Sprawozdanie z projektu Big Data

Szymon Rećko, Hubert Ruczyński, Mateusz Ziemła

13.01.2023

Geneza projektu

Kryptowaluty stały się niesamowicie popularne w ciągu ostatnich kilku lat wśród wielu konsumentów i inwestorów. Zawdzięczają to głównie cenie i szybkości wykonywania transakcji z ich użyciem. Aby móc efektywnie działać na rynku kryptowalut należy posiadać narzędzia umożliwiające jego analizę. Szczególną uwagę należy poświęcić analizie podejścia użytkowników, które w znaczącym stopniu wpływa na wartość kryptowaluty.

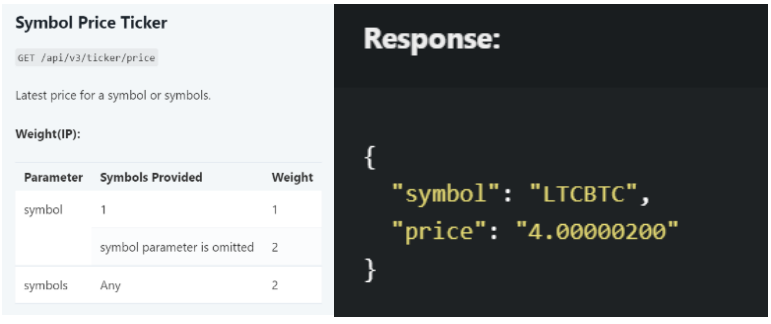
Celem naszego projektu jest zbadanie zależności między ilością tweetów publikowanych na Twitterze przez użytkowników, a wartościami kilku najpopularniejszych kryptowalut. Kryptowaluty, które zostaną wzięte pod uwagę to:

- Bitcoin (BTC)
- Ethereum (ETH)
- Dogecoin (DOGE)
- Cardano (ADA)
- Polygon (MATIC)
- XRP (XRP)

Dane

Akcje

Dane o wartościach kryptowalut są pobierane za pomocą "Binance API". Korzystając z ogólno dostępnej metody "*GET /api/v3/ticker/price"* możliwe jest pobranie jednym zapytaniem cen kryptowalut z przeliczeniem na różne rzeczywiste waluty krajowe (takie jak: USD, EUR, GBP). Dane są zwracane w ustalonym formacie JSON. Darmowa wersja API pozwala na wysłanie 1200 zapytań na minutę, przy czym różne zapytania mają różną "wagę" więc, aby nie przekroczyć limitu i nie dostać blokady na IP, wywołujemy zapytania dla wszystkich kryptowalut raz na 15 sekund.



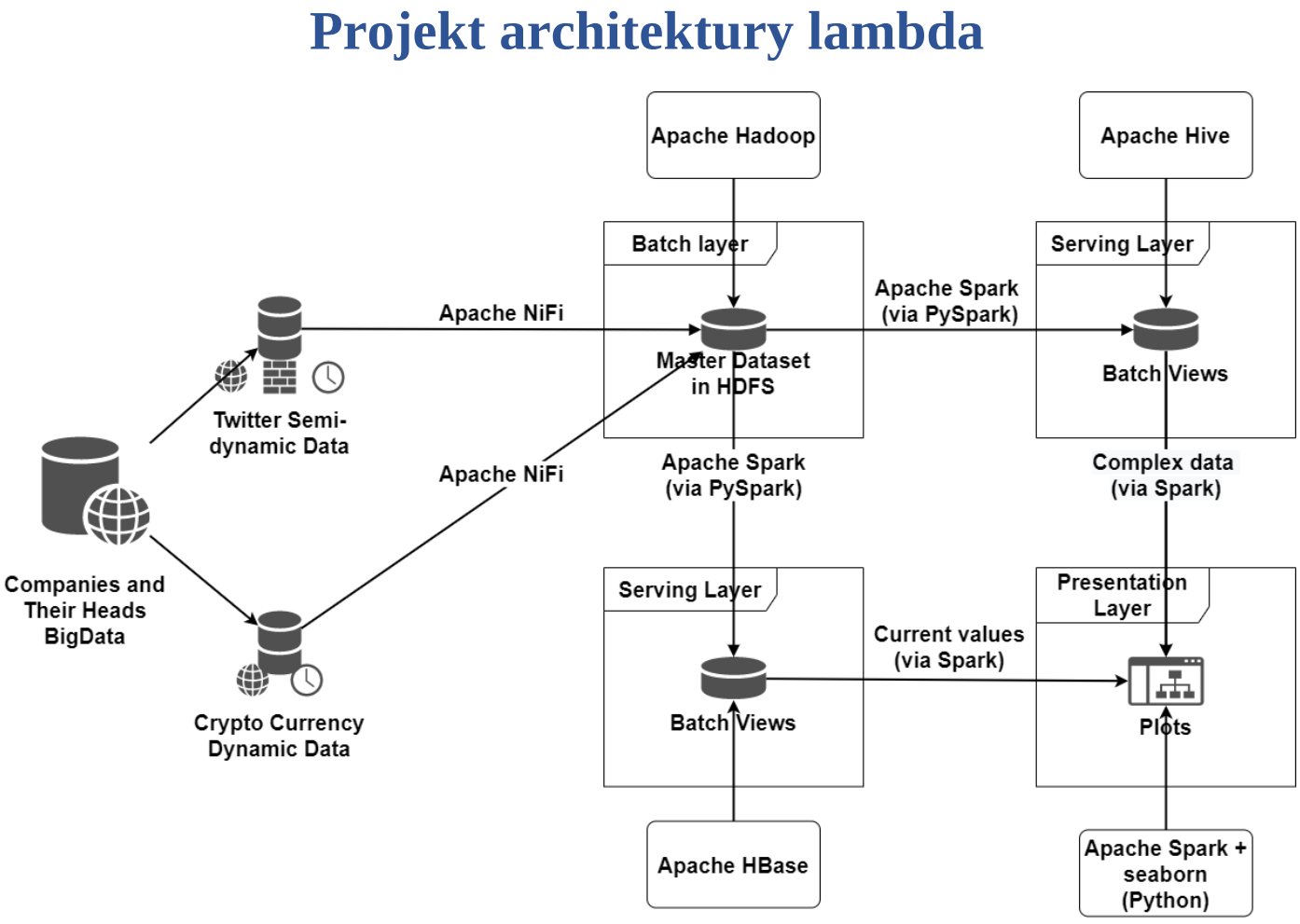
Rysunek 1 Binance API – przykładowe zapytanie

Tweety

Tweety powiązane treścią z kryptowalutami są pobierane za pomocą "Twitter API v2". Do filtrowania tweetów wykorzystujemy frazę 'crypto', tak aby wyodrębnić interesujące nas tweety. Dzięki temu, raz na minutę pobierana jest lista tweetów zawierająca timestamp, treść tweeta, oraz jego identyfikator. Wszystkie zapytania zwracają dane w postaci JSON.

Endpoints	Rate limit ⓘ	Tweet cap ⓘ	Special attributes
RECENT TWEET COUNTS			
GET /2/tweets/counts/recent	300 requests / 15 mins PER APP	no	• 512 query length ⓘ • core operators ⓘ

Rysunek 2 Twitter API – przykładowe zapytanie



Rysunek 3 Architektura Lambda

Składowanie danych

Apache Hadoop - Jako warstwę składowania master data set ‘u wybraliśmy Apache Hadoop. Jedną z kluczowych zalet Hadoop Distributed File System (HDFS) jest jego zdolność do zapewnienia wysokiej dostępności i odporności na uszkodzenia. Dane przechowywane są w wielu węzłach w klastrze w więcej niż jednej kopii, co oznaczą, że mimo awarii jednego z serwerów nadal można uzyskać dostęp do danych z innych węzłów. Dodatkową zaletą jest skalowalność. Wraz ze wzrostem ilości danych w głównym zbiorze, do klastra można dodać więcej węzłów, co pozwala systemowi obsługiwać większe ilości napływających danych. Ponadto użycie Hadoop’a pozwala nam obsługiwać zarówno dane strukturalne i niestrukturalne, a także SQLowe i NO-SQLowe, zatem nie musimy dodawać kolejnych narzędzi tak jak byłoby to w przypadku Hive. Dane zawarte w głównym zbiorze są odświeżane co 15 sekund. Pliki znajdują się w formacie CSV i dla łatwiejszego dalszego przetwarzania zawierają będą timestampy zarówno w nazwie jak i wewnątrz zawartości. Na poniższym zrzucie ekranu widać przykładową listę plików, które są składowane w HDFS. Z łatwością wskazać można sposób rozróżniania danych dotyczących Twittera oraz kryptowalut.

```
-rw-r--r--   1 root supergroup        755 2023-01-02 09:22 /user/BDprojekt/crypto_2023-01-02-09-22-54.csv
-rw-r--r--   1 root supergroup        753 2023-01-02 09:22 /user/BDprojekt/crypto_2023-01-02-09-22-59.csv
-rw-r--r--   1 root supergroup        755 2023-01-02 09:23 /user/BDprojekt/crypto_2023-01-02-09-23-05.csv
-rw-r--r--   1 root supergroup        101 2023-01-04 14:41 /user/BDprojekt/twitter_2023-01-04-14-41-06.csv
-rw-r--r--   1 root supergroup        101 2023-01-04 14:42 /user/BDprojekt/twitter_2023-01-04-14-42-04.csv
-rw-r--r--   1 root supergroup        101 2023-01-04 14:43 /user/BDprojekt/twitter_2023-01-04-14-43-04.csv
```

Rysunek 4 Przykład danych w HDFS

Apache Hive – W Hive składujemy będziemy widoki pochodzące z głównego zbioru danych. Zdecydowaliśmy się na Hive, gdyż łatwo sięga się do niego po dane z perspektywy aplikacji klienckiej. Każde uruchomienie skryptu w Sparku składuje w Hive obecny stan wszystkich znanych danych. W Hive przechowujemy dane w postaci rzędowej, dla łatwego dostępu do wszystkich danych, z partycjonowaniem po czasie, z którego pochodzą dane, w przypadku obu tabel.

crypto_data.symbol	crypto_data.price	crypto_data.time
BTCUSDT	16733.69	2023-01-02-09-22-22
ETHUSDT	1218.22	2023-01-02-09-22-22
ADAUSDT	0.2549	2023-01-02-09-22-22
XRPUSDT	0.345	2023-01-02-09-22-22
MATICUSDT	0.7798	2023-01-02-09-22-22
DOGEUSDT	0.0723	2023-01-02-09-22-22
BTCEUR	15562.84	2023-01-02-09-22-22
ETHEUR	1132.98	2023-01-02-09-22-22
XRPEUR	0.321	2023-01-02-09-22-22
BTCGBP	13872.73	2023-01-02-09-22-22

twitter_data.end	twitter_data.start	twitter_data.tweet_count	twitter_data.time
2023-01-02	2023-01-02	179	2023-01-02-09-22-22
2023-01-02	2023-01-02	174	2023-01-02-09-27-05
2023-01-02	2023-01-02	344	2023-01-02-09-32-04
2023-01-02	2023-01-02	156	2023-01-02-09-37-04
2023-01-02	2023-01-02	77	2023-01-02-09-42-04
2023-01-02	2023-01-02	65	2023-01-02-09-47-04
2023-01-02	2023-01-02	55	2023-01-02-09-52-04
2023-01-02	2023-01-02	56	2023-01-02-09-57-04
2023-01-02	2023-01-02	162	2023-01-02-10-02-04
2023-01-02	2023-01-02	59	2023-01-02-10-07-04
2023-01-02	2023-01-02	52	2023-01-02-10-12-04
2023-01-02	2023-01-02	84	2023-01-02-10-17-04
2023-01-02	2023-01-02	65	2023-01-02-10-22-04
2023-01-02	2023-01-02	53	2023-01-02-10-27-04
2023-01-02	2023-01-02	75	2023-01-02-10-32-04
2023-01-02	2023-01-02	58	2023-01-02-10-37-04
2023-01-02	2023-01-02	60	2023-01-02-10-42-04
2023-01-02	2023-01-02	90	2023-01-02-12-45-02
2023-01-03	2023-01-03	123	2023-01-03-12-57-57
2023-01-03	2023-01-03	119	2023-01-03-12-58-46

Rysunek 5 Przykład danych w Hive

Apache HBase - Jako platformę NO-SQL wybraliśmy HBase, do którego ładujemy dane pozyskiwane z API dotyczącego kryptowalut. Zdecydowaliśmy się na niego, ponieważ informacje o nim poznaliśmy na zajęciach, a jest także jednym z narzędzi sprawnie działających razem z Hadoop’em oraz ze Sparkiem. Dla łatwiejszego dostępu stworzyliśmy dodatkowe kolumny, symbol_1 oraz symbol_2, wskazujące osobno na symbole użyte w parach handlowych. Pogrupowaliśmy kolumny według tego czy opisują dane o cenie w danym momencie (price, time) oraz symboli walut których dotyczą dane.

```
hbase(main):002:0> scan 'projekt'
ROW                           COLUMN+CELL
 crypto_data_0                column=price_data:price, timestamp=2023-01-09T11:20:04.162, value=16736.58984375
 crypto_data_0                column=price_data:time, timestamp=2023-01-09T11:20:04.162, value=2023-01-02-10-19-11
 crypto_data_0                column=symbols:symbol_1, timestamp=2023-01-09T11:20:04.162, value=BTC
 crypto_data_0                column=symbols:symbol_2, timestamp=2023-01-09T11:20:04.162, value=USDT
 crypto_data_0                column=symbols:trade_symbol, timestamp=2023-01-09T11:20:04.162, value=BTCUSDT
 crypto_data_1                column=price_data:price, timestamp=2023-01-09T11:20:04.187, value=1218.75
 crypto_data_1                column=price_data:time, timestamp=2023-01-09T11:20:04.187, value=2023-01-02-10-19-11
 crypto_data_1                column=symbols:symbol_1, timestamp=2023-01-09T11:20:04.187, value=ETH
 crypto_data_1                column=symbols:symbol_2, timestamp=2023-01-09T11:20:04.187, value=USDT
 crypto_data_1                column=symbols:trade_symbol, timestamp=2023-01-09T11:20:04.187, value=ETHUSDT
```

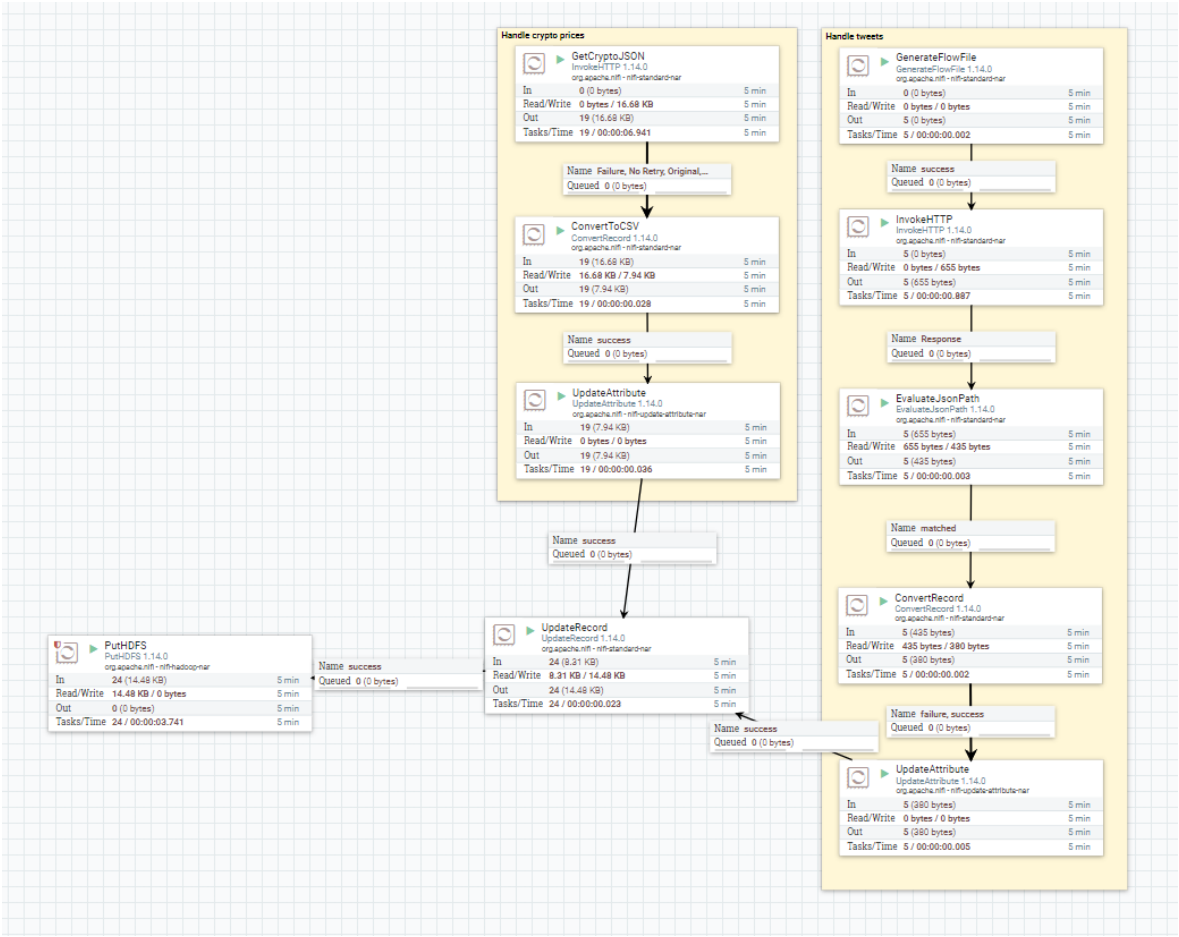
```
hbase(main):001:0> describe 'projekt'
Table projekt is ENABLED
projekt
COLUMN FAMILIES DESCRIPTION
{NAME => 'price_data', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA
_BLOCK_ENCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'false', BLOCKSIZ
E => '65536', REPLICATION_SCOPE => '0'}

{NAME => 'symbols', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BL
OCK_ENCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'false', BLOCKSIZE =
> '65536', REPLICATION_SCOPE => '0'}
```

Rysunek 6 Przykład danych w HBase

Przepływ danych

Apache NiFi – NiFi służyć nam będzie w procesach pozyskiwania danych z API oraz ich wstępnego preprocessing 'u tak aby do HDFS'a trafiały już wstępnie przygotowane dane. NiFi pobierać będzie zatem tweety i komunikować się z Binance oraz Twitter API. Głównym powodem użycia Apache NIFI w tym projekcie była jego zdolność do obsługi szerokiej gamy źródeł i formatów danych. Jest to możliwe dzięki wbudowanym procesorom, które zapewniają również możliwość manipulacji danymi np. filtrowanie i agregację. Ponadto można go łatwo integrować z innymi użytymi przez nas systemami. Apache NIFI dostarcza dodatkowo interfejs użytkownika ułatwiający monitorowanie i zarządzanie przepływem danych. Jest to przydatne do inspekcji zbieranych danych, a także rozwiązywania wszelkich problemów, które mogą pojawić się na tym etapie. Nasze flow w NIFI dla kryptowalut składa się kolejno z: pobrania danych w formacie JSON, konwersji do formatu CSV i zmiany rozszerzenia pliku na ".csv". Flow dla tweetów jest analogiczne z tą różnicą, że na początku generowany jest flowfile z danymi potrzebnymi do wykonania zapytania do API. Niezależnie od ścieżki, do danych zostaje dodany timestamp, a następnie dane umieszczane są w HDFS.



Rysunek 7 Architektura NiFI

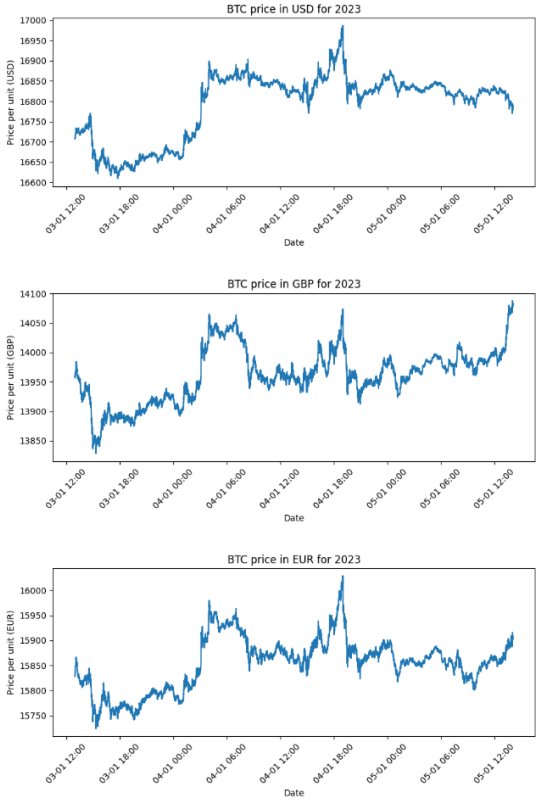
Apache Spark – Spark wykorzystywany będzie w pozostałych przypadkach, gdzie potrzebne będą bardziej skomplikowane operacje. Pierwszym jego zadaniem będzie przetworzenie danych z HDFS tak aby do Hive'a trafiały już w formie sensownych Batch Views. Ponadto za pomocą Sparka właśnie przekazywać będziemy dane z HBase i Hive do warstwy prezentacji. Jako że numpy średnio radzi sobie z przetwarzaniem dużych danych te przetwarzane będą wewnątrz Sparka z poziomu Python'owego Notebook'a i biblioteki PySpark.

Wsadowa analiza danych

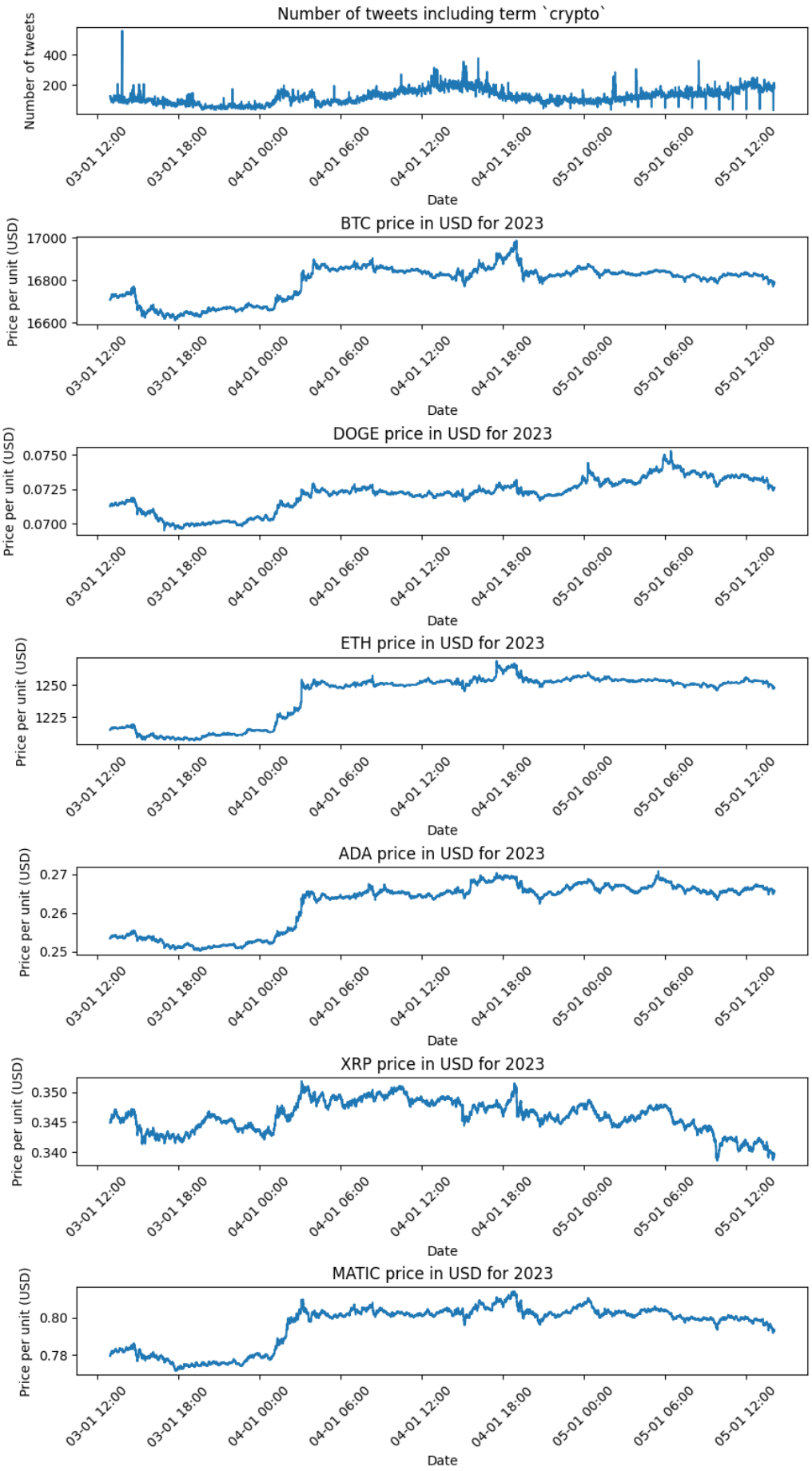
Dane pobierane są z utworzonych wcześniej Hive oraz HBase poprzez Apache Spark, następnie przerzucane są do ramki pandas w Python, aby użyć biblioteki seaborn do stworzenia wizualizacji odpowiadających na pytania:

1. Czy względne zmiany wartości kryptowalut zależą od waluty państwowej na którą tą wartość rzutujemy?
2. Czy częstsze tweetowanie o kryptowalutach (zawierające słowo ‘crypto’) wpływa na kursy kryptowalut?
3. Czy kurs kryptowalut wpływa na częstsze tweetowanie o kryptowalutach?
4. Czy kursy kryptowalut w czasie różnią się istotnie?
5. Czy da się zabezpieczać przy inwestycji w kryptowaluty poprzez kupowanie odwrotnie skorelowanych walut?
6. Jaka waluta się najmniej zmienia i jest najbezpieczniejsza?

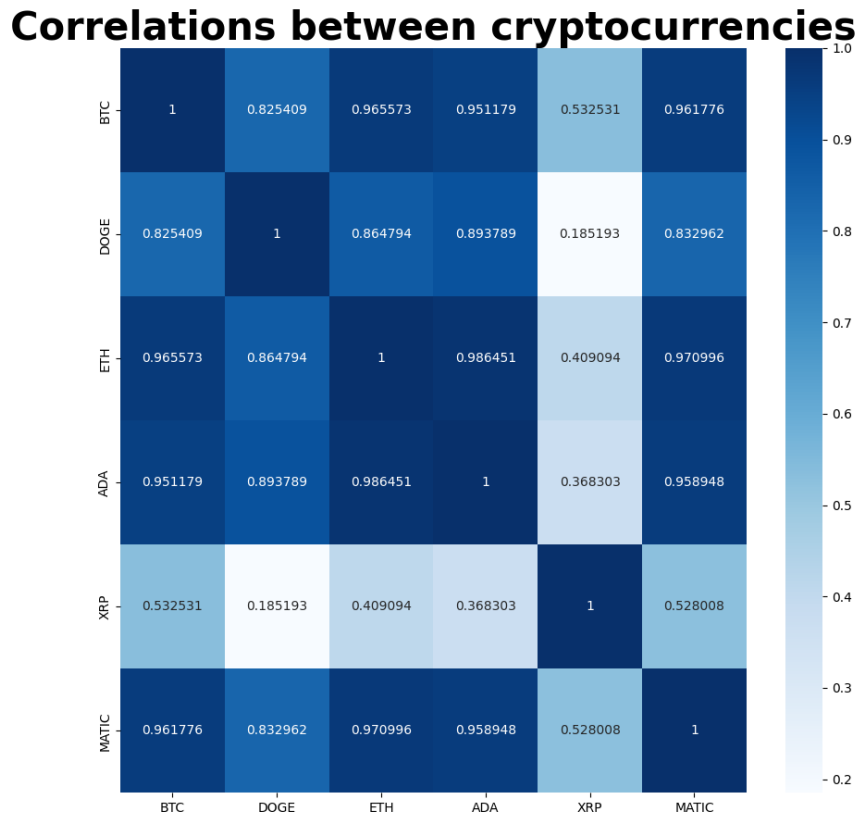
Na przykładzie bitcoina zaprezentowanego na pierwszym wykresie Rysunek 8 widać, że kursy w pewnym stopniu różnią się od siebie w zależności od waluty w której kupić chcemy daną kryptowalutę. Różnice widoczne są szczególnie na drugim peaku. Na podstawie wykresu Rysunek 9 jesteśmy w stanie stwierdzić, że to liczba tweetów wpływa na wartości kryptowalut. Co więcej zauważyć można, że kolejne kursy w czasie wyglądają bardzo podobnie, co potwierdza następna wizualizacja Rysunek 10 prezentująca korelacje między kursami kryptowalut. Warto wyróżnić tutaj walutę XRP, która jako jedyne zachowuje się niejako odwrotnie od reszty, przez co może zostać użyta jako zabezpieczanie wymieniane w punkcie 5. Potwierdzenie tej relacji widoczne jest na ostatnim wykresie Rysunek 11. Dodatkowo stworzyliśmy analizę, która określa w jakiej walucie odnotowujemy najmniejsze zmiany – jest najbardziej stabilna, co widoczne jest w tabelce Tabela 1.



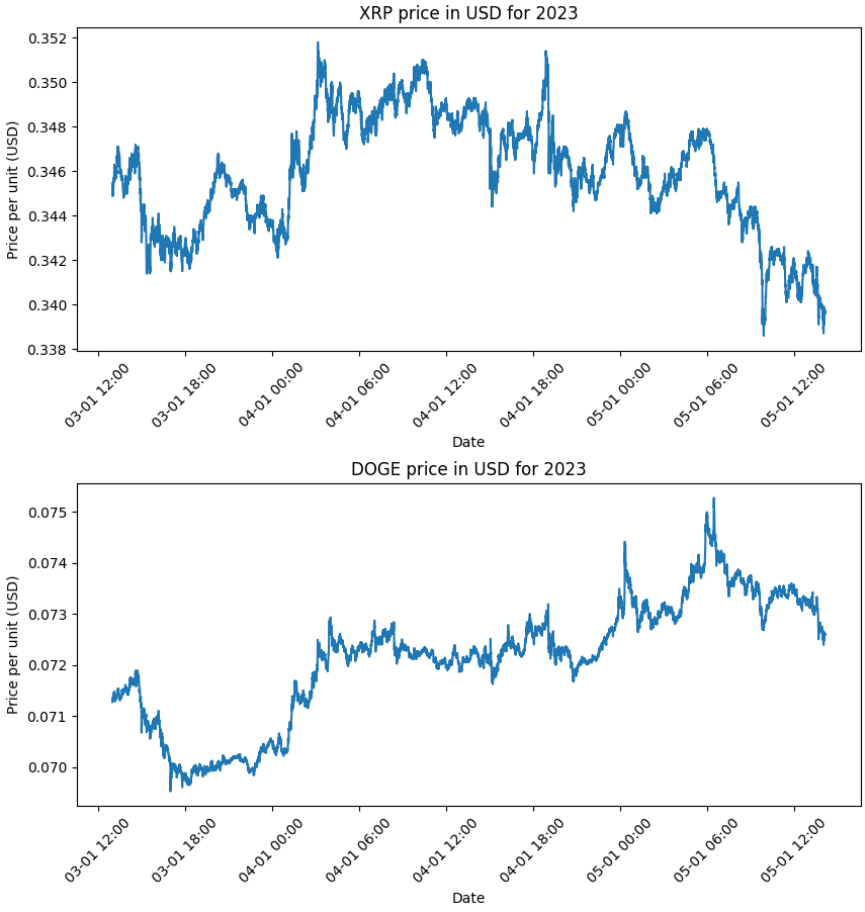
Rysunek 12 Porównanie ceny kryptowaluty w różnych walutach krajowych



Rysunek 13 Porównanie wartości kryptowalut z ilością tweetów w czasie



Rysunek 14 Macierz korelacji kryptowalut



Rysunek 15 Porównanie kursów XRP oraz DOGE Coin

	Crypto	Min	Min_rel	Mean	Max	Max_rel
0	BTCUSDT	16609.220703	0.989404	16787.099751	16986.230469	1.011862
1	ETHUSDT	1206.599976	0.964140	1239.648720	1268.550049	1.043884
2	DOGEUSDT	0.069520	0.964140	0.072106	0.075270	1.043884
3	DOTUSDT	4.481000	0.976397	4.589323	4.685000	1.020848
4	XRPUSDT	0.338600	0.978972	0.345873	0.351800	1.017136
5	ADAUSDT	0.250200	0.956561	0.261562	0.270800	1.035319

Tabela 2 Analiza stabilności kryptowalut

Repozytorium

Do pracy nad projektem zdecydowaliśmy się wykorzystać środowisko, w którym każdy z nas czuje się swobodnie, a mianowicie GitHuba. Repozytorium jest publiczne, projekt tworzony jest jako Open Source, a poniżej zamieszczony jest link do repozytorium: https://github.com/HubertR21/BigDataProject.

Testy

Obsługa testów

Aby rozpocząć pipeline testowania należy podążać za niniejszą instrukcją. Pierwszym krokiem jest stworzenie folderu do testowania wewnątrz hdfs za pomocą komendy: *hadoop fs -mkdir /user/testowanie*. Następnie sprawdzamy, że folder jest pusty za pomocą: *hadoop fs -ls /user/testowanie,* co wygląda tak:

```
vagrant@node1:~$ hadoop fs -ls /user/testowanie
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/im
pl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLog
gerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
vagrant@node1:~$
```

Następnie należy dostać się do NiFI za pomocą: http://localhost:9443/nifi/ , zlokalizować grupę procesów o nazwie testowanie, przejść do niej i włączyć wszystkie znajdujące się w niej procesory. Dane zaciągać w ten sposób przez co najmniej 3,5 minuty (4 krotne stworzenie pliku potrzebnego do zaciągnięcia danych przez procesor GenerateFlowFile dla tweetów) i wyłączyć po tym czasie. Aby sprawdzić wyniki działania NiFI wywołujemy następującą komendę: *hadoop fs -ls /user/testowanie.* W efekcie uzyskujemy wynik podobny do tego:

```
vagrant@node1:~$ hadoop fs -ls /user/testowanie
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/im
pl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLog
gerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 18 items
-rw-r--r--   1 root supergroup        753 2023-01-10 10:49 /user/testowanie/crypto_2023-01-10-10-49-37.csv
-rw-r--r--   1 root supergroup        752 2023-01-10 10:49 /user/testowanie/crypto_2023-01-10-10-49-52.csv
-rw-r--r--   1 root supergroup        752 2023-01-10 10:50 /user/testowanie/crypto_2023-01-10-10-50-07.csv
-rw-r--r--   1 root supergroup        752 2023-01-10 10:50 /user/testowanie/crypto_2023-01-10-10-50-22.csv
-rw-r--r--   1 root supergroup        753 2023-01-10 10:50 /user/testowanie/crypto_2023-01-10-10-50-37.csv
-rw-r--r--   1 root supergroup        752 2023-01-10 10:50 /user/testowanie/crypto_2023-01-10-10-50-53.csv
-rw-r--r--   1 root supergroup        754 2023-01-10 10:51 /user/testowanie/crypto_2023-01-10-10-51-08.csv
-rw-r--r--   1 root supergroup        751 2023-01-10 10:51 /user/testowanie/crypto_2023-01-10-10-51-23.csv
-rw-r--r--   1 root supergroup        750 2023-01-10 10:51 /user/testowanie/crypto_2023-01-10-10-51-38.csv
-rw-r--r--   1 root supergroup        751 2023-01-10 10:51 /user/testowanie/crypto_2023-01-10-10-51-54.csv
-rw-r--r--   1 root supergroup        750 2023-01-10 10:52 /user/testowanie/crypto_2023-01-10-10-52-10.csv
-rw-r--r--   1 root supergroup        753 2023-01-10 10:52 /user/testowanie/crypto_2023-01-10-10-52-26.csv
-rw-r--r--   1 root supergroup        752 2023-01-10 10:52 /user/testowanie/crypto_2023-01-10-10-52-42.csv
-rw-r--r--   1 root supergroup        755 2023-01-10 10:52 /user/testowanie/crypto_2023-01-10-10-52-57.csv
-rw-r--r--   1 root supergroup        101 2023-01-10 10:49 /user/testowanie/twitter_2023-01-10-10-49-37.csv
-rw-r--r--   1 root supergroup        101 2023-01-10 10:50 /user/testowanie/twitter_2023-01-10-10-50-35.csv
-rw-r--r--   1 root supergroup        101 2023-01-10 10:51 /user/testowanie/twitter_2023-01-10-10-51-35.csv
-rw-r--r--   1 root supergroup        101 2023-01-10 10:52 /user/testowanie/twitter_2023-01-10-10-52-35.csv
```

Rysunek 16 Test_NiFI

Kolejnym krokiem jest przetestowanie przetwarzania danych za pomocą PySpark do Hive oraz HBase wykonywane przez plik *projekt/sparkito_test.py*. Domyślnie odpowiednie tabele (`twitter_data_test, crypto_data_test, oraz projekt_test`) nie znajdują się w Hive oraz HBase, co sprawdzić możemy poprzez komendy:

beeline -e 'show tables;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant

```
vagrant@node1:~$ beeline -e 'show tables;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant
Connecting to jdbc:hive2://localhost:10000
23/01/10 10:44:35 INFO jdbc.Utils: Supplied authorities: localhost:10000
23/01/10 10:44:35 INFO jdbc.Utils: Resolved authority: localhost:10000
Connected to: Apache Hive (version 2.3.8)
Driver: Hive JDBC (version 2.3.7)
Transaction isolation: TRANSACTION_REPEATABLE_READ
+---------------+
|   tab_name    |
+---------------+
| crypto_data   |
| employees     |
| salaries      |
| twitter_data  |
| wifi          |
+---------------+
5 rows selected (0.71 seconds)
Beeline version 2.3.7 by Apache Hive
Closing: 0: jdbc:hive2://localhost:10000
vagrant@node1:~$
```

/usr/local/hbase/bin/hbase shell

list

```
vagrant@node1:~$ /usr/local/hbase/bin/hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/apache-tez-0.9.1-bin/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hbase-2.3.5/lib/client-facing-thirdparty/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.5, rfd3fdc88d1cd43eb3432a1a70d31c3aece6ecabe, Thu Mar 25 20:50:15 UTC 2021
Took 0.0008 seconds
hbase(main):001:0> list
TABLE
books
employees
person
prog_langs
projekt
test
test2
trains
trees
tweets
wifi
11 row(s)
Took 0.3828 seconds
=> ["books", "employees", "person", "prog_langs", "projekt", "test", "test2", "trains", "trees", "tweets", "wifi"]
hbase(main):002:0>
```

Aby rozpocząć proces przetwarzania należy wykonać następującą komendę: *python projekt/sparkito_test.py*. W efekcie skrypt stworzy tabele w Hive: `twitter_data_test` oraz `crypto_data_test,` natomiast w HBase stworzona zostanie tabela `projekt_test.` Obserwacje pochodzące z dwóch pierwszych wypisywane są także podczas działania skryptu. Wyniki sprawdzić można za pomocą następujących komend, w efekcie których powinniśmy otrzymać wyniki podobne do tych:

beeline -e 'show tables;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant

*beeline -e 'SELECT * FROM crypto_data_test LIMIT 10;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant*

*beeline -e 'SELECT * FROM twitter_data_test LIMIT 10;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant*

```
vagrant@node1:~$ beeline -e 'show tables;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant
Connecting to jdbc:hive2://localhost:10000
23/01/10 11:03:23 INFO jdbc.Utils: Supplied authorities: localhost:10000
23/01/10 11:03:23 INFO jdbc.Utils: Resolved authority: localhost:10000
Connected to: Apache Hive (version 2.3.8)
Driver: Hive JDBC (version 2.3.7)
Transaction isolation: TRANSACTION_REPEATABLE_READ
+---------------------+
|      tab_name       |
+---------------------+
| crypto_data         |
| crypto_data_test    |
| employees           |
| salaries            |
| twitter_data        |
| twitter_data_test   |
| wifi                |
+---------------------+
7 rows selected (0.116 seconds)
Beeline version 2.3.7 by Apache Hive
Closing: 0: jdbc:hive2://localhost:10000
vagrant@node1:~$
```

```
vagrant@node1:~$ beeline -e 'SELECT * FROM crypto_data_test LIMIT 10;' -u jdbc:hive2://localhost:10000 -n vagrant p vagrant
Connecting to jdbc:hive2://localhost:10000
23/01/10 11:05:32 INFO jdbc.Utils: Supplied authorities: localhost:10000
23/01/10 11:05:32 INFO jdbc.Utils: Resolved authority: localhost:10000
Connected to: Apache Hive (version 2.3.8)
Driver: Hive JDBC (version 2.3.7)
Transaction isolation: TRANSACTION_REPEATABLE_READ
+---------------------------+--------------------------+--------------------------+
| crypto_data_test.symbol   | crypto_data_test.price   | crypto_data_test.time    |
+---------------------------+--------------------------+--------------------------+
| BTCUSDT                   | 17264.91                 | 2023-01-10-10-49-37      |
| ETHUSDT                   | 1332.81                  | 2023-01-10-10-49-37      |
| ADAUSDT                   | 0.3193                   | 2023-01-10-10-49-37      |
| XRPUSDT                   | 0.3499                   | 2023-01-10-10-49-37      |
| MATICUSDT                 | 0.849                    | 2023-01-10-10-49-37      |
| DOGEUSDT                  | 0.07664                  | 2023-01-10-10-49-37      |
| BTCEUR                    | 16090.24                 | 2023-01-10-10-49-37      |
| ETHEUR                    | 1242.08                  | 2023-01-10-10-49-37      |
| XRPEUR                    | 0.3263                   | 2023-01-10-10-49-37      |
| BTCGBP                    | 14191.17                 | 2023-01-10-10-49-37      |
+---------------------------+--------------------------+--------------------------+
10 rows selected (0.371 seconds)
Beeline version 2.3.7 by Apache Hive
Closing: 0: jdbc:hive2://localhost:10000
vagrant@node1:~$
```

```
vagrant@node1:~$ beeline -e 'SELECT * FROM twitter_data_test LIMIT 10;' -u jdbc:hive2://localhost:10000 -n vagrant p vag
rant
Connecting to jdbc:hive2://localhost:10000
23/01/10 11:11:12 INFO jdbc.Utils: Supplied authorities: localhost:10000
23/01/10 11:11:12 INFO jdbc.Utils: Resolved authority: localhost:10000
Connected to: Apache Hive (version 2.3.8)
Driver: Hive JDBC (version 2.3.7)
Transaction isolation: TRANSACTION_REPEATABLE_READ
+-------------------------+---------------------------+--------------------------------+--------------------------+
| twitter_data_test.end   | twitter_data_test.start   | twitter_data_test.tweet_count  | twitter_data_test.time   |
+-------------------------+---------------------------+--------------------------------+--------------------------+
| 2023-01-10              | 2023-01-10                | 117                            | 2023-01-10-10-49-37      |
| 2023-01-10              | 2023-01-10                | 126                            | 2023-01-10-10-50-35      |
| 2023-01-10              | 2023-01-10                | 137                            | 2023-01-10-10-51-35      |
| 2023-01-10              | 2023-01-10                | 130                            | 2023-01-10-10-52-35      |
+-------------------------+---------------------------+--------------------------------+--------------------------+
4 rows selected (0.372 seconds)
Beeline version 2.3.7 by Apache Hive
Closing: 0: jdbc:hive2://localhost:10000
```

Rysunek 17 Tests Hive

/usr/local/hbase/bin/hbase shell

list

scan ‘projekt_test’

describe ‘projekt_test’

```
hbase(main):003:0> list
TABLE
books
employees
person
prog_langs
projekt
projekt_test
test
test2
trains
trees
tweets
wifi
12 row(s)
Took 0.0205 seconds
=> ["books", "employees", "person", "prog_langs", "projekt", "projekt_test", "test", "test2", "trains", "trees", "tweets
", "wifi"]
```

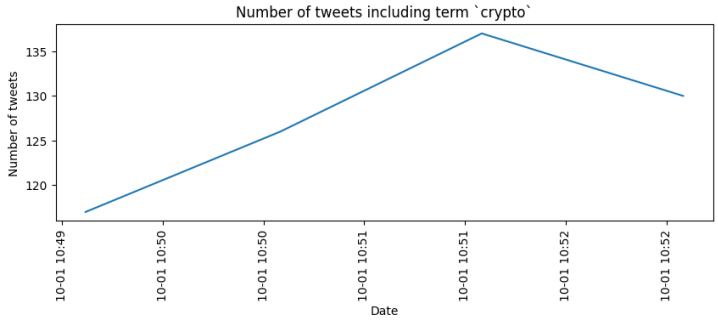
```
hbase(main):001:0> scan 'projekt_test'
ROW                    COLUMN+CELL
 crypto_data_0         column=price_data:price, timestamp=2023-01-10T10:55:50.529, value=17268.330078125
 crypto_data_0         column=price_data:time, timestamp=2023-01-10T10:55:50.529, value=2023-01-10-10-52-57
 crypto_data_0         column=symbols:symbol_1, timestamp=2023-01-10T10:55:50.529, value=BTC
 crypto_data_0         column=symbols:symbol_2, timestamp=2023-01-10T10:55:50.529, value=USDT
 crypto_data_0         column=symbols:trade_symbol, timestamp=2023-01-10T10:55:50.529, value=BTCUSDT
 crypto_data_1         column=price_data:price, timestamp=2023-01-10T10:55:50.547, value=1332.8699951171875
 crypto_data_1         column=price_data:time, timestamp=2023-01-10T10:55:50.547, value=2023-01-10-10-52-57
 crypto_data_1         column=symbols:symbol_1, timestamp=2023-01-10T10:55:50.547, value=ETH
 crypto_data_1         column=symbols:symbol_2, timestamp=2023-01-10T10:55:50.547, value=USDT
 crypto_data_1         column=symbols:trade_symbol, timestamp=2023-01-10T10:55:50.547, value=ETHUSDT
```

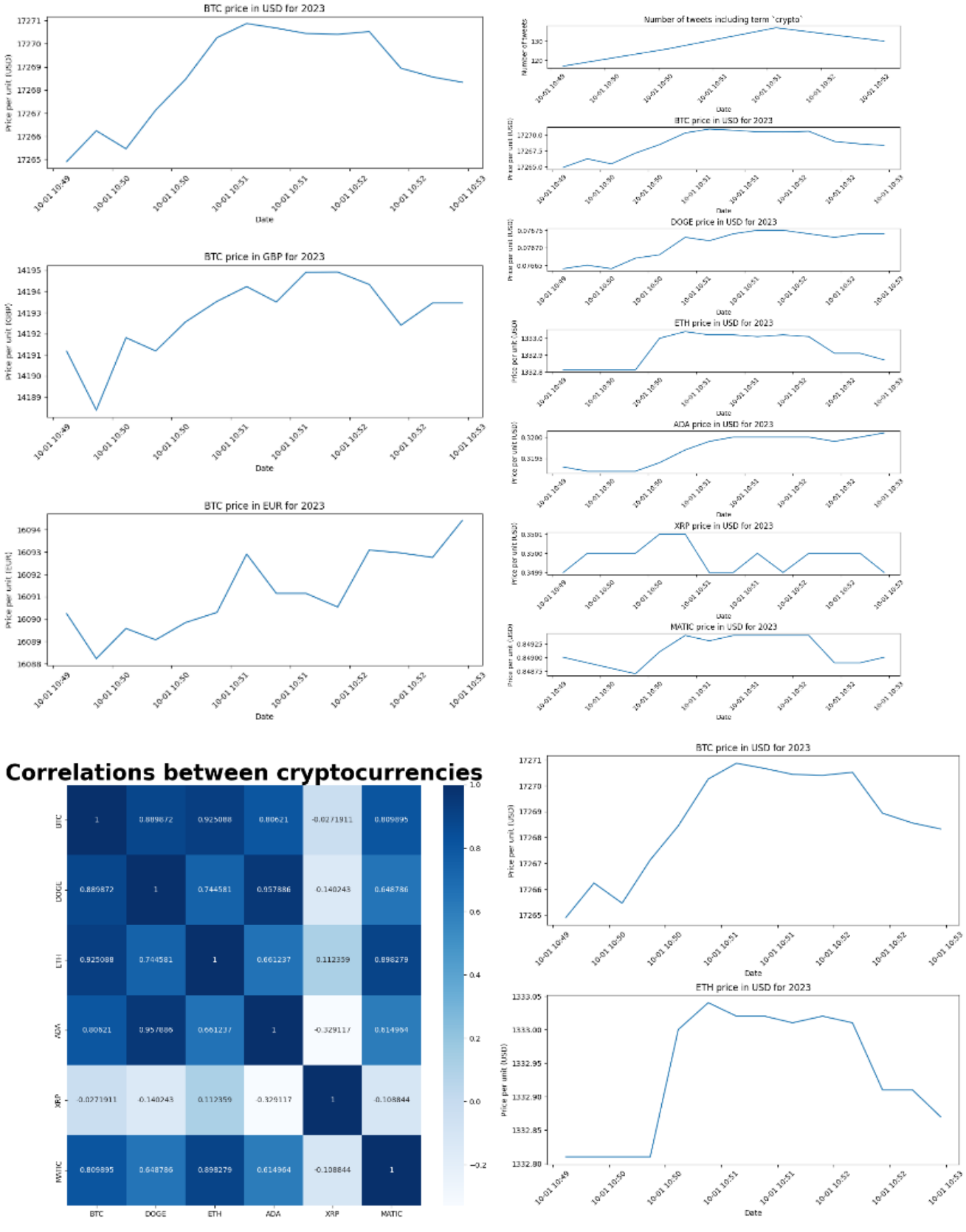
```
hbase(main):002:0> describe 'projekt_test'
Table projekt_test is ENABLED
projekt_test
COLUMN FAMILIES DESCRIPTION
{NAME => 'price_data', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA
_BLOCK_ENCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'false', BLOCKSIZ
E => '65536', REPLICATION_SCOPE => '0'}

{NAME => 'symbols', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BL
OCK_ENCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'false', BLOCKSIZE =
> '65536', REPLICATION_SCOPE => '0'}
```

Rysunek 18 Tests HBase

Ostatnim etapem jest tworzenie wizualizacji, które testowane jest za pomocą przejścia przez kolejne komórki w pliku `Plots_test.ipynb`. Opisane tam działania sprawdzają czy jesteśmy w stanie odczytywać dane z Hive i HBase oraz czy wykresy generują się poprawnie. Aby uruchomić notatnik należy wpisać komendę *jupyter notebook*, następnie wejść do folderu projekt i otworzyć plik `Plots_test.ipynb`. Wizualizacje powinny wyglądać podobnie do zamieszczonych poniżej:





Rysunek 19 Tests Python

Wyniki

Testowane narzędzie	Cel testu	Jak działa test?	Spodziewany rezultat	Przykładowy rezultat
NiFI	Sprawdzenie pobierania za pomocą API i transformacji danych w NiFI.	Użytkownik podąża za kolejnymi poleceniami [x] tworząc folder hdfs, włączając procesory testowe oraz sprawdza czy dane się pobrały.	Wewnątrz HDFS pojawiają się dane pobrane przez NiFI z odpowiednimi nazwami (crypto/twitter_date).	Rysunek 16 Test_NiFI
Spark Hive HBase	Sprawdzenie czy skrypt napisany w PySpark poprawnie obsługuje dane.	Użytkownik podąża za kolejnymi poleceniami [x] sprawdzając stan przed wykonaniem skryptu, wywołując skrypt, oraz sprawdzając stan po jego wywołaniu.	Wewnątrz Hive pojawiają się dwie tabele `twitter_data_test` oraz `crypto_data_test`, a wewnątrz HBase jedna `projekt_test`.	Rysunek 17 Tests Hive Rysunek 18 Tests HBase
Python (wykresy)	Sprawdzenie czy wykresy są generowane.	Użytkownik włącza jupyter notebook, otwiera odpowiedni plik ipynb i wywołuje przygotowane komórki.	Wykresy łączą się z danymi i poprawnie generują (bez błędów).	Rysunek 19 Tests Python

Podział pracy

Szymon Rećko:

- Rozplanowanie projektu.
- NiFi: komunikacja z API, wstępny preprocessing danych, zapisanie danych do HDFS.
- Konfiguracja HDFS'a.

Hubert Ruczyński:

- Rozplanowanie projektu.
- Zbieranie danych za pomocą Sparka do warstwy prezentacji.
- Stworzenie notebook'a jako warstwy prezentacji (wizualizacja).
- Nadzorowanie i debugowanie każdego etapu.

Mateusz Ziemła:

- Konfiguracja HBase'a.
- Konfiguracja Hive'a.
- Konfiguracja Sparka między Hadoop'em, a Hive'em.