

**PES UNIVERSITY**  
100 feet Ring Road, BSK 3<sup>rd</sup> Stage  
Bengaluru 560085



Department of Computer Science and Engineering  
B. Tech. CSE - 6<sup>th</sup> Semester  
Jan – May 2023

UE20CS343  
**DATABASE TECHNOLOGIES (DBT)**  
**Project Report**

# Analysis of real time price of Bitcoin using Kafka and Spark

TEAM  
PES1UG20ME009: Alleti Rohan Reddy  
PES1UG20CS043: Ananya Menon  
Class of Prof. Suresh Jamadagni

# Table of Contents

1. Introduction
2. Installation of Software [include version #s and URLs]
  - a. Streaming Tools Used
  - b. DBMS Used
3. Problem Description
4. Architecture Diagram
5. Input Data
  - a. Source
  - b. Description
6. Streaming Mode Experiment
  - a. Windows
  - b. Workloads
  - c. Code like SQL Scripts
  - d. Inputs and Corresponding Results
7. Batch Mode Experiment
  - a. Description
  - b. Data Size
  - c. Results
8. Comparison of Streaming & Batch Modes
  - a. Results and Discussion
9. Conclusion
10. References
  - a. URLs

## 1. Introduction

Bitcoin is a digital cryptocurrency that has gained a lot of attention in the last few years. The price of bitcoin fluctuates constantly, and it is essential for traders and investors to keep track of the real-time price of bitcoin. In this report, we analyze the real-time price of bitcoin using Spark and Kafka.

Spark is a fast and flexible big data processing engine that allows for real-time analysis of large data sets. Kafka is a distributed streaming platform that is used for building real-time data pipelines and streaming applications. Together, they provide a powerful platform for analyzing real-time data.

The objective of this report is to analyze the real-time price of bitcoin using Spark and Kafka. We will build a real-time data pipeline that collects bitcoin price data, processes it in real-time using Spark, and produces insights that will help traders and investors make informed decisions.

## 2. Installation of Software

Services such as spark, kafka, zookeeper and postgres database are being run in docker containers connected over a single bridge docker network. Code is written in python along with few packages installed using pip.

source code:

<https://github.com/RohanJnr/bitcoin-analysis-spark-kafka>

The following are the required softwares to run the project.

- Docker
- Docker compose
- Python 3.11
- Python packages
  - websocket-client==1.5.1
  - pyspark==3.3.2
  - kafka-python==2.0.2
  - psycopg2-binary==2.9.6
- Softwares Images running in docker containers
  - confluentinc/cp-zookeeper:7.3.2
    - <https://hub.docker.com/r/confluentinc/cp-zookeeper>
  - confluentinc/cp-kafka:7.3.2
    - <https://hub.docker.com/r/confluentinc/cp-kafka>
  - bitnami/spark:3.3.
    - <https://hub.docker.com/r/bitnami/spark>
  - postgres:15-alpine
    - [https://hub.docker.com/\\_/postgres](https://hub.docker.com/_/postgres)
    - Database is created with 2 tables:
      - “stock\_data”: This table stores real time bitcoin price and volume of purchase.
      - “stock\_aggregates”: This table contains average data of price and sum of volume of purchase of bitcoin

between a certain time interval which is set by spark sliding window.

- Command to run all services:  
`docker compose up`

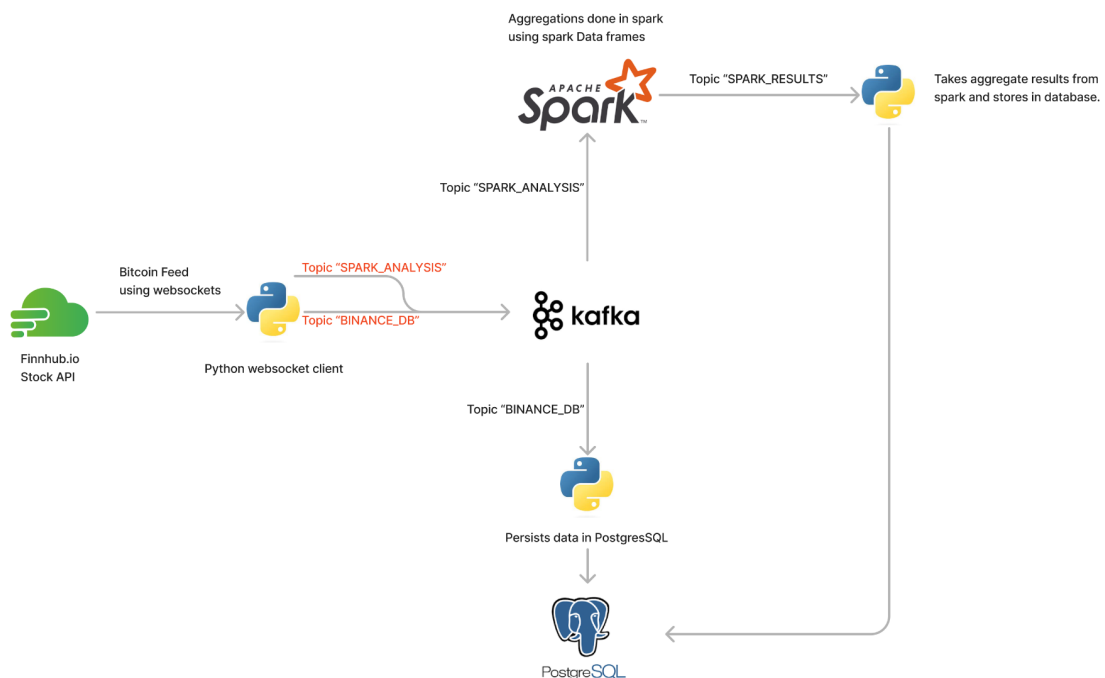
### 3. Problem Description

- Bitcoin price live / real time data is being fetched from <https://finnhub.io>. They provide a websocket URL from which the feed is received (data comes in every 500ms-1000ms).
- The feed is then published to 2 kafka topics
  - Topic "BINANCE\_DB"
    - The consumer of this topic stores the real time data in a database under the table "stock\_data".
  - Topic "SPARK\_ANALYSIS"
    - Spark is the consumer for this topic. Spark analysis this data, does aggregations and then sends the aggregated to the kafka topic "SPARK\_RESULTS"
- The topic "SPARK\_RESULTS" is received by another consumer which stores the results in the database.
- For Batch processing, the spark instances reads the database "stock\_data" table and computes the following:
  - Overall average price
  - Average price per minute (since the data coming in high velocity: every 500ms-1000ms)
  - Total purchases
  - Total purchases per minute

## 4.1 Stream processing Flow Chart

Link to figma diagram:

<https://www.figma.com/file/7vopZh1Q0YByug4QRINsr8/Spark-Streaming?node-id=0%3A1&t=uFjFYhnZ5HgkwZYx-1>



## 4.2 Batch Processing Flow Chart



## 5. Input Data

- Source and description
  - Finnhub.io site was used for live input data.
  - <https://finnhub.io/docs/api/websocket-trades>
  - The input data consists of
    - **Bitcoin price**
    - **timestamp**
    - **volume purchased**
    - symbol
    - conditions

## 6. Streaming mode experiment

- Install all dependencies by creating a python and virtualenv and install packages using requirements.txt

```

(venv) ➔ dbt_project gti@nato pip install -r requirements.txt
Requirement already satisfied: asttokens==2.2.1 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 1)) (2.2.1)
Requirement already satisfied: autospell==2.0.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 2)) (2.0.2)
Requirement already satisfied: backcall==2.0.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 3)) (0.2.0)
Requirement already satisfied: cachetools==5.3.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 4)) (5.3.0)
Requirement already satisfied: certifi==2022.12.7 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 5)) (2022.12.7)
Requirement already satisfied: charset-normalizer==3.1.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 6)) (3.1.0)
Requirement already satisfied: comm==0.1.3 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 7)) (0.1.3)
Requirement already satisfied: debugpy==1.6.7 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 8)) (1.6.7)
Requirement already satisfied: decorator==5.1.1 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 9)) (5.1.1)
Requirement already satisfied: executing==1.2.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 10)) (1.2.0)
Requirement already satisfied: gevent==22.10.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 11)) (22.10.2)
Requirement already satisfied: google-api-core==2.11.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 12)) (2.11.0)
Requirement already satisfied: google-api-python-client==2.86.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 13)) (2.86.0)
Requirement already satisfied: google-auth==2.17.3 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 14)) (2.17.3)
Requirement already satisfied: google-auth-httplib2==0.1.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 15)) (0.1.0)
Requirement already satisfied: googleapis-common-protos==1.59.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 16)) (1.59.0)
Requirement already satisfied: greenlet==2.0.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 17)) (2.0.2)
Requirement already satisfied: grpcio==1.54.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 18)) (1.54.0)
Requirement already satisfied: grpcio-status==1.54.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 19)) (1.54.0)
Requirement already satisfied: httplib2==22.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 20)) (22.0)
Requirement already satisfied: idna==3.4 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 21)) (3.4)
Requirement already satisfied: ipkernel==6.22.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 22)) (6.22.0)
Requirement already satisfied: ipython==8.12.1 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 23)) (8.12.1)
Requirement already satisfied: jedi==0.18.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 24)) (0.18.2)
Requirement already satisfied: jupyter-client==8.2.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 25)) (8.2.0)
Requirement already satisfied: jupyter-core==5.3.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 26)) (5.3.0)
Requirement already satisfied: kafka-python==2.0.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 27)) (2.0.2)
Requirement already satisfied: matplotlib==3.5.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 28)) (3.5.0)
Requirement already satisfied: nest-asyncio==1.5.6 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 29)) (1.5.6)
Requirement already satisfied: numpy==1.24.3 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 30)) (1.24.3)
Requirement already satisfied: packaging==23.1 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 31)) (23.1)
Requirement already satisfied: pandas==2.0.1 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 32)) (2.0.1)
Requirement already satisfied: parso==0.8.3 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 33)) (0.8.3)
Requirement already satisfied: pexpect==4.8.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 34)) (4.8.0)
Requirement already satisfied: pickleshare==0.7.5 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 35)) (0.7.5)
Requirement already satisfied: platformdirs==2.5.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 36)) (2.5.0)
Requirement already satisfied: prompt-toolkit==3.0.38 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 37)) (3.0.38)
Requirement already satisfied: protobuf==4.22.3 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 38)) (4.22.3)
Requirement already satisfied: psutil==5.9.5 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 39)) (5.9.5)
Requirement already satisfied: pycocot-binary==2.9.6 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 40)) (2.9.6)
Requirement already satisfied: ptyprocess==0.7.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 41)) (0.7.0)
Requirement already satisfied: pure-eval==0.2.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 42)) (0.2.2)
Requirement already satisfied: pydantic==1.10.9 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 43)) (1.10.9)
Requirement already satisfied: pyarrow==11.0.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 44)) (11.0.0)
Requirement already satisfied: pysnmp==5.0.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 45)) (5.0.0)
Requirement already satisfied: pysnmp-modules==0.3.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 46)) (0.3.0)
Requirement already satisfied: pycodestyle==2.10.0 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 47)) (2.10.0)
Requirement already satisfied: pygments==2.15.1 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 48)) (2.15.1)
Requirement already satisfied: pyrsing==3.0.9 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 49)) (3.0.9)
Requirement already satisfied: pyspark==3.3.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 50)) (3.3.2)
Requirement already satisfied: python-dateutil==2.8.2 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 51)) (2.8.2)
Requirement already satisfied: pytz==2023.3 in ./venv/lib64/python3.11/site-packages (from -r requirements.txt (line 52)) (2023.3)

```

- Run all docker containers: Includes zookeeper, kafka, spark and postgres via docker compose



The first screenshot shows the output of `docker compose up` for a Bitnami Spark cluster. It details the creation and startup of containers for Spark Master, Spark Worker, Zookeeper, and PostgreSQL. The logs show the Spark Master starting in master mode and the PostgreSQL database directory being skipped.

The second screenshot shows the output of `docker ps`, displaying a table of running containers:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b1e6ed5b7a38	postgres:15-alpine	"docker-entrypoint.s..."	2 minutes ago	Up 2 minutes	0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	dbt-project-postgres-1
65c17489eada	bitnami/spark:latest	"/opt/bitnami/script..."	4 hours ago	Up 2 minutes	0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	dbt-project-spark-1
d48648c919f2	bitnami/spark:latest	"/opt/bitnami/script..."	4 hours ago	Up 2 minutes	0.0.0.0:7077->7077/tcp, :::7077->7077/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp	dbt-project-spark-worker-1
2ae878145c8a	confluentinc/cp-kafka:7.3.2	"/etc/confluent/dock..."	8 hours ago	Up 2 minutes	0.0.0.0:9092->9092/tcp, :::9092->9092/tcp	dbt-project-spark-1
f7f78d2c2716	confluentinc/cp-zookeeper:7.3.2	"/etc/confluent/dock..."	8 hours ago	Up 2 minutes	2181/tcp, 2888/tcp, 3888/tcp	zookeeper

- We need to now run the following files:
  - **kafka/rt\_stock\_producer.py** (receives real time data and publishes to kafka topics)
  - **kafka/persists\_db\_consumer.py** (Persists data from topic “BINANCE\_DB” to database)
  - **kafka/results\_consumer.py** (Kafka consumer to receive results after calculations/aggregations from spark)
  - **spark/spark\_streaming.py** (Listens to bitcoin data from kafka and performs aggregations)
- The above four files can be run by running “streaming.sh” file. The bash script is already configured to run the 4 files and logs the output to the “log/” directory.

```

Activities  Terminal  Sun Apr 30 19:59
./streaming.sh

pgadmin4  docker compose up  ./streaming.sh

(venv) → dbt_project git:(main) ./streaming.sh
ivy Default Cache set to: /home/rohan/.ivy2/cache
The jars for the packages stored in: /home/rohan/.ivy2/jars
org.apache.spark:spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark:spark-submit-parent-c02676b1-cadc-4d6e-9ad8-19064712795b:1.0
  confs: [default]
  found org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0 in central
  found org.apache.spark:spark-token-provider-kafka-0-10_2.12:3.2.0 in central
  found org.apache.kafka:kafka-clients:2.8.0 in central
  found org.lz4:lz4-java:1.7.1 in central
  found org.xerial.snappy:snappy-java:1.1.8.4 in central
  found org.slf4j:slf4j-api:1.7.30 in central
  found org.apache.hadoop:hadoop-client-runtime:3.3.1 in central
  found org.spark-project.spark:unused:1.0.0 in central
  found org.apache.hadoop:hadoop-client-api:3.3.1 in central
  found org.apache.htrace:htrace-core4:4.1.0-incubating in central
  found commons-logging:commons-logging:1.1.3 in central
  found com.google.code.findbugs:jsr305:3.0.0 in central
  found org.apache.commons:commons-pool2:2.6.2 in central
  :: resolution report :: resolve 360ms :: artifacts dl 10ms
  :: modules in use:
    com.google.code.findbugs:jsr305:3.0.0 from central in [default]
    commons-logging:commons-logging:1.1.3 from central in [default]
    org.apache.commons:commons-pool2:2.6.2 from central in [default]
    org.apache.hadoop:hadoop-client-api:3.3.1 from central in [default]
    org.apache.hadoop:hadoop-client-runtime:3.3.1 from central in [default]
    org.apache.htrace:htrace-core4:4.1.0-incubating from central in [default]
    org.apache.kafka:kafka-clients:2.8.0 from central in [default]
    org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0 from central in [default]
    org.apache.spark:spark-token-provider-kafka-0-10_2.12:3.2.0 from central in [default]
    org.lz4:lz4-java:1.7.1 from central in [default]
    org.slf4j:slf4j-api:1.7.30 from central in [default]
    org.spark-project.spark:unused:1.0.0 from central in [default]
    org.xerial.snappy:snappy-java:1.1.8.4 from central in [default]

  | conf | number | search | downloaded | evicted | number | downloaded |
  |-----|-----|-----|-----|-----|-----|-----|
  | default | 13 | 0 | 0 | 0 | 13 | 0 |

  :: retrieving :: org.apache.spark:spark-submit-parent-c02676b1-cadc-4d6e-9ad8-19064712795b
    0 artifacts copied, 13 already retrieved (0kB/6ms)
  confs: [default]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

```

- 
- The logs of interest are in “logs/spark\_streaming.log”.

```

Activities  Visual Studio Code  Sun Apr 30 20:02
spark_streaming.log - dbt_project - Visual Studio Code

logs > spark_streaming.log
1  :: Loading settings :: url = jar:file:/home/rohan/dev/dbt_project/venv/lib/python3.11/site-packages/pyspark/jars/ivy-2.5.1-jar/org/apache/ivy/core/settings/ivysettings.xml
2  23/04/30 19:59:35 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
3  23/04/30 19:59:38 WARN Utils: Service 'SparkUI' could not bind to port 4040. Attempting port 4041.
4  23/04/30 19:59:38 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
5  23/04/30 19:59:38 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
6  -----
7  Batch: 0
8  -----
9  +-----+-----+
10 | window | avg(data.p) | sum(data.v) |
11 +-----+-----+
12
13
14
15 Batch: 1
16 -----
17 +-----+-----+
18 | window | avg(data.p) | sum(data.v) |
19 +-----+-----+
20 | [2023-04-30 19:59:30, 2023-04-30 19:59:40] | 29272.0048828125 | 0.0032899999641813338 |
21 +-----+-----+
22
23
24 Batch: 2
25 -----
26 +-----+-----+
27 | window | avg(data.p) | sum(data.v) |
28 +-----+-----+
29 | [2023-04-30 19:59:40, 2023-04-30 19:59:50] | 29272.003255208332 | 0.15730000071926042 |
30 | [2023-04-30 19:59:30, 2023-04-30 19:59:40] | 29272.003255208332 | 0.006270000070799142 |
31 +-----+-----+
32
33
34 Batch: 3
35 -----
36 +-----+-----+
37 | window | avg(data.p) | sum(data.v) |
38 +-----+-----+
39 | [2023-04-30 19:59:40, 2023-04-30 19:59:50] | 29272.001085869445 | 1.672020002210047 |
40 | [2023-04-30 19:59:50, 2023-04-30 20:00:00] | 29271.9658203125 | 0.07324000069638714 |
41 +-----+-----+
42
43
44 Batch: 4
45 -----
46 +-----+-----+
47 | window | avg(data.p) | sum(data.v) |
48 +-----+-----+
49 | [2023-04-30 19:59:50, 2023-04-30 20:00:00] | 29271.96651785714 | 0.11096000066027045 |

```

- 
- From the screenshot, Spark is receiving data from kafka and processing them in batches with a sliding window set to a duration of 10 seconds.

- The results will be available in the database as well. We can see this by running the query: **select \* from stock\_data order by price\_datetime;**

The screenshot shows the PGAdmin interface with the query 'select \* from stock\_data order by price\_datetime;' executed. The results are displayed in a table with the following columns: id, price, symbol, price\_datetime, and volume. The data is sorted by price\_datetime in ascending order.

id	price	symbol	price_datetime	volume
1	29257.75	BINANCE BTCUSD	2023-04-30 14:46:05.943	0.0026
2	29257.75	BINANCE BTCUSD	2023-04-30 14:46:06.501	0.00179
3	29257.76	BINANCE BTCUSD	2023-04-30 14:46:06.782	0.00035
4	29257.76	BINANCE BTCUSD	2023-04-30 14:46:07.047	0.0011
5	29257.76	BINANCE BTCUSD	2023-04-30 14:46:08.662	0.00196
6	29257.76	BINANCE BTCUSD	2023-04-30 14:46:08.891	0.00354
7	29257.76	BINANCE BTCUSD	2023-04-30 14:46:08.988	0.00224
8	29257.75	BINANCE BTCUSD	2023-04-30 14:46:09.071	0.14291
9	29257.75	BINANCE BTCUSD	2023-04-30 14:46:09.071	0.04569
10	29257.76	BINANCE BTCUSD	2023-04-30 14:46:09.089	0.01554
11	29257.75	BINANCE BTCUSD	2023-04-30 14:46:09.375	0.00505
12	29257.75	BINANCE BTCUSD	2023-04-30 14:46:09.84	0.00107
13	29257.76	BINANCE BTCUSD	2023-04-30 14:46:10.582	0.00089
14	29257.76	BINANCE BTCUSD	2023-04-30 14:46:11.786	0.00242
15	29257.76	BINANCE BTCUSD	2023-04-30 14:46:11.295	0.00169
16	29257.76	BINANCE BTCUSD	2023-04-30 14:46:11.321	0.02463
17	29257.75	BINANCE BTCUSD	2023-04-30 14:46:11.722	0.00131
18	29257.75	BINANCE BTCUSD	2023-04-30 14:46:11.751	0.00499
19	29257.75	BINANCE BTCUSD	2023-04-30 14:46:12.762	0.00045
20	29257.75	BINANCE BTCUSD	2023-04-30 14:46:14.469	0.03278
21	29257.76	BINANCE BTCUSD	2023-04-30 14:46:14.468	0.00069
22	29257.76	BINANCE BTCUSD	2023-04-30 14:46:14.493	0.00069
23	29257.76	BINANCE BTCUSD	2023-04-30 14:46:14.494	0.00069
24	29257.76	BINANCE BTCUSD	2023-04-30 14:46:14.498	0.00069

- Query to see results after sliding window aggregations: **select \* from stock\_aggregates;**

The screenshot shows the PGAdmin interface with the query 'select \* from stock\_aggregates;' executed. The results are displayed in a table with the following columns: id, start\_datetime, end\_datetime, average\_price, and total\_volume. The data is sorted by start\_datetime in ascending order.

id	start_datetime	end_datetime	average_price	total_volume
1	2023-04-30 18:13:30	2023-04-30 18:13:40	29218.3962890625	0.23467000556411222
2	2023-04-30 18:13:20	2023-04-30 18:13:30	29218.19870283019	0.554520004283404
3	2023-04-30 18:13:10	2023-04-30 18:13:20	29214.031065244933	7.301870099260774
4	2023-04-30 18:13:20	2023-04-30 18:13:30	29218.200774016204	0.560600040925201
5	2023-04-30 18:13:30	2023-04-30 18:13:40	29218.43799765625	0.768730009556748
6	2023-04-30 18:13:40	2023-04-30 18:13:50	29218.612771739132	0.5564400142757222
7	2023-04-30 18:13:40	2023-04-30 18:13:50	29218.612630208332	0.7162700145272538
8	2023-04-30 14:48:40	2023-04-30 14:48:50	29238.773552389706	0.985909999594036
9	2023-04-30 14:48:50	2023-04-30 14:49:00	29238.776448567707	0.13638999976683408
10	2023-04-30 14:48:50	2023-04-30 14:49:00	29238.77679869186	0.31641999987186864
11	2023-04-30 14:49:00	2023-04-30 14:49:10	29236.0142852531	8.522540056263097
12	2023-04-30 14:49:00	2023-04-30 14:49:10	29235.19437839674	12.969140039553167
13	2023-04-30 14:49:10	2023-04-30 14:49:20	29227.511637369793	2.653469992259488
14	2023-04-30 14:49:10	2023-04-30 14:49:20	29226.635270003433	7.888760021432972
15	2023-04-30 14:49:20	2023-04-30 14:49:30	29223.73340992647	0.9616200021991972
16	2023-04-30 14:49:20	2023-04-30 14:49:30	29224.378300107757	3.0222900276712608
17	2023-04-30 14:49:30	2023-04-30 14:49:40	29227.814208984375	0.33127000188687816
18	2023-04-30 14:49:30	2023-04-30 14:49:40	29227.814587823275	0.3475700017297404
19	2023-04-30 14:49:40	2023-04-30 14:49:50	29227.81351902174	0.22136999771464616
20	2023-04-30 14:49:50	2023-04-30 14:50:00	29228.33033511513	0.9645999968888646
21	2023-04-30 14:49:50	2023-04-30 14:50:00	29227.814689867424	0.5141499887686223
22	2023-04-30 14:49:50	2023-04-30 14:50:00	29229.609316986385	1.6871899955085646
23	2023-04-30 14:50:00	2023-04-30 14:50:10	29234.356693097016	0.9605500013858546

- Database is being populated as spark is outputting data after aggregations.

## Relevant Code

- Code for sql tables

```
• CREATE TABLE IF NOT EXISTS stock_data (  
•     id SERIAL PRIMARY KEY,  
•     price DECIMAL,  
•     symbol VARCHAR(128),  
•     price_datetime TIMESTAMP WITHOUT TIME ZONE,  
•     volume DECIMAL  
• );  
•  
•  
• CREATE TABLE IF NOT EXISTS stock_aggregates (  
•     id SERIAL PRIMARY KEY,  
•     start_datetime TIMESTAMP WITHOUT TIME ZONE,  
•     end_datetime TIMESTAMP WITHOUT TIME ZONE,  
•     average_price DECIMAL,  
•     total_volume DECIMAL  
• )
```

- Code for spark aggregations

```
• # Read data from Kafka as a DataFrame  
• df =  
•     spark.readStream.format("kafka").options(**kafka_params).load()  
•  
•  
• kafka_df = df.selectExpr("CAST(value AS STRING)", "CAST(timestamp  
•     AS TIMESTAMP)").withWatermark(  
•     "timestamp", "20 seconds")  
•  
•  
•
```

```
● parsed_df = kafka_df.select(from_json("value",
● schema).alias("data"))
●
● windowed_stream = parsed_df \
● .groupBy(window(col("data.datetime"), "10 seconds", "10
● seconds")) \
● .agg(
●     avg(col("data.p")),
●     _sum(col("data.v"))
● )
●
● # Write to console
● console_write = ( windowed_stream.writeStream \
●     .outputMode("update") \
●     .format("console") \
●     .option("truncate", False) \
●     .trigger(processingTime="10 seconds") \
●     .start()
● )
●
● # Write to kafka topic
● kafka_write = windowed_stream \
●     .selectExpr("to_json(struct(*)) AS value") \
●     .writeStream \
●     .outputMode("update") \
●     .trigger(processingTime="10 seconds") \
●     .format("kafka") \
●     .option("kafka.bootstrap.servers", "localhost:9092") \
●     .option("topic", "SPARK_RESULTS") \
●     .start()
```

```

•
• kafka_write.awaitTermination()
• console_write.awaitTermination()
•

```

## 7. Batch mode experiment

- Install all dependencies by creating a python and virtualenv and install packages using requirements.txt as shown previously
- Start all docker containers as shown previously
- We will now run “**spark/spark\_batch.py**” which will put data from the database and perform analysis on the data in spark.

```

(venv) → dbt_project git:(main) python spark/spark_batch.py
23/04/30 20:20:01 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/04/30 20:20:01 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Time intervals for the following calculations.
Start Timestamp
+-----+
| min(price_datetime)|
+-----+
|2023-04-30 14:46:...|
+-----+

End Timestamp
+-----+
| max(price_datetime)|
+-----+
|2023-04-30 20:01:...|
+-----+

Overall average price
+-----+
|      avg(price)|
+-----+
|29239.60886630336...|
+-----+

Calculating average price per minute
+-----+
|      date|      avg(price)|
+-----+
|2023-04-30 18:39|29263.97522598870...|
|2023-04-30 18:35|29242.12655303030...|
|2023-04-30 18:00|29197.10410326086...|
|2023-04-30 18:09|29205.90596551724...|
|2023-04-30 17:12|29186.83859296482...|
|2023-04-30 14:46|29257.75787500000...|
|2023-04-30 17:31|29197.68438709677...|
|2023-04-30 18:36|29251.87102564102...|
|2023-04-30 19:09|29272.21846153846...|
|2023-04-30 17:20|29194.13819767441...|
|2023-04-30 19:29|29291.88873239436...|
|2023-04-30 18:37|29265.36465181058...|
|2023-04-30 17:46|29208.35416083916...|
|2023-04-30 18:07|29190.48142857142...|
|2023-04-30 18:11|29205.31230158730...|
|2023-04-30 19:19|29291.17620183486...|
|2023-04-30 18:46|29266.35148401826...|
|2023-04-30 17:18|29186.52013752455...|
|2023-04-30 18:42|29263.08375000000...|
|2023-04-30 17:27|29199.53582568807...|
+-----+

```

only showing top 20 rows

```

Total Purchases
+-----+
|          sum(volume) |
+-----+
|1782.088850000000...|
+-----+

Total purchase volume per minute
+-----+
|          date|          sum(volume) |
+-----+
|2023-04-30 20:01|12.289600000000000...|
|2023-04-30 20:00|16.469560000000000...|
|2023-04-30 19:59|3.664210000000000000|
|2023-04-30 19:33|1.188640000000000000|
|2023-04-30 19:32|3.370210000000000000|
|2023-04-30 19:31|14.176230000000000...|
|2023-04-30 19:30|16.137700000000000...|
|2023-04-30 19:29|14.664380000000000...|
|2023-04-30 19:28|11.748330000000000...|
|2023-04-30 19:27|5.830970000000000000|
|2023-04-30 19:26|2.410910000000000000|
|2023-04-30 19:25|5.089010000000000000|
|2023-04-30 19:24|3.810780000000000000|
|2023-04-30 19:23|11.872010000000000...|
|2023-04-30 19:22|7.235710000000000000|
|2023-04-30 19:21|5.147940000000000000|
|2023-04-30 19:20|2.359600000000000000|
|2023-04-30 19:19|25.385230000000000...|
|2023-04-30 19:18|11.463920000000000...|
|2023-04-30 19:17|3.244940000000000000|
+-----+
only showing top 20 rows

(venv) → dbt_project git:(main) █

```

- The above gives insights on average prices, number of purchases and prices and purchases of bitcoin per minute.

## Relevant Code

```
# Start timestamp
print("Start Timestamp")
start_timestamp = df.agg({"price_datetime": "min"})
start_timestamp.show()

# End timestamp
print("End Timestamp")
end_timestamp = df.agg({"price_datetime": "max"})
end_timestamp.show()

# Overall average price
print("Overall average price")
average = df.agg({"price": "avg"})
average.show()

# Calculating average price per minute
print("Calculating average price per minute")
df = df.withColumn("date", date_format(col("price_datetime"), "yyyy-MM-dd
HH:mm"))
average_per_min = df.groupBy("date").mean("price")
average_per_min.show()

# Total purchase volume
print("Total Purchases")
volume_sum = df.agg({"volume": "sum"})
volume_sum.show()

# Total purchase volume per minute
print("Total purchase volume per minute")
sum_volume_per_min = df.groupBy("date").sum("volume").orderBy(desc("date"))
sum_volume_per_min.show()
```



## 8. Comparison of streaming and batch mode

In Spark, batch mode and streaming mode are two different processing models that are used to process data.

Batch processing refers to processing a fixed amount of data at a time. In Spark, batch processing involves reading a set of data, processing it, and then outputting the results. Batch processing is typically used for non-real-time applications where processing time is not critical. In this application, batch mode was done on already collected data from the bitcoin api.

Streaming processing, on the other hand, refers to processing data in real-time as it is generated. In Spark, streaming processing involves reading data in small batches, processing it, and then outputting the results. Streaming processing is typically used for real-time applications where processing time is critical. In this application, streaming processing was done on real time, high velocity data of bitcoin information.

## 9. Conclusion

The real time analysis of bitcoin prices and purchase volume done using Python, Spark, Kafka and postgres showed interesting insights on the trend of bitcoin prices and purchase volume. Value of bitcoin is extremely unpredictable and hence this analysis helps us understand the market and how the price fluctuates. The purchase volume is quite random and the price fluctuates heavily due to this high purchase by 100s of dollars.

Streaming analysis was helpful in understanding the sub-minute fluctuation in bitcoin and this can be very useful for day traders to instantly monitor and sell/buy bitcoin.

Batch mode processing has been demonstrated on a per minute basis which can be easily extended to hourly or daily batch processing whose insights can be useful for long term buyers of bitcoin.

## 10. References

- <https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html>
- <https://developer.confluent.io/quickstart/kafka-docker/>
- <https://www.psycopg.org/docs/>
- <https://spark.apache.org/docs/latest/api/python/>
- <https://kafka-python.readthedocs.io/en/master/>