90640ab6d1a2c0f073c'), 'created_at': 'Fri Mar 08 17:40:19 +0000 2019', 'id': 1104074378931777537, 'id_str': '1104074378931777537', 'text': 'RT @TheBlockchain: Fixing the financial literacy gap will attract more women to Bitcoin https://t.co/loP5AjYuVm #bitcoin #blockchain #fintech', 'source': '<a href="http://discussionexpress.com" rel="nofollow">Information Critical</a>', 'truncated': False, 'in_reply_to_status_id': None, 'in_reply_to_status_id_str': None, 'in_reply_to_user_id': None, 'in_reply_to_user_id_str': None, 'user': {'id': 16967457, 'id_str': '16967457', 'name': 'Don Robinson', 'screen_name': 'greentechdon', 'location': 'New Jersey', 'url': None, 'description': 'The byproducts of innovative thinking are assembled into revolutionary products and services. #crosstraining #wellness\n #bigdata #poetry #environment', 'translator_type': 'none', 'protected': False, 'verified': False, 'followers_count': 629, 'friends_count': 422, 'listed_count': 16, 'favourites_count': 90, 'statuses_count': 16313, 'created_at': 'Sat Oct 25 15:13:11 +0000 2008', 'utc_offset': None, 'time_zone': None, 'geo_enabled': False, 'lang': 'en', 'contributors_enabled': False, 'is_translator': False, 'profile_background_color': '9AE4E8', 'profile_background_image_url': 'http://abs.twimg.com/images/themes/theme1/bg.png', 'profile_background_image_url_https': 'https://abs.twimg.com/images/themes/theme1/bg.png', 'profile_background_tile': False, 'profile_link_color': '0084B4', 'profile_sidebar_border_color': 'BDDCAD', 'profile_sidebar_fill_color': 'DDFFCC', 'profile_text_color': '333333', 'profile_use_background_image': True, 'profile_image_url': 'http://pbs.twimg.com/profile_images/1088467528379260928/Jpqavmr_normal.jpg', 'profile_image_url_https': 'https://pbs.twimg.com/profile_images/1088467528379260928/Jpqavmr_normal.jpg', 'profile_banner_url': 'https://pbs.twimg.com/profile_banners/16967457/1548345008', 'default_profile': False, 'default_profile_image': False, 'following': None, 'follow_request_sent': None, 'notifications': None, 'geo': None, 'coordinates': None, 'place': None, 'contributors': None, 'retweeted_status': {'created_at': 'Fri Mar 08 17:37:26 +0000 2019', 'id': 1104073655171145728, 'id_str': '1104073655171145728', 'text': 'Fixing the financial literacy gap will attract more women to Bitcoin https://t.co/loP5AjYuVm #bitcoin #blockchain #fintech', 'source': '<a href="http://discussionexpress.com" rel="nofollow">Information Critical</a>'}}
```

In the example below only the fields username, created\_at, and text are printed for tweets containing the keyword “tron”

```
myquery={"text": {'$regex': 'tron', '$options': '$i'}}
mydoc = mycol.find(myquery)

for x in mydoc:
    print(x['user']['name'], x['created_at'], x['text'])
```

```
margarita semit Fri Mar 08 17:40:12 +0000 2019 RT @justinsuntron: Wanna know more information about #TRON Japan community? Do n't forget to follow these channels. #TRON #TRX $TRX https://t.co/loP5AjYuVm
Tron Price ($TRX) Fri Mar 08 17:40:18 +0000 2019 #TRON $TRX:
Price (USD) $0.0228882242
Price (BTC) 0.00000582

Exchange on Binance with 50% discount trading.. https://t.co/CGc6ih8cgh
laverne franchuk Fri Mar 08 17:40:24 +0000 2019 RT @justinsuntron: Excited to be in Hong Kong to talk about the future of #decentralized ecosystems and methods of tokenizing the content s...
ashely wood Fri Mar 08 17:49:28 +0000 2019 RT @justinsuntron: Thanks for the support from #TRONICS. https://t.co/3vXXEPCRL7
melody parr Fri Mar 08 17:49:29 +0000 2019 RT @justinsuntron: #TRON weekly report 02.23-03.01 International Version 🌐 🌐 🌐 #TRX $TRX https://t.co/QwCdh1saak
Seim Kuruc Fri Mar 08 17:49:44 +0000 2019 RT @sesameseed_SR: $SEED is now available on @HuobiWallet via @TronTrade #DApp. Download Huobi Wallet during the promotion period to share...
margarita semit Fri Mar 08 17:49:46 +0000 2019 RT @justinsuntron: According to https://t.co/YoLhmhtq9y, as of this week, #TRON has over 187 #Dapps with more than 464 smart contracts. The...
nga kawahara Fri Mar 08 17:49:53 +0000 2019 RT @justinsuntron: Excited to be in Hong Kong to talk about the future of #decentralized ecosystems and methods of tokenizing the content s...
sohidul islam Fri Mar 08 17:49:55 +0000 2019 RT @HuobiWallet: Huobi Wallet supports #Tron DApps now, and has become one of the most Tron-friendly wallets in the world. To celebrated th...
david myers Fri Mar 08 17:49:56 +0000 2019 RT @justinsuntron: I'm supporting the @alsassociation with a $250,000 donation and voicing my love to the #TRON and #BitTorrent community!...
david myers Fri Mar 08 17:50:24 +0000 2019 RT @justinsuntron: According to https://t.co/YoLhmhtq9y, as of this week, #TRON has over 187 #Dapps with more than 464 smart contracts. The...
Rasit Fri Mar 08 17:50:32 +0000 2019 RT @TRON_topia: Happy international Women's day !

To celebrate #WomenOfTRON we are giving away 10000 $TRX and some rose emojis 🌹🌹🌹
```

Similarly, for coinmarketcap the keyword search was done using the “\$regex” operator. Since the data consist of only the top 100 cryptocurrencies, the query returns only one entry which is related to the Ripple currency. The coin collection contains only one entry for each cryptocurrency.

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mongodb"]
mycol2 = mydb["coin"]

#searching for keyword ripple
myquery2={"name": {'$regex':'xrp','$options':'$i'}}

mydoc2 = mycol2.find(myquery2)

#printing coinmarketcap data
for x in mydoc2:
    print(x)

{'_id': ObjectId('5c86c00f40ab6d3f30406662'), 'id': 'ripple', 'name': 'XRP', 'symbol': 'XRP', 'rank': '3', 'price_usd': '0.3102233279', 'price_btc': '0.00007976', '24h_volume_usd': '665937124.526', 'market_cap_usd': '12853216953.0', 'available_supply': '41432141931.0', 'total_supply': '99991683860.0', 'max_supply': '100000000000', 'percent_change_1h': '-0.27', 'percent_change_24h': '-1.16', 'percent_change_7d': '1.5', 'last_updated': '1552334705'}
```

Keyword search or pattern searching is relatively easier in MongoDB because of the functionality provided by the string pattern matching “\$regex” operator.

## (b) Searching for alternate forms of the same keyword

Alternate forms of the same keyword can be searched using the “\$or” operator. In the example below, I searched for the cryptocurrency TRON, another word used to identify the TRON currency is TRX.

```
myquery = {'$or':[{"text": {'$regex':'tron','$options':'$i'}}, {"text": {'$regex':'trx','$options':'$i'}}]}
```

```
#searching for alternative words of keyword
myquery = {'$or':[{"text": {'$regex':'tron','$options':'$i'}}, {"text": {'$regex':'trx','$options':'$i'}}]}
```

```
mydoc = mycol.find(myquery)

#printing only the username, created_at and text column of the tweets
for x in mydoc:
    print(x['user']['name'],x['created_at'],x['text'])
```

The output contains all the tweets that have both TRON and TRX in the text field. Similarly, for other cryptocurrencies we can search for other terms that are used to talk about those currencies using the “\$or” operator.




margarita semit Fri Mar 08 17:40:12 +0000 2019 RT @justinsuntron: Wanna know more information about #TRON Japan community? Do n't forget to follow these channels. #TRON #TRX \$TRX <https://t.co/CGcGih8cgh>

Tron Price (\$TRX) Fri Mar 08 17:40:18 +0000 2019 #TRON \$TRX:  
 Price (USD) \$0.0228882242  
 Price (BTC) 0.00000582

Exchange on Binance with 50% discount trading... <https://t.co/CGcGih8cgh>

laverne franchuk Fri Mar 08 17:40:24 +0000 2019 RT @justinsuntron: Excited to be in Hong Kong to talk about the future of #decentralized ecosystems and methods of tokenizing the content s...

ashely wood Fri Mar 08 17:49:28 +0000 2019 RT @justinsuntron: Thanks for the support from #TRONICS. <https://t.co/3vXXEPCRL7>

melody parr Fri Mar 08 17:49:29 +0000 2019 RT @justinsuntron: #TRON weekly report 02.23-03.01 International Version    #TRX \$TRX <https://t.co/QWcdhisaak>

Seim Kuruc Fri Mar 08 17:49:44 +0000 2019 RT @sesameseed\_SR: \$SEED is now available on @HuobiWallet via @TronTrade #DApp. Download Huobi Wallet during the promotion period to share...

margarita semit Fri Mar 08 17:49:46 +0000 2019 RT @justinsuntron: According to <https://t.co/YoLhmhtq9y>, as of this week, #TRON has over 187 #Dapps with more than 464 smart contracts. The...

nga kawahara Fri Mar 08 17:49:53 +0000 2019 RT @justinsuntron: Excited to be in Hong Kong to talk about the future of #decentralized ecosystems and methods of tokenizing the content s...

sohidul islam Fri Mar 08 17:49:55 +0000 2019 RT @HuobiWallet: Huobi Wallet supports #Tron DApps now, and has become one of the most Tron-friendly wallets in the world. To celebrated th...

davida myers Fri Mar 08 17:49:56 +0000 2019 RT @justinsuntron: I'm supporting the @alsassociation with a \$250,000 donation and voicing my love to the #TRON and #BitTorrent community!...

davida myers Fri Mar 08 17:50:24 +0000 2019 RT @justinsuntron: According to <https://t.co/YoLhmhtq9y>, as of this week, #TRON has over 187 #Dapps with more than 464 smart contracts. The...

Rasit Fri Mar 08 17:50:32 +0000 2019 RT @TRON\_topia: Happy international Women's day !

To celebrate #WomenOfTRON we are giving away 10000 \$TRX and some rose emojis 🌹🌹🌹

To par...

nga kawahara Fri Mar 08 17:50:32 +0000 2019 RT @justinsuntron: Tether will release a new stablecoin on #TRON blockchain. The issuer of stablecoin @Tether to and one of the largest blo...

The query below returns data from the collection “coin” that was extracted from coinmarketcap, to demonstrate searching for alternate forms of the same keyword I searched for XRP in both upper and lower case and removed the option field which is used to specify case-insensitive. Since Ripple is represented as “XRP” querying only “xrp” will return results results.

```
#searching for alternative words of keyword
```

```
myquery = {'$or': [{'name': {'$regex': 'XRP'}}, {'text': {'$regex': 'xrp'}}]}
mydoc2 = mycol2.find(myquery)
```

```
#printing data for Ripple
```

```
for x in mydoc2:
    print(x)
```

```
{ '_id': ObjectId('5c86c00f40ab6d3f30406662'), 'id': 'ripple', 'name': 'XRP', 'symbol': 'XRP', 'rank': '3', 'price_usd': '0.3102233279', 'price_btc': '0.00007976', '24h_volume_usd': '665937124.526', 'market_cap_usd': '12853216953.0', 'available_supply': '41432141931.0', 'total_supply': '99991683860.0', 'max_supply': '100000000000', 'percent_change_1h': '-0.27', 'percent_change_24h': '-1.16', 'percent_change_7d': '1.5', 'last_updated': '1552334705' }
```

## (c) Aggregation sums and count

In the below query with the count() function the total number of documents present in the collection can be validated.

```
mydoc=mycol.find()
#count of total number of documents
count = mydoc.count()
print(count)
```

374

There are 54 documents containing the term “Bitcoin”

```
#searching for keyword Bitcoin
myquery={"text": {'$regex': 'Bitcoin', '$options': '$i'}}
mydoc=mycol.find(myquery)
#count of number of documents containing the word bitcoin
count = mydoc.count()
print(count)
```

54

For coinmarketcap data the id field was searched for “bitcoin”

```
#searching for keyword Bitcoin
myquery={"id": 'bitcoin'}
mydoc=mycol2.find(myquery)
#count of number of documents containing the word bitcoin
count = mydoc.count()
print(count)
```

1

By grouping the tweets based on username we can get the count of the number of tweets posted by the users. From, the screenshot below we can see that “Vell Chapo” has 2 tweets.

```
#grouping the tweets based on username and finding the number of tweets per user
list(mycol.aggregate([{"$group": {"_id": "$user.name", "count": {"$sum": 1}}]))
```

```
[{'_id': 'Twon♣', 'count': 1},
 {'_id': 'Vell Chapo', 'count': 2},
 {'_id': 'Jenna Harner', 'count': 1},
 {'_id': 'Jake😄', 'count': 1},
 {'_id': 'luigi', 'count': 1},
 {'_id': 'Kevin Wynne', 'count': 1},
 {'_id': 'J.', 'count': 1},
 {'_id': 'Nate', 'count': 1},
 {'_id': '👁️ Jenny 👁️', 'count': 1},
 {'_id': 'Richard Hernandez', 'count': 1},
 {'_id': 'Brent', 'count': 1},
```

After checking MongoDB for the same user, we can see that there are 2 documents for “Vell Chapo”

The screenshot shows the MongoDB Compass interface for a database named 'mongodb.tweets'. The top bar indicates there are 374 documents, a total size of 1.9MB, an average size of 5.1KB, and 1 index with a total size of 36.0KB and an average size of 36.0KB. The 'Documents' tab is selected. The query bar shows a filter: {'user.name': 'Vell Chapo'}. Below the query bar, there are fields for 'PROJECT' (field: 0) and 'SORT' (field: -1). At the bottom, there are buttons for 'INSERT DOCUMENT', 'VIEW', 'LIST', and 'TABLE'. The status bar at the bottom right says 'Displaying documents 1 - 2 of 2'.



The query below aggregates the currencies from coinmarketcap based on “id” and counts how many times each has occurred

```
: #grouping the currencies based on id
list(mycol2.aggregate([{"$group":{"_id": "$id","count": {"$sum": 1}}}))

: [{"_id": 'linkey', 'count': 1},
  {'_id': 'power-ledger', 'count': 1},
  {'_id': 'wanchain', 'count': 1},
  {'_id': 'qash', 'count': 1},
  {'_id': 'pivx', 'count': 1},
  {'_id': 'decentraland', 'count': 1},
  {'_id': 'aelf', 'count': 1},
  {'_id': 'waltonchain', 'count': 1},
  {'_id': 'loom-network', 'count': 1},
  {'_id': 'maximine-coin', 'count': 1},
  {'_id': 'kucoin-shares', 'count': 1},
  {'_id': 'electroneum', 'count': 1},
  {'_id': 'hypercash', 'count': 1},
  {'_id': 'storj', 'count': 1},
  {'_id': 'qubitica', 'count': 1},
  {'_id': 'gxchain', 'count': 1},
```

#### (d) Conditions and order

The query below orders the tweets based on the time they were posted and prints the fields created\_at and text

```
#searching for keyword ripple
myquery={"text": {'$regex':'ripple','$options':'$i'}}

#ordering the tweets based on the time they were created
mydoc=mycol.find(myquery)
order = mydoc.sort("created_at")

for i in order:
    print(i['created_at'],i['text'])

Fri Mar 08 17:40:32 +0000 2019 RT @Arbitr0n: Coinbase was bribed by the #XRPArmy #XRPCommunity to list #Cripple. Jokes on them
when the @SEC Enforcement finds out about L...
Fri Mar 08 17:49:26 +0000 2019 So excited with #telcoin these days...keep rolling team!!
#Telfam #Bitcoin #Btc #ETH #xrp #ripple #TopInfluence... https://t.co/FU2hSN2F5i
Mon Mar 11 18:27:00 +0000 2019 Try out real time massive scale stateful ripples
Mon Mar 11 18:27:38 +0000 2019 The latest CRYPTO WATCHERS NEWS! https://t.co/EACPD7Xnq3 Thanks to @XRP_REYRIVAS @Lichking78 @Ji
llRTeamXRP #ripple #bitcoin
Mon Mar 11 18:27:38 +0000 2019 Founder of Ripple-Backed Startup Pushes for Mass XRP Adoption at 3.3 Million Online Stores | The
Daily Hodl... https://t.co/FAwv8NyMi1
Mon Mar 11 18:27:40 +0000 2019 RT @_Crypto_Maniac_: ⚠ IMPORTANT #XRP DATE ADDED ⚠

#Ripple at Money 20/20 March 20 - Singapore 11:05 (30 Minutes Long) https://t.co/01Z...
Mon Mar 11 18:27:56 +0000 2019 RT @CKJCryptonews: 8-Hour Trading Halt on World's Leading Bitcoin and Crypto Exchange - Plus Rip
ple and XRP, Ethereum, Tron, Litecoin, Stel...
Thu Mar 14 00:17:50 +0000 2019 RT @Journeyto100k_: My top 5 picks for 2019 in no particular order:

- Ripple $XRP
- TRON $TRX
```

The query below sorts the names of the cryptocurrencies from the coin collection in alphabetical order

```
: #ordering the crypto in alphabetical order
mydoc2=mycol2.find()
order = mydoc2.sort("name")

for i in order:
    print(i['name'])
```

0x  
ABBC Coin  
Aeternity  
Ardor  
Ark  
Augur  
Aurora  
Basic Attention Token  
Binance Coin  
BitShares  
Bitcoin  
Bitcoin Cash  
Bitcoin Diamond  
Bitcoin Gold  
Bitcoin SV  
Bytecoin  
Bytom  
Cardano  
Chainlink  
Crypto.com  
Crypto.com Chain  
Cryptonex  
Dai

### (e) Code complexity and query time

Pattern or keyword searches weren't too complex in MongoDB mainly because of the "\$regex" operator, by specifying the field and keywords code could be easily written to query the database of tweets. Since very few documents were extracted the queries did not take much time to fetch results, they were instantaneous. However, in real world scenarios for actual data analysis the datasets are incredibly large, and queries will take a much longer time to execute.

Python script for MongoDB queries: mongo.ipynb



## Data Exploration in Cassandra

First, we connect to Cassandra in Python using cluster from the cassandra library and calling the cluster() function

### (a) Searching for keywords

In order to perform a keyword search in Cassandra the column on which the search is done has to be indexed first, then using the LIKE operator a query can be written in cql. I indexed the “text” column of the tweets table and then searched for tweets that contained the term “Bitcoin”.

```
--
63 CREATE CUSTOM INDEX idx_text ON cassandra.tweets1 (text)
64 USING 'org.apache.cassandra.index.sasi.SASIIndex'
65 WITH OPTIONS = {
66   'mode': 'CONTAINS',
67   'analyzer_class': 'org.apache.cassandra.index.sasi.analyzer.NonTokenizingAnalyzer',
68   'case_sensitive': 'false'};
69
70
```

```
#connect to cassandra
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('cassandra')

#Keyword search using the LIKE operator in cassandra
#where text like '%bitocin%'
rows=session.execute("select * from tweets1 where text like '%bitcoin%'")
for user_row in rows:
    print (user_row.text)
```

The latest The People's Bitcoin! <https://t.co/aSkBnkHbF4> #bitcoin #btc  
RT @Airdropnotecom: #Bitcoin #Satoshi #Earn\_Money #cryptocurrency #blockchain #Airdrop  
New Airdrop #Kryptono 📢

Lamborghini to Win 🚗💰

Kry...

A Blockchain to Connect All Blockchains, Cosmos Is Now Officially Live

<https://t.co/Z440oKsSd6>

#Bitcoin... <https://t.co/x42YgeN8zL>  
RT @coinairdropall: #Bitcoin #crypto #Airdrop  
New Airdrop #BIGtoken 📢

Earn rewards when you Sign Up for #BIG and share your data

1. Sign...

Similarly, for data stored on the coin table the name column was indexed and a query using the like operator was used to search for currency that have the term Bitcoin, 5 currencies were returned.

```
--
92 CREATE CUSTOM INDEX idx_name ON cassandra.coin1 (name)
93 USING 'org.apache.cassandra.index.sasi.SASIIndex'
94 WITH OPTIONS = {
95   'mode': 'CONTAINS',
96   'analyzer_class': 'org.apache.cassandra.index.sasi.analyzer.NonTokenizingAnalyzer',
97   'case_sensitive': 'false'};
98
99
```

```
#connect to cassandra
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('cassandra')

#Keyword search using the LIKE operator in cassandra for coinmarketcap
#where text like '%bitocin%'
rows=session.execute("select * from coin1 where name like '%bitcoin%'")
for user_row in rows:
    print (user_row.name)
```

```
Bitcoin SV
Bitcoin Cash
Bitcoin
Bitcoin Diamond
Bitcoin Gold
```

## (b) Searching for alternate forms of the same keyword

It was difficult to search for alternate forms of the same keyword. So, individual queries had to be written to look for different keywords such as “Bitcoin”, “bit” and “BTC” from the tweets.

```
#searching for alternative forms of the same keyword bitcoin, bit, btc
rows=session.execute("select * from tweets1 where text like '%bitcoin%'")
rows1=session.execute("select * from tweets1 where text like '%bit%'")
rows2=session.execute("select * from tweets1 where text like '%btc%'")
for user_row in rows:
    print (user_row.text)
for user_row in rows1:
    print (user_row.text)
for user_row in rows2:
    print (user_row.text)
```

The latest The People's Bitcoin! <https://t.co/aSkBnkhbf4> #bitcoin #btc  
RT @Airdropnote.com: #Bitcoin #Satoshi #Earn\_Money #cryptocurrency #blockchain #Airdrop  
New Airdrop #Kryptono 📢

Lamborghini to Win 🚗💰

Kry...

A Blockchain to Connect All Blockchains, Cosmos Is Now Officially Live

<https://t.co/Z440oKsSd6>

#Bitcoin... <https://t.co/x42YgeN8zL>

RT @coinairdropall: #Bitcoin #crypto #Airdrop

New Airdrop #BIGtoken 📢

Earn rewards when you Sign Up for #BIG and share your data

1. Sign...

I was also able to use the IN operator to search for specific terms from the table coin

```
#searching for alternate forms of the same keyword
rows=session.execute("select * from coin1 where ID in ('bitcoin-sv','bitcoin-cash','bitcoin','bitcoin-diamond','bitcoin-gold')")
for user_row in rows:
    print (user_row.name)
```

```
Bitcoin
Bitcoin Cash
Bitcoin Diamond
Bitcoin Gold
Bitcoin SV
```

### (c) Aggregation sums and count

The query below returns the count of the total number of tweets stored, in total 2264 tweets were stored on the table tweets

```
#counting total number of tweets
rows=session.execute("select count(*) from tweets1")
for user_row in rows:
    print (user_row)
```

```
Row(count=2264)
```

The query below returns the count of all the tweets that contain the term “Bitcoin”, 506 tweets contained the term Bitcoin

```
#counting total number of tweets containing the term bitcoin
rows=session.execute("select count(*) from tweets1 where text like '%bitcoin%'")
for user_row in rows:
    print (user_row)
```

```
Row(count=506)
```

The top 100 cryptocurrencies were retrieved from coinmarketcap and stored on the coin1 table on Cassandra. Out of those currencies 5 of them contain the term Bitcoin in their name

```
#counting total number of records
rows=session.execute("select count(*) from coin1")
for user_row in rows:
    print (user_row)
```

Row(count=100)

```
#counting records that have the word bitcoin
#where text like '%bitocin%'
rows=session.execute("select count(*) from coin1 where name like '%bitcoin%'")
for user_row in rows:
    print (user_row)
```

Row(count=5)

#### (d) Conditions and order

In the query below the username “iotworkers” is filtered and the tweets are arranged in increasing order of timestamp. In Cassandra order by is not supported with secondary index, therefore searching for patterns and ordering the timestamp was not possible

In order to be able to sort with the created\_at column the created\_at column had to be specified as primary key and then with clustering order by.

```
139 CREATE TABLE tweets2 (
140     tweet_id bigint,
141     name varchar,
142     screen_name varchar,
143     created_at varchar,
144     text text,
145     followers int,
146     friends int,
147     PRIMARY KEY (name, created_at))
148 WITH CLUSTERING ORDER BY (created_at desc);
149
```

```
#ordering the queries by created_at time for user "iotworkers"
rows=session.execute("select * from tweets2 where name='iotworkers' order by created_at asc")
for user_row in rows:
    print (user_row.name, user_row.created_at)
```

```
iotworkers 2019-03-14 02:02:40+00:00
iotworkers 2019-03-14 02:03:02+00:00
iotworkers 2019-03-14 02:03:24+00:00
iotworkers 2019-03-14 02:13:41+00:00
iotworkers 2019-03-14 02:13:55+00:00
iotworkers 2019-03-14 02:14:38+00:00
iotworkers 2019-03-14 02:38:35+00:00
iotworkers 2019-03-14 02:38:49+00:00
iotworkers 2019-03-14 02:39:17+00:00
```

The data from coin market cap consist of only 100 records, however they all the records could not be ordered because Cassandra only supports ordering after filtering the data with an IN or = clause and there was not way to filter group of cryptocurrencies

Similarly, for the coin2 table rank was specified as a primary key and then in the clustering order by statement

```
106 CREATE TABLE coin2
107 (
108   sr_no int,
109   "24h_volume_usd" decimal,
110   available_supply double,
111   id varchar,
112   last_updated double,
113   market_cap_usd double,
114   max_supply double,
115   name varchar,
116   percent_change_1h decimal,
117   percent_change_24h decimal,
118   percent_change_7d decimal,
119   price_btc decimal,
120   price_usd decimal,
121   rank int,
122   symbol varchar,
123   total_supply decimal,
124   PRIMARY KEY (id,rank))
125   WITH CLUSTERING ORDER BY (rank desc);
126
```

```
#finding the rank of the currency binance-coin
rows=session.execute("select * from coin2 where id='binance-coin' order by rank asc")
for user_row in rows:
    print (user_row.rank)
```

7

### (e) Code complexity and query time

Since the syntax of CQL is like SQL it was quite easy to write queries. However, Cassandra is not a relational database management system it was difficult to query non-key columns and perform task such as pattern searching and sorting. In order to be able to search for patterns the columns had to be indexed first and to sort columns they must be included as primary keys. Since, only a few records were stored the queries did not take much time to execute, however when dealing with real-time data query might take much longer to run.

Python script for Cassandra queries: `cassandra.ipynb`

## Data Exploration Challenges

It was relatively easy to store and query data in MongoDB, on the other hand performing the same operations in Cassandra were quite challenging. Primarily because Cassandra appears as a relational database but behaves differently. The first challenge was performing a keyword or a pattern search, the “LIKE” clause could not be applied to columns that are not indexed, to resolve this I indexed the text column of tweets and the name column of the coin table, thus creating 2 secondary indexes. This simplified the querying enabled me to search for keywords in the tweets.

The other major challenge was sorting the data based on a column, in order to do this the column had to be specified as a primary key and then with clustering specify the column which should be used for ordering. It was not possible to order all the data of one table.

**Conclusion:** It was easier to store and retrieve data from MongoDB, since MongoDB can store unstructured data all the information gathered was in JSON format and stored directly on MongoDB in JSON format. Pattern or keywords searches were quite easy to perform because of the regex operator, even searching for multiple forms of the same keyword could be easily done with the or operator. On the other hand, Cassandra is not a very flexible database and there were lots of challenges right from data gathering, storage and retrieval. The data gathering had to be done through .csv files and then imported through cqlsh, while extracting and importing the data care had to be taken of the encoding to ensure that data could be properly loaded into the tables. In addition, running queries on non-indexed columns was a major challenge especially for while performing keyword searches and sorting data.