

RAPPORT DU PROJET
MACHINE LEARNING ET BIG DATA

Par

CHEBAANE Zeineb et OMRANE Karim

Prédiction de prix de Bitcoin

Encadrant :

Monsieur Maher Heni

Année universitaire 2022-2023

Table des matières

1 Cadre général du projet	1
1.1 Présentation de Bitcoin	2
1.2 Problématique	2
1.3 Solution	3
1.4 Les technologies utilisées	3
1.5 Architecture	3
1.6 Décomposition du projet	4
2 Réalisation et test	5
2.1 Obtenir une clé d'API	6
2.2 Producer	7
2.3 Consumer	12
2.4 Indexage des données à l'aide de Elasticsearch	16
2.5 Visualisation des données avec Kibana	21
2.6 Prédiction de prix de bitcoin avec le model de Machine Learning (LSTM)	24

Table des figures

1.1	Architecture du projet	4
2.1	API pour la récupération des données	6
2.2	Dataset	7
2.3	Script de producer1	8
2.4	Script de producer2	9
2.5	Capture pour le lancement de Zookeeper	9
2.6	Capture pour le lancement du 1er serveur Kafka	10
2.7	Capture pour le lancement du 2éme serveur Kafka	10
2.8	Capture pour la création de topic "bitcoin"	11
2.9	Vérifier la création du topic "bitcoin"	11
2.10	Augmenter le nombre de partitions de topic "bitcoin"	12
2.11	Résultat de producer1	12
2.12	Script de consumer1	12
2.13	Résultat de consumer	13
2.14	Importer les bibliothéques	13
2.15	Créer un objet SparkConf	14
2.16	Créer Streaming DataFrame	14
2.17	Lire les données	15
2.18	Lancer le producer	15
2.19	Exécuter consumer2.py	15
2.20	Résultat affiché	16
2.21	Script pour l'indexage des données	17
2.22	Script pour l'indexage des données(mapping)	18
2.23	Script pour l'indexage des données	19
2.24	Résultat de l'indexagee des données	20
2.25	Visualisation de prix de bitcoin	21
2.26	les prix de bitcoin	21
2.27	Prédiction de prix de bitcoin	22
2.28	Modèle d'évaluation de la regression	22
2.29	Prévision de prix de Bitcoin	23
2.30	Analyse de prix de Bitcoin	23
2.31	Analyse de prix de Bitcoin	24

2.32 Dataset YahooFinance API	24
2.33 L'analyse de données de l'année 2015	25
2.34 L'analyse de données de l'année 2015	25
2.35 L'analyse globale de prix de Bitcoin	26
2.36 Prédiction de prix de Bitcoin	26
2.37 Prédiction de prix de Bitcoin	27
2.38 Prédiction de prix de Bitcoin	27

Introduction

Dans ce chapitre on va présenter au premier lieu Bitcoin et on va parler de la problématique reconnue et la solution proposée pour mettre fin aux problèmes rencontrés. Dans un second lieu on va parler des technologies utilisées dans ce projet et l'architecutre proposée.

1.1 Présentation de Bitcoin

Bitcoin est une "monnaie" virtuelle créée en 2009 par un développeur inconnu sous le pseudonyme de Satoshi Nakamoto. C'est un véritable système de paiement dont l'organisation ne repose sur aucune entité centrale, comme peut l'être par exemple une banque centrale dans le cadre des monnaies traditionnelles. La gestion des transactions et la création de bitcoins est prise en charge collectivement par le réseau informatique mondial.

Une crypto-monnaie comme Bitcoin élimine l'obligation pour les tiers de s'emmêler dans les transferts d'argent en agissant comme de l'argent numérique et une forme de paiement qui n'est basée sur aucun individu, groupe ou autre institution. Il est livré aux mineurs de blockchain en paiement de leur aide à la confirmation des transactions, et il peut être acheté sur de nombreux sites.

Un développeur ou un groupe de développeurs non identifié a présenté Bitcoin au public et depuis lors, il est devenu la crypto-monnaie la plus connue au monde. De nombreuses crypto-monnaies supplémentaires ont été développées en raison de leur popularité. Ces rivaux veulent le remplacer comme moyen de paiement ou sont employés dans d'autres chaînes de blocs et technologies financières de pointe comme jetons utilitaires ou de sécurité. Investir dans les crypto-monnaies est une possibilité nouvelle et passionnante, mais il peut être difficile de les comprendre.

Le Bitcoin est l'une des crypto-monnaies les plus connues et la prévision du prix du Bitcoin est un moyen de voir comment il en sera à l'avenir.

1.2 Problématique

Depuis des décennies, notre système financier a toujours été épaulé par un tiers de confiance, la banque, pour assurer la sécurité de nos transactions. Ce tiers de confiance a été remis en cause lors la crise de 2008. Nous avons été témoins d'un risque d'effondrement total du la machine financière. D'où la nécessité d'un système de paiement alternatif fiable, dénationalisé, libre de toute autorité et à faible coût. En 2008, le bitcoin a été créé pour répondre à ce besoin. En effet, les cryptomonnaies ne dépendent d'aucun intermédiaire. Les transactions de cryptomonnaies sont vérifiées à l'aide de la technologie « blockchain » qui sécurise les échanges. Depuis le lancement du bitcoin en 2009, aucun hacker n'a pu infiltrer sa blockchain démontrant un haut niveau de sécurité, mais aussi l'énorme potentiel technologique de son réseau. Toutefois, les deux monnaies (cryptomonnaie et fiat) acquièrent

principalement leur valeur de l'acceptation et la confiance des parties prenantes. De toute évidence, le sentiment des investisseurs à propos du bitcoin a une valeur informationnelle importante pour expliquer les variations de son cours, ceci explique d'une autre part sa grande volatilité.

On peut le constater tous les jours, le cours du Bitcoin et des autres crypto actifs a toujours tendance à fluctuer de façon importante. Il n'est pas rare de voir des augmentations ou des chutes, voire davantage, pendant une journée, ce qui est rarissime pour d'autres classes d'actifs comme les actions ou encore les métaux précieux.

Comme pour ces autres actifs, la valeur du Bitcoin est déterminée par l'offre et la demande sur des places de marché. À n'importe quel moment, des acteurs économiques lancent des offres d'achat et de vente avec un prix qu'ils déterminent en fonction de leur propre analyse. Lorsqu'un acheteur et un vendeur sont d'accord sur un prix, une transaction s'effectue et le cours est fixé au prix auquel a eu lieu cette dernière transaction.

Ceci est la réponse simple. Il serait bien plus intéressant de se demander : Comment est fixé le cours du Bitcoin ? Quels sont les facteurs qui influencent l'analyse des différents acteurs et qui les poussent à lancer un ordre d'achat ou de vente à tel ou tel prix ?

1.3 Solution

Notre solution permet de prédire le prix de bitcoin à l'aide des algorithmes de Machine Learning appliqués sur des données tirées à l'aide d'une API .

1.4 Les technologies utilisées

- **Kafka** : C'est une plateforme open source d'agents de messages (brokers) en temps réel. Cette plateforme permet à la fois de diffuser des données à grande échelle (event streaming) et d'effectuer des traitements sur ces données en temps réel (stream processing).
- **Spark** : C'est est un moteur d'analyse unifié et ultra-rapide pour le traitement de données à grande échelle. Il permet d'effectuer des analyses de grande ampleur par le biais de machines de Clusters. Il est essentiellement dédié au Big Data et Machine Learning.).
- **Elasticsearch** : C'est un logiciel utilisant Lucene pour l'indexation et la recherche de données.
- **Kibana** : C'est une interface utilisateur gratuite et ouverte qui nous permet de visualiser nos données

1.5 Architecture

La figure ci-dessous présente l'architecture du notre projet.

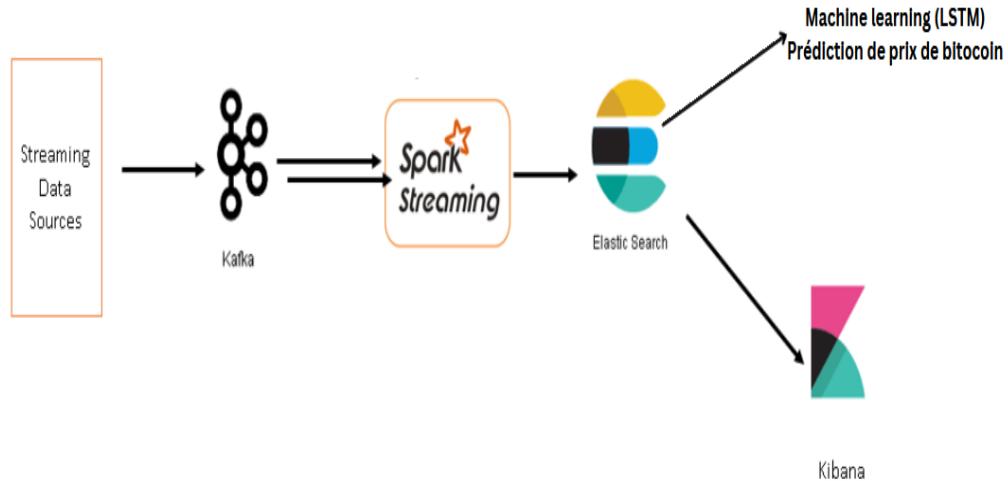


FIGURE 1.1 : Architecture du projet

1.6 Décomposition du projet

Notre projet est décomposé en 6 parties :

- **Choisir une API pour récupérer les données**
- **Producer** : Dans l'écosystème Apache Kafka, un Producer désigne un système qui va publier des messages vers un topic du cluster. On a récupéré les données et les publier dans un topic "bitcoins".
- **Consumer** : qui permet de lire les données d'un cluster kafka.
- **Indexage des données à l'aide de Elasticsearch**
- **Visualisation des données à l'aide de Kibana**
- **Prédiction de prix de Bitcoin à l'aide de l'algorithme de Machine Learning (LSTM)**

Introduction

Dans ce chapitre on va présenter les étapes réalisées pour atteindre notre objectif du projet qui est la prédiction de prix de Bitcoin.

2.1 Obtenir une clé d'API

Dans cette section du chapitre, nous nous intéressons à présenter la première partie du projet qui consiste à récupérer une clé d'API. On a testé qu'elle fonctionne correctement en obtenant en réponse un gros morceau de JSON affiché ci-dessous qui présente un ensemble de données.

```

{
  "status": {
    "elapsed": 3,
    "timestamp": "2023-01-09T09:18:17.607981002Z"
  },
  "data": {
    "Asset": {
      "id": "1e31218a-e44e-4285-820c-8282ee222035",
      "serial_id": 6057,
      "symbol": "BTC",
      "name": "Bitcoin",
      "slug": "bitcoin",
      "contract_addresses": null,
      "_internal_temp_agora_id": "9793eae6-f374-46b4-8764-c2d224429791"
    },
    "market_data": {
      "price_usd": 17202.845721661743,
      "price_btc": 1,
      "price_eth": 13.106438626324033,
      "volume_last_24_hours": 4775661151.531321,
      "real_volume_last_24_hours": 3265550542.4111414,
      "volume_last_24_hours_overstatement_multiple": 1.4624367589806706,
      "percent_change_usd_last_1_hour": null,
      "percent_change_btc_last_1_hour": null,
      "percent_change_eth_last_1_hour": null,
      "percent_change_usd_last_24_hours": 1.5010821440255844,
      "percent_change_btc_last_24_hours": -0.025271951060185482,
      "percent_change_eth_last_24_hours": -2.3552846893994275,
      "ohlcv_last_1_hour": null,
      "ohlcv_last_24_hour": {
        "open": 16948.15437781874,
        "high": 16949.83413834782,
        "low": 16947.22875629321,
        "close": 16948.43577849905,
        "volume": 36756559.58143179,
        "last_trade_at": "2023-01-09T09:18:16Z"
      }
    }
  }
}

```

FIGURE 2.1 : API pour la récupération des données

Dans cet ensemble de données, nous avons trouvé "timestamp" qui nous a donné une information sur la date, "price-usd", "price-btc", "price-eth", qui nous informent sur les prix de bitcoin précédents. On a travaillé alors particulièrement avec ces quatres champs.

L'objectif de notre projet est la prédiction de prix de bitcoin, donc on a besoin d'une source de données riche qui donne des informations sur le prix de bitcoin pour des diverses années. Puisque notre source de données est insuffisante pour avoir une prédiction parfaite, on a cherché aussi une autre source de

données qui nous a aidé à avoir un prédition meilleure. La nouvelle source de données contient six champs : "Data", "Open" qui représente le premier prix dans un jour particulier, "High" qui représente le prix le plus grand dans un jour particulier, "Low" qui représente le prix le plus bas dans un jour particulier, "close" qui représente le dernier prix dans un jour particulier , "Adj close" et "Volume". On a choisi la période à partir de 14 septembre 2014 jusqu'à aujourd'hui.

Time Period: Sep 17, 2014 - Jan 10, 2023		Show: Historical Prices		Frequency: Daily		Apply
Currency in USD						
Date	Open	High	Low	Close*	Adj Close**	Volume
Jan 10, 2023	17,203.87	17,439.56	17,165.80	17,422.63	17,422.63	16,125,792,256
Jan 09, 2023	17,093.99	17,389.96	17,093.99	17,196.55	17,196.55	18,624,736,866
Jan 08, 2023	16,954.15	17,091.14	16,924.05	17,091.14	17,091.14	9,768,827,914
Jan 07, 2023	16,952.12	16,975.02	16,914.19	16,955.08	16,955.08	7,714,767,174
Jan 06, 2023	16,836.47	16,991.99	16,716.42	16,951.97	16,951.97	14,413,662,913
Jan 05, 2023	16,863.47	16,884.02	16,790.28	16,836.74	16,836.74	13,692,758,566
Jan 04, 2023	16,680.21	16,964.59	16,667.76	16,863.24	16,863.24	18,421,743,322
Jan 03, 2023	16,688.85	16,760.45	16,622.37	16,679.86	16,679.86	13,903,079,207
Jan 02, 2023	16,625.51	16,759.34	16,572.23	16,688.47	16,688.47	12,097,775,227
Jan 01, 2023	16,547.91	16,630.44	16,521.23	16,625.08	16,625.08	9,244,361,700
Dec 31, 2022	16,603.67	16,628.99	16,517.52	16,547.50	16,547.50	11,239,186,456

FIGURE 2.2 : Dataset

2.2 Producer

On a stocké ensuite les données relatives à bitcoin dans des messages Kafka : chacun des éléments de la liste renvoyés par l'appel à l'API ci-dessus va être stocké dans Kafka sous la forme d'une chaîne de caractères au format JSON. Pour cela, nous avons crée le script "producer.py" présenté ci-dessous.

```

producer.py M X consumer.py commands.txt
producer.py > ...
You, yesterday | 1 author (You)
1 from kafka import KafkaProducer
2 import urllib.request
3 import json
4 import time
5
6 producer = KafkaProducer(bootstrap_servers=['localhost:9092', 'localhost:9093'], api_version=(0, 10))
7
8 url = "https://data.messari.io/api/v1/assets/btc/metrics/market-data"
9
10 if __name__ == "__main__":
11     print("Sending Data to Kafka")
12     while True:
13         print("***** New")
14         bitcoin_market_data = urllib.request.urlopen(url).read()
15         print(json.loads(bitcoin_market_data))
16         producer.send('Bitcoin', bitcoin_market_data)
17         time.sleep(3)
18

```

FIGURE 2.3 : Script de producer1

Dans ce script, on a crée un producer qui va réaliser un appel à l'API toutes les 3 secondes (time.sleep(3)). Chacune des données de bitcoin contenues dans la réponse de l'API sera redirigée vers le topic "bitcoin" de Kafka (producer.send("bitcoin", bitcoin-market-data)).

Pour exécuter ce script on a besoin du package kafka-python qu'on installé en exécutant la commande "pip install kafka-python".

La figure ci-dessous présente le script de producer de la deuxième dataset.

On a également besoin d'un cluster Kafka minimal, ainsi que d'un topic "bitcoin". Nous avons lancé un cluster et on a crée un toc en suivant les étapes suivantes :

— Lancer Zookeeper

Un cluster Kafka consiste typiquement en plusieurs courtiers (Brokers) pour maintenir la répartition de charge. Ces courtiers sont stateless, c'est pour cela qu'ils utilisent Zookeeper pour gérer et coordonner les courtiers Kafka et maintenir l'état du cluster.

```
2 import urllib.request
3 import json
4 import time
5 import pandas
6 from pandas_datareader import data as pdr
7 import yfinance as yf
8
9 producer = KafkaProducer(bootstrap_servers=['localhost:9092', 'localhost:9093'], api_version=(0, 10))
10
11 # url = "https://data.messari.io/api/v1/assets/btc/metrics/market-data"
12
13 yf.pdr_override()
14
15 bitcoin_prices = pdr.get_data_yahoo('BTC-USD', '2014-09-17', '2023-01-10')
16 bitcoin_prices.reset_index(inplace = True)
17
18 if __name__ == "__main__":
19     print("Sending Data to Kafka")
20     while True:
21         # print("***** New Data *****")
22         # bitcoin_market_data = urllib.request.urlopen(url).read()
23         # print(json.loads(bitcoin_market_data))
24         # producer.send('bitcoin', bitcoin_market_data)
25         # time.sleep(3)
26         for index, row in bitcoin_prices.iterrows():
27             print("***** New Data *****")
28             data = {
29                 "Date": row['Date'],
30                 "Open": row['Open'],
31                 "High": row['High'],
32                 "Low": row['Low'],
33                 "Close": row['Close'],
34                 "Adj Close": row['Adj Close'],
35                 "Volume": row['Volume']
36             }
37             producer.send('bitcoin', json.dumps({
38                 "Date": str(row['Date']),
39                 "Open": float(row['Open']),
40                 "High": float(row['High']),
41                 "Low": float(row['Low']),
42                 "Close": float(row['Close']),
43                 "Adj Close": float(row['Adj Close']),
44                 "Volume": float(row['Volume'])
45             }).encode('utf-8'))
46             # time.sleep(1)
```

FIGURE 2.4 : Script de producer2

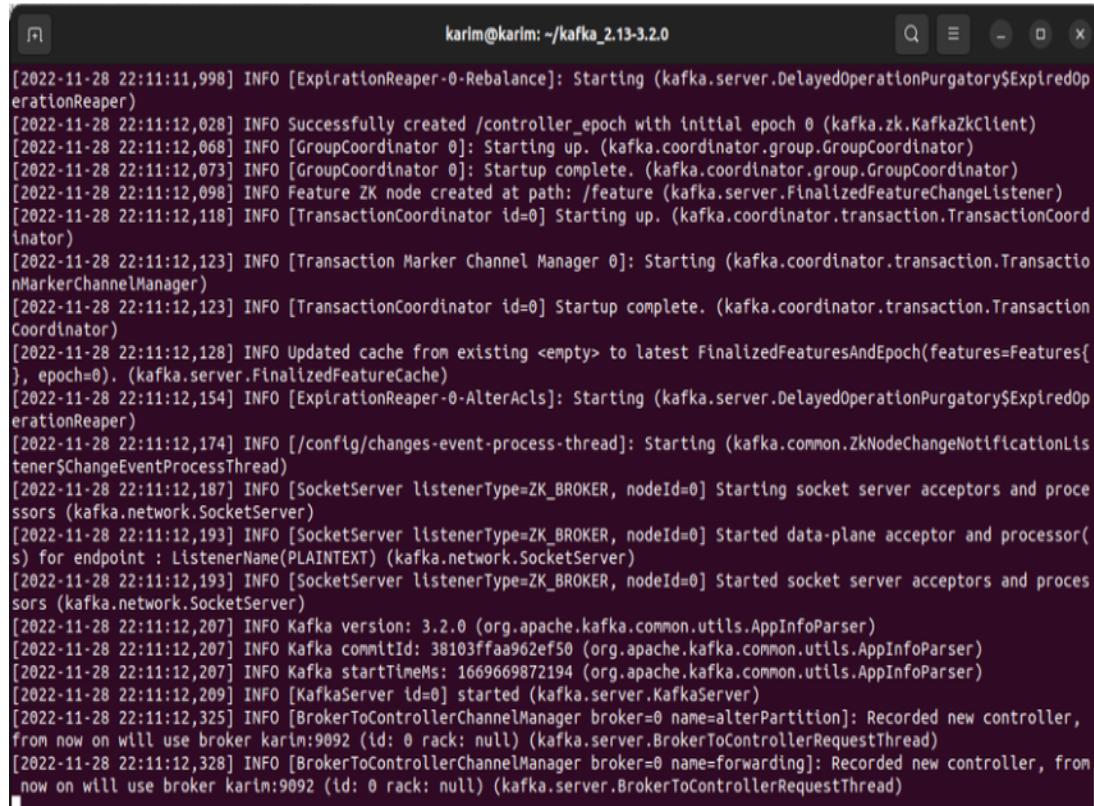
```
[karim@karim:~/kafka_2.13-5.2.0$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
[2022-11-28 22:09:24,437] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerC
[2022-11-28 22:09:24,440] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper
[2022-11-28 22:09:24,450] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,452] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,452] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,452] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zooke
[epServer.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,455] INFO aut purge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-11-28 22:09:24,461] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-11-28 22:09:24,601] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-11-28 22:09:24,602] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.serve
r.quorum.QuorumPeerMain)
[2022-11-28 22:09:24,675] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2022-11-28 22:09:24,700] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerC
onfig)
[2022-11-28 22:09:24,701] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper
[server.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,702] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,702] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,702] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,703] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zooke
[epServer.quorum.QuorumPeerConfig)
[2022-11-28 22:09:24,704] INFO ZooKeeper is starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2022-11-28 22:09:24,794] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@358c99
f5 (org.apache.zookeeper.server.ServerMetrics)
[2022-11-28 22:09:24,807] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2022-11-28 22:09:24,854] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-28 22:09:24,854] INFO _____ - (org.apache.zookeeper.server.ZooK
eeperserver)
[2022-11-28 22:09:24,854] INFO |__ / | | (org.apache.zookeeper.server.ZooK
eeperserver)
[2022-11-28 22:09:24,855] INFO / / ____ _ | | | (org.apache.zookeeper.server.ZooK
eeperserver)
[2022-11-28 22:09:24,855] INFO / / _ \ / _ \ | | / | / _ \ / _ \ | | (org.apache.zookeeper.server.ZooK
eeperserver)
[2022-11-28 22:09:24,855] INFO / / _ \ | ( ) | | | < | / | / | | | | | (org.apache.zookeeper.server.ZooK
eeperserver)
[2022-11-28 22:09:24,855] INFO / | \ / | \ / | | \ | | | | | | | | (org.apache.zookeeper.server.ZooK
eeperserver)
[2022-11-28 22:09:24,855] INFO | | | | | | | | | | | | | | | | | | | | (org.apache.zookeeper.server.ZooK
eeperserver)
[2022-11-28 22:09:24,855] INFO | | | | | | | | | | | | | | | | | | | | (org.apache.zookeeper.server.ZooK
eeperserver)
```

FIGURE 2.5 : Capture pour le lancement de Zookeeper

— Lancer le 1er serveur Kafka

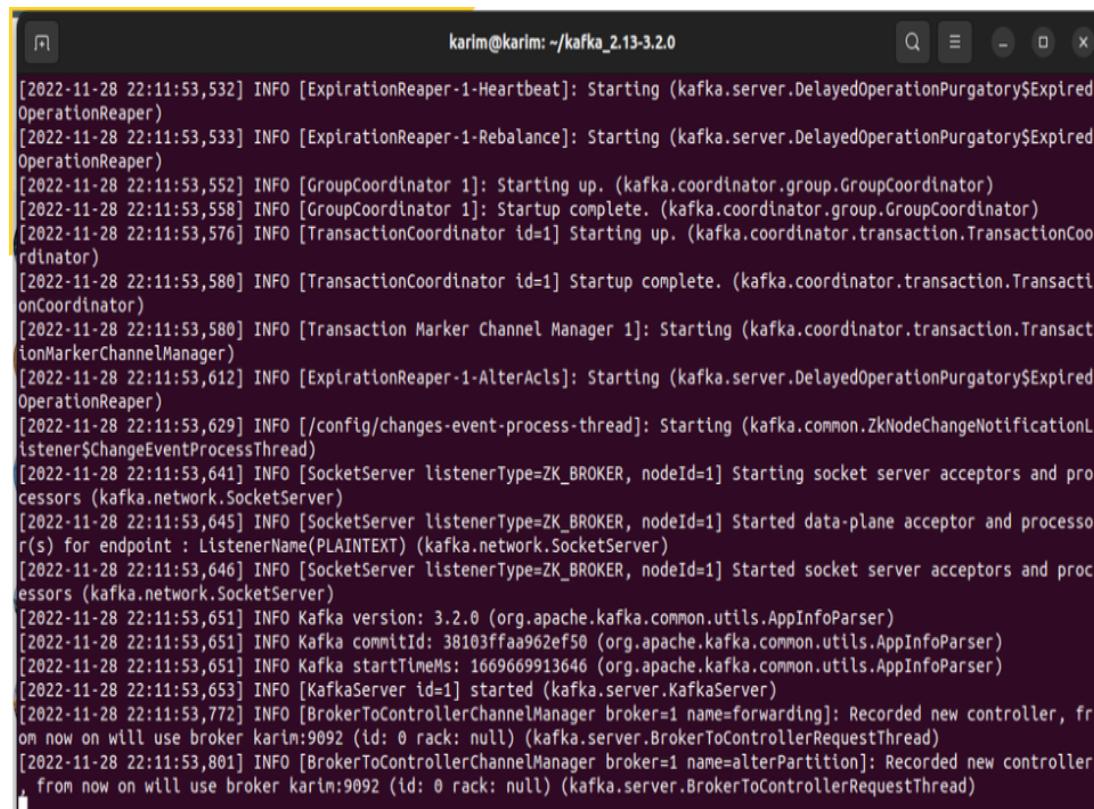
On a travaillé avec kafka version 2.2.0.

— Lancer le 2ème serveur Kafka



```
karim@karim: ~/kafka_2.13-3.2.0
[2022-11-28 22:11:11,998] INFO [ExpirationReaper-0-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-11-28 22:11:12,028] INFO Successfully created /controller_epoch with initial epoch 0 (kafka.zk.KafkaZkClient)
[2022-11-28 22:11:12,068] INFO [GroupCoordinator 0]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2022-11-28 22:11:12,073] INFO [GroupCoordinator 0]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2022-11-28 22:11:12,098] INFO Feature ZK node created at path: /feature (kafka.server.FinalizedFeatureChangeListener)
[2022-11-28 22:11:12,118] INFO [TransactionCoordinator id=0] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2022-11-28 22:11:12,123] INFO [Transaction Marker Channel Manager 0]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2022-11-28 22:11:12,123] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2022-11-28 22:11:12,128] INFO Updated cache from existing <empty> to latest FinalizedFeaturesAndEpoch(features=Features{}, epoch=0). (kafka.server.FinalizedFeatureCache)
[2022-11-28 22:11:12,154] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-11-28 22:11:12,174] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2022-11-28 22:11:12,187] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Starting socket server acceptors and processors (kafka.network.SocketServer)
[2022-11-28 22:11:12,193] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started data-plane acceptor and processor(s) for endpoint : ListenerName(PLAINTEXT) (kafka.network.SocketServer)
[2022-11-28 22:11:12,193] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started socket server acceptors and processors (kafka.network.SocketServer)
[2022-11-28 22:11:12,207] INFO Kafka version: 3.2.0 (org.apache.kafka.common.utils.AppInfoParser)
[2022-11-28 22:11:12,207] INFO Kafka commitId: 38103ffaa962ef50 (org.apache.kafka.common.utils.AppInfoParser)
[2022-11-28 22:11:12,207] INFO Kafka startTimeMs: 1669669872194 (org.apache.kafka.common.utils.AppInfoParser)
[2022-11-28 22:11:12,209] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2022-11-28 22:11:12,325] INFO [BrokerToControllerChannelManager broker=0 name=alterPartition]: Recorded new controller, from now on will use broker karim:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2022-11-28 22:11:12,328] INFO [BrokerToControllerChannelManager broker=0 name=forwarding]: Recorded new controller, from now on will use broker karim:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
```

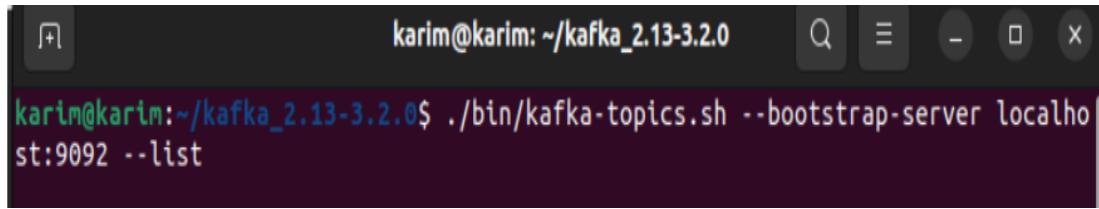
FIGURE 2.6 : Capture pour le lancement du 1er serveur Kafka



```
karim@karim: ~/kafka_2.13-3.2.0
[2022-11-28 22:11:53,532] INFO [ExpirationReaper-1-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-11-28 22:11:53,533] INFO [ExpirationReaper-1-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-11-28 22:11:53,552] INFO [GroupCoordinator 1]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2022-11-28 22:11:53,558] INFO [GroupCoordinator 1]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2022-11-28 22:11:53,576] INFO [TransactionCoordinator id=1] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2022-11-28 22:11:53,580] INFO [TransactionCoordinator id=1] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2022-11-28 22:11:53,580] INFO [Transaction Marker Channel Manager 1]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2022-11-28 22:11:53,612] INFO [ExpirationReaper-1-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-11-28 22:11:53,629] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2022-11-28 22:11:53,641] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1] Starting socket server acceptors and processors (kafka.network.SocketServer)
[2022-11-28 22:11:53,645] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1] Started data-plane acceptor and processor(s) for endpoint : ListenerName(PLAINTEXT) (kafka.network.SocketServer)
[2022-11-28 22:11:53,646] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1] Started socket server acceptors and processors (kafka.network.SocketServer)
[2022-11-28 22:11:53,651] INFO Kafka version: 3.2.0 (org.apache.kafka.common.utils.AppInfoParser)
[2022-11-28 22:11:53,651] INFO Kafka commitId: 38103ffaa962ef50 (org.apache.kafka.common.utils.AppInfoParser)
[2022-11-28 22:11:53,651] INFO Kafka startTimeMs: 1669669913646 (org.apache.kafka.common.utils.AppInfoParser)
[2022-11-28 22:11:53,653] INFO [KafkaServer id=1] started (kafka.server.KafkaServer)
[2022-11-28 22:11:53,772] INFO [BrokerToControllerChannelManager broker=1 name=forwarding]: Recorded new controller, from now on will use broker karim:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2022-11-28 22:11:53,801] INFO [BrokerToControllerChannelManager broker=1 name=alterPartition]: Recorded new controller, from now on will use broker karim:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
```

FIGURE 2.7 : Capture pour le lancement du 2éme serveur Kafka

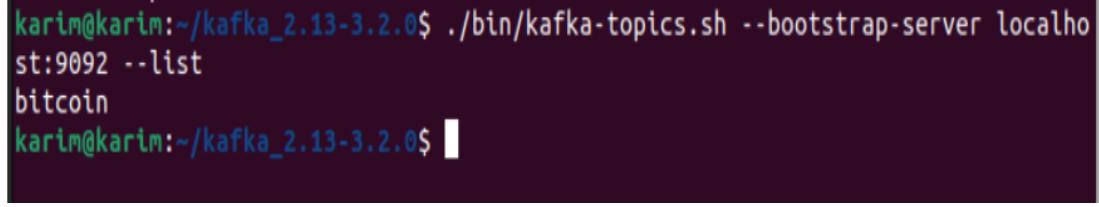
- Créer le topic "bitcoin"



```
karim@karim:~/kafka_2.13-3.2.0$ ./bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
```

FIGURE 2.8 : Capture pour la création de topic "bitcoin"

On a vérifié que le topic est créé en exécutant la commande suivante :



```
karim@karim:~/kafka_2.13-3.2.0$ ./bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
bitcoin
karim@karim:~/kafka_2.13-3.2.0$
```

FIGURE 2.9 : Vérifier la création du topic "bitcoin"

Un producer ajoute des messages à un topic doté d'une seule partition et les messages sont récupérés par un unique consumer.

On a voulu augmenter le nombre de consumers, ce qui signifie mathématiquement qu'il va falloir augmenter le nombre de partitions de notre topic. On passe à 10 partitions à l'aide de la commande suivante :

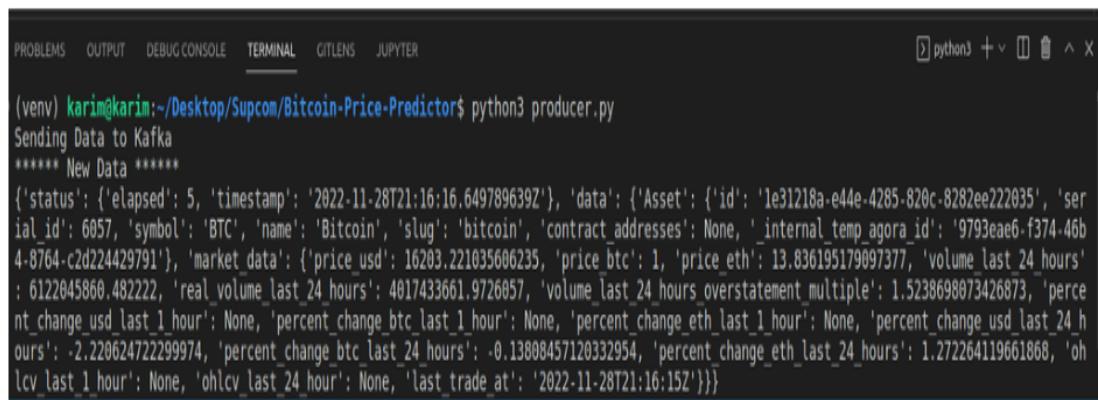


```
karim@karim:~/kafka_2.13-3.2.0$ ./bin/kafka-topics.sh --bootstrap-server localhost:9092 --list
karim@karim:~/kafka_2.13-3.2.0$ ./bin/kafka-topics.sh --create --topic bitcoin --bootstrap-server localhost:9093 --replication-factor 2 --partitions 10
Created topic bitcoin.
```

FIGURE 2.10 : Augmenter le nombre de partitions de topic "bitcoin"

— Exécuter le script "producer1.py"

Notre topic Kafka se remplit progressivement et il nous reste à créer un consumer qui va lire les données de notre topic.



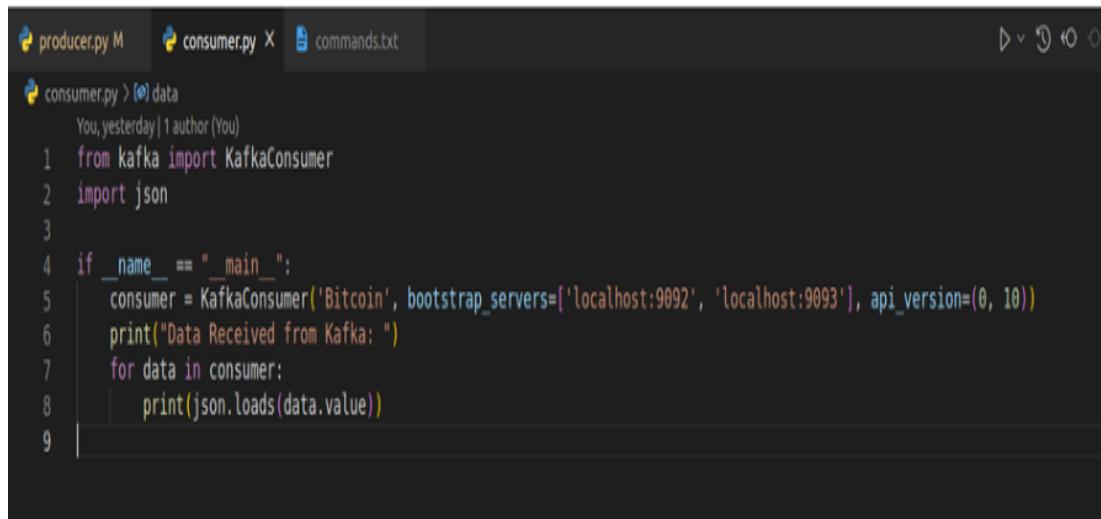
```
(venv) karim@karim:~/Desktop/Supcom/Bitcoin-Price-Predictor$ python3 producer.py
Sending Data to Kafka
***** New Data *****
{'status': {'elapsed': 5, 'timestamp': '2022-11-28T21:16:16.649789639Z'}, 'data': {'Asset': {'id': '1e31218a-e44e-4285-820c-8282ee222035', 'serial_id': 6057, 'symbol': 'BTC', 'name': 'Bitcoin', 'slug': 'bitcoin', 'contract_addresses': None, 'internal_temp_agora_id': '9793eaef-f374-46b4-8764-c2d224429791'}, 'market_data': {'price_usd': 16203.221035606235, 'price_btc': 1, 'price_eth': 13.836195179097377, 'volume_last_24_hours': 6122045860.482222, 'real_volume_last_24_hours': 4017433661.9726057, 'volume_last_24_hours_overstatement_multiple': 1.5238698073426873, 'percent_change_usd_last_1_hour': None, 'percent_change_btc_last_1_hour': None, 'percent_change_eth_last_1_hour': None, 'percent_change_usd_last_24_hours': -2.220624722299974, 'percent_change_btc_last_24_hours': -0.13808457120332954, 'percent_change_eth_last_24_hours': 1.272264119661868, 'ohlcv_last_1_hour': None, 'ohlcv_last_24_hours': None, 'last_trade_at': '2022-11-28T21:16:15Z'}}}
```

FIGURE 2.11 : Résultat de producer1

2.3 Consumer

On a également utilisé le package kafka-python pour développer un consumer.

Voici le script "consumer.py" présenté ci-dessous. Dans ce script, on a créé un consumer Kafka pour



```
producer.py M consumer.py X commands.txt

consumer.py > [0]data
You, yesterday | 1 author (You)
1 from kafka import KafkaConsumer
2 import json
3
4 if __name__ == "__main__":
5     consumer = KafkaConsumer('Bitcoin', bootstrap_servers=['localhost:9092', 'localhost:9093'], api_version=(0, 10))
6     print("Data Received from Kafka: ")
7     for data in consumer:
8         print(json.loads(data.value))
9
```

FIGURE 2.12 : Script de consumer1

le topic "bitcoin".

— Exécuter le script "consumer.py"

Après l'exécution du script "consumer.py" on a obtenu le résultat suivant.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS JUPYTER + v ^ x

source /home/karim/Desktop/Supcom/Bitcoin-Price-Predictor/venv/bin/activate
karim@karim:~/Desktop/Supcom/Bitcoin-Price-Predictor$ source /home/karim/Desktop/Supcom/Bitcoin-Price-Predictor/venv/bin/
activate
(venv) karim@karim:~/Desktop/Supcom/Bitcoin-Price-Predictor$ python3 consumer.py
Data Received from Kafka:
{'status': {'elapsed': 5, 'timestamp': '2022-11-28T21:17:18.271880585Z'}, 'data': {'Asset': {'id': '1e31218a-e44e-4285-82
0c-8282ee222035', 'serial id': 6057, 'symbol': 'BTC', 'name': 'Bitcoin', 'slug': 'bitcoin', 'contract addresses': None, '_internal_temp_agora_id': '9793ea6-f374-46b4-8764-c2d22429791'}, 'market data': {'price_usd': 16202.334303965094, 'pric
e_btc': 1, 'price_eth': 13.836066687483873, 'volume_last_24_hours': 6125431956.69196, 'real_volume_last_24_hours': 401991
3862.8571973, 'volume_last_24_hours_overstatement_multiple': 1.5237719427993524, 'percent_change_usd_last_1_hour': None,
'percent_change_btc_last_1_hour': None, 'percent_change_eth_last_1_hour': None}, 'order': 1}
```

FIGURE 2.13 : Résultat de consumer

— Connecter Spark avec le topic Kafka "bitcoin"

Cette étape permet la lecture des données en streaming avec `readStream` à partir de topic "bitcoin" et les stocker dans une streaming spark DataFrame. Voici le script ci-dessous qui permet de connecter Spark avec le topic Kafka "bitcoin".

On a commencé par l'importation des bibliothèques nécessaires.

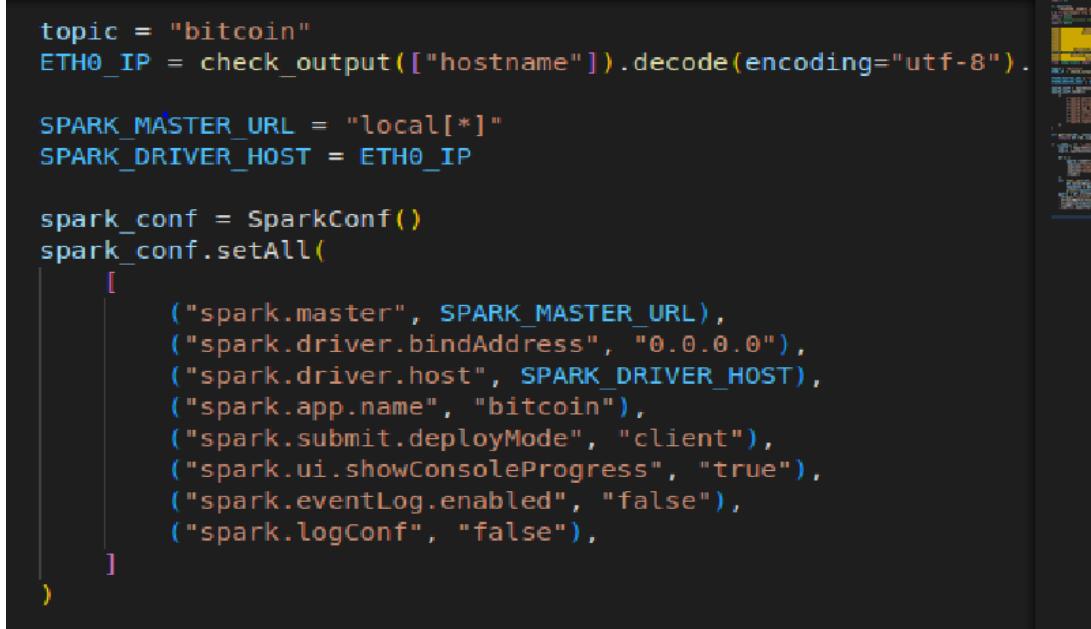
```
import os

os.environ[
    "PYSPARK_SUBMIT_ARGS"
] = "--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3
import json
#from collections import defaultdict
import math

from pyspark.conf import SparkConf
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.sql.functions import from_json, col
from pyspark import SparkContext
#from pyspark import StreamingContext
from kafka import KafkaConsumer
from elasticsearch import Elasticsearch
from subprocess import check_output
```

FIGURE 2.14 : Importer les bibliothéques

Ensuite on a crée un objet SparkConf avec SparkConf(), qui chargera également les valeurs des propriétés système Spark. Dans ce cas, tous les paramètres qu'on a définissé directement sur l'objet SparkConf sont prioritaires sur les propriétés système.



```

topic = "bitcoin"
ETH0_IP = check_output(["hostname"]).decode(encoding="utf-8").strip()

SPARK_MASTER_URL = "local[*]"
SPARK_DRIVER_HOST = ETH0_IP

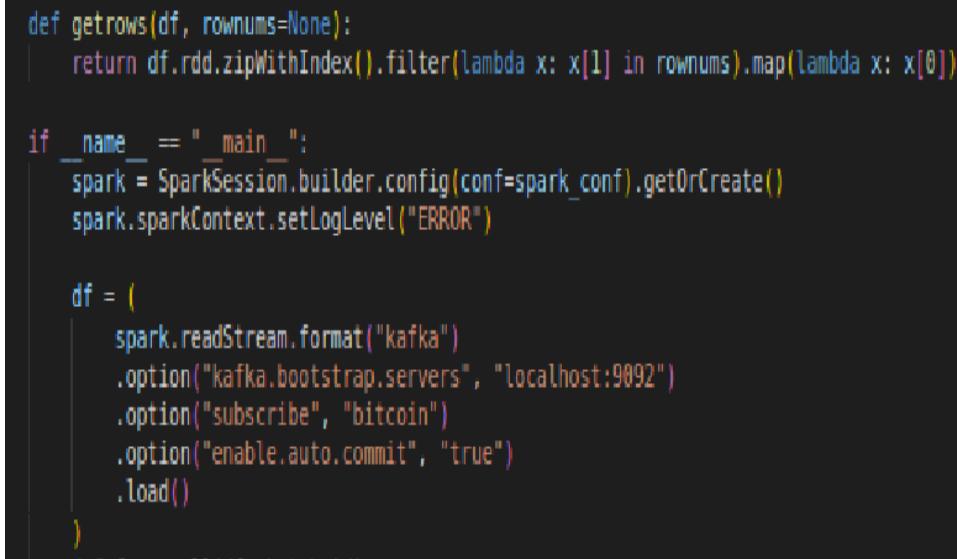
spark_conf = SparkConf()
spark_conf.setAll(
    [
        ("spark.master", SPARK_MASTER_URL),
        ("spark.driver.bindAddress", "0.0.0.0"),
        ("spark.driver.host", SPARK_DRIVER_HOST),
        ("spark.app.name", "bitcoin"),
        ("spark.submit.deployMode", "client"),
        ("spark.ui.showConsoleProgress", "true"),
        ("spark.eventLog.enabled", "false"),
        ("spark.logConf", "false"),
    ]
)

```

FIGURE 2.15 : Créer un objet SparkConf

— Créer Streaming DataFrame

On a crée Streaming DataFrame en utilisant la méthode "Spark.readStream" et on a spécifié les détails de la source : format de données, schéma et options comme indiqué ci-dessous.



```

def getrows(df, rownums=None):
    return df.rdd.zipWithIndex().filter(lambda x: x[1] in rownums).map(lambda x: x[0])

if __name__ == "__main__":
    spark = SparkSession.builder.config(conf=spark_conf).getOrCreate()
    spark.sparkContext.setLogLevel("ERROR")

    df = (
        spark.readStream.format("kafka")
            .option("kafka.bootstrap.servers", "localhost:9092")
            .option("subscribe", "bitcoin")
            .option("enable.auto.commit", "true")
            .load()
    )

```

FIGURE 2.16 : Créer Streaming DataFrame

— Lire les données

On a enfin lu les données à partie de DataFrame en utilisant la méthode "writeStream" et les afficher. D'une part on a lancé le producer et on s'assuré que les données sont envoyés vers Kafka. D'autre part on a exécuté le script "consumer.py" en exécutant la commande suivante :

```

def func_call(df, batch_id):
    df.selectExpr("CAST(value AS STRING) as json")
    requests = df.rdd.map(lambda x: x.value).collect()
    print(requests)
    query = df.writeStream \
        .format('console') \
        .foreachBatch(func_call) \
        .trigger(processingTime="30 seconds") \
        .start().awaitTermination()

```

FIGURE 2.17 : Lire les données

```

zeineb@zeineb-VirtualBox:~/Desktop$ python3 producer.py
sending data to kafka
{'status': {'elapsed': 20, 'timestamp': '2022-12-11T16:37:24.586986063Z'}, 'data': {'Asset': {'id': '1e31218a-e44e-4285-82c-8282ee222035', 'serial_id': 6057, 'symbol': 'BTC', 'name': 'Bitcoin', 'slug': 'bitcoin', 'contract_addresses': None, '_internal_temp_agora_id': '9793eae6-f374-46b4-8764-c2d224429791'}, 'market_data': {'price_usd': 17161.47679319596, 'price_btc': 1, 'price_eth': 13.484346539906817, 'volume_last_24_hours': 3116948895.4202824, 'real_volume_last_24_hours': 2191764212.694909, 'volume_last_24_hours_overstatement_multiple': 1.422157642151765, 'percent_change_usd_last_1_hour': None, 'percent_change_btc_last_1_hour': None, 'percent_change_eth_last_1_hour': None, 'percent_change_usd_last_24_hours': -0.16514684935478638, 'percent_change_btc_last_24_hours': -0.035006105321230295, 'percent_change_eth_last_24_hours': 0.08832358197025032, 'ohlcv_last_1_hour': None, 'ohlcv_last_24_hour': None, 'last_trade_at': '2022-12-11T16:37:23Z'}}}
{'status': {'elapsed': 4, 'timestamp': '2022-12-11T16:37:28.562499892Z'}, 'data': {'Asset': {'id': '1e31218a-e44e-4285-82c-8282ee222035', 'serial_id': 6057, 'symbol': 'BTC', 'name': 'Bitcoin', 'slug': 'bitcoin', 'contract_addresses': None, '_internal_temp_agora_id': '9793eae6-f374-46b4-8764-c2d224429791'}, 'market_data': {'price_usd': 17161.38449118186, 'price_btc': 1, 'price_eth': 13.484142392111082, 'volume_last_24_hours': 3116988661.289043, 'real_volume_last_24_hours': 2191714454.9456477, 'volume_last_24_hours_overstatement_multiple': 1.4221654897860958, 'percent_change_usd_last_1_hour': None, 'percent_change_btc_last_1_hour': None, 'percent_change_eth_last_1_hour': None, 'percent_change_usd_last_24_hours': -0.16568380536005328, 'percent_change_btc_last_24_hours': -0.035006105321230295, 'percent_change_eth_last_24_hours': 0.08680828342926537, 'ohlcv_last_1_hour': None, 'ohlcv_last_24_hour': None, 'last_trade_at': '2022-12-11T16:37:27Z'}}}
{'status': {'elapsed': 1, 'timestamp': '2022-12-11T16:37:32.534806222Z'}, 'data': {'Asset': {'id': '1e31218a-e44e-4285-82c-8282ee222035', 'serial_id': 6057, 'symbol': 'BTC', 'name': 'Bitcoin', 'slug': 'bitcoin', 'contract_addresses': None, '_internal_temp_agora_id': '9793eae6-f374-46b4-8764-c2d224429791'}, 'market_data': {'price_usd': 17161.40019893652, 'price_btc': 1, 'price_eth': 13.484142392111082, 'volume_last_24_hours': 3116988661.289043, 'real_volume_last_24_hours': 2191714454.9456477, 'volume_last_24_hours_overstatement_multiple': 1.4221654897860958, 'percent_change_usd_last_1_hour': None, 'percent_change_btc_last_1_hour': None, 'percent_change_eth_last_1_hour': None, 'percent_change_usd_last_24_hours': -0.16568380536005328, 'percent_change_btc_last_24_hours': -0.035006105321230295, 'percent_change_eth_last_24_hours': 0.08680828342926537, 'ohlcv_last_1_hour': None, 'ohlcv_last_24_hour': None, 'last_trade_at': '2022-12-11T16:37:27Z'}}}

```

FIGURE 2.18 : Lancer le producer

```

zeineb@zeineb-VirtualBox:~/spark-3.3.1-bin-hadoop3/bin$ spark-submit --packaging org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.1 /home/zeineb/Desktop/consumer.py

22/12/11 18:06:35 WARN Utils: Your hostname, zeineb-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
22/12/11 18:06:35 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
:: loading settings :: url = jar:file:/opt/spark/jars/ivy-2.5.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/zeineb/.ivy2/cache
The jars for the packages stored in: /home/zeineb/.ivy2/jars
org.apache.spark#spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-219056e7-541a2-b4c6-aae176f92602;1.0
  confs: [default]

```

FIGURE 2.19 : Exécuter consumer2.py

Finalement les données sont affichés comme présenté ci-dessous.

```
[bytarray(b'{"status": {"elapsed": 4, "timestamp": "2022-12-11T17:33:47.049946383Z"}, "data": {"Asset": {"id": "1e31218a-e44e-4285-820c-8282ee222035", "serial_id": 6057, "symbol": "BTC", "name": "Bitcoin", "slug": "bitcoin", "contract_addresses": null, "internal_temp_agora_id": "9793eae6-f374-46b4-8764-c2d224429791"}, "market_data": {"price_usd": 17173.1503076173, "price_btc": 1, "price_eth": 13.467754578948682, "volume_last_24_hours": 3074063742.405888, "real_volume_last_24_hours": 2167078770.0517516, "volume_last_24_hours_overstatement_multiple": 1.4185288439388277, "percent_change_usd_last_1_hour": 0.10567441102999516, "percent_change_btc_last_1_hour": -0.006447891252888586, "percent_change_eth_last_1_hour": -0.13201334814251178, "percent_change_usd_last_24_hours": 0.02310629332953848, "percent_change_btc_last_24_hours": 0.1090980877535715, "percent_change_eth_last_24_hours": -0.06846250498905897, "ohlcv_last_1_hour": {"open": 17156.452975338736, "high": 17156.851791648132, "low": 17153.17912182981, "close": 17155.021839326524, "volume": 3848206.349331313}, "ohlcv_last_24_hour": null, "last_trade_at": "2022-12-11T17:33:45Z"}}}), bytarray(b'{"status": {"elapsed": 214, "timestamp": "2022-12-11T17:33:38.975166557Z"}, "data": {"Asset": {"id": "1e31218a-e44e-4285-820c-8282ee222035", "serial_id": 6057, "symbol": "BTC", "name": "Bitcoin", "slug": "bitcoin", "contract_addresses": null, "internal_temp_agora_id": "9793eae6-f374-46b4-8764-c2d224429791"}, "market_data": {"price_usd": 17174.016834447993, "price_btc": 1, "price_eth": 13.465847105630242, "volume_last_24_hours": 3073984524.7602086, "real_volume_last_24_hours": 2167042470.5592694, "volume_last_24_hours_overstatement_multiple": 1.418516049649399, "percent_change_usd_last_1_hour": 0.11072556653891799, "percent_change_btc_last_1_hour": -0.006447891252888586, "percent_change_eth_last_1_hour": -0.14615791238950718, "percent_change_usd_last_24_hours": 0.028153282597086503, "percent_change_btc_last_24_hours": 0.1090980877535715, "percent_change_eth_last_24_hours": -0.08261607010824644, "ohlcv_last_1_hour": {"open": 17156.452975338736, "high": 17156.851791648132, "low": 17153.17912182981, "close": 17155.021839326524, "volume": 3848206.349331313}, "ohlcv_last_24_hour": null, "last_trade_at": "2022-12-10T17:33:37Z"}}}), bytarray(b'{"status":
```

FIGURE 2.20 : Résultat affiché

2.4 Indexage des données à l'aide de Elasticsearch

Elasticsearch utilise une structure de données appelée index inversé et conçue pour faire des recherches full-text très rapides. Un index inversé liste chaque mot unique qui apparaît dans n'importe quel document et identifie tous les documents dans lesquels chaque mot apparaît. Au cours du processus d'indexation, Elasticsearch stocke des documents et fabrique un index inversé pour permettre de rechercher les données du document en temps quasi réel.

Dans ElasticSearch, l'unité est le document. En d'autres termes, l'index dans ElasticSearch c'est l'ensemble des documents qui sont stockés dans le moteur. Les documents sont enregistrés au format JSON. La structure hiérarchique du format JSON permet à ElasticSearch d'indexer tout le contenu des documents. À titre de rappel, le contenu d'un document JSON est formellement organisé sous forme d'objets. Chaque objet est l'équivalent d'une ligne dans une table de base de données relationnelle. Les documents sont organisés en séries de propriétés, qui sont l'équivalent des colonnes dans le monde des bases de données relationnelles.

Puisqu'on va utiliser la deuxième dataset pour prédire le prix de bitcoin on a finalisé le script qui permet l'indexage des données comme présenté ci-dessous.

```
3  from pyspark.sql.types import *
4  from pyspark.sql.functions import *
5  from pyspark.sql import SparkSession
6  from pyspark.conf import SparkConf
7  import math
8  from _collections import defaultdict
9  import json
10 from datetime import datetime
11 import os
12 os.environ[
13     "PYSPARK_SUBMIT_ARGS"
14 ] = "--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.1,org.elasticsearch:elasticsearch-spark-20_2.12:7.17.5 pyspark-shell"
15
16 topic = "bitcoin"
17 ETH0_IP = check_output(["hostname"]).decode(encoding="utf-8").strip()
18 SPARK_MASTER_URL = "local[*]"
19 SPARK_DRIVER_HOST = ETH0_IP
20 spark_conf = SparkConf()
21 spark_conf.setAll(
22     [
23         ("spark.master", SPARK_MASTER_URL),
24         ("spark.driver.bindAddress", "0.0.0.0"),
25         ("spark.driver.host", SPARK_DRIVER_HOST),
26         ("spark.app.name", "BitcoinConsumer"),
27         ("spark.submit.deployMode", "client"),
28         ("spark.ui.showConsoleProgress", "true"),
29         ("spark.eventLog.enabled", "false"),
30         ("spark.logConf", "false"),
31     ]
32 )
```

FIGURE 2.21 : Script pour l'indexage des données

On a ainsi structré la partie "mapping", en effet le mappage est un processus de définition de la manière dont un document et les champs qu'il contient sont stockés et indexés. Chaque document est une collection de champs, qui ont chacun leur propre type de données . Lors du mappage de nos données, on a crée une définition de mappage, qui contient une liste de champs pertinents pour le document comme "Date", "Open", "High", "Low", "Close", "Adj Close" et "Volume".

```
def getrows(df, rownums=None):
    return df.rdd.zipWithIndex().filter(lambda x: x[1] in rownums).map(lambda x: x[0])

elastic_index = "bitcoin_kafka"
elastic_document = "_doc"
es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'scheme': 'http'}])
es.options(ignore_status=[400, 404]).indices.delete(index=elastic_index)

es_mapping = {'mappings': {
    "properties": {
        "Date": {
            "type": "date"
        },
        "Open": {
            "type": "double"
        },
        "High": {
            "type": "double"
        },
        "Low": {
            "type": "double"
        },
        "Close": {
            "type": "double"
        },
        "Adj Close": {
            "type": "double"
        },
        "Volume": {
            "type": "double"
        }
    }
}}
}

response = es.indices.create(
    index=elastic_index,
    mappings=es_mapping,
    ignore=400 # Ignore Error 400 Index already exists
```

FIGURE 2.22 : Script pour l'indexage des données(mapping)

```

78     spark = SparkSession.builder.config(conf=spark_conf).getOrCreate()
79     spark.sparkContext.setLogLevel("ERROR")
80
81     df = (
82         spark.readStream.format("kafka")
83             .option("kafka.bootstrap.servers", "localhost:9092")
84             .option("subscribe", topic)
85             .option("enable.auto.commit", "true")
86             .load()
87     )
88
89     schema = StructType([
90         StructField("Date", StringType()),
91         StructField("Open", DoubleType()),
92         StructField("High", DoubleType()),
93         StructField("Low", DoubleType()),
94         StructField("Close", DoubleType()),
95         StructField("Adj Close", DoubleType()),
96         StructField("Volume", DoubleType())
97     ])
98
99     def func_call(df, batch_id):
100        valueRdd = df.rdd.map(lambda x: x[1])
101        strList = valueRdd.map(lambda x: json.loads(x)).collect()
102        tupleList = []
103        for batch in strList:
104            tupleList = []
105            _source = {}
106            _source['Date'] = batch['Date']
107            _source['Open'] = batch['Open']
108            _source['High'] = batch['High']
109            _source['Low'] = batch['Low']
110            _source['Close'] = batch['Close']
111            _source['Adj Close'] = batch['Adj Close']
112            _source['Volume'] = batch['Volume']
113            tupleList.append(_source)
114        tweetDF = spark.createDataFrame(data=tupleList, schema=schema)
115        tweetDF.write \
116            .format("org.elasticsearch.spark.sql") \
117            .mode("append") \
118            .option("es.nodes", "localhost").save(elastic_index)
119
120    query = df.writeStream \
121        .foreachBatch(func_call) \
122        .start().awaitTermination()

```

FIGURE 2.23 : Script pour l'indexage des données

Finalement on a obtenu le résultat affiché ci-dessous :

```
    "took": 11,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 3038,
            "relation": "eq"
        },
        "max_score": 1,
        "hits": [
            {
                "_index": "bitcoin_kafka",
                "_type": "_doc",
                "_id": "p4gJm4UBVe-fuqasrBll",
                "_score": 1,
                "_source": {
                    "High": 468.174011,
                    "Date": "2014-09-17",
                    "Open": 465.864014,
                    "@timestamp": "2014-09-17T00:00:00.000+01:00",
                    "Low": 452.421997,
                    "Volume": 21056800,
                    "Adj Close": 457.334015,
                    "Close": 457.334015
                }
            },
            {
                "_index": "bitcoin_kafka",
                "_type": "_doc",
                "_id": "qIgJm4UBVe-fuqasrBll",
                "_score": 1,
                "_source": {
                    "High": 456.859985,
                    "Date": "2014-09-18",
                    "Open": 456.859985,
                    "@timestamp": "2014-09-18T00:00:00.000+01:00",
                    "Low": 413.104004,
                    "Volume": 34483200,
                    "Adj Close": 424.440002,
                    "Close": 424.440002
                }
            }
        ],
        "_index": "bitcoin_kafka",
        "_score": 1
    }
}
```

FIGURE 2.24 : Résultat de l'indexagee des données

2.5 Visualisation des données avec Kibana

On a visualisé les données en utilisant Kibana. Dabord on a visualisé le prix de Bitcoin en fonction de l'année comme présenté ci-dessous.

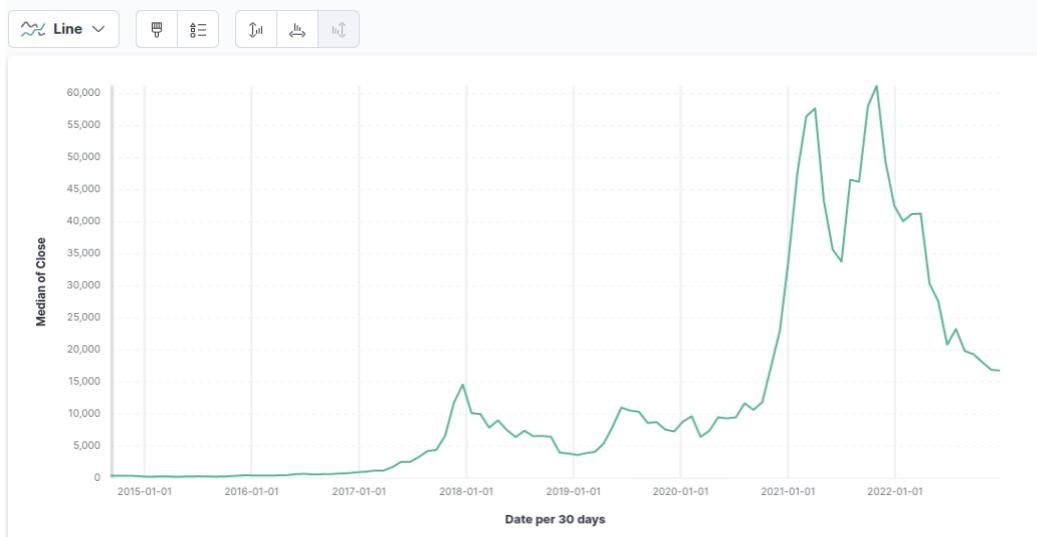


FIGURE 2.25 : Visualisation de prix de bitcoin

On a remarqué d'après cette courbe l'augmentation de prix de Bticoen à partir de l'année 2021 et après 2021 le prix de Bitcoin est entrain de diminuer.

On a pu aussi visualiser la majorité des prix de Bitcoin qui existent.

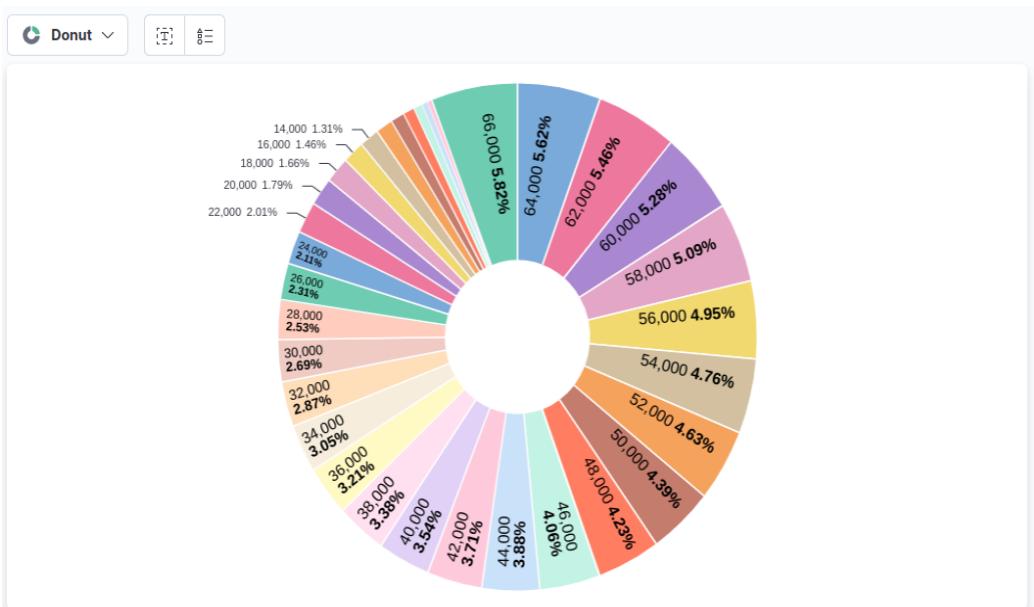


FIGURE 2.26 : les prix de bitcoin

On a obtenu une prédition de prix de Bitcoin à l'aide des outils de Kibana comme présenté ci-dessous.

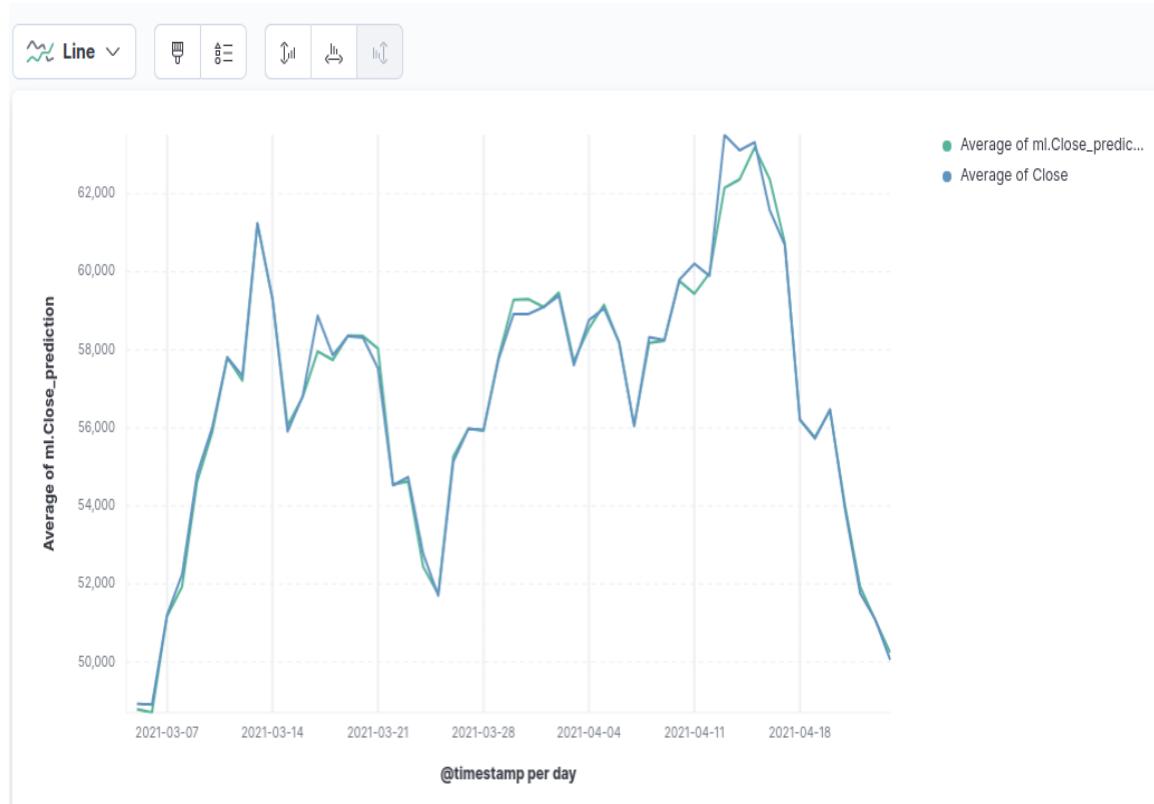


FIGURE 2.27 : Prédiction de prix de bitcoin

Encore on a employé un modèle d'évaluation de la regression utilidé dans Kibana

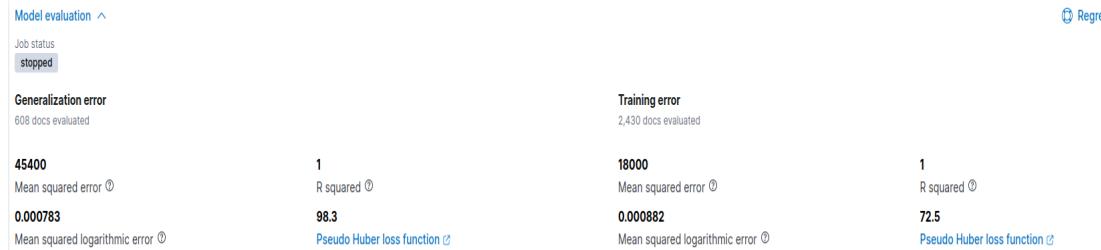


FIGURE 2.28 : Modèle d'évaluation de la regression

On a obtenu le résultat de prévision de prix de bitcoin de l'année 2023 comme présenté ci-dessous.



FIGURE 2.29 : Prévision de prix de Bitcoin

On a creusé de plus dans l'analyse des données en détectant la hausse et la chute de prix de bitcoin comme présenté ci-dessous.



FIGURE 2.30 : Analyse de prix de Bitcoin

Time	Severity ↓	Detector	Actual Ⓜ	Typical Ⓜ	Description	Actions
› December 6th 2017	● 97	mean(Close)	17,899.699	13,017.834	↑ 1.4x higher	⚙️
› June 15th 2016	● 94	mean(Close)	766.308	525.89	↑ 1.5x higher	⚙️
› November 3rd 2015	● 93	mean(Close)	411.563	287.354	↑ 1.4x higher	⚙️
› December 21st 2017	● 91	mean(Close)	13,831.8	18,790.357	↓ 1.4x lower	⚙️
› December 7th 2017	● 86	mean(Close)	16,569.4	13,546.847	↑ 1.2x higher	⚙️
› June 18th 2016	● 86	mean(Close)	763.781	533.227	↑ 1.4x higher	⚙️
› June 17th 2016	● 86	mean(Close)	756.227	530.859	↑ 1.4x higher	⚙️
› June 16th 2016	● 86	mean(Close)	748.909	528.397	↑ 1.4x higher	⚙️
› January 13th 2015	● 85	mean(Close)	178.103	351.179	↓ 2x lower	⚙️
› December 23rd 2017	● 81	mean(Close)	13,925.8	18,302.628	↓ 1.3x lower	⚙️
› November 2nd 2015	● 79	mean(Close)	403.417	285.833	↑ 1.4x higher	⚙️
› December 5th 2017	● 78	mean(Close)	14,291.5	11,548.095	↑ 1.2x higher	⚙️
› December 10th 2017	◆ 74	mean(Close)	16,936.801	15,201.547	↑ 1.1x higher	⚙️
› December 24th 2017	◆ 70	mean(Close)	14,026.6	18,288.366	↓ 1.3x lower	⚙️
› March 5th 2017	◆ 69	mean(Close)	1,272.83	1,067.68	↑ 1.2x higher	⚙️
› January 16th 2015	◆ 67	mean(Close)	199.26	351.138	↓ 2x lower	⚙️
› December 11th 2017	◆ 67	mean(Close)	17,415.4	15,888.43	↑ 1.1x higher	⚙️
› June 19th 2016	◆ 64	mean(Close)	737.226	535.279	↑ 1.4x higher	⚙️
› June 12th 2016	◆ 64	mean(Close)	704.376	519.321	↑ 1.4x higher	⚙️

FIGURE 2.31 : Analyse de prix de Bitcoin

2.6 Prédiction de prix de bitcoin avec le model de Machine Learning (LSTM)

Comme travail supplémentaire, on a utilisé l'algorithme LSTM afin de prédire le prix de Bitcoin.

Tout d'abord on a importé le Dataset à patir de YahooFinance API

Importing dataset (YahooFinance)

```
[1]: import yfinance as yf
from pandas_datareader import data as pdr

yf.pdr_override()

maindf = pdr.get_data_yahoo('BTC-USD', '2014-09-17', '2023-01-11')
maindf.reset_index(inplace = True)
```

[*****100%*****] 1 of 1 completed

FIGURE 2.32 : Dataset YahooFinance API

Ensuite, on a fait une analyse de différentes années à partir de l'année 2015 jusqu'à l'année 2023.

La figure ci-dessous présente l'analyse de données de l'année 2015.

Monthwise comparision between Stock open and close price

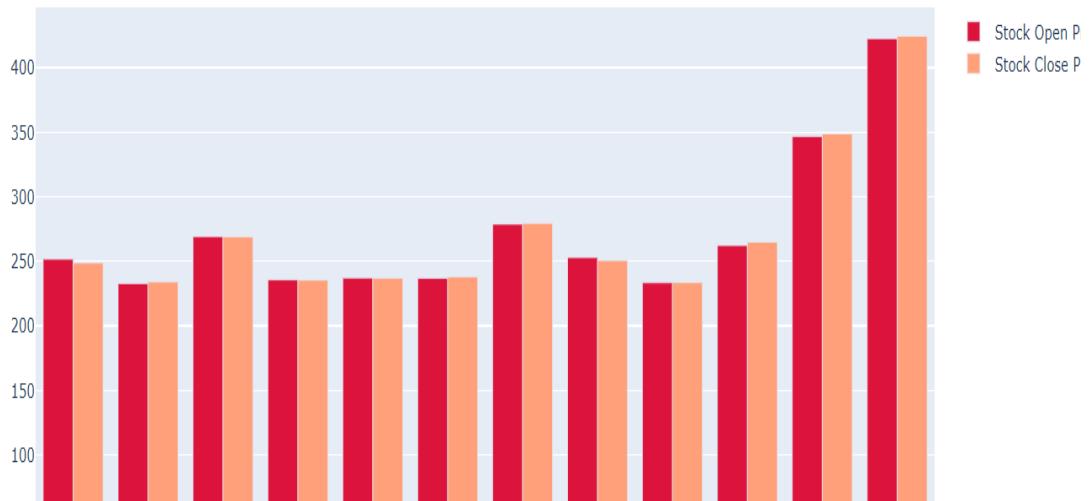


FIGURE 2.33 : L'analyse de données de l'année 2015

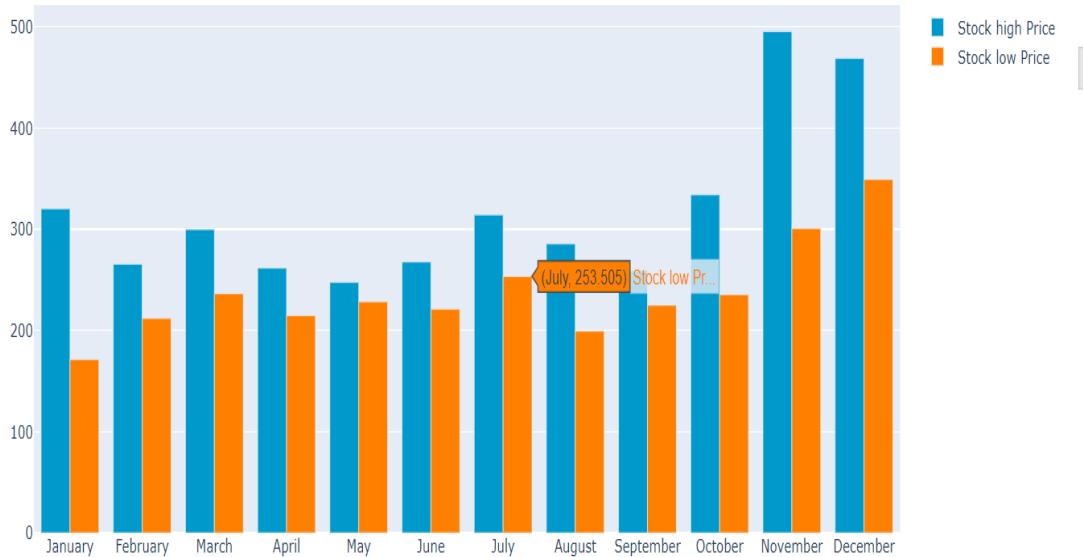
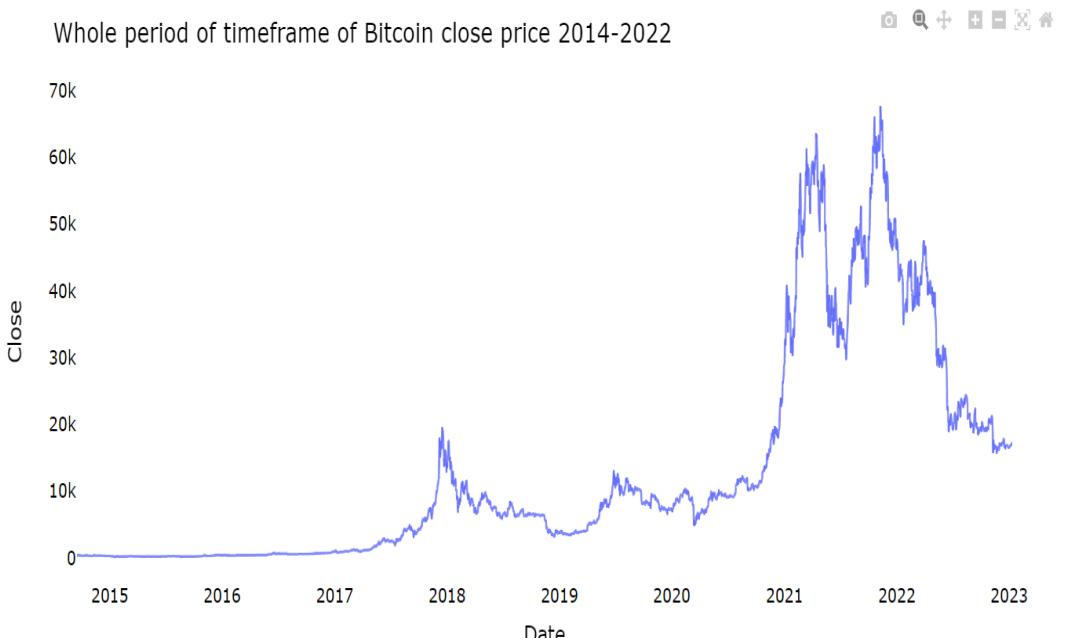


FIGURE 2.34 : L'analyse de données de l'année 2015

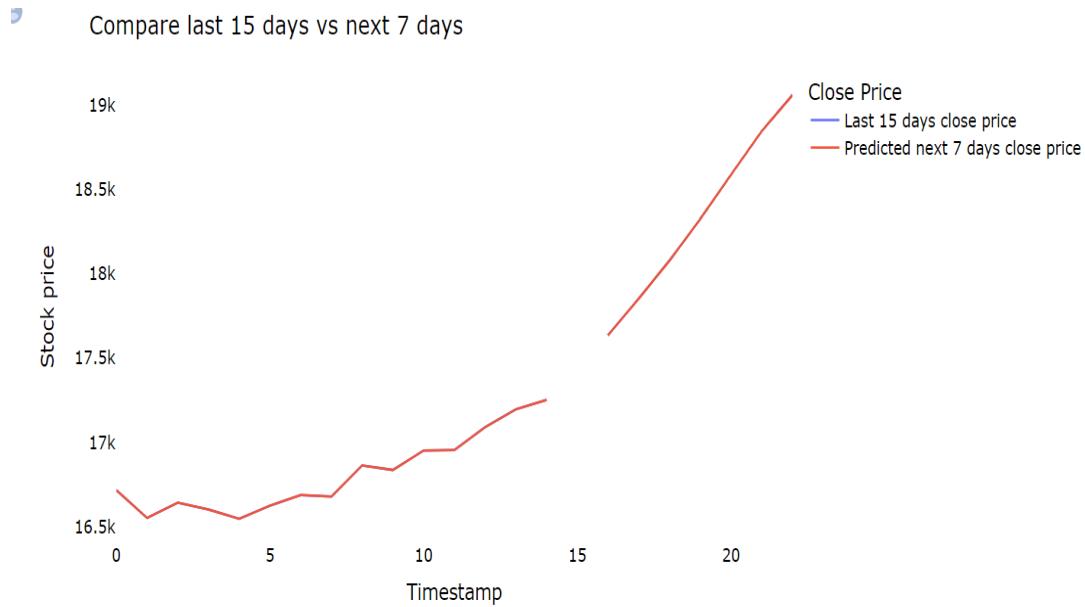
Après on a réalisé une analyse globale de prix de Bitcoin à partir de l'année 2014 jusqu'au aujourd'hui.

On a obtenu le résultat suivant :

**FIGURE 2.35 :** L'analyse globale de prix de Bitcoin

Finalement, on a crée le modèle et on a fait le training et on a obtenu le prédition de pix de Bitcoin comme présenté dans les figures ci-dessous.

**FIGURE 2.36 :** Prédiction de prix de Bitcoin

**FIGURE 2.37 :** Prédiction de prix de Bitcoin**FIGURE 2.38 :** Prédiction de prix de Bitcoin

Conclusion

La prévision à la demande est la toute nouvelle fonctionnalité de Machine Learning. Jusqu'ici, la solution Machine Learning d'Elasticsearch était conçue pour exploiter les données historiques et prévoir la plage de valeurs normales à laquelle on pouvait s'attendre "maintenant". Alors grâce à des outils de Kibana on a pu faire une prévision de prix de Bitcoin. On a pas se contenter seulement sur l'analyse offerte par Kibana, mais on a fait une prédiction de prix de Bitcoin dans les 7 prochains jours.

