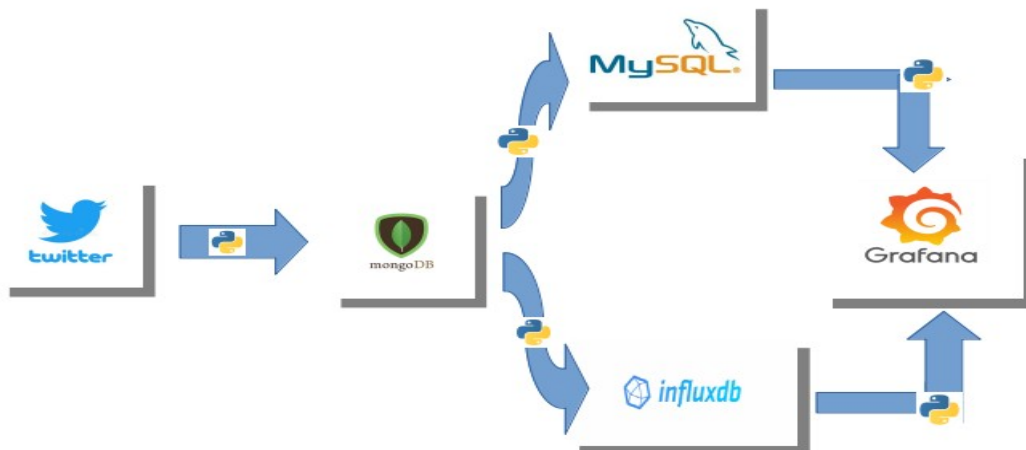


INTRODUCTION

Dans ce projet de l'unité d'enseignement **Analyse des réseaux sociaux**, notre objectif est de collecter les Tweets comportant le hashtag #COVID19 à l'aide de **Tweepy** puis les stocker dans une base de données **MongoDB** en local en suite pré-traiter et analyser les données avec **python** et stocker les résultats dans les bases de données **Mysql** et **InfluxDB** selon la visualisation avec **Grafana**. Il est claire de souligner que grafana est un puissant outils de visualisation, pour l'utiliser je l'ai paramétré aux base de données Mysql et InfluxDB. Ci dessous on peut voir l'architecture de ce projet :



Commençons par la collection des Tweets.

COLLECTIONS

Nous collectionnons les Twittes comportant des hashtag #COVID19 en utilisant Tweepy puis nous les stockons dans une base de données mongoDB. Tweepy permet de faciliter l'utilisation de l'API de streaming Twitter en gérant l'authentification, la connexion, la création et la destruction de la session, la lecture des messages entrant et le routage partiel des messages. Ci dessous nous importons les librairies et définissons les clés d'aces de notre application crée sur Twitter.

Le fichier **config.py** contient les clés d'authentifications de l'application crée dans mon compte Twitter développeur pour avoir à l'API de streaming Twitter.

```
In[2]: # Importation des librairies
...: from tweepy.streaming import StreamListener
...: from tweepy import OAuthHandler
...: from tweepy import Stream
...: import config
...: import time
...: import json
...: from pymongo import MongoClient
...:
...: # les clés d'autification pour l'accès à l'Api streaming de Twitter
...: consumer_key = config.consumer_key
...: consumer_secret = config.consumer_secret
...: access_token = config.access_token
...: access_secret = config.access_secret
...:
```

L' API de streaming Twitter est utilisé pour télécharger les messages Twitter en temps réel.

Dans la librairie **Tweepy** , une instance de **tweepy.Stream** établit une session de streaming et achemine les messages vers l'instance **StreamListener**. La méthode **on_data()** d'un écouteur de flux reçoit tous les messages et appelle des fonctions en fonction du type de message. Ci dessous la création de la classe **StreamingListener** héritant de la classe **StreamListener**

```
In[2]: class StreamingListener(StreamListener):
...:
...:     def on_data(self, tweet_data):
...:
...:         tweet_data_json = json.loads(tweet_data)
...:
...:         client = MongoClient('localhost:27017')
...:         COVID = client.COVID
...:         Collection = COVID["Tweets"]
...:         Collection.insert_one(tweet_data_json)
...:
...:         print(tweet_data_json)
...:
...:         return True
...:
...:     def on_error(self, status):
...:         print("Printing in on_error function :" + status)
...:
```

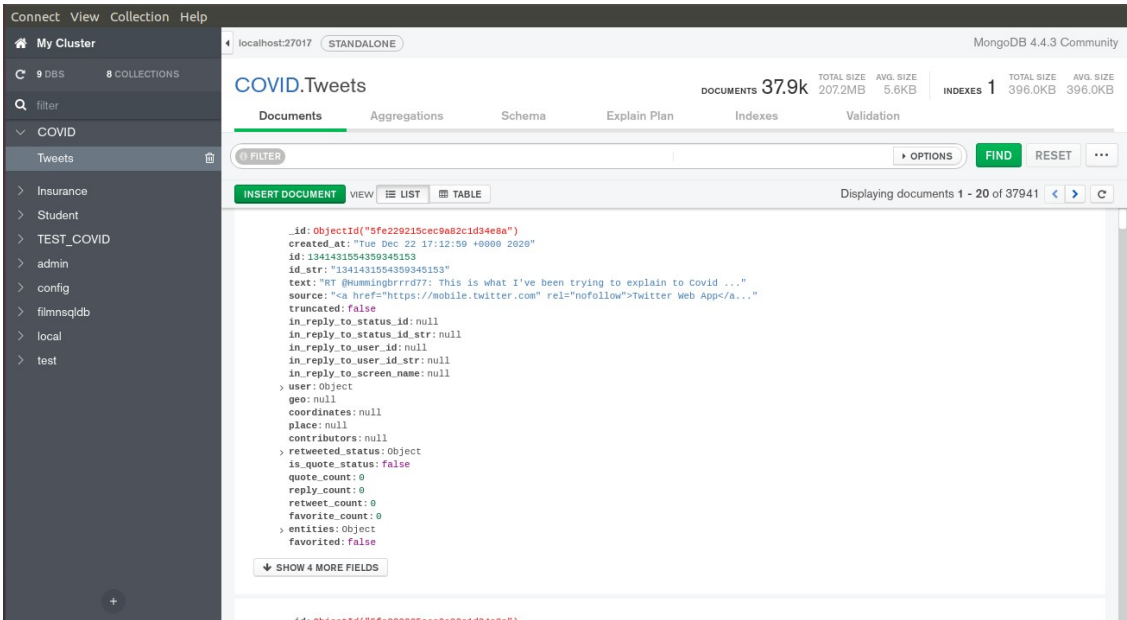
La méthode **on_data()** reçoit tout les messages puis les charges en json par la suite on insert les données en format json dans la base de données **COVID** dans une collection appelée **Tweets**. Nous sommes maintenant à mesure de créer notre objet flux en utilisant un filtre pour diffuser tous les messages contenant: **corona**, **coronavirus**, **covid**, **covid-19**.

```
In[2]: if __name__ == '__main__':
...:     print("Twitter Streaming Application Started")
...:
...:     covidStreamingListening = StreamingListener()
...:     auth = OAuthHandler(consumer_key, consumer_secret)
...:     auth.set_access_token(access_token, access_secret)
...:
...:     stream = Stream(auth, covidStreamingListening)
...:
...:     stream.filter(track=['corona', 'covid', 'Coronavirus', 'Covid-19'])
...:
```

Après exécutions nous pouvons voir les tweets:

{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693753634742272',	'id_str':	'1354693753634742272',	'text':	'RT @pudy.betrayed: remember we
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693753747857408',	'id_str':	'1354693753747857408',	'text':	'RT @BJP4Gujarat: We've seen eff
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'13546937538387073159',	'id_str':	'13546937538387073159',	'text':	'RT @benizli_SM: COVID-19 AŞI ÜY
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'13546937540080843521',	'id_str':	'13546937540080843521',	'text':	'@nickkangwa: There is differen
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693754213392838',	'id_str':	'1354693754213392838',	'text':	'https://t.co/rHISgyHWrw', 'sour
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693754242818728',	'id_str':	'1354693754242818728',	'text':	'RT @lokeshsharma: #Rajasthan 1
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693754347728896',	'id_str':	'1354693754347728896',	'text':	'RT @SundayTimez2A: The pharme
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693754653978624',	'id_str':	'1354693754653978624',	'text':	'RT @dardogaspape: Se habla mu
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693754712514362',	'id_str':	'1354693754712514362',	'text':	'RT @Isabellazovella: Why CCP ha
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693754922397696',	'id_str':	'1354693754922397696',	'text':	'"Firstly, we need to continue t
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693755040432515',	'id_str':	'1354693755040432515',	'text':	'RT @MollyJongFast: This is an e
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'135469375545107074',	'id_str':	'135469375545107074',	'text':	'RT @Libertyninja: @JonathanTur
{	'created_at':	'Thu Jan 28 07:32:14 +0000 2021',	'id':	'1354693755565310786',	'id_str':	'1354693755565310786',	'text':	'RT @Jeff-Jacoby: "With a heart
{	'created_at':	'Thu Jan 28 07:32:15 +0000 2021',	'id':	'1354693755786440704',	'id_str':	'1354693755786440704',	'text':	'RT @Kinda-Libyan: "once covid 1
{	'created_at':	'Thu Jan 28 07:32:15 +0000 2021',	'id':	'1354693755937255429',	'id_str':	'1354693755937255429',	'text':	'RT @tnnthailand: #สัปดาห์ท
{	'created_at':	'Thu Jan 28 07:32:15 +0000 2021',	'id':	'1354693756021846306',	'id_str':	'1354693756021846306',	'text':	'RT @SmartHabitsGirl: Very impor
{	'created_at':	'Thu Jan 28 07:32:15 +0000 2021',	'id':	'1354693756444762113',	'id_str':	'1354693756444762113',	'text':	'RT @ScalvinTanner: Fuck Covid.
{	'created_at':	'Thu Jan 28 07:32:15 +0000 2021',	'id':	'1354693756654649349',	'id_str':	'1354693756654649349',	'text':	'RT @doctor_oxford: Terrible new
{	'created_at':	'Thu Jan 28 07:32:15 +0000 2021',	'id':	'1354693756833020214',	'id_str':	'1354693756833020214',	'text':	'RT @BibBabylon: Daily: Core Updat

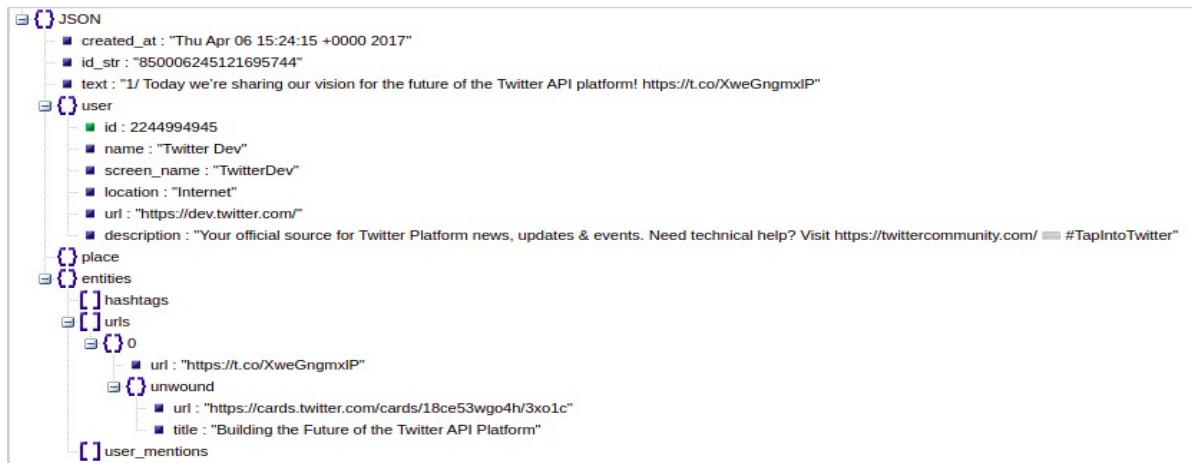
À présent nos Twittes sont bien stockés dans **MongoDB**:



Une première étape de notre projet est effectué passons à l'étape du pré-traitement des Twittes.

PRÉ-TRAITEMENTS

D'abord il nous connaissons la structure json d'un tweete afin d'extraire les champs qui vont nous servir au cours de cette étude, ci dessous la structure d'un fichier json:



Les champs qui vont nous être utile dans notre étude sont : **id**, **created_at**, **name**, **location**, **text**, **followers**, **hashtags**. Dans le code ci dessous nous extrayons les champs cités ci dessous puis nous intéressent puis nous les sauvegardons dans un fichier nommé **Twitter_data.csv** :

```

In[2]: from pymongo import MongoClient
      import pandas as pd
      client = MongoClient('mongodb://localhost:27017/')
      COVID = client['COVID']
      collection = COVID['Tweets']
      documents = []
      for doc in collection.find():
          if "limit" in doc.keys():
              documents.append({'_id':doc['_id'], "created_at": "2020", "id": "0", "user": {"name": "nan", "location": "nan", "followers_count": 0}, "entities": doc['entities']})
          else:
              documents.append(doc)

```

```

_Id = []
Id = []
Created_at = []
Name = []
Location = []
Followers = []
Messages = []
hashtags = []
for i in range(len(documents)):
    _Id = [documents[i]['_id'] for i in range(len(documents))]
    Id = [documents[i]['id'] for i in range(len(documents))]
    Created_at = [documents[i]['created_at'] for i in range(len(documents))]
    Name = [documents[i]['user']['name'] for i in range(len(documents))]
    Location = [documents[i]['user']['location'] for i in range(len(documents))]
    Followers = [documents[i]['user']['followers_count'] for i in range(len(documents))]
    hashtags = [documents[i]['entities']['hashtags'] for i in range(len(documents))]
    Messages = [documents[i]['text'] for i in range(len(documents))]
list = [_Id, Id, Created_at, Name, Location, Followers, hashtags, Messages]
columns = ['_Id', 'Id', 'Created_at', 'Name', 'Location', 'Followers', 'hashtags', 'Messages']
data = pd.DataFrame(list, columns).T
df = data.to_csv("Twitter_data.csv")

```

Une idée du contenu du fichier **Twitter_data.csv**:

_id	id	Created_at	Name	Location	Followers	Messages
5fe229215cec9a82c1d34e8a	1341431554359345153	Tue Dec 22 17:12:59 +0000 2020	Vamo' Nessa CABR	Rio de Janeiro	595	RT @Hummingbrrrrd77: This is w
5fe229225cec9a82c1d34e8c	1341431554355077121	Tue Dec 22 17:12:59 +0000 2020	Kishor	nan	8	RT @Gaj79976684: "गरिबी" लुटी कय
5fe229225cec9a82c1d34e8e	1341431554543763456	Tue Dec 22 17:12:59 +0000 2020	Brenda	nan	110	RT @brbarry1: Veto stimulus bil
5fe229225cec9a82c1d34e90	1341431554673958913	Tue Dec 22 17:12:59 +0000 2020	Time for Change CA	NY	4836	RT @ErinBrockovich: No Covid r
5fe229225cec9a82c1d34e92	1341431554556440581	Tue Dec 22 17:12:59 +0000 2020	Casillas86	Loading...	1	@velardedaoiz2 El Reino Unido
5fe229225cec9a82c1d34e94	1341431554644398080	Tue Dec 22 17:12:59 +0000 2020	Niraj Sheth	Panvel, India	55	RT @ANI: A nationwide lockdow
5fe229225cec9a82c1d34e96	1341431554652864512	Tue Dec 22 17:12:59 +0000 2020	Oscar	American Canyon, California	165	RT @rashadthewizard: The rest
5fe229225cec9a82c1d34e98	1341431554816565250	Tue Dec 22 17:12:59 +0000 2020	Incognita CA (@ CA)	nan	100	RT @realTuckFrumper: Democr.
5fe229225cec9a82c1d34e9a	1341431554866868227	Tue Dec 22 17:12:59 +0000 2020	Sandra V. Fellous	Paris	24083	RT @mugiwaraleelo: Y'a 1 truc
5fe229225cec9a82c1d34e9c	1341431554812293121	Tue Dec 22 17:12:59 +0000 2020	Lee Becker	nan	586	Former @USNavy @NavyMedici
5fe229225cec9a82c1d34e9e	1341431554992726023	Tue Dec 22 17:12:59 +0000 2020	Wear A Mask	London	362	RT @Independent: Covid varian
5fe229225cec9a82c1d34ea0	1341431554946547715	Tue Dec 22 17:12:59 +0000 2020	Jérôme OLIVIER	Hauts-de-Seine, Ile-de-France	117	RT @AllanBARTÉ: Voilà qui devr
5fe229225cec9a82c1d34ea2	1341431555005292550	Tue Dec 22 17:12:59 +0000 2020	not inklessPW	nan	3541	now do this tweet but replace @
5fe229225cec9a82c1d34ea4	1341431555294515200	Tue Dec 22 17:13:00 +0000 2020	Rodolfo	nan	7163	RT @Gabyosoriohdz: Se embri
5fe229225cec9a82c1d34ea6	1341431555219202052	Tue Dec 22 17:13:00 +0000 2020	John Willow	Wisconsin, USA	1779	RT @michaeljknowles: Presiden
5fe229225cec9a82c1d34ea8	1341431555286167552	Tue Dec 22 17:13:00 +0000 2020	LovefromMoxie	nan	1120	RT @pcbrynn: This is how I expl
5fe229235cec9a82c1d34ea	1341431555168690176	Tue Dec 22 17:13:00 +0000 2020	The Blessed Cat Daddy	TEXAS, USA	3889	That was the plan from the begi
5fe229235cec9a82c1d34eac	1341431555256963072	Tue Dec 22 17:13:00 +0000 2020	DADAS	Provence-Alpes-Côte d'Azur	125	Bonsoir, non
5fe229235cec9a82c1d34eae	1341431555143680005	Tue Dec 22 17:12:59 +0000 2020	Carol	United States	812	EMT who gave CPR to man on fl
5fe229235cec9a82c1d34eab	1341431555374407682	Tue Dec 22 17:13:00 +0000 2020	FF	Buenos Aires	326	Y cuando digo lo peor que año,
5fe229235cec9a82c1d34eab	134143155538775690	Tue Dec 22 17:13:00 +0000 2020	🌞	Daxir (Daxir)	167	Je vais tout à fait jamais réunir

● Insertion des pays dans InfluxDB

Nous allons nettoyer la colonne **Location** qui représente la situation géographique puis ajouter la **Longitude**, **Latitude**, **Metric**.

On va extraire le nom du pays associé à la location et compter le nombre de fois qu'il se répète en suite lui ajouté ses coordonnées spatiales notamment Longitude, Latitude. Par la suite nous sauvegardons dans un fichier **train_geo_countries.csv**

```
In[2]: import pandas as pd
      import numpy as np
      from geopy.geocoders import Nominatim
      from influxdb import InfluxDBClient
      client = InfluxDBClient(host='localhost', port='8086')
      client.switch_database('covid19')
      data = pd.read_csv("countries_update_2.csv")
      pays = data["Country"].tolist()
      rest_pay = data["Country"].iloc[len(pays):].to_list()
      latitude = []
      longitude = []
      for name in pays:
          geolocator = Nominatim(user_agent="Agent1")
          location = geolocator.geocode(name)
          latitude.append(location.latitude)
          longitude.append(location.longitude)
```

```
In[2]: latitud = []
      longitud = []
      for name in rest_pay:
          geolocator = Nominatim(user_agent="Agent2")
          location = geolocator.geocode(name)
          latitud.append(location.latitude)
          longitud.append(location.longitude)
      location = pays[:1269] + rest_pay
      latitude = latitude + latitud
      longitude = longitude + longitud
      countries = pd.DataFrame([location, latitude, longitude], ["Location", "Latitude", "Longitude"]).T
      countries['Metric'] = countries.groupby('Location')['Location'].transform('count')
      countries.to_csv("train_geo_countries.csv")
```

on insert le résultat dans une base de données influxDB, comme le montre la figure ci-dessous :

```
In[3]: """ Insertion des pays de la donnée train dans influxdb pour la visualisation sur la map du monde """
      countries = pd.read_csv("train_geo_countries.csv")
      for row_ind, row in countries.iloc[0:].iterrows():
          json_body = [{
              'measurement' : 'train_country_tweets',
              'tags':{'Id':row[0]},
              'fields':{'
                  "name":row[1],
                  "latitude":row[2],
                  "longitude":row[3],
                  "metric":row[4]
              }
          }]
          client.write_points(json_body)
      print("done")
```

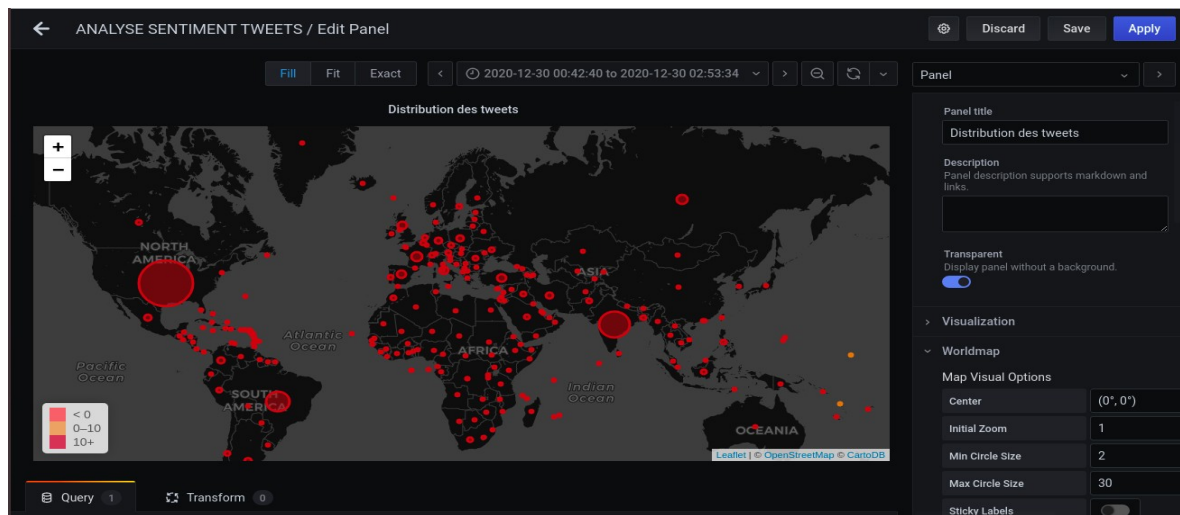
Par la figure ci dessous on peut voir les données dans la base de données InfluxDB :

```
> select * from train_country_tweets limit 10;
name: train_country_tweets
time                Id latitude      longitude      metric name
----                -  -
1611346836774449693 0  -10.333333300000001 -53.2         32  Brasil
1611346837445254114 1  22.3511148         78.6677428    66  India
1611346838021405243 2  22.3511148         78.6677428    66  India
1611346838077072178 3  46.603353999999996 1.8883334999999999 45  France
1611346838243120234 4  52.5310214         -1.2649062    118 England
1611346838377388869 5  37.09024          -95.712891    628 America
1611346838552038368 6  37.09024          -95.712891    628 America
1611346838668036044 7  37.09024          -95.712891    628 America
1611346838756012371 8  -72.8438691        0             1  Antarctica
1611346838975391452 9  52.500169799999995 5.7480820999999995 12  Holland
>
```

On connecte en paramétrant la base de donnée InfluxDB à Grafana pour visualiser la position de chaque personne qui a twitté.

Voici la localisation des utilisateurs qui ont twitté sur le covid19 lors de l'extraction :

On peut voir que plus d'utilisateurs en Amérique, Brésil et en Inde ont twitté sur le covid à cet instant.



Nous remarquons que la colonne **Messages** du fichier **Twitter_data.csv** contient les tweets en plusieurs langues et contient des caractères spéciaux que nous devons supprimer. D'abord chargeons les données puis ajoutons une colonne **sentiment** signifiant si le tweets est neutre , négatif (le tweet déplore le coronavirus) ou positive (le tweet parle en bien du coronavirus). Puis créons une colonne qui étiquette les sentiments en trois classes notamment **4 : positive**, **0 : négative** et **2 : neutre**

```
In[1]: #Fonction pour charger le jeu de données
def load_dataset(filename):
    dataset = pd.read_csv(filename, encoding='latin-1')
    dataset['sentiment'] = dataset["Messages"].apply(lambda x: sentiment.polarity_scores(str(x))['compound'])
    dataset['sentiment_label'] = dataset["sentiment"].apply(lambda x: 4 if (x > 0) else (0 if (x < 0) else 2))
    return dataset
```

La fonction suivante supprime les colonnes dont nous n'avons pas besoin dans notre étude.

```
In[3]: #Suppression des colonnes indésirables
def remove_unwanted_cols(dataset, columns):
    for column in columns:
        del dataset[column]
    return dataset
```

La fonction ci-dessous supprime les caractères spéciaux, les url, les ponctuations, les mots vides, les chiffres puis crée un sac de mots.

```

In[3]: #Pré-traitements des tweets
def preprocess_tweet_text(tweet):
    tweet = str(tweet).lower()
    # suppressions des urls
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', str(tweet), flags=re.MULTILINE)
    # Suppression des utilisateurs @ de references et les '#' venant des tweets
    tweet = re.sub(r'\@w+|\#', '', tweet)
    tweet = re.sub('[^A-Za-z0-9]+', '', tweet)
    # Suppressions des punctuations
    tweet = tweet.translate(str.maketrans('', '', string.punctuation))
    # Suppression des mots vides
    tweet_tokens = word_tokenize(tweet)
    filtered_words = [w for w in tweet_tokens if not w in stop_words]

    ps = PorterStemmer()
    stemmed_words = [ps.stem(w) for w in filtered_words]
    lemmatizer = WordNetLemmatizer()
    lemma_words = [lemmatizer.lemmatize(w, pos='a') for w in stemmed_words]

    return " ".join(filtered_words)

```

La fonction ci dessous permet de visualiser le nuage des mots :

```

In[2]: def world_cloud(data,column):
    tweet_words = ''
    stopwords = set(STOPWORDS)

    for val in data[column]:
        val = str(val)

        tokens = val.split()

        for i in range(len(tokens)):
            tokens[i] = tokens[i].lower()

        tweet_words += " ".join(tokens) + " "
    wordcloud = WordCloud(width=800, height=800,
                           background_color='black',
                           stopwords=stopwords,
                           min_font_size=10).generate(tweet_words)

    # Affichage du nuage de mot
    plt.figure(figsize=(8, 8), facecolor=None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad=0)

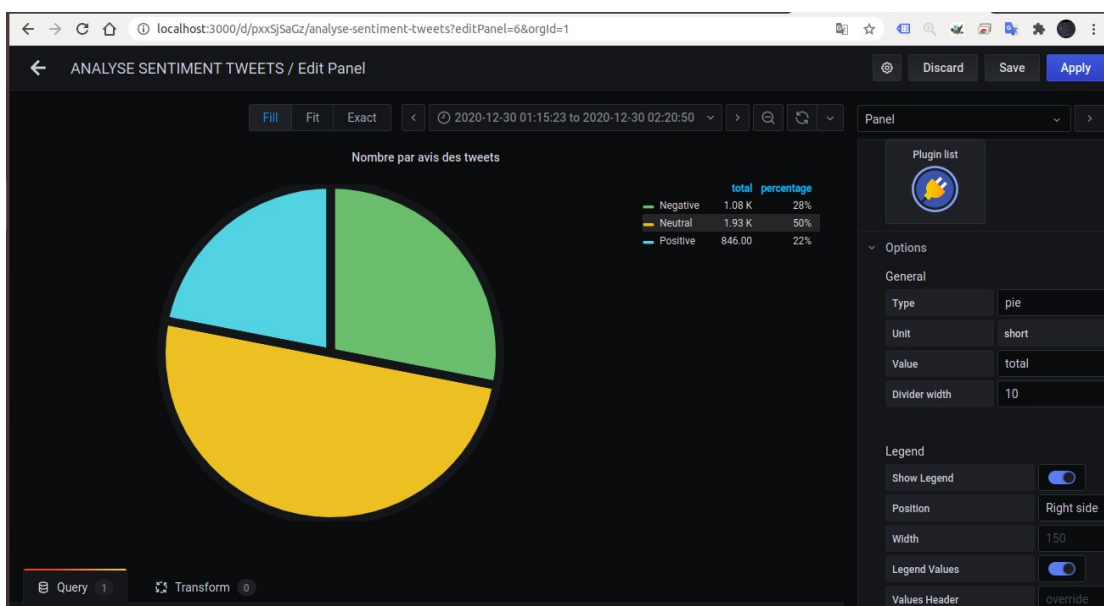
    plt.show()

```

Comme vous pouvez le voir ci dessous le nuage des mots :



Une représentation des données en diagramme circulaire en pourcentage nous donne une idée plus fine du volume de chaque classe.



On peut toute suite voir sur la figure ci dessous qu'on a **50 %** de tweets neutres, **28 %** de tweets négatifs et **22 %** de tweets positifs.

Représentons la distribution de la polarité des tweets avec un histogramme.



Nous allons entraîner ces données sur des modèles de machines learning.
Cette fonction permet de transformer les tokens en vecteurs numériques

```
In[2]: #Conversion des tokens en vecteurs
...: def get_feature_vector(train_fit):
...:     vector = TfidfVectorizer(sublinear_tf=True)
...:     vector.fit(train_fit)
...:     return vector
...:
```

Nous allons étiqueter le label en string :

```
In[2]: #Ré-étiquetons en string les labels.
...: def int_to_string(sentiment):
...:     if sentiment == 0:
...:         return "Negative"
...:     elif sentiment == 2:
...:         return "Neutral"
...:     else:
...:         return "Positive"
```

Chargeons le jeu de données , pour l'entraînement seul les colonnes **tweets** et les **sentiment_label** seront utilisés et les autres colonnes seront supprimés. Nous appliquons le prétraitement de la fonction **preprocess_tweet_text()**

```
In[2]: #Fonction pour charger le jeu de données
...: dataset = load_dataset("Twitter_data.csv")
...:
...: #dataset['sentiment'].to_csv("polarity.csv")
...:
...:
...: #Suppression des colonnes indésirables
...: n_dataset = remove_unwanted_cols(dataset, ['Unnamed: 0', '_Id', 'Id', 'Created_at', 'Name', 'Location', 'Followers', 'sentiment'])
...:
...:
...: #Pré-traitements des tweets
...: n_dataset.Messages = n_dataset['Messages'].apply(preprocess_tweet_text)
...:
```

Nous vectorisons les tweets en vecteurs numériques et les séparons des labels.

X : représente les variables, c'est les tweets

y : représente le label

```
In[2]: #Vectorisation des tweets en vecteurs numériques
...: tf_vector = get_feature_vector(np.array(n_dataset.iloc[:, 0]).ravel())
...: X = tf_vector.transform(np.array(n_dataset.iloc[:, 0]).ravel())
...: y = np.array(n_dataset.iloc[:, 1]).ravel()
...:
```

Construction du modèle en combinant plusieurs modèles d'apprentissage automatique.

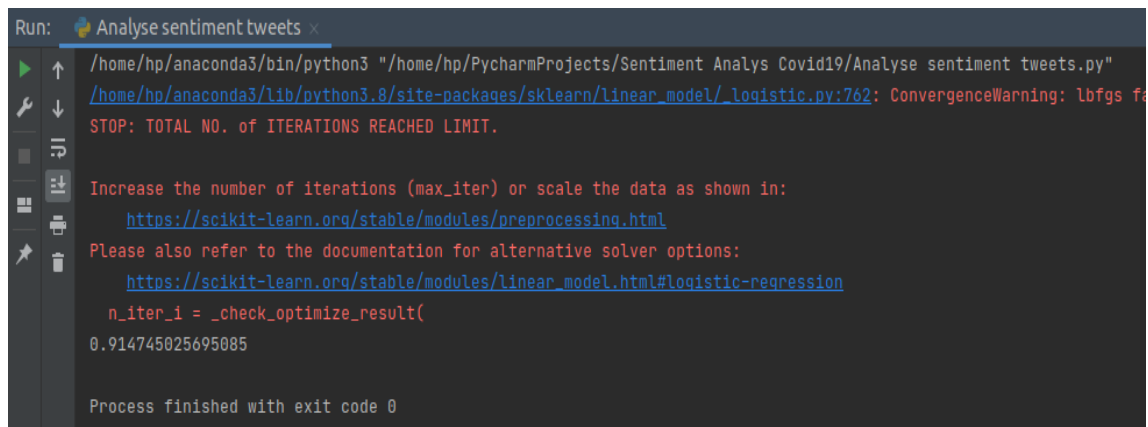
Le stacking est une technique d'apprentissage d'ensemble permettant de classer plusieurs modèles de classification via un méta-classificateur. Les modèles de classification individuels sont formés sur la base de l'ensemble de formation complet, ensuite, le méta-classificateur est ajusté en fonction des sorties méta-caractéristiques.

```
In[2]: # Création du modèle
...: estimators = [('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
...:               ('svr', LinearSVC(random_state=42)),
...:               ('NB', MultinomialNB()),
...:               ('SGDClassifier', SGDClassifier()),
...:               ('GBC', GradientBoostingClassifier()),
...:               ]
...: stack = StackingClassifier(estimators=estimators,
...:                           final_estimator=LogisticRegression())
...:
```

Divisons le jeu de données en 80 % de données d'apprentissage et 20 % de données de test.

```
In[2]: #Division du jeu de données en Train, Test
...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
...:
...:
...: stack.fit(X_train, y_train)
...: y_predict = stack.predict(X_test)
...: print(accuracy_score(y_test, y_predict))
...: |
```

Une fois le modèle entraîné appliquons le sur les données tests. Sur les données tests on obtient un score de : **91%** , ce qui signifie que à **91%** le modèle classe bien les nouveaux tweets entrant. La figure ci dessous montre le score du modèle obtenu.



```
Run: Analyse sentiment tweets x
/home/hp/anaconda3/bin/python3 "/home/hp/PycharmProjects/Sentiment Analys Covid19/Analyse sentiment tweets.py"
/home/hp/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
  https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
  https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
0.914745025695085

Process finished with exit code 0
```

CONCLUSION

Dans cette étude nous avons extraire des tweets que nous avons stocké dans différent base de donnée puis nous les avons pré traité et analysé. Il faut noter que le modèle entraîné pour classifier le sentiment du tweet nous donne un meilleur score. Mon objectif en procédant ainsi était de détecter en temps réel les tweets négatifs puis visualiser leur origine et évolution avec **grafana** mais j'étais limité en ressources matériels c'est à dire capacité d'ordinateur insuffissant. J'espère refaire un tutoriel encore plus sophistiqué et à temps réel une fois que j'aurai une bonne capacité d'ordinateur à pouvoir supporter tout les calculs découlant et les outils. Voici le lien du projet sur mon github et ci dessous le tableau de bord des visualisations.