AA 2021/2022
Università degli Studi di Milano Bicocca

Data Management

Database - NFT & Tweets

Potertì Daniele 844892

Sanvito Alessio 844785

Sanvito Simone 844794

SOMMARIO

PANORAMICA

Cosa sono gli NFT? Come funzionano? Come si fa a investire? Queste e altre domande rimbalzano da più parti da quando gli NFT hanno iniziato a riscuotere un successo clamoroso.

Il termine NFT sta per Non-Fungible Token, che in italiano vuol dire Gettone digitale non fungibile, non riproducibile. Si tratta di tipi speciali di token crittografici (smart contract) che rappresentano l'atto di proprietà e il certificato di autenticità scritto su Blockchain. Con NFT si intende un modo per identificare in modo univoco, sicuro e senza dubbi un prodotto digitale creato su internet. NFT può essere qualsiasi oggetto digitale: un video, una foto, una GIF, un testo, un articolo, un audio.

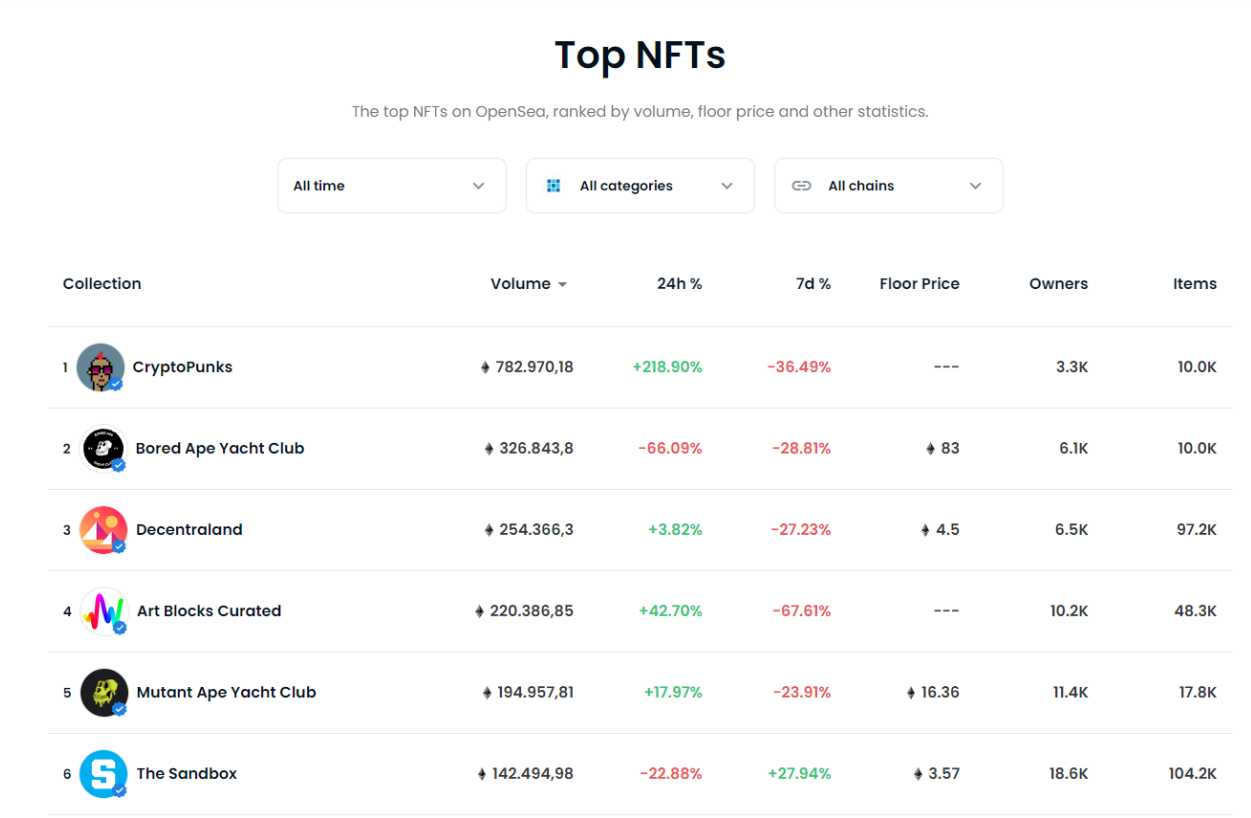
Gli NFT sono asset digitali unici che vengono acquistati e venduti online utilizzando criptovalute. Sono dei prodotti molto discussi al momento in quanto hanno fatto irruzione sui diversi mercati di tutto il mondo, dal settore artistico a quello dei giochi, fino agli eventi e alle assicurazioni.

OBIETTIVI

L’idea che sta dietro alla realizzazione di questo progetto è la seguente: dal momento che gli NFT sono un fenomeno molto recente e molto discusso in rete, è stato deciso di creare un database che associa ad ogni collection NFT circa un migliaio (ove possibile) di tweet presi nell’ultima settimana dal momento dell’inizio della ricerca. L’obiettivo è quello di ottenere delle metriche di popolarità per ogni progetto NFT per analizzare quanto “hype” ci sia intorno a quel determinato progetto e analizzare quali possano essere i progetti potenzialmente più soggetti a crescita di valore (la popolarità è una metrica che influenza pesantemente il valore). Per fare ciò, i dati sono stati integrati con le API di Twitter, raccogliendo circa 300 mila tweets per le prime 1532 collections NFT considerate. Sono stati scaricati circa 1000 tweets per ogni collezione NFT e, in seguito all’integrazione dei dataset, sono stati associati tutti i tweets riguardanti una collection alla collection stessa, aumentandone (in alcuni casi) il numero. Tramite queste operazioni si può contare il numero di tweets che ogni collezione ha raccolto in un determinato periodo di tempo, in modo da valutarne la popolarità rispetto ad un’altra che potrebbe aver raccolto meno tweets o lo stesso numero (circa) in un periodo di tempo più lungo.

FONTI DEI DATI

Scraping



Top NFTs

The top NFTs on OpenSea, ranked by volume, floor price and other statistics.

All time | All categories | All chains

Collection	Volume	24h %	7d %	Floor Price	Owners	Items
1 CryptoPunks	782.970,18	+218.90%	-36.49%	---	3.3K	10.0K
2 Bored Ape Yacht Club	326.843,8	-66.09%	-28.81%	83	6.1K	10.0K
3 Decentraland	254.366,3	+3.82%	-27.23%	4.5	6.5K	97.2K
4 Art Blocks Curated	220.386,85	+42.70%	-67.61%	---	10.2K	48.3K
5 Mutant Ape Yacht Club	194.957,81	+17.97%	-23.91%	16.36	11.4K	17.8K
6 The Sandbox	142.494,98	-22.88%	+27.94%	3.57	18.6K	104.2K

Come prima operazione è stato effettuato uno scraping dei diversi dati riguardanti le collezioni NFT.
Lo scraping viene effettuato sul sito https://opensea.io/, in particolare è stato fatto lo scraping sulla tabella presente all'interno della sezione "Stats" selezionando come intervallo temporale "All time".

All'interno di questa pagina, ad ogni collection è associato il volume, la percentuale di variazione del volume nelle ultime 24 ore e negli ultimi 7 giorni, il prezzo più basso, il numero di persone che possiedono un NFT della collection e il numero di item che fanno parte della collezione.

Lo scraping è stato effettuato usando Selenium tramite il web driver di Chrome, ovvero Chromium. Per salvare i dati scaricati tramite questa tecnica si è deciso di utilizzare MongoDB che permette di creare database documentali.

```
i = 0

while True:

    bot = OpenSeaBot()

    time.sleep(10)

    currentNFTS = bot.scrapePage(i)

    dataframe =  pd.DataFrame(currentNFTS, columns=["Collection", "Volume", "Floor
Price", "Owners", "Items"])

    dict = dataframe.to_dict(orient = "record")

    collection.insert_many(dict)

    bot.webdriver.quit()

    i += 1
```

È stato racchiuso il codice in un `while True` poiché non si era a conoscenza del numero di collezioni disponibili per lo scraping. L'idea era quella di prendere un buon numero di progetti NFT. Lo scraper è riuscito a raccogliere le prime 3900 collezioni ordinate per volume, dopodichè il sito non ha più caricato collezioni quindi è stato interrotto il programma tramite `KeyboardInterrupt`.

L'istruzione `bot = OpenSeaBot()` si occupa di creare un'istanza del webDriver di Chromium e quindi aprirà il browser e si posizionerà sulla pagina sulla quale dovranno essere presi i dati, ovvero https://opensea.io/rankings?sortBy=total_volume.

Tramite l'istruzione `currentNFTS = bot.scrapePage(i)` il webDriver si posizionerà sulla i-esima pagina della classifica e effettuerà lo scraping completo delle collezioni. La funzione scrapePage restituirà una lista con dentro tutte le collezioni raccolte.

Dopodichè tale lista viene trasformata in DataFrame con l'istruzione `dataframe = pd.DataFrame(currentNFTS, columns=["Collection", "Volume", "Floor Price", "Owners", "Items"])` e successivamente viene ricavato un dictionary tramite `dict = dataframe.to_dict(orient = "record"). Non` sono state considerate quindi le colonne relative alla percentuale di variazione del volume nelle ultime 24 ore (24h%) e negli ultimi 7 giorni (7d%). La prima è stata esclusa dal momento che considera solamente le variazioni di volume nelle ultime 24 ore e dal momento che la nostra ricerca si focalizza sui tweet di una settimana, quindi non sarebbe stata consistente. La seconda, invece, nonostante mappi un valore settimanale, non è stata presa in considerazione poiché, nella nostra analisi, non si vanno a prendere **tutti** i tweets dell'ultima settimana, ma solamente 1000 tweets (circa) per collection che possono essere raggiunti in un intervallo di tempo variabile, non necessariamente nell'arco di una settimana.

Finalmente tramite l'istruzione `collection.insert_many(dict)` sono inserite tutte le collezioni prelevate all'interno della collezione "nft" in MongoDB.

Tramite l'istruzione `bot.webdriver.quit()` è stata chiusa l'applicazione Chromium: questo per evitare un crash del browser dovuto ad una esecuzione continuativa.

Twitter

Successivamente sono stati presi in considerazione i tweet relativi ad ogni collezione NFT.

Sfruttando i vantaggi forniti dalle API v1.1 e v2 sono stati scaricati i tweet che, all'interno della sezione full_text o retweeted_status.full_text o all'interno dell'url allegato al tweet, presentassero il nome della collection in questione.

Inizialmente sono state create 2 funzioni principali:

1. `fromTweetsToDF(tweets)`: permette di salvare i tweet scaricati all'interno di un dataframe
2. `getTweets(query)`: funzione vera e propria che permette di scaricare i tweet

Tramite l'istruzione `dataframe = getTweets(document['Collection'])` sono ricevuti i tweets relativi alla collection scaricati tramite le API v2 di Twitter.

```
status = api.get_status(row['id'], tweet_mode = "extended")

res = collectionTweets.insert_one(status._json)
```

I tweets sono stati scaricati nella modalità "extended" e sono stati inseriti uno ad uno nella collection su MongoDB. Sono state salvate tutte le informazioni di questi tweet con particolare attenzione a quelle feature che potessero essere utili per fornire delle misure di popolarità (come numero di retweet, data di creazione del tweet, ecc.).

```
try:
     dataframe = getTweets(document['Collection'])
     for index, row in dataframe.iterrows():
         try:
             status = api.get_status(row['id'],  tweet_mode = "extended")
             res = collectionTweets.insert_one(status._json)
         except Exception as e:
             print(e)
 except Exception as e:
```

```
print(e)

if str(e) ==  "429 Too Many Requests":

    print("TIMEOUT 15 minuti")

    time.sleep(15*60)
```

I due try servono per intercettare eventuali errori che emergono dalla richiesta dei tweets: il primo try (più esterno) serve per trovare gli errori di tipo “Nonetype” oppure gestire i timeout imposti da Twitter "429 Too Many Requests"; il secondo try (più interno) serve per gestire i tweets di una collezione di NFT che danno errore e passare al tweet successivo (così facendo non si passa direttamente alla collezione successiva, ma semplicemente al tweet successivo).

ANALISI ESPLORATIVA

Si è passati poi ad una fase di analisi esplorativa dei dati: sono stati analizzati sia alcuni aspetti dei dati grezzi che alcuni aspetti dei dati appartenenti al dataset finale integrato.

Query

Di seguito si mostrano le query create, realizzate tramite *Robo3T*:

1. **Contare quanti tweet sono duplicati (ogni istanza va contata)**

```
db.getCollection('tweets').aggregate([
  { "$group": {
    "_id": { "id_str": "$id_str"},
    "dups": { "$addToSet": "$_id" },
    "count": { "$sum": 1 }
  }},
  { "$match": {
    "count": { "$gt": 1 }
  }},
  {"$group": {"_id": 0,
      "somma": {"$sum": "$count"}}}
])
```

2. **Contare le istanze singole che generano uno o più duplicati**

```
db.getCollection('tweets').aggregate([
  { "$group": {
    "_id": { "id_str": "$id_str"},
    "dups": { "$addToSet": "$_id" },
    "count": { "$sum": 1 }
  }},
  { "$match": {
    "count": { "$gt": 1 }
  }},
  {"$count": "dups"}
```

```
])
```

Di conseguenza, per sapere quanti tweets duplicati sono da eliminare, è necessario fare la sottrazione tra i risultati della prima query e della seconda, ovvero 17076 - 7410; quindi, i duplicati che vengono eliminati, sono 9666.

3. **Contare il numero di collezioni che hanno floor price pari a null**

```
db.getCollection('nft_tweets').find({'Floor Price': NaN}).count()
```

4. **Contare il numero di tweet con più di 100000 retweet**

```
db.getCollection('tweets').find({"retweet_count": {$gt: 100000}}).count()
```

5. **Contare il numero di collection che hanno associati 0 tweet**

```
db.getCollection('nft_tweets').find({"TweetsNum": 0}).count()
```

6. **Contare e definire il nome delle collection i cui tweet sono stati salvati con GridFS**

```
db.getCollection('nft_tweets').find({"IsGridFile": true})
```

7. **Selezionare e contare il numero di collezioni NFT con volume maggiore di 15000 e numero di owners minore di 2000**

```
db.getCollection('nft_tweets').find({$and: [{"Volume": {$gt: 15000}},{"Owners": {$lt: 2000}}]})
```

8. **Contare le collezioni con floor price “basso” (< 0.2 ethereum) e un numero alto di tweet (>1500)**

```
db.getCollection('nft_tweets').find({$and: [{"Floor Price": {$lt: 0.2}}, {"TweetsNum": {$gt: 1500}}]}).count()
```

Risultati

Vengono ora presentati i risultati prodotti dalle interrogazioni:

1. 17076
2. 7410
3. 202
4. 21
5. 614
6. 5
   Doodles, The Sandbox, Decentraland, NFT Art, Binance

7. 5
   Bored Ape Chemistry Club, Cryptovoxels, Sneaky Vampire Syndicate, Gutter Cat Gang, Damien Hirst - The Currency.
8. 5

MODELLAZIONE/CLEANING/INTEGRAZIONE

Modellazione

Ai fini del nostro progetto si è deciso di lavorare con un modello documentale, in particolare con il modello non relazionale *MongoDB*.

È stata fatta questa scelta, preferendo questo modello a quello relazionale, in quanto l'uso principale del database dovrebbe essere quello di lettura dei dati: non sarebbe stato conveniente perciò adottare il modello relazionale perché le query di lettura sarebbero potute essere poco performanti.

Un altro aspetto vantaggioso dei modelli NoSQL è che si tratta di modelli schema free, quindi non richiedono agli sviluppatori di assumere impegni iniziali nei confronti dei modelli di dati.
Inoltre questi modelli garantiscono scalabilità orizzontale e leggerezza computazionale; al contrario di un modello relazionale (dove i dati vengono scritti in un solo posto e dove esiste una e una sola tabella), un modello documentale spinge lo sviluppatore a replicare il dato in diversi posti, in modo che la lettura possa essere eseguita considerando un solo modello dati, senza quindi costrutti simili alla JOIN.

Utilizzare un modello documentale è stata ritenuta perciò la scelta più opportuna.

Cleaning & Data Quality

Durante il processo di creazione del database sono stati fatti degli assessment di qualità, in modo tale da migliorare la qualità dei dati all'interno del dataset.

1. **Errors from data entry / Completezza / Quantità**

Durante la fase di ricerca dei tweets contenenti i nomi delle collezioni NFT sono stati rilevati due tipi di errore: errore di tipo "None Type", il quale sta ad indicare che per quella collezione sono stati trovati 0 tweet; errore di tipo "Bad Request", il quale sta ad indicare che la stringa ricercata non permette di trovare risultati.

Si è deciso, perciò, di portare un miglioramento della qualità del dataset aumentando il numero di collezioni che, quando ricercate, portano dei risultati: sono stati eliminati i caratteri speciali (quindi i caratteri non alfanumerici) dalle stringhe contenenti i nomi delle collezioni NFT.

```
q = ''.join(e for e in query if e.isalnum())
```

Inoltre, sono stati eliminati gli spazi nelle stringhe, in modo da effettuare una ricerca che avesse come focus la selezione dei tweets con l'hashtag relativo al nome (per intero) della collezione cercata.

```
q = q.replace("  ", ' ')
```

Così facendo è stata migliorata sia la completezza dei risultati (vengono cercati i tweet anche delle collezioni che senza manipolazione sarebbero risultate prive di tweets) che la pertinenza con gli standard di ricerca di Twitter (facendo una ricerca hashtag-based).

2. Ridondanza

Nella creazione della collection contenente tutti i tweets trovati nelle varie ricerche è stato necessario, una volta raccolti tutti i tweets, controllare la qualità dei risultati trovati. In particolare sono stati trovati dei tweets duplicati che sono stati eliminati per evitare ridondanza. Di seguito il codice:

```
db = MongoClient()['DatMan']
duplicates = []
cursor = db.tweets.aggregate([
  { "$group": {
    "_id": { "id_str": "$id_str"},
    "dups": { "$addToSet": "$_id" },
    "count": { "$sum": 1 }
  }},
  { "$match": {
    "count": { "$gt": 1 }
  }}
],
allowDiskUse = True
)
for doc in cursor:
    del doc['dups'][0]
    for dupId in doc['dups']:
        duplicates.append(dupId)
print(duplicates)
db.tweets.delete_many({"_id":{"$in":duplicates}})
```

3. Usability - NFT

Per quanto riguarda la fase di scraping è stata necessaria la modifica di alcune variabili, in modo da migliorare la comprensibilità e l'usabilità dei dati importati. In particolare, sono state effettuate le seguenti modifiche:

- trasformati i tipi delle variabili uniformandoli al tipo (appunto) necessario per quella variabile specifica (nomi delle collection in String, valori numerici in numeric ecc.)
- questa operazione ha portato a dover rendere coerenti i valori delle variabili con i tipi delle variabili: ad esempio, all'interno dei record relativi al volume, è stato sostituito il punto, usato per dividere l'unità delle centinaia dalla prima unità delle migliaia, con una stringa vuota, rendendo il record compatibile con il tipo del dato associato al volume, eccetera.

```
nftsDataframe = nftsDataframe.astype({"Collection": str})
nftsDataframe["Volume"] = nftsDataframe["Volume"].map(
    lambda x: x.replace('.', ''))
nftsDataframe["Volume"] = nftsDataframe["Volume"].map(
    lambda x: x.replace(',', '.'))
nftsDataframe["Volume"] = pd.to_numeric(nftsDataframe["Volume"])
nftsDataframe["Floor Price"] = nftsDataframe["Floor Price"].map(
    lambda x: '' if x == '---' else x)
nftsDataframe["Floor Price"] = nftsDataframe["Floor Price"].map(
    lambda x: '0.01' if x == '< 0.01' else x)
nftsDataframe["Floor Price"] = pd.to_numeric(nftsDataframe["Floor Price"])
nftsDataframe["Items"] = nftsDataframe["Items"].map(
    lambda x: x.replace('.', ''))
nftsDataframe["Items"] = pd.to_numeric(nftsDataframe["Items"])
#Owners ha un record di tipo String, quindi viene cambiato il tipo in Double
nftsDataframe['Owners'] = nftsDataframe['Owners'].astype(float)
nftsDataframe["Owners"] = nftsDataframe["Owners"].map(lambda x: int(x * 1000))
```

4. Usability - Numero tweets e isGrid

All'interno del database finale sono state inserite due colonne per aumentare l'usabilità e la chiarezza di alcuni aspetti del dataset.

```
document['TweetsNum'] = len(tweets)
```

La prima colonna serve per contare il numero di tweets raccolti per ogni collezione, in modo da migliorare la usability: in questo modo l'utente non deve contare a mano o scorrere tutti i tweets per saperlo.

```
document['IsGridFile'] = isGrid
```

La seconda colonna, invece, serve per specificare se i tweet per quella collezione NFT siano stati salvati utilizzando GridFS (se la dimensione del documento supera 16MB) oppure no.

Integrazione

Per effettuare l'integrazione, per ogni collezione, è stato creato un documento dove vengono definiti gli attributi Collection, Volume, Floor Price, Owners e Items.

Successivamente è stato necessario includere i tweets relativi al progetto NFT, cercandoli all'interno della collection "tweets".

```
q = ''.join(e for e in record['Collection'] if e.isalnum() or e==" ")

q = q.replace(" ", '\s?')

regx = re.compile(q, re.IGNORECASE)

tweetsCursor = db.tweets.find({"$or": [{"full_text": regx},
                          {"retweeted_status.full_text": regx},
             {"entities.urls": {"$elemMatch":{"expanded_url": regx}}}]})
```

Per fare ciò sono state utilizzate le RegEx, in modo da evitare problemi relativi a spaziature all'interno del nome della collezione o relativi al fatto che i nomi presentano una sequenza di caratteri maiuscoli e/o minuscoli.

```
tweetsCursor = db.tweets.find({"$or": [{"full_text": regx},

                          {"retweeted_status.full_text": regx},

             {"entities.urls": {"$elemMatch":{"expanded_url": regx}}}]})
```

Per ogni collezione viene interrogato MongoDB con lo scopo di cercare i tweets che corrispondono nel loro testo o nell'URL al pattern proposto dalla RegEx. Questi tweet vengono messi nel cursor.

```
isGrid = False

tweets = list(tweetsCursor)

document['TweetsNum'] = len(tweets)

document['Tweets'] = tweets

try:
```

```
    document['IsGridFile'] = isGrid
    collectionNFTGrid.insert_one(document)
except:
    print("File troppo grosso")
    isGrid = True
    document['IsGridFile'] = isGrid
    s = json_util.dumps(tweets)
    res = fs.put(s, encoding = "utf-8")
    document['Tweets'] = res
    collectionNFTGrid.insert_one(document)
```

Dal cursor vengono convertiti in list e messi nel document, nel campo “Tweets”, che viene poi caricato all’interno della base di dati, nella collezione “nft_tweets”.
Nel caso la lista di tweets fosse eccessivamente grande, il documento viene caricato tramite FSGrid.
Per accedere ai tweet salvati in questi documenti si possono eseguire due operazioni, una get e una read che permettono di mostrare i tweet con le relative informazioni coerentemente con i tweet che non superano la dimensione di 16MB.

MEMORIZZAZIONE SUL DBMS

È stato creato un database su MongoDB denominato DatMan. All'interno di questo DB sono state create 3 collezioni distinte, ovvero: nft, tweets e nft_tweets.

Durante il processo di scraping sono stati collezionati i progetti NFT raccolti all'interno della collezione "nft", per un totale di 3900 documenti. Le collezioni sono state salvate nella collection di MongoDB denominata "nft".

Dopodiché, partendo dalle collezioni acquisite, sono state interrogate le API di Twitter per ottenere i tweets legati alle collezioni. Sono stati ottenuti quindi 300.000 tweets per 1532 collezioni.
A causa dei timeout imposti dalle API e dei limiti dell'account Elevated questo processo ha impiegato 3 giorni di elaborazione.

I tweets sono stati salvati nella collezione "tweets".

Infine, avendo tutti dati a disposizione è stata effettuata l'integrazione dei due dataset associando le collezioni NFT ai tweets raccolti.
L'integrazione è stata effettuata tramite un matching tra il nome della collezione e il testo del tweet o titolo del sito citato nel tweet.
Per eseguire questo matching sono state usate regular expression.

Quindi, ogni documento al suo interno ha i campi della collection nft, più un campo tweets che contiene un array di tweets associati al progetto NFT.
Non è stato però possibile archiviare tutti i documenti associati in mongoDB, poiché alcuni richiedevano un documento di dimensione maggiore di 16MB, eccedendo il limite imposto dalla piattaforma.
Di conseguenza l'archiviazione dei tweet per questi documenti è stata effettuata tramite GridFS. GridFS è una specifica per memorizzare e recuperare file che superano il limite di dimensione del documento BSON di 16 MB. Invece di memorizzare un file in un singolo documento, GridFS divide il file in parti, o chunks, e memorizza ogni chunk come un documento separato.

In questi documenti, invece di esserci un array di tweets, ci sarà il riferimento che permetterà di recuperare i tweets salvati tramite GridFS.
Tutti i documenti sono stati salvati nella collezione "nft_tweets"

Il database finale, quindi, è costituito dalle seguenti colonne:

- **_id**: chiave univoca associata ad ogni documento
- **Collection**: nome della collezione NFT

- **Volume**: quantità monetaria scambiata relativa a una collection
- **Floor Price**: è il prezzo più basso per gli articoli della collezione
- **Owners**: numero di persone che possiedono o hanno posseduto almeno un oggetto di quella collezione NFT
- **Items**: numero di oggetti che costituiscono una collection
- **TweetsNum**: numero di tweet associato alla ricerca del nome della collection su Twitter
- **Tweets**: array di oggetti contenente i tweet con le specifiche informazioni (id tweet, full_text, retweeted_status, retweet_count, favourite_count, ecc.)
- **IsGridFile**: segnala se i tweets relativi alla collezione NFT sono stati salvati usando GridFS (valore True) oppure no (valore False).

CONCLUSIONI E SVILUPPI FUTURI

In conclusione, è stato creato, in primis, un dataset che raccogliesse, dal sito opensea, le informazioni riguardanti 3900 collezioni NFT; poi è stato creato un dataset che, per le prime 1532 collezioni, andasse a trovare circa 1000 tweets (dove fossero disponibili), considerando i tweet dell'ultima settimana, prendendoli in formato esteso e prendendo tutte le informazioni rese disponibili dalle API v1.1 e v2 per quel tweet. Infine, è stata fatta l'integrazione dei due dataset per crearne uno solo a partire da questi due: l'integrazione è stata fatta tramite delle RegEx e ha permesso di creare una colonna relativa ai tweets che all'interno contiene un array di oggetti, ovvero tutti i tweets per quella collezione con tutte le informazioni relative al tweet. Sul dataset, come richiesto, sono state effettuate operazioni di data exploration, data quality, data integration (appunto) e data modeling.

Possibili sviluppi futuri potrebbero essere quelli di aggiungere più collezioni NFT, effettuando uno scraping che possa scaricare tutte le collection presenti (dovrebbero essere circa 6000) e tutte le informazioni presenti relative alle collection (esempio 24h%, 7d%,); un altro sviluppo potrebbe consistere nell'aggiungere più tweet oltre ai circa 300000 scaricati fino a riuscire ad ricercare tweet per ogni collezione.
Inoltre si potrebbero effettuare le ricerche su Twitter considerando anche i nomi delle collection con gli spazi, evidenziando quindi anche i tweet che presentano all'interno del testo il nome della collection (questa ricerca potrebbe essere fuorviante in quanto potrebbero essere trovati tweet non inerenti all'ambito di ricerca per alcuni nomi di alcune collezioni).
Infine, facendo uso delle API di Twitter che hanno l'accesso massimo, ovvero l'academic research, si potrebbero considerare i tweet in un range temporale più ampio, considerando quelli pubblicati negli ultimi 30 giorni (e non negli ultimi 7 come succede con l'accesso di tipo elevated).

RIFERIMENTI

- https://www.money.it/NFT-cosa-sono-come-funzionano-come-investire
- https://n26.com/it-it/blog/nft-cosa-sono
- https://it.wikipedia.org/wiki/Non-fungible_token
- https://blog.mia-platform.eu/it/modello-relazionale-o-modello-documentale-quale-database-adottare
- https://docs.mongodb.com/manual/core/gridfs/#:~:text=GridFS%20is%20a%20specification%20for,size%20limit%20of%2016%20MB.&text=Instead%20of%20storing%20a%20file,chunk%20as%20a%20separate%20document