UCLouvain
FUCaM MONS

LOUVAIN
School of Management

Louvain School of Management

La collaboration des internautes peut-elle influencer le marché boursier ou est-ce ce dernier qui influence les internautes ?

Annexes

Auteur : Hugo Dendievel
Promoteur : François Fouss
Année académique 2021-2022
Master d'Ingénieur de gestion - Business Analytics

Liste des annexes

Annexe A

Codes

Cette section contient les différents codes Python réalisés et utilisés au sein de ce mémoire.

A.1 Extraction des données

A.1.1 Reddit

Code A.1 – Reddit - Extraction des données

```
import time
from pmaw import PushshiftAPI
import pandas as pd
import datetime as dt

class submissions:

    @staticmethod
    def submissions_pushshift_pmaw(subreddit, start=None,\
          end=None, limit=100, extra_query=""):

        utc_offset = 28800
        now = int(time.time())
        start = max(int(start) + utc_offset if start else 0, 0)
        end = min(int(end) if end else now, now) + utc_offset

        api = PushshiftAPI()

        posts = api.search_submissions(subreddit=subreddit, \
  limit=limit, before=end, after=start, mem_safe=True)

        post_list = [post for post in posts]
```

```
        return post_list

    @staticmethod
    def reddit_extract(subreddit, start=None, end=None, \
              limit=None):

        utc_offset = 28800
        now = int(time.time())
        start = max(int(start) + utc_offset if start else 0, 0)
        end = min(int(end) if end else now, now) + utc_offset

        api = PushshiftAPI()

        posts = api.search_submissions(subreddit=subreddit, \
limit=limit, before=end, after=start, mem_safe=True)

        posts_df = pd.DataFrame(posts)

        return posts_df

    @staticmethod
    def export(subreddit, start=None, end=None, limit=None):

        posts_df = self.reddit_extract(subreddit, start, end, \
                   limit)

        if start != None:
            date = dt.datetime.utcfromtimestamp(start) \
            .strftime('%Y-%m-%d')
        else:
            date = "Nodate"
        posts_df.to_csv(subreddit+"_"+date+".csv", index=False)

    @staticmethod
    def get_titles(subreddit, start=None, end=None, \
           limit=100, extra_query=""):
        title = list()
        for post in self.submissions_pushshift_pmaw(subreddit, \
         start, end, limit, extra_query):
            title.append(post["title"])
        return title

    @staticmethod
    def get_text(subreddit, start=None, end=None, \
    limit=100, extra_query=""):
        text = list()
        for post in self.submissions_pushshift_pmaw(subreddit, \
```

```
                start, end, limit, extra_query):
            try:
                text.append(post["selftext"])
            except KeyError:
                text.append("NOT AVAILABLE")
        return text
```

Code A.2 – Reddit - Sélection des actions

```
import json
import re
from tqdm import tqdm
from collections import Counter
from itertools import chain
from typing import Set
import pandas as pd

import datetime as dt
import time
from dateutil.relativedelta import *

from submissions import submissions

class reddit_tickers_count:

    def __init__(self,subreddits,month_count=1):

        self.subreddits = subreddits
        self.month_count = month_count

        stop_words = ["I", "ME", "MY", "MYSELF", "WE", "OUR",
        "OURS", "OURSELVES", "YOU", "YOU'RE", "YOU'VE",
        "YOU'LL", "YOU'D", "YOUR", "YOURS", "YOURSELF",
         "YOURSELVES", "HE", "HIM", "HIS", "HIMSELF",
         "SHE", "SHE'S", "HER", "HERS", "HERSELF", "IT",
         "IT'S", "ITS", "ITSELF", "THEY", "THEM", "THEIR",
         "THEIRS", "THEMSELVES", "WHAT", "WHICH",
         "WHO", "WHOM", "THIS", "THAT", "THAT'LL",
         "THESE", "THOSE", "AM", "IS", "ARE", "WAS",
         "WERE", "BE", "BEEN", "BEING", "HAVE", "HAS",
         "HAD", "HAVING", "DO", "DOES", "DID", "DOING",
         "A", "AN", "THE", "AND", "BUT", "IF", "OR", "BECAUSE",
         "AS", "UNTIL", "WHILE", "OF", "AT", "BY", "FOR","WITH",
         "ABOUT", "AGAINST", "BETWEEN", "INTO", "THROUGH",
         "DURING", "BEFORE", "AFTER", "ABOVE", "BELOW",
         "TO", "FROM","UP", "DOWN", "IN", "OUT", "ON", "OFF",
         "OVER", "UNDER", "AGAIN", "FURTHER", "THEN",
         "ONCE", "HERE", "THERE", "WHEN","WHERE",
```

```
        "WHY", "HOW", "ALL", "ANY", "BOTH", "EACH", "FEW",
        "MORE", "MOST", "OTHER", "SOME", "SUCH", "NO",
        "NOR", "NOT","ONLY", "OWN", "SAME", "SO", "THAN",
        "TOO", "VERY", "S", "T", "CAN", "WILL", "JUST", "DON",
        "DON'T", "SHOULD", "SHOULD'VE","NOW", "D", "LL",
        "M", "O", "RE", "VE", "Y", "AIN", "AREN", "AREN'T",
        "COULDN", "COULDN'T", "DIDN", "DIDN'T", "DOESN",
        "DOESN'T", "HADN", "HADN'T", "HASN", "HASN'T",
        "HAVEN", "HAVEN'T", "ISN", "ISN'T", "MA", "MIGHTN",
        "MIGHTN'T", "MUST", "MUSTN", "MUSTN'T", "NEEDN",
        "NEEDN'T", "SHAN", "SHAN'T", "SHOULDN",
        "SHOULDN'T", "WASN", "WASN'T", "WEREN",
        "WEREN'T","WON", "WON'T", "WOULDN", "WOULDN'T"]

    block_words = ["DIP", "", "$", "RH", "YOLO", "PORN",
    "BEST", "MOON", "HOLD", "FAKE", "WISH", "USD",
    "EV", "MARK", "RELAX", "LOL", "LMAO", "LMFAO",
    "EPS", "DCF", "NYSE", "FTSE", "APE", "CEO",
    "CTO", "FUD", "DD", "AM", "PM", "FDD", "EDIT",
    "TA", "UK", "IPO","GO","GME","AMC","TD","AI","ITM"]

    with open('path_to_json') as f:
        tickers = set(json.load(f))

    exclude = set(stop_words + block_words)

    self.keep_tickers = tickers - exclude

    self.start_date = dt.datetime(2021, 10, 1)
    self.lastnMonth = self.start_date \
    +  relativedelta(months=-month_count)
    self.lastnMonth_str = self.lastnMonth \
    .strftime('%d/%m/%Y')
    self.lastnMonth_unix = time.mktime \
    (self.lastnMonth.timetuple())

def get_date(self):
    return self.lastnMonth_unix

def get_tickers(self):
    return self.keep_tickers

def extract_ticker(self, text: str, pattern: \
str = r'(?<=\$)[A-Za-z]+|[A-Z]{2,}') -> Set[str]:
    ticks = set(re.findall(pattern, str(text)))
    return ticks & self.keep_tickers
```

```
def get_posts(self):

    titles = dict()
    text = dict()

    submission = submissions()

    for subreddit in self.subreddits:
        print("Working with "+subreddit)
        titles[subreddit]= submission.get_titles(subreddit, \
         start = self.lastnMonth_unix,limit=None)
        text[subreddit]= submission.get_text(subreddit, \
        start = self.lastnMonth_unix,limit = None)

    return titles, text

def get_posts_csv(self):

    titles = dict()
    text = dict()

    for subreddit in self.subreddits:
        print("Working with "+subreddit)
        sub = pd.read_csv("../Data/"+subreddit+ \
        "_last12months.csv",low_memory=False)
        titles[subreddit]= sub[sub["created_utc"] > \
        self.lastnMonth_unix]["title"].to_list()
        text[subreddit]= sub[sub["created_utc"] > \
        self.lastnMonth_unix]["selftext"].to_list()


    return titles, text

def get_data(self,csv=False):
    print("Extraction of most spoken stocks")
    print("Start date: "+self.lastnMonth_str)

    if csv:
        title_dict, text_dict = self.get_posts_csv()
    else:
        title_dict, text_dict = self.get_posts()

    title_array = list()
    text_array = list()

    for sub in title_dict.keys():
        title_array += title_dict[sub]
```

```
            text_array += text_dict[sub]

        title_text_array = {"Title":title_array, \
        "Text":text_array}

        df_posts=pd.DataFrame(title_text_array)

        # Extract tickers from titles & count them
        print("Extracting tickers from titles...")
        tickers = df_posts["Title"].apply(self.extract_ticker)
        counts = Counter(chain.from_iterable(tickers))

        # Extract tickers from text
        print("Extracting tickers from text...")
        tickers_text = df_posts["Text"] \
        .apply(self.extract_ticker)
        counts_text = Counter(chain.from_iterable(tickers_text))

        # Create DataFrame of just mentions & remove
        # any occurring less than 3 or less
        df_tick = pd.DataFrame(counts.items(), \
        columns=['Ticker', 'Mentions in title'])
        df_tick = df_tick[df_tick['Mentions in title'] > 3]
        self.df_tick = df_tick.sort_values(by= \
        ['Mentions in title'], ascending=False)

        df_tick2 = pd.DataFrame(counts_text.items(), \
        columns=['Ticker', 'Mentions in text'])
        df_tick2 = df_tick2[df_tick2['Mentions in text'] > 3]
        self.df_tick2 = df_tick2.sort_values(by= \
        ['Mentions in text'], ascending=False)

        output_path = '../Output/' + \
         f'{dt.date.today()}_title_stocks.csv'
        output_path2 = '../Output/' + \
         f'{dt.date.today()}_text_stocks.csv'

        self.df_tick.to_csv(output_path, index=False)
        self.df_tick2.to_csv(output_path2,index=False)
        print("Data extracted and saved !")

        self.get_stats()

    def get_stats(self):

        print("Stats for the last "+str(self.month_count)+\
        " month(s):\n" )
```

```
        print("Top 10 tickers in titles:")
        print(self.df_tick.head(10),'\n')

        print("Total mentions: "+str(sum(self.df_tick.head(10) \
        ["Mentions in title"]))+"\n")

        print("Top 10 tickers in text:")
        print(self.df_tick2.head(10),'\n')

        print("Total mentions: "+str(sum(self.df_tick2.head(10) \
        ["Mentions in text"]))+"\n")

if __name__ == '__main__':
    #Subreddits
    subreddits=["wallstreetbets","investing","finance","stocks"]

    #Get most spoken stocks from pushshift API
    #ticker = reddit_tickers_count(["finance"],1)
    #ticker.get_data(csv=False)

    #Get most spoken stocks from csv
    ticker = reddit_tickers_count(subreddits,12)
    ticker.get_data(csv=True)
```

A.1.2 Twitter

Code A.3 – Twint - Extraction des données

```
import twint
import nest_asyncio
nest_asyncio.apply()
import pandas as pd
import os
import datetime as dt

scrapper = twint.Config()
scrapper.Hide_output = True

top_tickers_title = pd.read_csv("path_to_csv")
cashtags_title = top_tickers_title.head(10)["Ticker"].to_list()

top_tickers_text = pd.read_csv("path_to_csv")
cashtags_text = top_tickers_text.head(10)["Ticker"].to_list()

offset = 7200
```

```
start_date = dt.datetime(2020, 10, 1)
date = start_date.timestamp() - offset
next_date = start_date.timestamp()
now = dt.datetime.now().timestamp()

cashtags = list(set(cashtags_text) | set(cashtags_title))

while date <= now:
    for cashtag in cashtags:
        try:
            os.mkdir("../Data/"+cashtag)
        except OSError as error:
            pass

        scrapper.Search = "\%24"+cashtag
        scrapper.Since = dt.datetime.fromtimestamp(date) \
        .strftime("%Y-%m-%d %H:%M:%S")

        next_date = dt.datetime.fromtimestamp(date) \
         + dt.timedelta(days=1)

        scrapper.Until = next_date \
        .strftime("%Y-%m-%d %H:%M:%S")

        scrapper.Store_csv = True
        scrapper.Output = "../Data/"+cashtag+"/"+cashtag+"_"+ \
        dt.datetime.fromtimestamp(date+offset) \
        .strftime("%d-%m-%Y")+".csv"

        twint.run.Search(scrapper)

    next_date = next_date.timestamp()
    date = next_date
```

Code A.4 – Twint - Construction de l'ensemble de données

```
import os
import pandas as pd

top_tickers_title = pd.read_csv("path_to_csv")
cashtags_title = top_tickers_title.head(10)["Ticker"].to_list()

top_tickers_text = pd.read_csv("path_to_csv")
cashtags_text = top_tickers_text.head(10)["Ticker"].to_list()

cashtags = list(set(cashtags_text) | set(cashtags_title))
cashtags_directories = ['../Data/'+x for x in cashtags]
```

```
def load_files(directory,filenames):
  for filename in filenames:
    yield pd.read_csv(directory+'/'+filename)


for directory in cashtags_directories:
    data_files = os.listdir(directory)
    if '.ipynb_checkpoints' in data_files:
        data_files.remove('.ipynb_checkpoints')
    data = pd.concat(load_files(directory,data_files))
    data.to_csv('../Output/'+directory[8:]+'_last12months.csv' \
    ,index=False)
```

A.1.3 Yahoo Finance

Code A.5 – yfinance - Extraction des données

```
import pandas as pd
import yfinance as yf

tickers = ["AAPL","AMD","BB","CLOV",
           "NAKD","NIO","NOK","PLTR","RKT",
           "SNDL","SPCE","SPY","TSLA"]

start_date = '2020-10-02' #To start on 2020-10-01
end_date = '2021-10-24'

for tick in tickers:
    if tick == "NAKD":
       ticker = yf.Ticker("CENN")
    else:
        ticker = yf.Ticker(tick)

    stock = ticker.history(start=start_date, end=end_date)
    stock.to_csv("path_to_csv")
```

A.2 Nettoyage et préparation des données

A.2.1 Reddit

Code A.6 – Reddit - Classification des messages et commentaires par action

```
import pandas as pd
```

```
tickers = [
    "AAPL",
    "AMD",
    "BB",
    "CLOV",
    "NAKD",
    "NIO",
    "NOK",
    "PLTR",
    "RKT",
    "SNDL",
    "SPCE",
    "SPY",
    "TSLA",
]

tickers_name = {
    "AAPL": ["aapl", "apple"],
    "AMD": ["amd", "advanced micro devices"],
    "BB": ["bb", "blackberry"],
    "CLOV": ["clov", "clover health"],
    "NAKD": ["nakd", "cenn", "naked brand", "cenntro electric group"],
    "NIO": ["nio"],
    "NOK": ["nok", "nokia"],
    "PLTR": ["pltr", "palantir technologies"],
    "RKT": ["rkt", "rocket companies"],
    "SNDL": ["sndl", "sundial growers"],
    "SPCE": ["spce", "virgin galactic"],
    "SPY": ["spy", "spdr s&p 500 trust etf", "s&p 500"],
    "TSLA": ["tsla", "tesla"],
}

comments = pd.read_csv("comments.csv")
submissions = pd.read_csv("submissions.csv")

# Date filter
final_date = 1635033600
submissions_date = submissions[submissions["created_utc"] <= final_date]
id_list = submissions[submissions["created_utc"] > final_date]["id"].
                                        to_list()
comments_date = comments[~comments["submission_id"].isin(id_list)]


# Tickers filter

for tick in tickers:
    sub = pd.DataFrame()
```

```
    for name in tickers_name[tick]:
        sub = sub.append(
            submissions_date[
                submissions_date["title"].str.contains(name, na=False,
                                                       case=False)
            ],
            ignore_index=True,
        )
    sub = sub.drop_duplicates()
    sub.to_csv("submissions/submissions_" + tick + ".csv")
    id = sub["id"].to_list()
    com = comments_date[comments_date["submission_id"].isin(id)]
    com.to_csv("comments/comments_" + tick + ".csv")
```

Code A.7 – Reddit - Concaténation des messages et commentaires

```
import pandas as pd

tickers = ["AAPL","AMD","BB","CLOV",
           "NAKD","NIO","NOK","PLTR","RKT",
           "SNDL","SPCE","SPY","TSLA"]

for tick in tickers:
    com = pd.read_csv("comments/comments_"+tick+".csv",lineterminator='\
                                        n')[["created_utc","body"]].
                                        rename(columns={"body": "text"})
    sub = pd.read_csv("submissions/submissions_"+tick+".csv",
                                        lineterminator='\n')[["
                                        created_utc","title"]].rename(
                                        columns={"title": "text"})
    df = pd.concat([sub,com],ignore_index=True)
    df.to_csv("final/"+tick+".csv")
```

A.2.2 Twitter

Code A.8 – Twitter - Vérification des dates

```
import pandas as pd
import datetime as dt
import os

tickers = ["AAPL","AMD","BB","CLOV",
           "NAKD","NIO","NOK","PLTR","RKT",
           "SNDL","SPCE","SPY","TSLA"]

dataframes = dict.fromkeys(tickers,)
final_date = dt.datetime.timestamp(dt.datetime(2021,10,24))
```

```
offset = 120*60

for ticker in tickers:
    df = pd.read_csv("path_to_csv", low_memory=False, \
             lineterminator='\n')
    dataframes[ticker] = df
    dataframes[ticker]["created_at"] = pd.to_datetime( \
          dataframes[ticker]["created_at"])
    dataframes[ticker]["created_utc"] = dataframes[ticker] \
    ["created_at"].apply(lambda x: x.value/10**9-offset)

if not os.listdir('path_to_folder'):
    for ticker in tickers:
        dataframes[ticker] = dataframes[ticker].loc \
        [dataframes[ticker]["created_utc"] <= final_date]
        dataframes[ticker].to_csv("path_to_csv")
```

A.2.3 Yahoo Finance

Code A.9 – Yahoo ! Finance - Préparation des données

```
import pandas as pd

tickers = [
    "AAPL",
    "AMD",
    "BB",
    "CLOV",
    "NAKD",
    "NIO",
    "NOK",
    "PLTR",
    "RKT",
    "SNDL",
    "SPCE",
    "SPY",
    "TSLA",
]

for tick in tickers:
    df = pd.read_csv("path_to_csv")
    df["Daily variation"] = df["Close"] - df["Open"]
    df.to_csv("path_to_csv", index=False)
```

A.3 Analyse exploratoire

Code A.10 – Analyse exploratoire des données

```
import pandas as pd
from IPython.display import display
import matplotlib.pyplot as plt
import july

tickers = [
    "AAPL",
    "AMD",
    "BB",
    "CLOV",
    "NAKD",
    "NIO",
    "NOK",
    "PLTR",
    "RKT",
    "SNDL",
    "SPCE",
    "SPY",
    "TSLA",
]

tick = tickers[12]

reddit = "../Reddit/subreddit-comments-dl/dataset/20220306120535/final/"
twitter = "Output/Twitter/"
yahoo = "../YahooFinance/Data/Final/"

# twit = pd.read_csv(
#     twitter + tick + ".csv",
#     low_memory=False,
#     lineterminator="\n",
#     index_col=0,
# )

redd = pd.read_csv(
    reddit + tick + ".csv",
    low_memory=False,
    lineterminator="\n",
    index_col=0,
)

redd["date"] = pd.to_datetime(redd["created_utc"], unit="s").dt.date
redd = redd[redd["created_utc"] <= 1635033600]
```

```
redd["date"] = pd.to_datetime(redd["date"])

yahoo = pd.read_csv(yahoo + tick + ".csv")

# print("Tweets for", tick)
# display(twit)

print("Posts for", tick)
display(redd)

print("Financial informations for", tick)
display(yahoo)

# print("Statistic description for tweets")
# display(twit.describe())

print("Statistic description for tweets")
display(redd.describe())

print("Statistic description for financial infos")
display(yahoo.describe())

# print("Number of tweets")
# print(len(twit))

print("Number of posts")
print(len(redd))

# print("Day of most tweets")
# plt.bar(twit["date"].value_counts()[:10].index, twit["date"].
                                    value_counts()[:10].values)
# plt.xticks(rotation=30)
# plt.title("Most tweets " + tick)
# plt.show()

print("Day of most posts")
plt.bar(
    redd["date"].dt.strftime("%Y-%m-%d").value_counts()[:10].index,
    redd["date"].value_counts()[:10].values,
)
plt.xticks(rotation=30)
plt.title("Most Reddit posts " + tick)
plt.show()

# july.heatmap(
#     twit["date"]
#     .value_counts()
```

```
#     .index,  # Here, we can remove extreme values with a slice
#     twit["date"].value_counts().values,
#     cmap="golden",
#     colorbar=True,
#     title="Tweets " + tick,
# )


july.heatmap(
    redd["date"]
    .value_counts()
    .index,  # Here, we can remove extreme values with a slice
    redd["date"].value_counts().values,
    cmap="golden",
    colorbar=True,
    title="Reddit posts " + tick,
)
```

A.4 Analyse de sentiments

Code A.11 – Analyse de sentiments

```
#####################Dependencies#########################
from transformers import AutoModelForSequenceClassification
from transformers import TFAutoModelForSequenceClassification
from transformers import AutoTokenizer
import transformers
transformers.utils.logging.set_verbosity_error


import numpy as np
from scipy.special import softmax
import shutil
import pandas as pd
import re
import reticker
from nltk.corpus import stopwords
from tqdm import tqdm
tqdm.pandas()
#####################Conditions#########################

twitter = False
reddit = True

export = True
```

```
PT = True
TF = False

####################Preprocessing#########################

def preprocess(text):
    remove_user_http = True
    remove_specialcharacters = False
    remove_tickers = False
    remove_stopwords = False

    new_text = []

    if remove_user_http:
        for t in str(text).split(" "):
            t = '@user' if t.startswith('@') and len(t) > 1 else t
            t = 'http' if t.startswith('http') else t
            new_text.append(t)

    if len(new_text)>0:
        new_text = " ".join(new_text)
    else:
        new_text = text

    if remove_specialcharacters:
        new_text = re.sub('[^A-Za-z]+', ' ', new_text)

    if remove_tickers:
        tickers = reticker.TickerExtractor().extract(new_text)
        tickers = '|'.join(tickers)
        new_text = re.sub(tickers, '', new_text)

    if remove_stopwords:
        pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english
                                                ')) + r')\b\s*')
        new_text = pattern.sub('', new_text)

    return new_text

#################Task and model selection###################

# Tasks: emoji, emotion, hate, irony, offensive, sentiment

shutil.rmtree("cardiffnlp",ignore_errors=True)

task='sentiment'
MODEL = f"cardiffnlp/twitter-roberta-base-{task}"
```

```
tokenizer = AutoTokenizer.from_pretrained(MODEL)

####################Label mapping############################

if task == "sentiment":
    labels = ["negative","neutral","positive"]
elif task == "emotion":
    labels = ["anger","joy","optimism","sadness"]

# labels=[]
# mapping_link = f"https://raw.githubusercontent.com/cardiffnlp/
                                    tweeteval/main/datasets/{task}/
                                    mapping.txt"
# with urllib.request.urlopen(mapping_link) as f:
#     html = f.read().decode('utf-8').split("\n")
#     csvreader = csv.reader(html, delimiter='\t')
# labels = [row[1] for row in csvreader if len(row) > 1]

def ranking(scores):
    ranking = np.argsort(scores)
    ranking = ranking[::-1]

    sentiment = np.where(scores == max(scores))
    l = labels[sentiment[0][0]]
    return l

####################Classifier############################
if PT:
    model = AutoModelForSequenceClassification.from_pretrained(MODEL)
    model.save_pretrained(MODEL)

    def pipeline(text):
        encoded_input = tokenizer(text,padding=True,truncation=True,
                                            max_length=512, verbose=
                                            False, return_tensors='pt')
        output = model(**encoded_input)
        scores = output[0][0].detach().numpy()
        scores = softmax(scores)
        return scores

elif TF:
    model = TFAutoModelForSequenceClassification.from_pretrained(MODEL)
    model.save_pretrained(MODEL)

    def pipeline(text):
        encoded_input = tokenizer(text, return_tensors='tf')
```

```
        output = model(encoded_input)
        scores = output[0][0].numpy()
        scores = softmax(scores)
        return scores


############################################################
#########################Main###############################
############################################################

def main():

     #["AAPL","AMD",
    tickers = ["BB","CLOV",
               "NIO","NOK","PLTR","RKT","SNDL",
               "SPCE","TSLA","SPY","NAKD"]

    for tick in tickers:

        ####################Data importation#####################
        if reddit:
            path = "../Reddit/subreddit-comments-dl/dataset/
                                                20220306120535/final/"+
                                                tick+".csv"
            name = "Reddit"
        elif twitter:
            path = "../Data/Output/Twitter/"+tick+"_last12months_dateOK.
                                                csv"
            name = "Twitter"

        print("Importing "+tick+ " data ("+name+")...")
        df = pd.read_csv(path,low_memory=False,lineterminator='\n')

        if reddit:
            data = df.loc[:, ('created_utc', 'text')]
        elif twitter:
            df["text"] = df["tweet"]
            data = df.loc[:, ('created_utc', 'text')]

        ####################Preprocessing#########################
        print("Preprocessing data...")
        data["text_preprocessed"] = data["text"].progress_apply(
                                            preprocess)

        ####################Classifying##########################
        print("Classifying...")
```

```
        data["scores"] = data["text_preprocessed"].progress_apply(
                                                pipeline)

        ######################Labeling#############################
        print("Labeling results...")
        data["sentiment"] = data["scores"].progress_apply(ranking)

        ######################Exporting#############################
        if export:
            print("Exporting data...")
            if reddit:
                data.to_csv("Reddit/"+tick+"_"+name+"_sentiment.csv",
                                                        index=False)
            elif twitter:
                data.to_csv("Twitter/"+tick+"_"+name+"_sentiment.csv",
                                                        index=False)

if __name__=="__main__":
    main()
```

A.5 Analyses finales

Code A.12 – Test de corrélation entre séries temporelles

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams["figure.dpi"] = 300

tickers = [
    "AAPL",
    "AMD",
    "BB",
    "CLOV",
    "NAKD",
    "NIO",
    "NOK",
    "PLTR",
    "RKT",
    "SNDL",
    "SPCE",
    "SPY",
    "TSLA",
]
```

```
# Create a custom palette
cmap = sns.diverging_palette(250, 15, s=75, l=40, n=9, center="light",
                                    as_cmap=True)

for ticker in tickers:
    df = pd.read_csv(f"../../Sentiment Analysis/Reddit_final/{ticker}.
                                    csv", index_col=0)

    # Compute corr matrix
    matrix = df[["negative", "positive", "neutral", "Daily variation", "
                                    Volume"]].corr(
        method="pearson"
    )
    # Create a mask
    mask = np.triu(np.ones_like(matrix, dtype=bool))

    fig, ax = plt.subplots(figsize=(7, 7))
    sns.heatmap(matrix, mask=mask, cmap=cmap, square=True, annot=True,
                                    fmt=".2f", ax=ax)
    plt.title(f"{ticker} - Correlation matrix on Reddit")
    plt.savefig(f"matrix/{ticker}_Reddit.png", facecolor="white",
                                    transparent=False)
```

Code A.13 – Régression linéaire

```
import pandas as pd
import numpy as np
from IPython.display import display

import warnings

warnings.filterwarnings("ignore")

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
                                    variance_inflation_factor as vif


class LinearRegression:
    def __init__(self, path: str):
        self.tickers = [
            "AAPL",
            "AMD",
            "BB",
            "CLOV",
            "NAKD",
            "NIO",
```

```
            "NOK",
            "PLTR",
            "RKT",
            "SNDL",
            "SPCE",
            "SPY",
            "TSLA",
        ]
        if path.lower() == "reddit":
            self.path = "../Sentiment Analysis/Reddit_final/{}.csv"
        elif path.lower() == "twitter":
            self.path = "../Sentiment Analysis/Twitter_final/{}.csv"

        self.summary = pd.DataFrame(columns=["const", "R_squared"])

    def main(self, x: list, y: str):

        for tick in self.tickers:
            df = pd.read_csv(self.path.format(tick), index_col=0)

            df["positive_log"] = df["positive"].apply(np.log)
            df["negative_log"] = df["negative"].apply(np.log)
            df["neutral_log"] = df["neutral"].apply(np.log)
            df["Volume_log"] = df["Volume"].apply(np.log)

            df = df[np.isfinite(df).all(1)]

            X = df[x].copy()
            Y = df[y]

            # Center data to deal with multicollinearity
            X = X.subtract(X.mean())

            X = sm.add_constant(X)

            # Check for multicollinearity
            collinearity = pd.Series(
                [vif(X.values, i) for i in range(X.shape[1])], index=X.
                                                        columns
            )

            # Considering a VIF of 10 for problematic multicollinearity
            multicol = [collinearity[x[i]] > 10 for i in range(len(x))]
            if any(multicol):
                print(tick, " Multicollinearity!")

            model = sm.OLS(Y, X).fit()
```

```
            params = model.params.round(decimals=3)

            pvalues = model.pvalues.round(decimals=3).to_frame(name=tick
                                                    ).transpose()
            pvalues["R_squared"] = model.rsquared
            pvalues["coef_" + x[0]] = params[x[0]]
            pvalues["coef_" + x[1]] = params[x[1]]
            self.summary = self.summary.append(pvalues)

        self.summary = self.summary.style.set_caption(f"{y} ~ " + x[0] +
                                                "+" + x[1])
        display(self.summary)

    def main_inverse(self, x: list, y: str):

        for tick in self.tickers:
            df = pd.read_csv(self.path.format(tick), index_col=0)

            df["positive_log"] = df["positive"].apply(np.log)
            df["negative_log"] = df["negative"].apply(np.log)
            df["neutral_log"] = df["neutral"].apply(np.log)
            df["Volume_log"] = df["Volume"].apply(np.log)

            df = df[np.isfinite(df).all(1)]

            X = df[x].copy()
            Y = df[y[0]]  # + df[y[1]]

            # Center data to deal with multicollinearity
            X = X.subtract(X.mean())

            X = sm.add_constant(X)

            # Check for multicollinearity
            collinearity = pd.Series(
                [vif(X.values, i) for i in range(X.shape[1])], index=X.
                                                        columns
            )

            model = sm.OLS(Y, X).fit()
            print(model.summary())
            params = model.params.round(decimals=3)

            pvalues = model.pvalues.round(decimals=3).to_frame(name=tick
                                                    ).transpose()
            pvalues["R_squared"] = model.rsquared
            pvalues["coef_" + x] = params[x]
```

```
            self.summary = self.summary.append(pvalues)

        self.summary = self.summary.style.set_caption(f"{y} ~ " + x)
        display(self.summary)


if __name__ == "__main__":
    x_values = ["neutral", "negative"]
    x_values_log = ["positive_log", "negative_log"]
    y_values = ["Daily variation", "Volume"]

    reg = LinearRegression("Reddit")

    reg.main(x_values, y_values[1])
    # reg.main(x_values_log, y_values[1])
    # reg.main(["positive", "negative"], "Daily variation")
    # reg.main_inverse(x="Volume", y=x_values)
```

Code A.14 – Test de causalité de Granger

```
import pandas as pd
from IPython.display import display
from typing import Union

import warnings

warnings.filterwarnings("ignore")
from statsmodels.tools.sm_exceptions import InterpolationWarning

warnings.simplefilter("ignore", InterpolationWarning)

from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.api import VAR
from statsmodels.stats.stattools import durbin_watson
import statsmodels.tsa.stattools as ts


class GrangerCausalityTest:
    def __init__(self, path: str, swap_x_y: bool = False):
        self.swap_x_y = swap_x_y

        if path.lower() == "reddit":
            self.path = "../Sentiment Analysis/Reddit_final/{}.csv"
        elif path.lower() == "twitter":
            self.path = "../Sentiment Analysis/Twitter_final/{}.csv"

        self.summary = pd.DataFrame(
```

```
            columns=["Stock", "y", "x", "Explained", "p-value", "lag"]
        )

        self.tickers = [
            "AAPL",
            "AMD",
            "BB",
            "CLOV",
            "NAKD",
            "NIO",
            "NOK",
            "PLTR",
            "RKT",
            "SNDL",
            "SPCE",
            "SPY",
            "TSLA",
        ]

        self.x = None
        self.y = None

    @staticmethod
    def check_stationarity(time_series: pd.DataFrame) -> bool:
        adf_test = adfuller(time_series)
        kpss_test = kpss(time_series)
        return adf_test[1] <= 0.05 and kpss_test[1] >= 0.05

    @staticmethod
    def check_best_lag(time_series: pd.DataFrame):
        model = VAR(time_series)
        aic_scores = []
        for i in range(1, 13):
            result = model.fit(i)
            aic_scores.append(result.aic)
        return aic_scores.index(min(aic_scores)) + 1, model

    @staticmethod
    def check_serial_correlation(model, best_lag, time_series):
        model_fitted = model.fit(best_lag)
        out = durbin_watson(model_fitted.resid)
        return [val for col, val in zip(time_series.columns, out)]

    @staticmethod
    def check_cointegration(ts1, ts2):
        result = ts.coint(ts1, ts2)
        return result[1]
```

```
    @staticmethod
    def granger_causality_test(time_series, best_lag):
        results = grangercausalitytests(time_series, maxlag=[best_lag],
                                        verbose=False)

        return min(
            [
                results[best_lag][0]["ssr_ftest"][1],
                results[best_lag][0]["ssr_chi2test"][1],
                results[best_lag][0]["lrtest"][1],
                results[best_lag][0]["params_ftest"][1],
            ]
        )

    def main(self, x: Union[str, list], y: Union[str, list]):

        print("Running Granger causality test for x:", x, ", y:", y)

        if self.swap_x_y:
            x, y = y, x

        self.x = x
        self.y = y

        for tick in self.tickers:

            df = pd.read_csv(self.path.format(tick), index_col=0)
            df_test = df.copy(deep=True)

            if isinstance(x, list):
                df["sum"] = df[x].sum(axis=1)
                df_test["sum"] = df_test[x].sum(axis=1)
                x = "sum"

            if isinstance(y, list):
                df["sum"] = df[y].sum(axis=1)
                df_test["sum"] = df_test[y].sum(axis=1)
                y = "sum"

            # Check if time series are stationary
            n = 1
            while not GrangerCausalityTest.check_stationarity(
                df_test[x]
            ) and not GrangerCausalityTest.check_stationarity(df[y]):
                df_test[x] = df_test[x] - df_test[x].shift(n)
                df_test[y] = df_test[y] - df_test[y].shift(n)
```

```
        df_test = df_test.dropna()
        n += 1

    one_stock = dict.fromkeys(
        ["Stock", "y", "x", "Explained", "p-value", "lag"]
    )
    one_stock["Stock"] = tick
    one_stock["y"] = y
    one_stock["x"] = x

    df.index = pd.DatetimeIndex(df.index).to_period("D")

    # Check best lag
    one_stock["lag"], model = GrangerCausalityTest.
                                        check_best_lag(df[[y, x]
                                        ])

    # Check for serial correlation
    vals = GrangerCausalityTest.check_serial_correlation(
        model, one_stock["lag"], df[[y, x]]
    )

    if not all(i >= 1.5 and i <= 2.5 for i in vals):
        print("Serial correlation in the residuals")

    one_stock["coint pvalue"] = format(
        GrangerCausalityTest.check_cointegration(df_test[y],
                                            df_test[x]), ".3E"
    )

    # Compute Granger Causality test
    one_stock["p-value"] = round(
        GrangerCausalityTest.granger_causality_test(
            df_test[[y, x]], one_stock["lag"]
        ),
        2,
    )
    one_stock["Explained"] = one_stock["p-value"] <= 0.05

    one_stock["p-value"] = format(
        GrangerCausalityTest.granger_causality_test(
            df_test[[y, x]], one_stock["lag"]
        ),
        ".3E",
    )

    self.summary = self.summary.append(one_stock, ignore_index=
```

```
                                                    True)

            x = self.x
            y = self.y

    def print_results(self):
        print("\nPrinting results...\n")
        print(
            f'{self.summary["Explained"].value_counts()[True]} explained
                                                     stocks, {self.summary["
                                                    Explained"].value_counts
                                                    ()[False]} unexplained
                                                    stocks'
        )

        self.summary = self.summary.style.set_caption(f"{self.y} ~ {self
                                                .x}")
        display(self.summary)


if __name__ == "__main__":

    y_values = ["Daily variation", "Volume"]
    x_values = ["positive", "negative", "neutral"]

    granger = GrangerCausalityTest("Reddit", True)

    granger.main(x_values, y_values[1])

    granger.print_results()
```

Annexe B

Analyse exploratoire

Cette section contient les figures générées pour l'analyse exploratoire de ce mémoire, tout d'abord pour Reddit et ensuite pour Twitter.

B.1 Reddit

B.1.1 Répartition annuelle des messages

Dans cette section, des cartes de chaleur représentent la répartition annuelle des messages sur Reddit pour nos différentes actions. La répartition des messages semble assez sporadique tout au long de l'année, quelle que soit l'action. Cela peut probablement être expliqué par le nombre inférieur de messages extraits sur la plateforme.

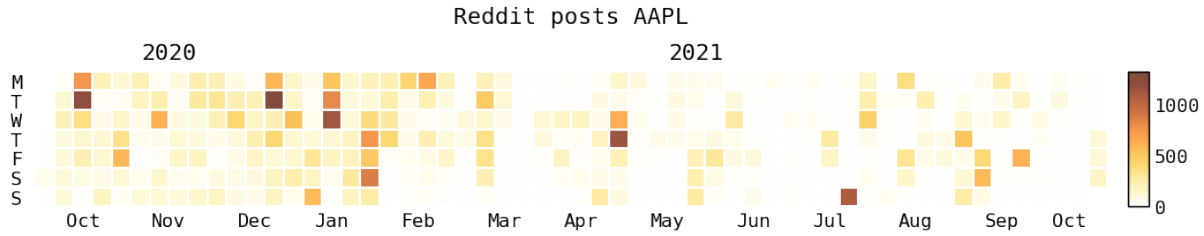

FIGURE 1 – AAPL - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

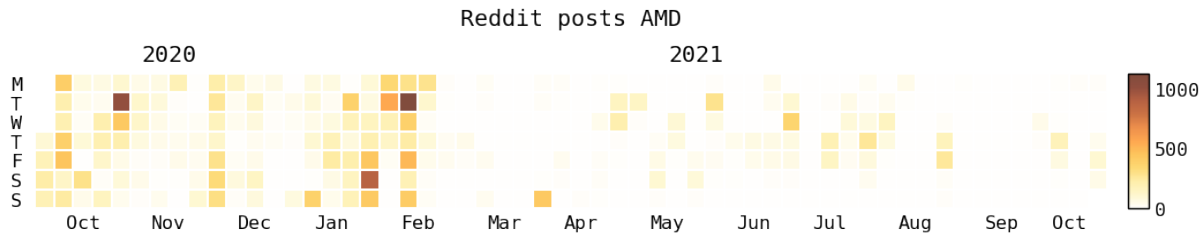

FIGURE 2 – AMD - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

FIGURE 3 – BB - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

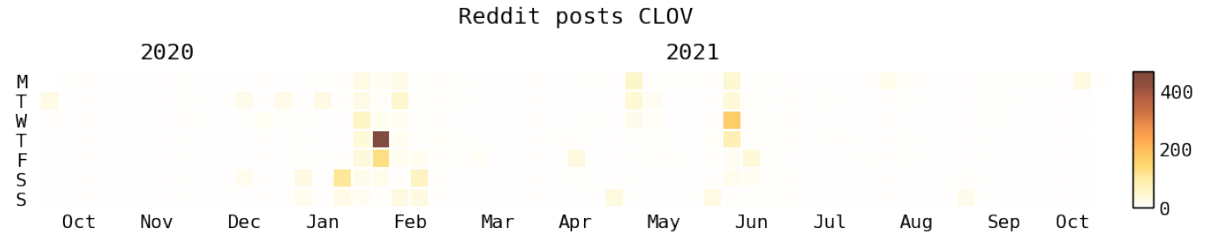

FIGURE 4 – CLOV - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

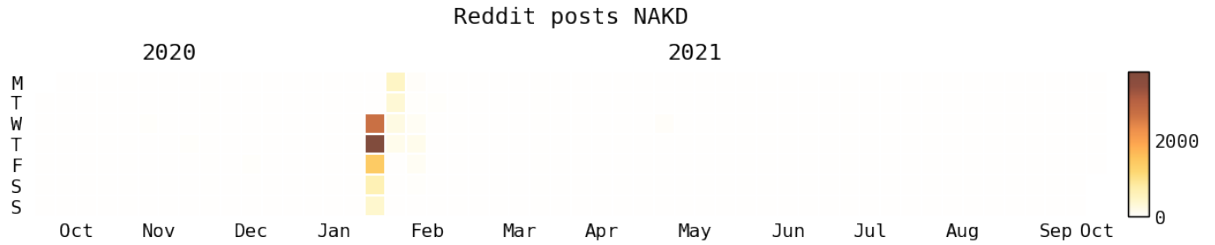

FIGURE 5 – CENN (NAKD) - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

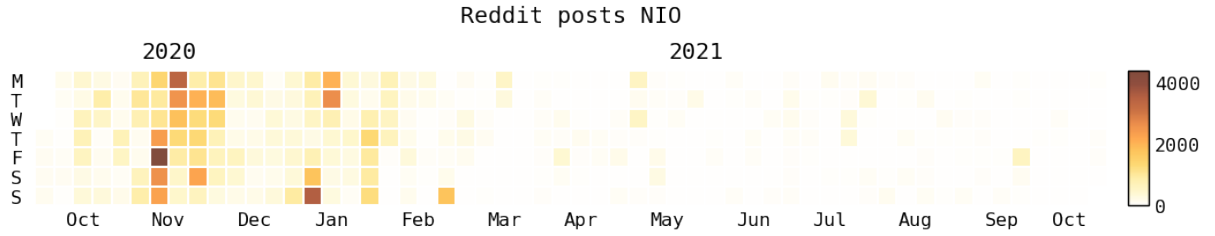

FIGURE 6 – NIO - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

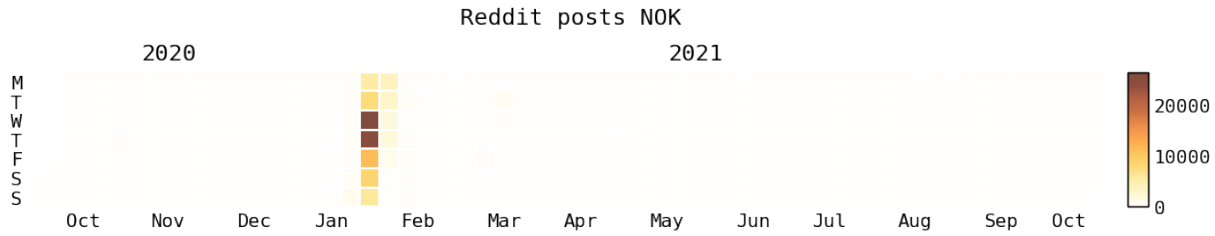

FIGURE 7 – NOK - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

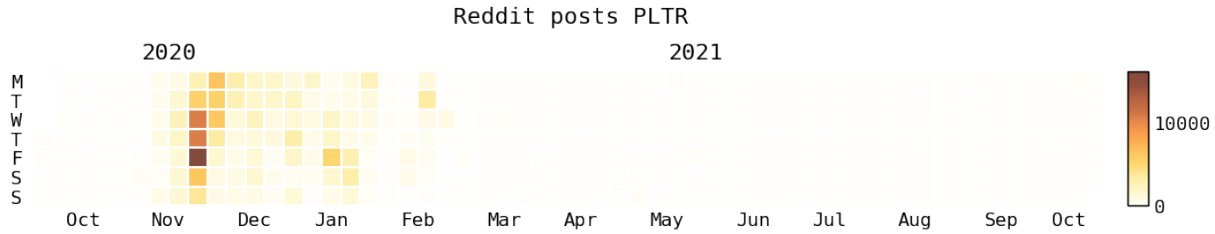

FIGURE 8 – PLTR - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

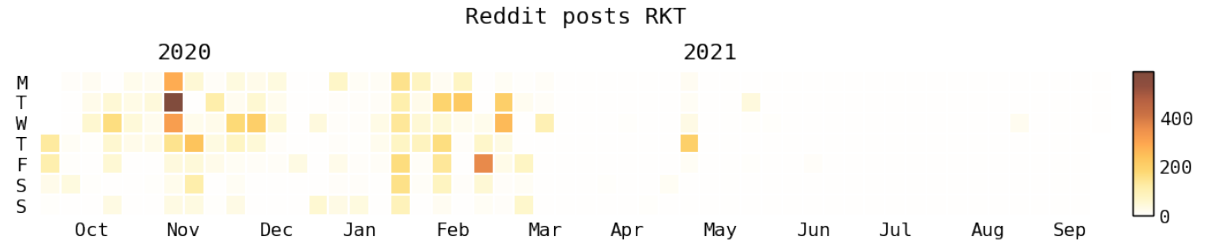

FIGURE 9 – RKT - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

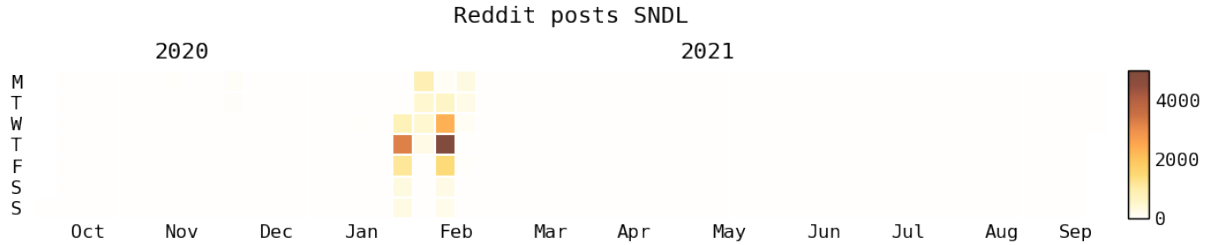

FIGURE 10 – SNDL - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

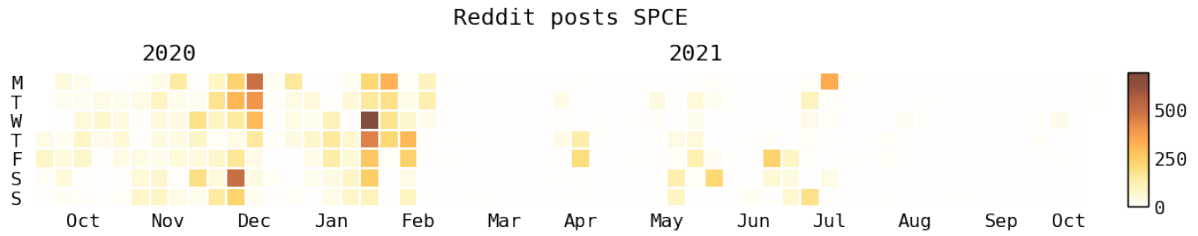

FIGURE 11 – SPCE - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

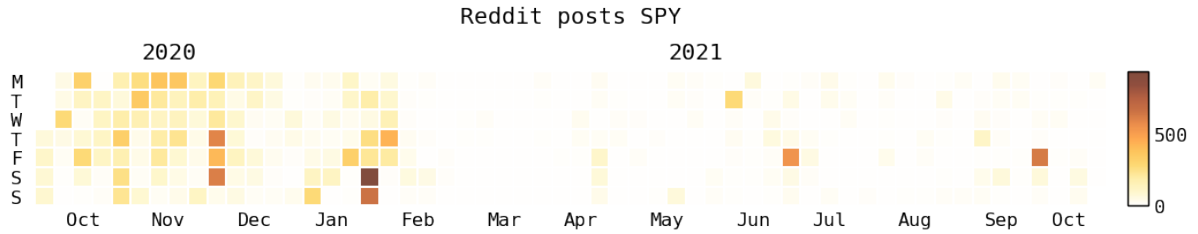

FIGURE 12 – SPY - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

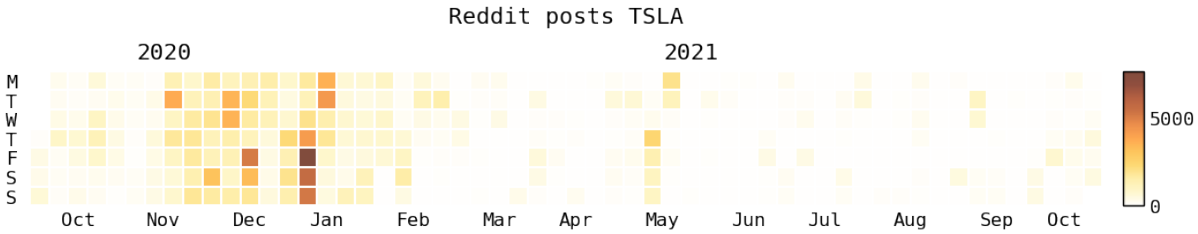

FIGURE 13 – TSLA - Carte de chaleur du nombre de messages sur Reddit entre le 01/10/2020 et le 24/10/2021

B.1.2 Jours présentant le plus de messages

Cette section s'intéresse aux jours pour lesquels le plus de messages ont été postés sur Reddit pour une action considérée. Nous observons de manière générale de grosses différences en termes de nombre de messages entre nos différents jours.

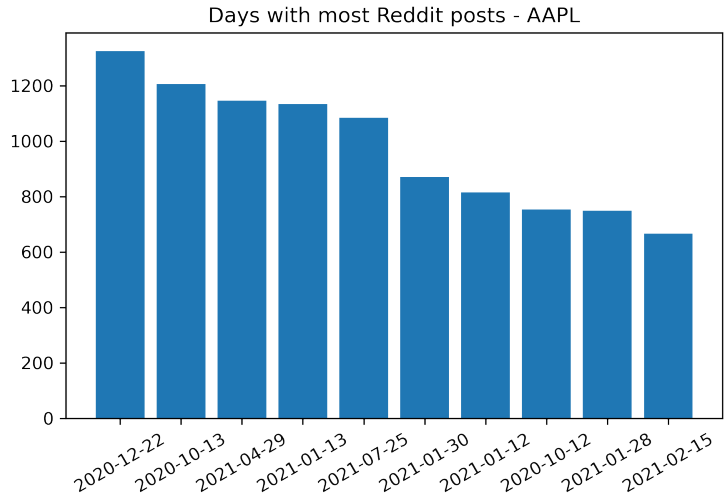

FIGURE 14 – AAPL - Top 10 du nombre de messages Reddit par jour

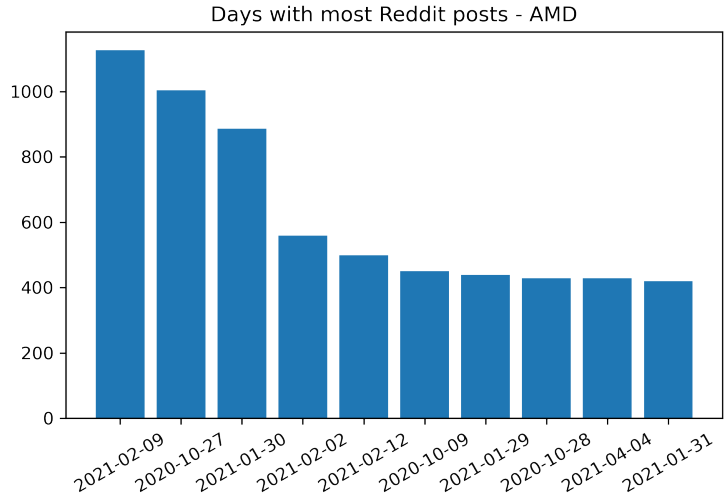

FIGURE 15 – AMD - Top 10 du nombre de messages Reddit par jour

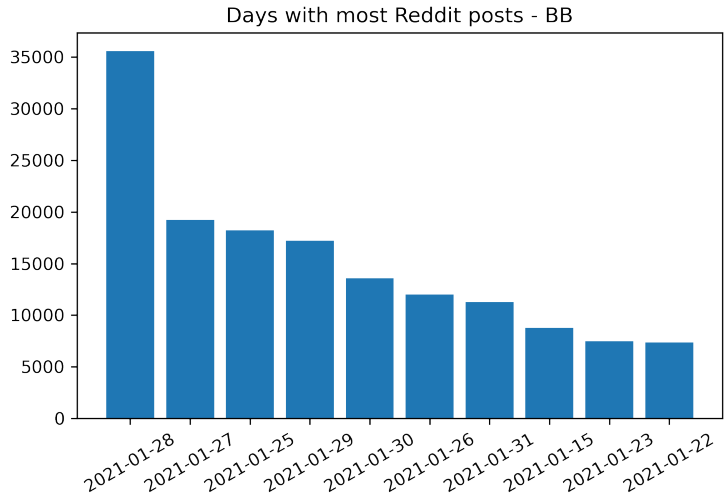

FIGURE 16 – BB - Top 10 du nombre de messages Reddit par jour

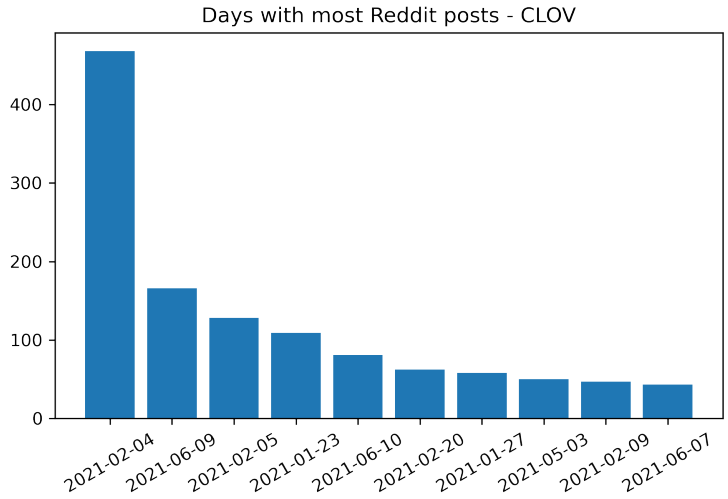

FIGURE 17 – CLOV - Top 10 du nombre de messages Reddit par jour

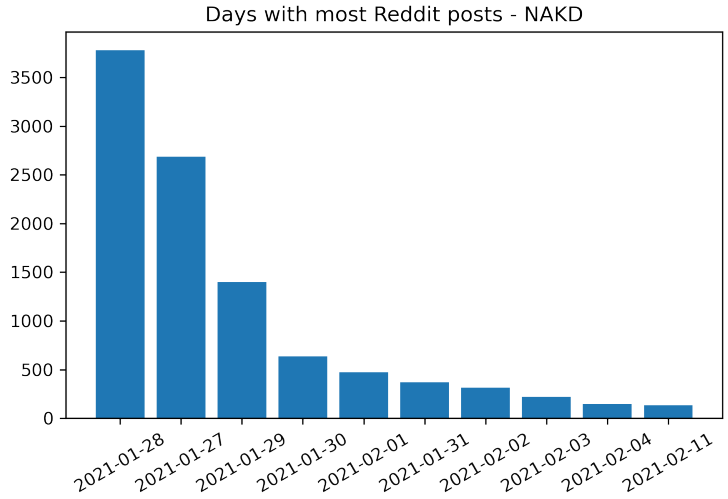

FIGURE 18 – CENN (NAKD) - Top 10 du nombre de messages Reddit par jour

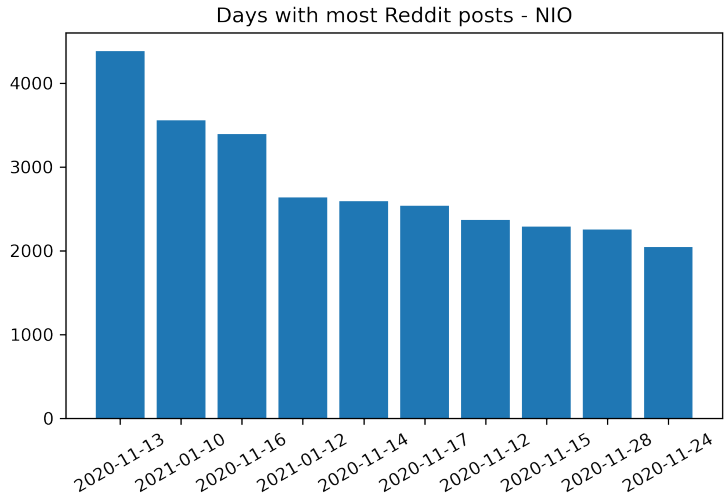

FIGURE 19 – NIO - Top 10 du nombre de messages Reddit par jour

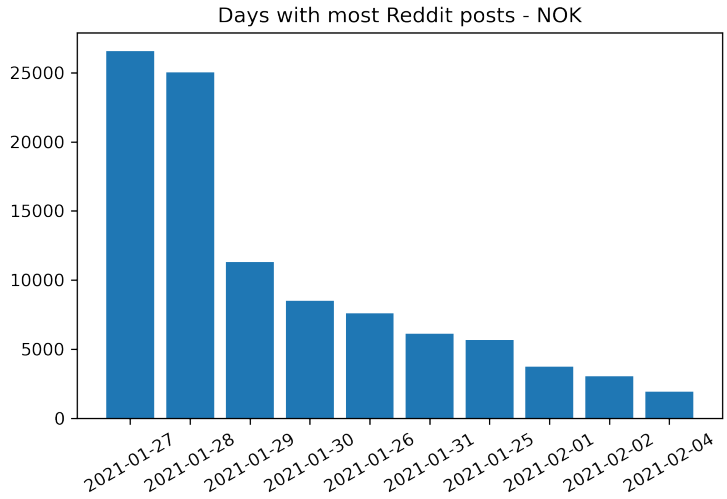

FIGURE 20 – NOK - Top 10 du nombre de messages Reddit par jour

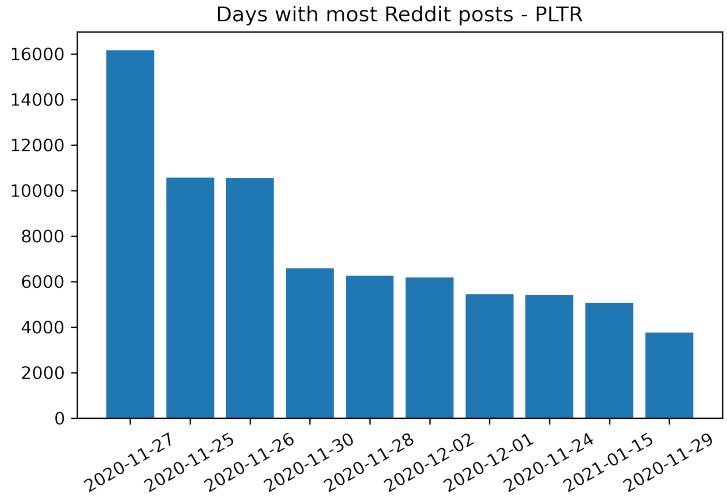

FIGURE 21 – PLTR - Top 10 du nombre de messages Reddit par jour

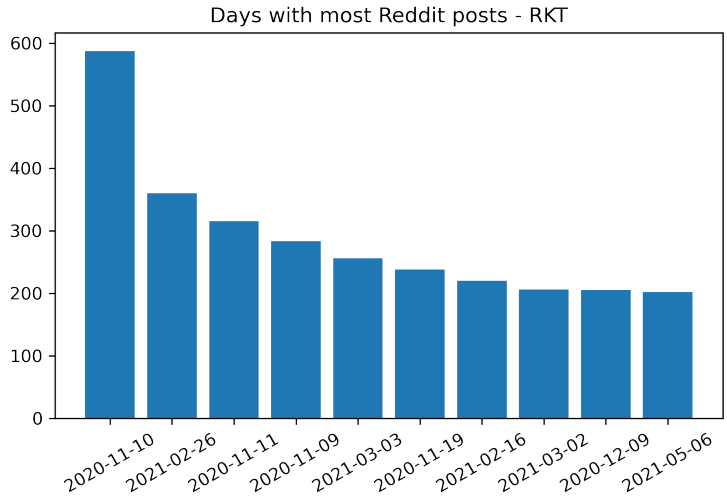

FIGURE 22 – RKT - Top 10 du nombre de messages Reddit par jour

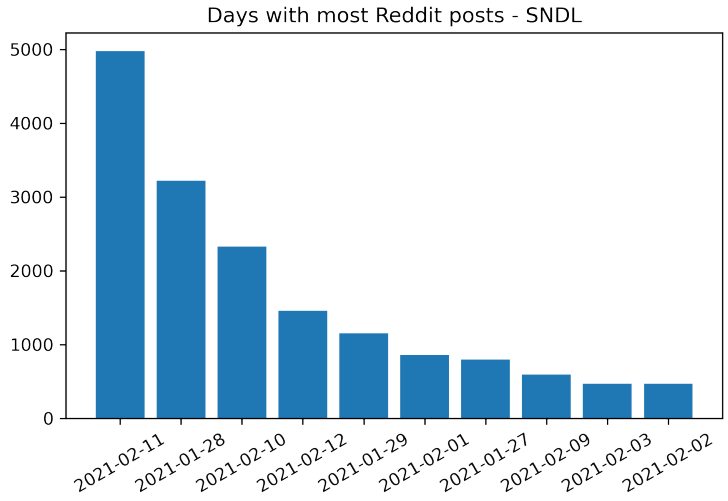

FIGURE 23 – SNDL - Top 10 du nombre de messages Reddit par jour

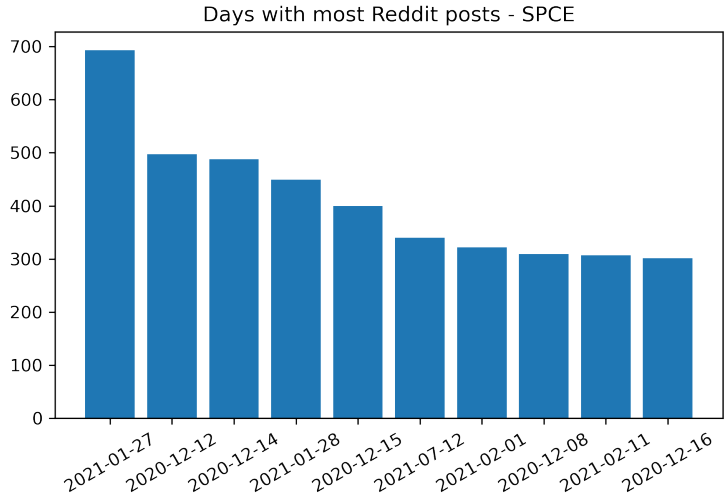

FIGURE 24 – SPCE - Top 10 du nombre de messages Reddit par jour

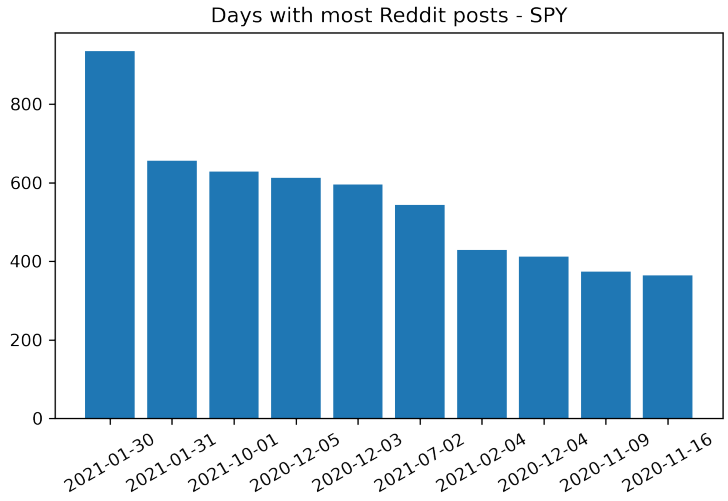

FIGURE 25 – SPY - Top 10 du nombre de messages Reddit par jour

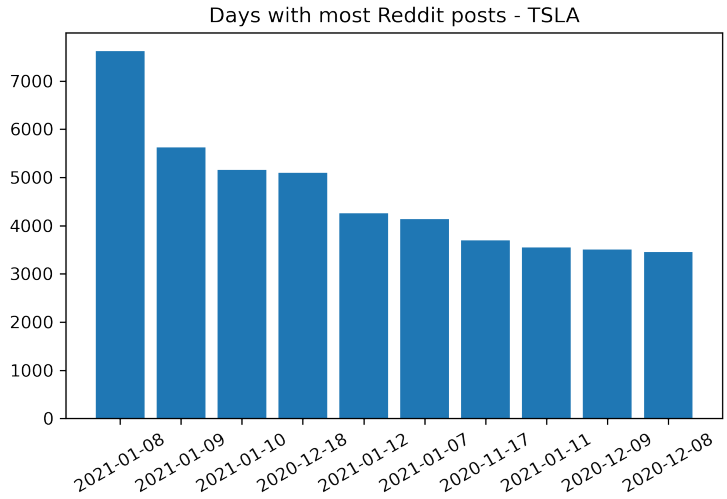

FIGURE 26 – TSLA - Top 10 du nombre de messages Reddit par jour

B.2 Twitter

B.2.1 Répartition annuelle des messages

Comme pour Reddit, la répartition annuelle des messages sur Twitter nous permet de réaliser certaines observations. La répartition est beaucoup plus constante que Reddit pour une partie des actions. Nous observons aussi que les messages sont principalement postés du lundi au vendredi.

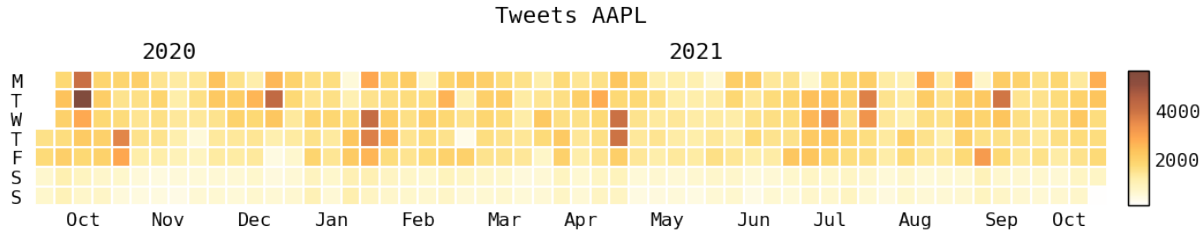

FIGURE 27 – AAPL - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

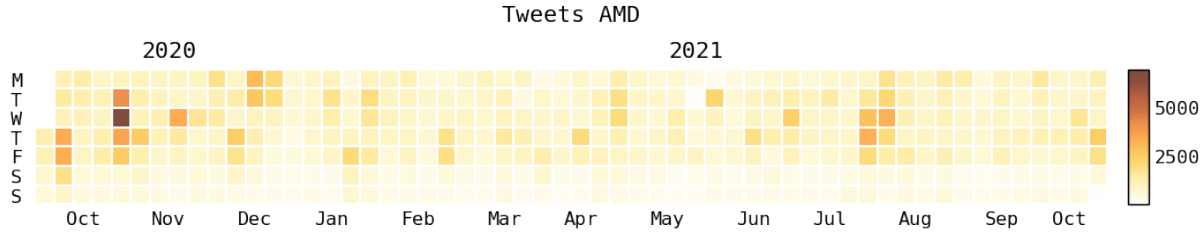

FIGURE 28 – AMD - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

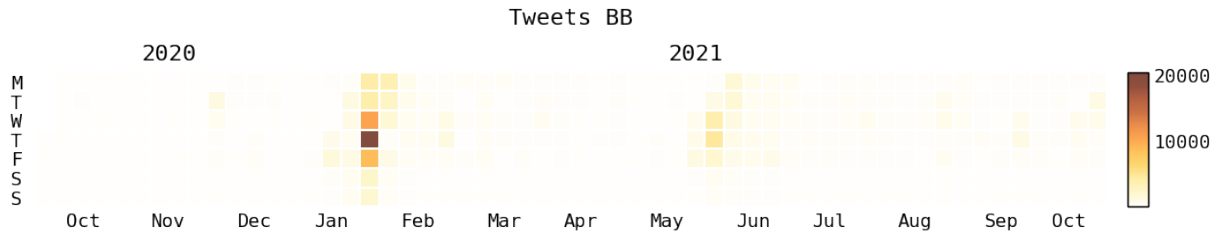

FIGURE 29 – BB - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

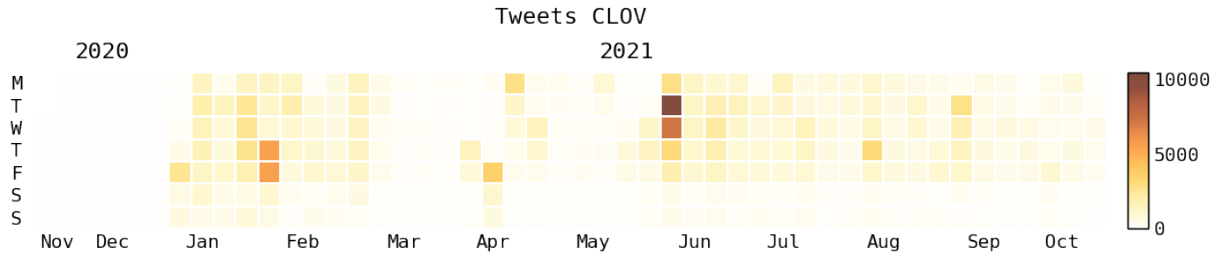

FIGURE 30 – CLOV - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

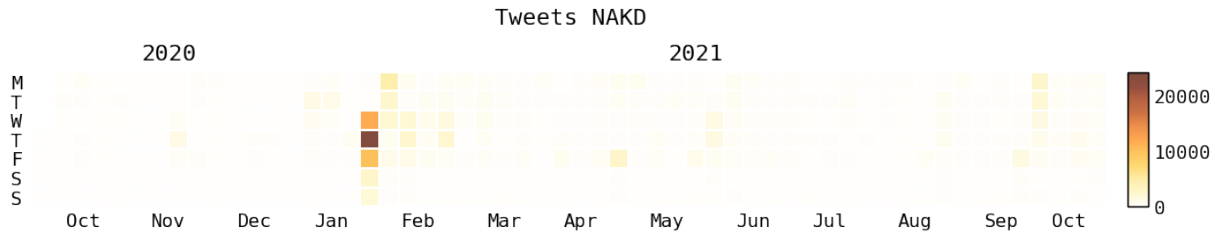

FIGURE 31 – CENN (NAKD) - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

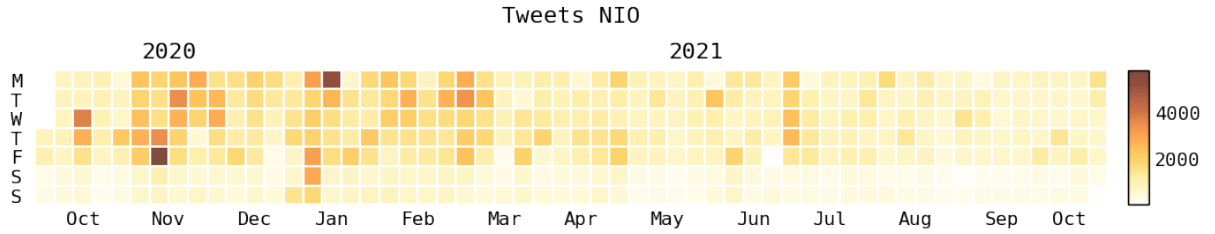

FIGURE 32 – NIO - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

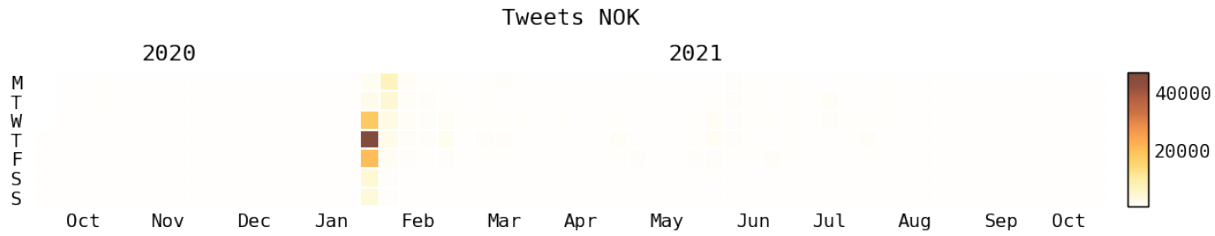

FIGURE 33 – NOK - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

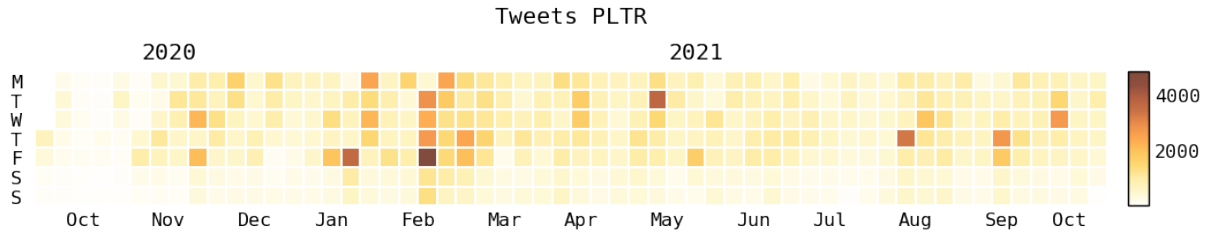

FIGURE 34 – PLTR - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

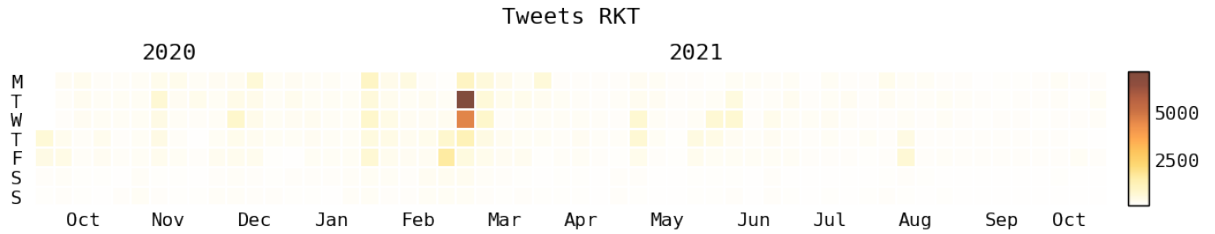

FIGURE 35 – RKT - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

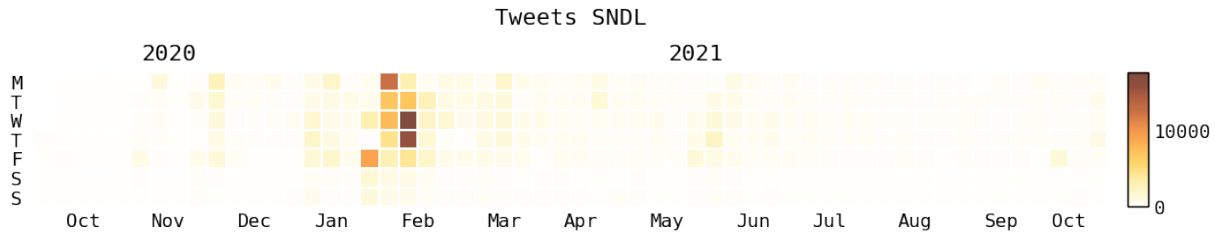

FIGURE 36 – SNDL - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

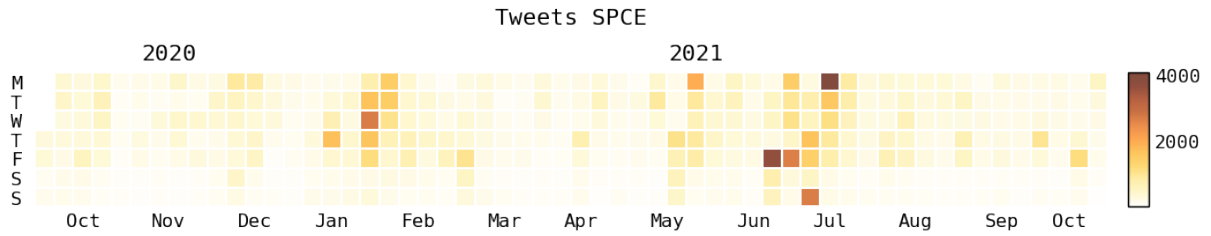

FIGURE 37 – SPCE - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

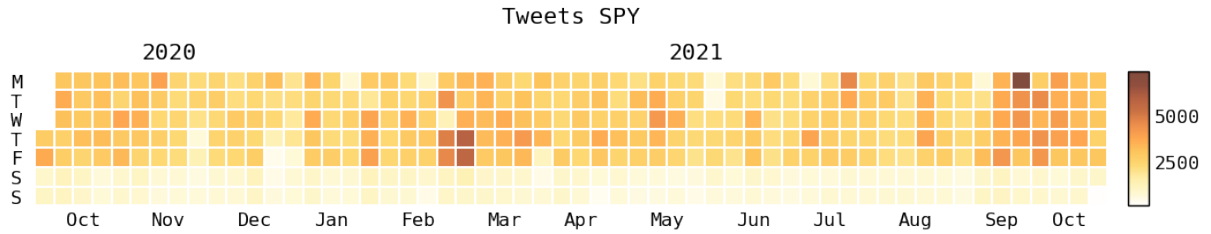

FIGURE 38 – SPY - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

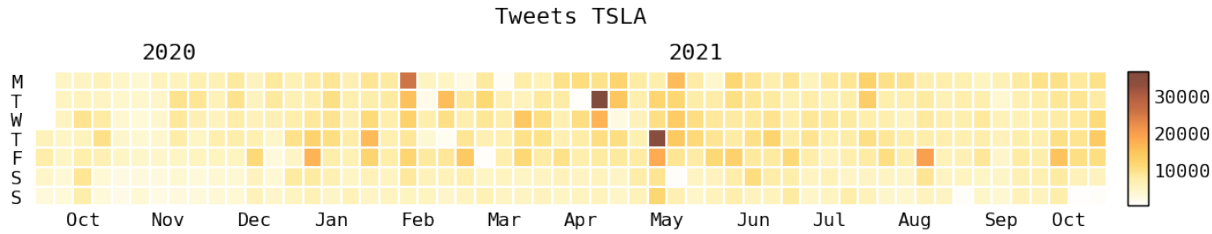

FIGURE 39 – TSLA - Carte de chaleur du nombre de tweets entre le 01/10/2020 et le 24/10/2021

B.2.2 Jours présentant le plus de messages

Si nous nous intéressons aux jours pour lesquels le plus de messages ont été postés sur Twitter, les actions pour lesquelles la répartition semblait plus constante lors de l'analyse des cartes de chaleurs semblent présenter peu de différence entre le nombre de messages pour ces différentes journées.

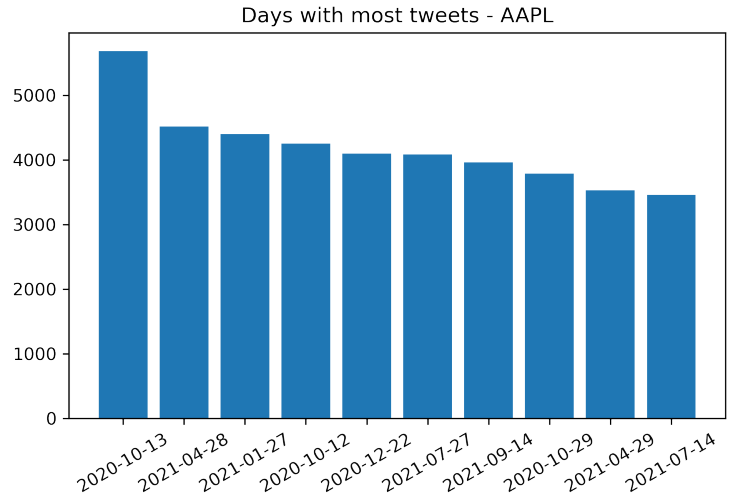

FIGURE 40 – AAPL - Top 10 du nombre de tweets par jour

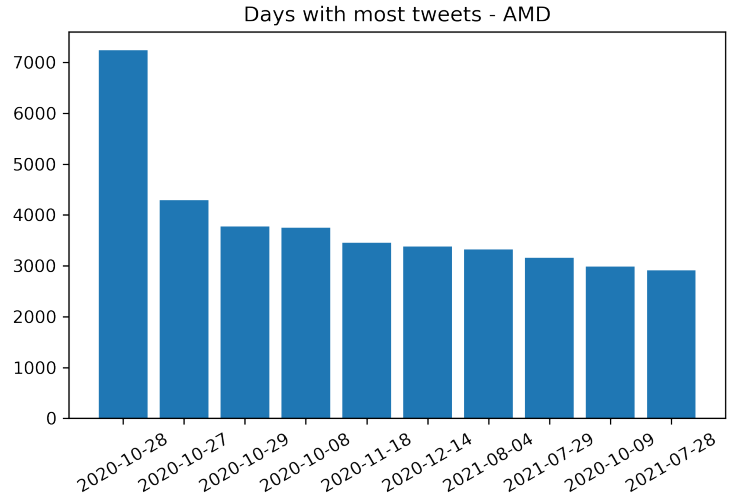

FIGURE 41 – AMD - Top 10 du nombre de tweets par jour

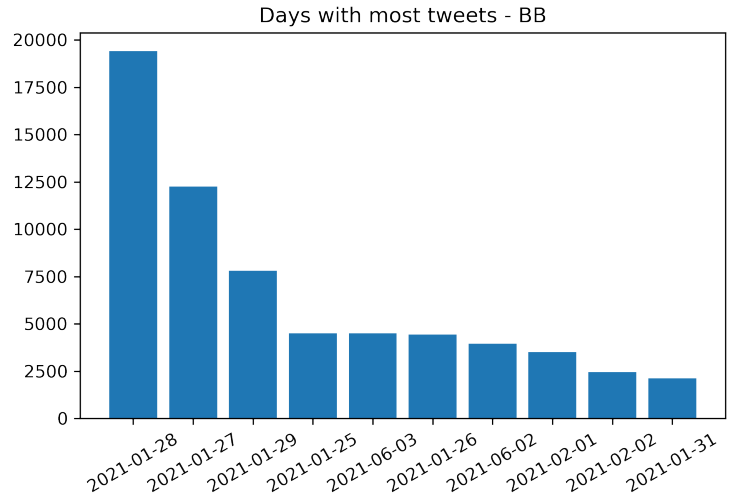

FIGURE 42 – BB - Top 10 du nombre de tweets par jour

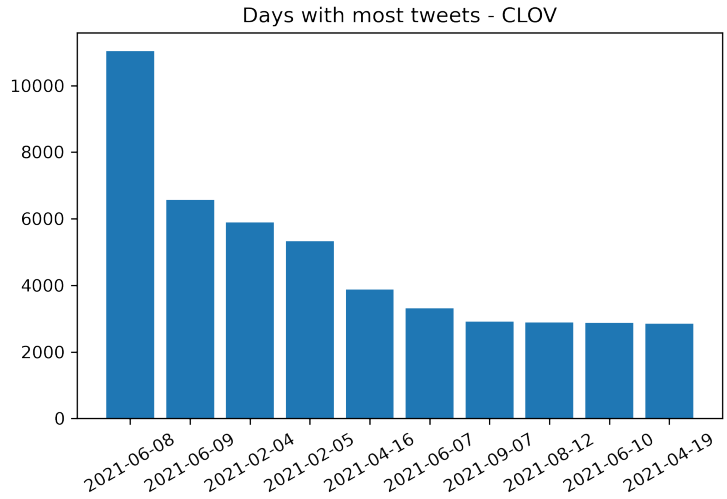

FIGURE 43 – CLOV - Top 10 du nombre de tweets par jour

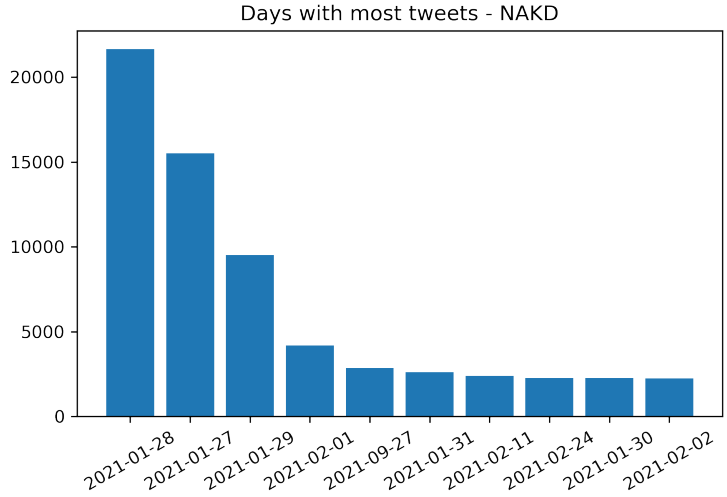

FIGURE 44 – CENN (NAKD) - Top 10 du nombre de tweets par jour

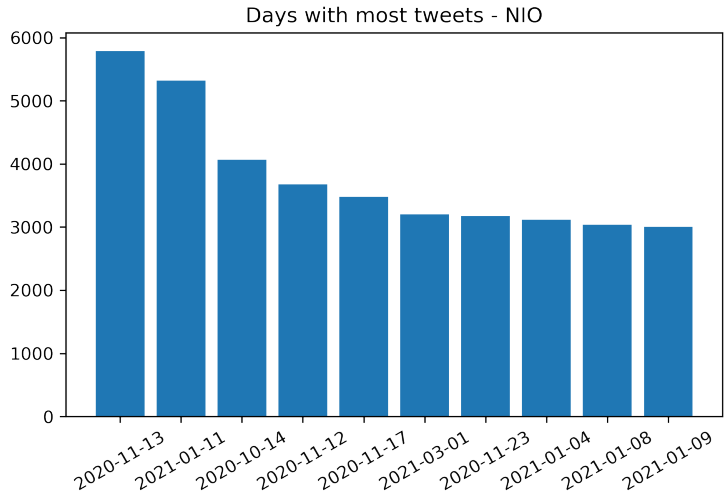

FIGURE 45 – NIO - Top 10 du nombre de tweets par jour

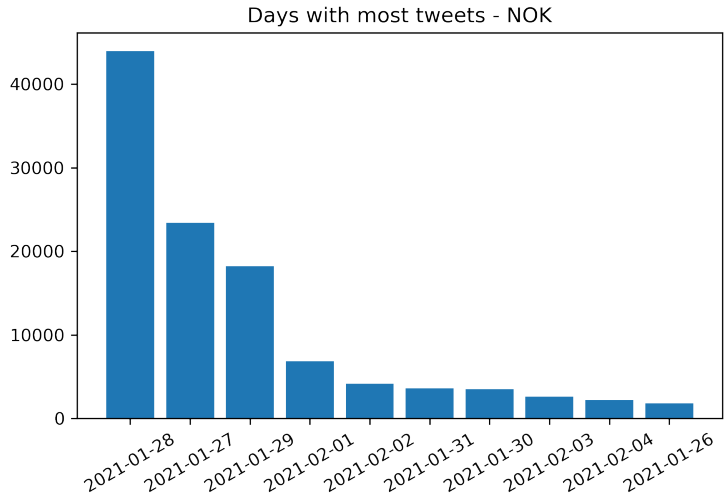

FIGURE 46 – NOK - Top 10 du nombre de tweets par jour

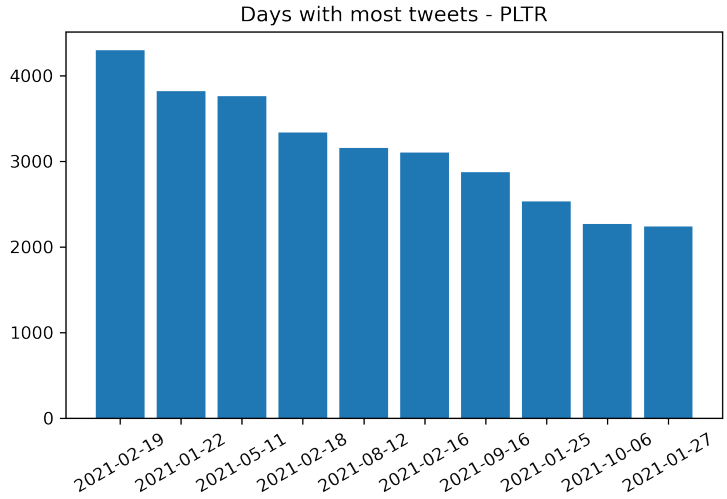

FIGURE 47 – PLTR - Top 10 du nombre de tweets par jour

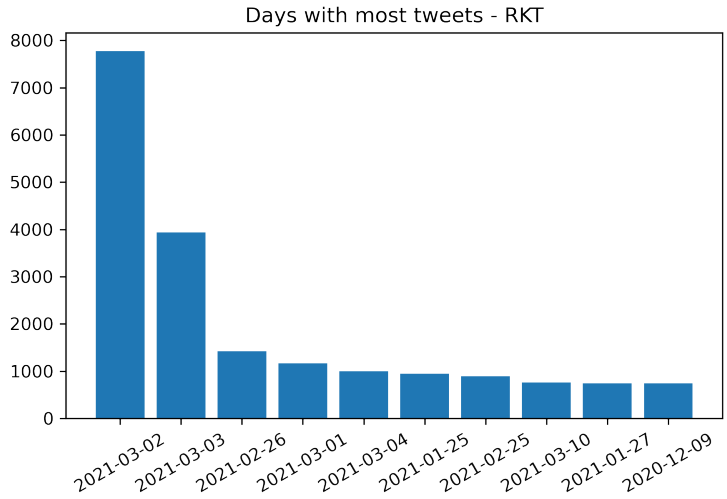

FIGURE 48 – RKT - Top 10 du nombre de tweets par jour

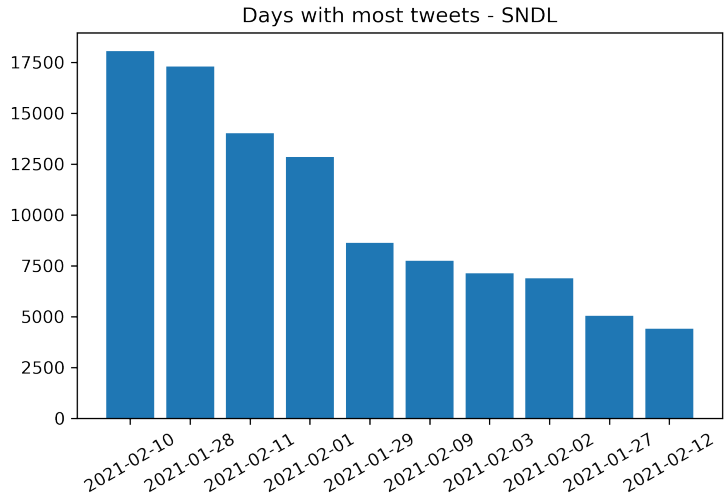

FIGURE 49 – SNDL - Top 10 du nombre de tweets par jour

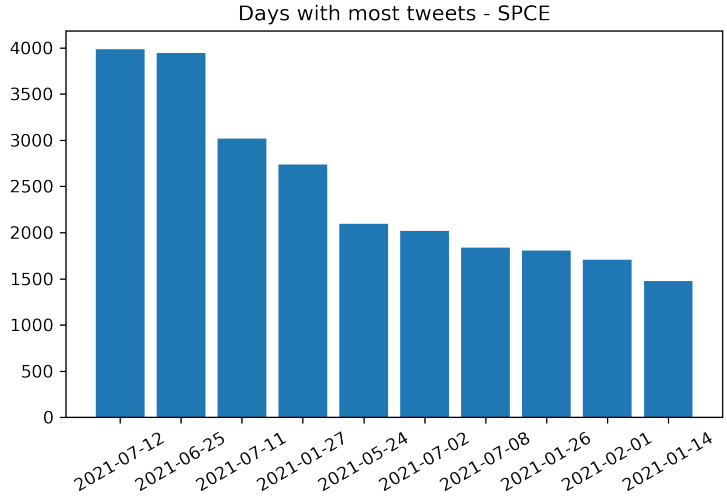

FIGURE 50 – SPCE - Top 10 du nombre de tweets par jour

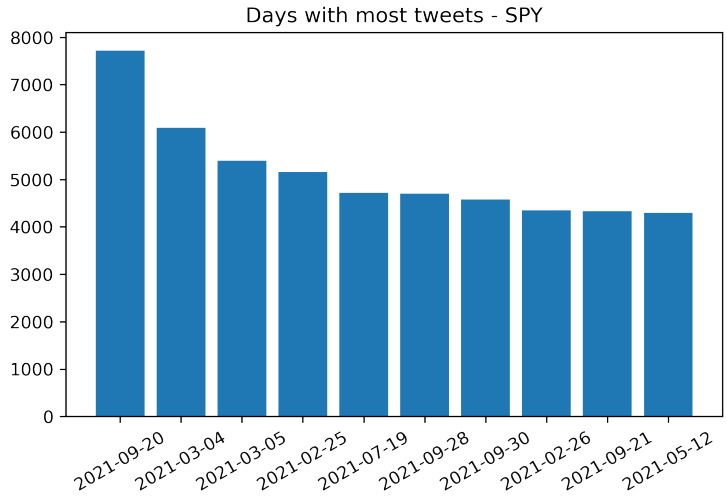

FIGURE 51 – SPY - Top 10 du nombre de tweets par jour

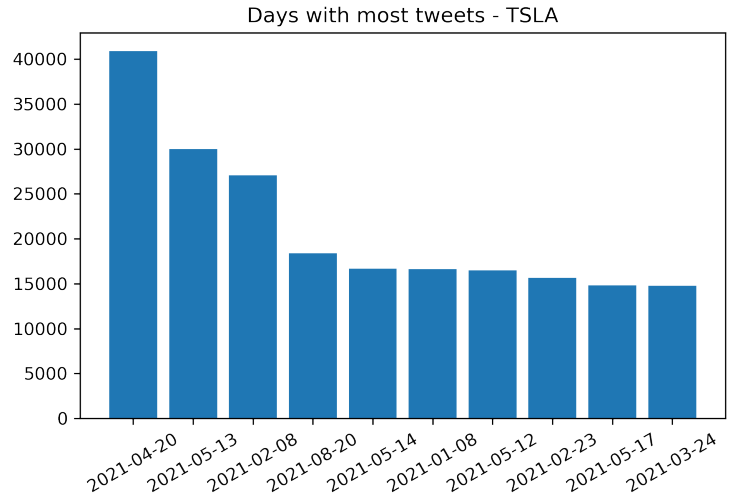

FIGURE 52 – TSLA - Top 10 du nombre de tweets par jour

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Louvain School of Management
Chaussée de Binche 151, 7000 Mons, Belgique | **www.uclouvain.be/lsm**