DIBUJAR TWEETS EN UN MAPA CON PYTHON

Contenido

[Objetivo de la práctica 2](#_Toc48981437)

[Requisitos 2](#_Toc48981438)

[Instalación de requisitos: 2](#_Toc48981439)

[Comprobar que Python está instalado: 2](#_Toc48981440)

[Instalar Package Installer for Python (PIP): 3](#_Toc48981441)

[Instalar paquetes: 4](#_Toc48981442)

[Token de acceso a la API de Twitter 5](#_Toc48981443)

[Ejecución de la práctica 6](#_Toc48981444)

[Credentials.py 6](#_Toc48981445)

[I\_DescargarTweets 6](#_Toc48981446)

[II\_ConvertirTweets.py 10](#_Toc48981447)

[III\_GraficarMapa.py 12](#_Toc48981448)

[IV\_AnalisisSentimientos.py 17](#_Toc48981449)

## Objetivo de la práctica

Graficar los tweets de una ciudad o zona específica, así como aplicar un análisis de sentimientos para determinar su polaridad (negativa o positiva).

## Requisitos

**Lenguaje:**

* Python 3.8.x

**Herramientas y paquetes:**

* Package Installer for Python (PIP)
* Token de acceso a la API de Twitter
* Spyder IDE (OPCIONAL)
* tweepy 3.9.0
* pandas 1.0.5
* numpy 1.19.0
* ipython 7.16.1
* osmviz 3.2.0
* Pillow 7.2.0
* colour 0.1.5
* imageio 2.9.0
* lxml 4.5.2
* nltk 3.5
* scikit-learn 0.23.2
* langdetect 1.0.8
* textblob 0.15.3
* joblib 0.16.0
* heatmap
* langid

## Instalación de requisitos:

### Comprobar que Python está instalado:

1. Abrir el símbolo de sistema o terminal y ejecutar el comando:

Python --version

Debe mostrar la versión de Python instalada

Python 3.8.3

En caso contrario, descargar e instalar Python del siguiente enlace: <https://www.python.org/downloads/>

### Instalar Package Installer for Python (PIP):

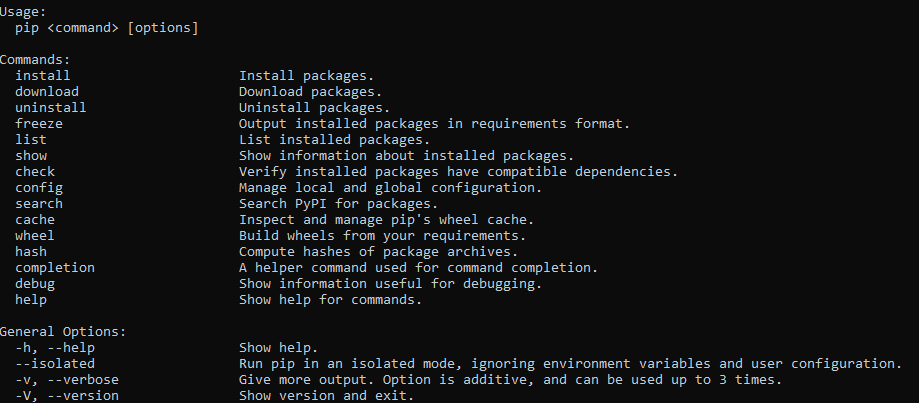
Es una herramienta de línea de comandos que permite instalar, reinstalar y desinstalar paquetes de Python.

Las últimas versiones de MacOS ya tienen integrado Python y PIP. En el caso de Windows, las últimas versiones de Python cuentan con PIP integrado.

Comprobar que PIP ya está instalado:

Abrir el símbolo del sistema o terminar y ejecutar el comando:

pip --help

Debe mostrar los comandos disponibles

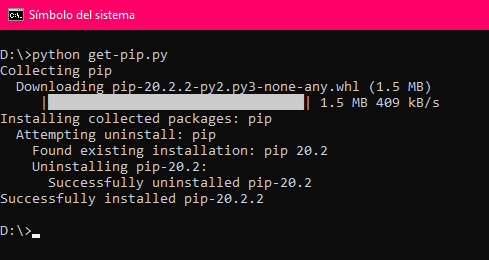
Esto quiere decir que ya tenemos PIP instalado, en caso contrario seguir con los siguientes pasos para Windows (si se tiene otro sistema operativo consultar: <https://www.neoguias.com/como-instalar-pip-python/#Como_instalar_PIP_en_Mac>):

1. Ir al siguiente enlace, dar clic derecho y guardar el script en la computadora

<https://bootstrap.pypa.io/get-pip.py>

1. Abrir el símbolo del sistema y navegar hasta el directorio en donde esta guardado el script anterior.
2. Ejecutar el siguiente comando:

python get-pip.py

Nos debe mostrar lo siguiente:

### Instalar paquetes:

#### Tweepy

Paquete que permite consumir la API de Twitter.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install tweepy

#### Pandas

Paquete que permite el análisis y la manipulación de datos.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install pandas

#### Numpy

Paquete de funciones matemáticas, vectorización, entre otros.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install numpy

#### Ipython

Shell interactivo que añade funcionalidades extra al modo interactivo incluido con Python.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install ipython

#### osmviz

Paquete que permite la visualización de OpenStreetMaps.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install osmviz

#### Pillow

Paquete para la manipulación de imágenes.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install Pillow

#### Colour

Paquete que permite convertir y manipular representaciones de color (RGB, HSL, …)

* Abrir el símbolo del sistema y ejecutar el comando:

pip install colour

#### Imageio

Paquete que permite leer y escribir un amplio rango de datos de imágenes.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install imageio

#### lxml

Paquete que permite la lectura de archivos XML, entre otros.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install lxml

#### nltk

Paquete para trabajar con datos de lenguaje natural, permite la tokenización. stemming, entre otros.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install nltk

#### Scikit-learn

Paquete para machine learning.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install scikit-learn

#### Langdetect

Paquete que permite detectar el idioma de un texto.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install langdetect

#### Textblob

Paquete para procesar datos textuales.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install textblob

#### Joblib

Paquete para programación paralela.

* Abrir el símbolo del sistema y ejecutar el comando:

pip install joblib

### Token de acceso a la API de Twitter

Se debe solicitar una cuenta de Twitter de desarrollador en el siguiente enlace:

<https://developer.twitter.com/en/apply-for-access>

Una vez aceptados, se nos proporcionará los siguientes datos de acceso a la API de Twitter: CONSUMER\_KEY, CONSUMER\_SECRET, ACCESS\_TOKEN, ACCESS\_SECRET.

## Ejecución de la práctica

A continuación, se mostrará el contenido, entradas y salidas de los scripts que constituyen a la práctica.

### Credentials.py

En este script solo se definen los datos de acceso a la API de Twitter.

CONSUMER\_KEY = "XXXXXXXXXX"

CONSUMER\_SECRET = "XXXXXXXX”

ACCESS\_TOKEN = "XXXXXXXXX"

ACCESS\_SECRET = "XXXXXXXXXX"

### I\_DescargarTweets

Este script permite descargar tweets en tiempo real de una ciudad o zona especificada.

* ***Paquetes necesarios.***

Entrada:

import json

import tweepy

from tweepy import Stream

from tweepy import OAuthHandler

from tweepy.streaming import StreamListener

* ***Importar los datos de acceso a la API de Twitter, definidos en el script Credentials.py.***

Entrada:

from credentials import \*

* ***Establecer acceso a la API de Twitter.***

Entrada:

auth = tweepy.OAuthHandler(CONSUMER\_KEY, CONSUMER\_SECRET)

auth.set\_access\_token(ACCESS\_TOKEN, ACCESS\_SECRET)

api = tweepy.API(auth)

* ***Definir las coordenadas del área del cual se quiere recuperar los tweets. En esta práctica las ciudades elegidas son Tampico y Ciudad Madero.***

Para consultar las coordenadas de una ciudad puede utilizar el siguiente sitio web: <https://www.geodatos.net/>



Entrada:

tampico\_madero = [-97.87777, 22.28519, -96.87777, 23.28519, -97.83623, 22.27228, -96.83623, 23.27228]

los primeros dos pares corresponden a las coordenadas de Tampico y los últimos 2 pares a Ciudad Madero.

\* Nota: las coordenadas deben proporcionarse de la siguiente manera:

[sw\_longitude, sw\_latitude, ne\_longitude, ne\_latitude]

* ***Definir la clase que permitirá el uso del Twitter Streaming API, el cual permite obtener tweets publicados en tiempo real.***

Entrada:

*class* listener(*tweepy*.*StreamListener*):

*def* \_\_init\_\_(self, numero\_tweets):

*self*.received\_tweets\_counter = 0

*self*.max\_number\_tweets = numero\_tweets

        #Directorio y nombre del archivo donde guardaremos los

        #tweets que se vayan publicando

*self*.file = open('tweets/tweets\_tampico\_madero.txt', 'a', encoding="UTF-8")

        super(listener, *self*).\_\_init\_\_()

*def* on\_data(self, data):

        if (*self*.received\_tweets\_counter < *self*.max\_number\_tweets):

*self*.received\_tweets\_counter += 1

            #La API de Twitter devuelve datos en formato JSON,

            #asi que hay que decodificarlos.

            try:

                decoded = json.loads(data)

            except Exception as e:

                print(e)

                return *True*

            #No todos los usuarios tienen habilitada la opcion de

#geolocalizacion

            #Por ello hay que dar formato a cuando no este disponible

            if decoded.get('geo') is not *None*:

                location = str(decoded.get('geo').get('coordinates'))

            else:

                location = '[,]'

            #Extraer los datos que nos interese de los tweets

            text = decoded['text'].replace('\n',' ')

            user = '@' + decoded.get('user').get('screen\_name')

            created = decoded.get('created\_at')

            #Escribir los tweets en el archivo de texto

*self*.file.write(user + "|" + location + "|" + created + "|" + text + "\n")

            return *True*

        else:

*self*.file.close()

            print('Done!')

            return *False*

    #def on\_status(self, status):

        #print(status.text)

*def* on\_error(self, status):

        print(status)

        #Debido a que el uso de la API de twitter tiene un limite diario

#debemos

        #desconectarnos del stream cuando excedamos dicho limite

        if status == 420:

            print('status code 420: ' + status)

*self*.file.close()

            #Retornando un False en on\_error Conseguimos desconectarnos de

#Stream

            return *False*

*self*.file.close()

* ***Comenzar a capturar los tweets en el archivo de texto (tweets\_tampico\_madero.txt) definido en la clase anterior.***

Entrada:

if \_\_name\_\_ == '\_\_main\_\_':

    print('Starting...')

    #Crear un Stream, estableciendo el número de

#tweets que se desean guardar

    twitterStream = tweepy.Stream(auth, listener(numero\_tweets = 1))

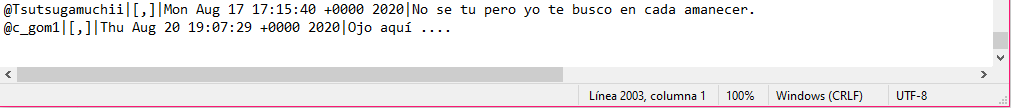
#Filtrar los tweets que coincidan con las coordenadas especificadas

    twitterStream.filter(locations = tampico\_madero)

Salida:

Starting…

Done!

Así como el archivo de texto con los tweets.



\* Nota 1: El proceso de filtrar y guardar tweets en el archivo de texto (tweets\_tampico\_madero.txt) puede tomar mucho tiempo en terminar, dependiendo del número de tweets establecidos para ser guardados.

\* Nota 2: Debido a que no todos los usuarios tienen habilitada la opción de “Acceso a la ubicación” en sus dispositivos, la información de las coordenadas viene vacía, por ello es recomendable guardar la mayor cantidad de tweets posibles para intentar conseguir un buen número de tweets con coordenadas.

Más información en: <https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/basic-stream-parameters>

### II\_ConvertirTweets.py

Este script convierte los tweets del archivo de texto (tweets\_tampico\_madero.txt) a un data frame y a un archivo CSV (tweets\_tampico\_madero.csv), para su posterior análisis y manipulación. También genera un archivo (tweets\_heatmap\_tam\_mad) que contiene solamente las latitudes y longitudes de cada tweet.

* ***Paquetes necesarios.***

Entrada:

import pandas as pd

import numpy as np

* ***Leer el archivo de texto (tweets\_tampico\_madero.txt) y definir un data frame para almacenar los tweets de ese archivo.***

Entrada:

tweets\_raw = pd.read\_table('tweets/tweets\_tampico\_madero.txt', header=*None*,iterator=*True*)

tweets\_2 = pd.DataFrame()

* ***Convertir los datos del archivo de texto (tweets\_tampico\_madero.txt) a un data frame y a un archivo CSV (tweets\_tampico\_madero.csv)***.

Entrada:

while 1:

    tweets = tweets\_raw.get\_chunk(1000) #1000 filas por chunk

    tweets.columns = ['tweets']

    tweets['len'] = tweets.tweets.apply(*lambda* x: len(x.split('|')))

    tweets[tweets.len < 4] = np.nan

    del tweets['len']

    tweets = tweets[tweets.tweets.notnull()]

    #Establecer las columnas y valores que tomarán

    tweets['user'] = tweets.tweets.apply(*lambda* x: x.split('|')[0])

    tweets['geo'] = tweets.tweets.apply(*lambda* x: x.split('|')[1])

    tweets['timestamp'] = tweets.tweets.apply(*lambda* x: x.split('|')[2])

    tweets['tweet'] = tweets.tweets.apply(*lambda* x: x.split('|')[3])

    tweets['lat'] = tweets.geo.apply(*lambda* x: x.split(',')[0].replace('[',''))

    tweets['lon'] = tweets.geo.apply(*lambda* x: x.split(',')[1].replace(']',''))

    del tweets['tweets']

    del tweets['geo']

    #Convertir las latitudes y longitudes de string a float

    tweets['lon'] = pd.to\_numeric(tweets['lon'], downcast="float")

    tweets['lat'] = pd.to\_numeric(tweets['lat'], downcast="float")

    #Cambiar la zona horaria de UTC a GMT-5

    tweets['timestamp'] = pd.to\_datetime(tweets['timestamp'], utc = *False*)

    tweets = tweets.set\_index('timestamp').tz\_convert('America/Mexico\_City').reset\_index()

    #Almacenar los tweets en el dataframe

    tweets\_2 = tweets\_2.append(tweets, ignore\_index = *True*)

    #Guardar los tweets en un archivo CSV

    tweets.to\_csv('tweets/tweets\_tampico\_madero.csv', mode='a', header=*False*,index=*False*)

Salida:

StopIteration

Así como un archivo CSV con los tweets

\* Nota: Ya que la cantidad de datos en el archivo de texto (tweets\_tampico\_madero.txt) es muy grande, se recomienda procesar cierta cantidad de filas, en lugar de cargar todo el archivo en memoria. En el ciclo while anterior se procesan 1000 filas del archivo de texto hasta que ya no haya más filas.

Dependiendo de la cantidad de tweets el proceso puede tardar en finalizar.

* ***Modificar el data frame con los tweets, para almacenar solo aquellos que tengan coordenadas y que a su vez se encuentren en el rango del área que deseamos.***

Entrada:

min\_lon = -97.99

min\_lat = 22.20

max\_lon = -96.0

max\_lat = 23.30

#Descartar las filas con latitud y longitud nulas

tweets\_2 = tweets\_2[(tweets\_2.lat.notnull()) & (tweets\_2.lon.notnull())]

#Filtrar los tweets que estén dentro de la zona que nos interese

tweets\_2 = tweets\_2[(tweets\_2.lon >= min\_lon) & (tweets\_2.lon <= max\_lon) & (tweets\_2.lat >= min\_lat) & (tweets\_2.lat <= max\_lat)]

* ***Guardar las coordenadas de los tweets en un archivo.***

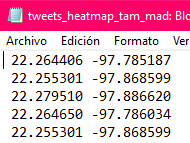
Entrada:

with open('heatmap-files/tweets\_heatmap\_tam\_mad','w') as file:

    file.write(tweets\_2[['lat','lon']].to\_string(header=*False*, index=*False*))

Salida:

Un archivo con las coordenadas de los tweets



### III\_GraficarMapa.py

Este script permite graficar las coordenadas del archivo (tweets\_heatmap\_tam\_mad) en un mapa, con ayuda del script heatmap.py

* ***Paquetes necesarios.***

Entrada:

import PIL

import osmviz

from PIL import Image

from PIL import ImageDraw, ImageFont

from IPython.display import Image

from IPython import get\_ipython

import numpy as np

import imageio

from colour import Color

import pandas as pd

from copy import deepcopy

* ***Graficar las coordenadas en un mapa.***

**OPCIÓN 1: Usando el símbolo del sistema**

Abrir el símbolo del sistema y navegar hasta el directorio de nuestro proyecto y ejecutar el siguiente comando.

Entrada:

python heatmap.py -o maps/map\_tweets\_tam\_mad\_01.png --width 1920 -p heatmap-files/tweets\_heatmap\_tam\_mad -b black -P equirectangular --osm --osm\_base http://tile.memomaps.de/tilegen --decay 0.8 -r 10 --zoom 0 --margin 15

-o Directorio y nombre del archivo.

--width Ancho de la imagen en pixeles.

-p Directorio y nombre del archivo que contiene las coordenadas.

--osm Especificar que se usará un mapa editable, OpenStreetMaps.

--osb\_base Estilo del mapa (tiles url). Puede consultar mas estilos en: <https://wiki.openstreetmap.org/wiki/Tile_servers>.

**OPCION 2:** Usando el IDE Spyder, Jupiter Qt console o Jupiter notebook

Simplemente ejecutar la siguiente línea

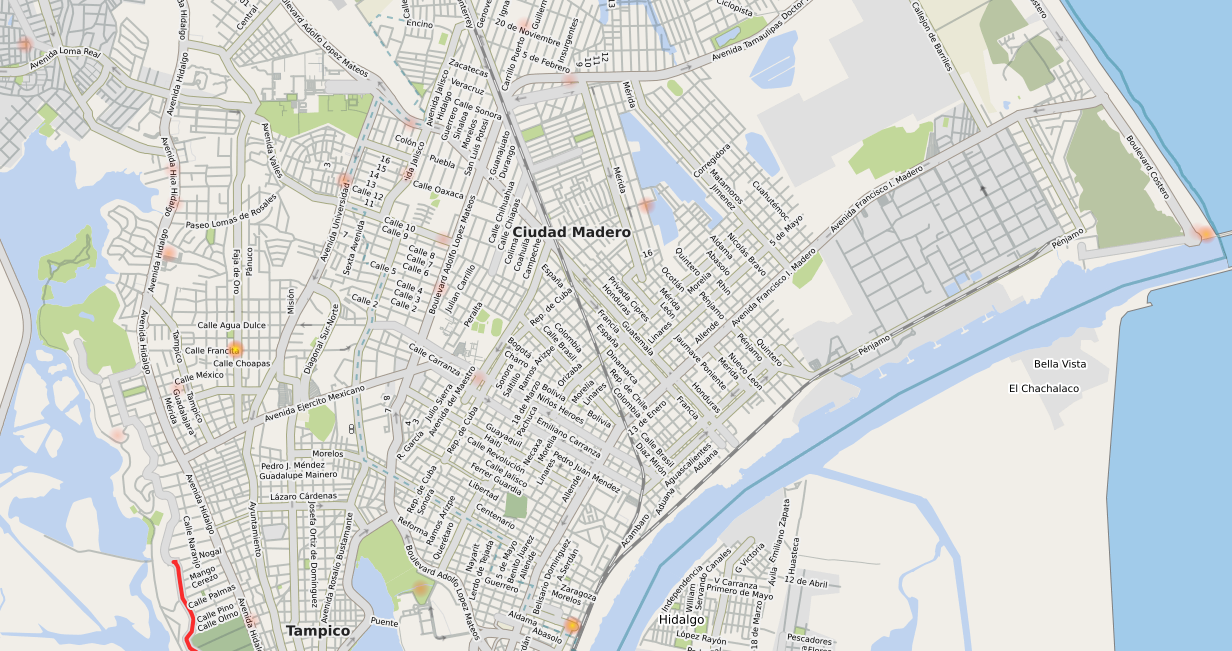
Entrada:

get\_ipython().system('python heatmap.py -o maps/map\_tweets\_tam\_mad\_01.png --width 1920 -p heatmap-files/tweets\_heatmap\_tam\_mad -b black -P equirectangular --osm --osm\_base http://tile.memomaps.de/tilegen --decay 0.8 -r 10 --zoom 0 --margin 15')

Salida:

Fetching tiles: 100%|█████████████████████████████████████████████████████████████████████████████████████████| 18/18 [00:00<00:00, 142.70tile/s]

Así como una imagen con las coordenadas graficadas en un mapa.



* ***Agregar una leyenda al mapa de los tweets; como el nombre de la ciudad, la cantidad de tweets graficados y la fecha***.

Entrada:

im = PIL.Image.open('maps/map\_tweets\_tam\_mad\_01.png')

draw = ImageDraw.Draw(im)

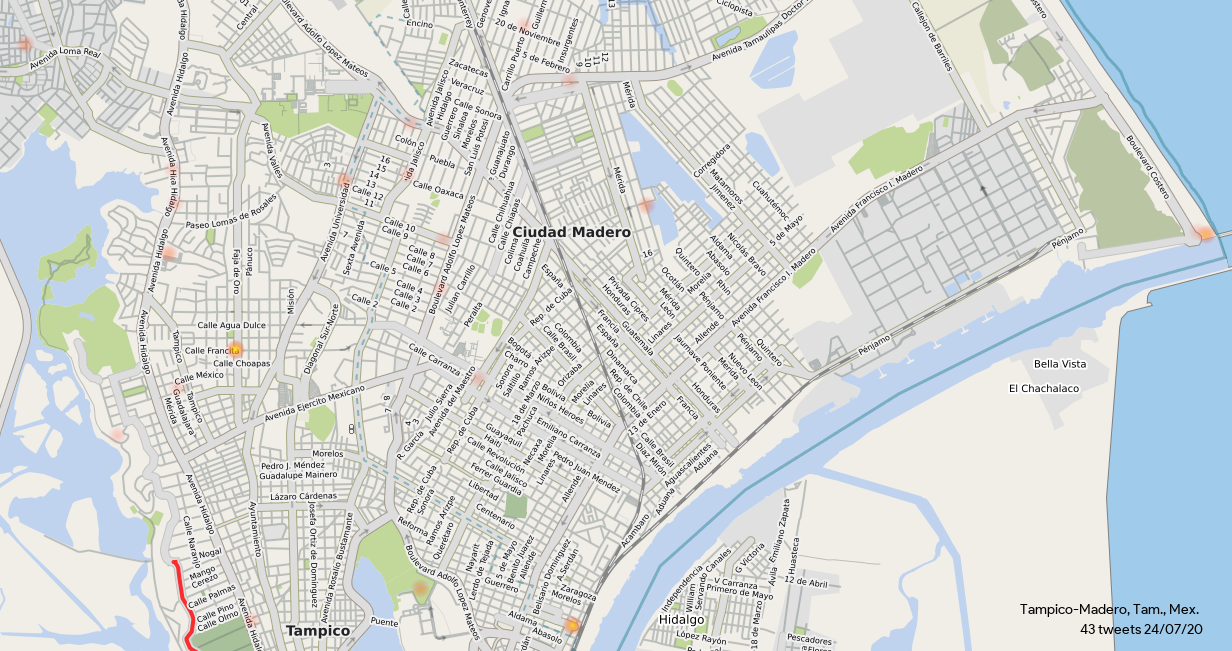
font = ImageFont.truetype("fonts/ProductSans.ttf", 14)

draw.text((1020, 600),"Tampico-Madero, Tam., Mex.", fill = "black", font=font)

draw.text((1080, 620),"43 tweets 24/07/20", fill = "black", font=font)

im.save('maps/map\_tweets\_tam\_mad\_01.png')

Salida:

La imagen de los puntos graficados en un mapa, pero ahora con el texto que agregamos.

* ***Definir un nuevo gradiente de color que sustituya los puntos amarillos que marca heatmap.py por defecto.***

Entrada:

hsva\_min = Color()

hsva\_min.hex\_l = '#008dcd'

hsva\_max = Color()

hsva\_max.hex\_l = '#4dccff'

color\_gradient = list(hsva\_max.range\_to(hsva\_min,256))

alpha = np.arange(0,256)[::-1]

gradient = []

for i, color\_point in enumerate(color\_gradient):

    rgb = list(color\_point.get\_rgb())

    rgb = [int(e \* 255) for e in rgb]

    rgb.append(alpha[i])

    gradient.append([rgb])

color\_gradient = np.array(gradient)

width = 43

from copy import deepcopy

color\_gradient\_row = deepcopy(color\_gradient)

for col in range(width-1):

    color\_gradient = np.hstack((color\_gradient, color\_gradient\_row))

imageio.imwrite('gradients/gradient\_blue.png', color\_gradient)

Salida:

Gradiente del color que definimos anteriormente



* ***Graficar las coordenadas utilizando el gradiente de color definido anteriormente.***

**OPCIÓN 1: Usando el símbolo del sistema**

Abrir el símbolo del sistema y navegar hasta el directorio de nuestro proyecto y ejecutar el siguiente comando.

Entrada:

python heatmap.py -G gradients/gradient\_blue.png -o maps/map\_tweets\_tam\_mad\_02.png --width 1920 -p heatmap-files/tweets\_heatmap\_tam\_mad -b black -P equirectangular --osm --osm\_base  https://basemaps.cartocdn.com/rastertiles/dark\_all --decay 0.8 -r 10 --zoom 0 --margin 15

-G Directorio y nombre de la imagen del gradiente.

**OPCION 2:** Usando el IDE Spyder, Jupiter Qt console o Jupiter notebook

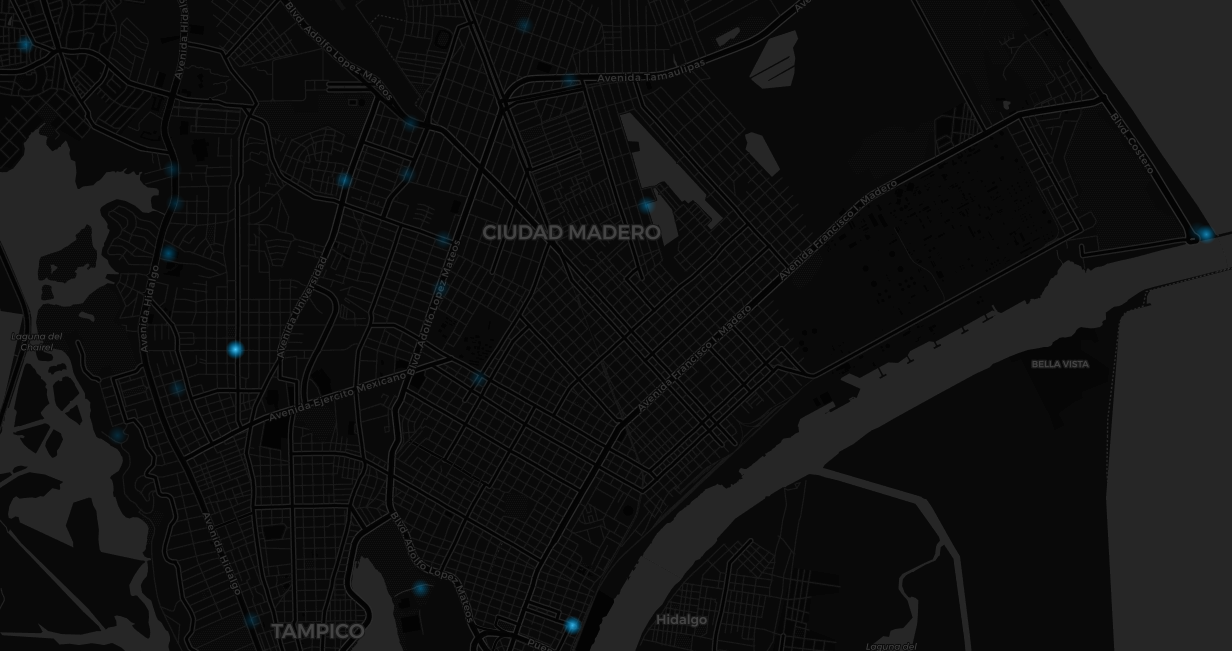
Simplemente ejecutar la siguiente línea

Entrada:

get\_ipython().system('python heatmap.py -G gradients/gradient\_blue.png -o maps/map\_tweets\_tam\_mad\_02.png --width 1920 -p heatmap-files/tweets\_heatmap\_tam\_mad -b black -P equirectangular --osm --osm\_base  https://basemaps.cartocdn.com/rastertiles/dark\_all --decay 0.8 -r 10 --zoom 0 --margin 15')

Salida:

Fetching tiles: 100%|█████████████████████████████████████████████████████████████████████████████████████████| 18/18 [00:00<00:00, 142.70tile/s]

Y un mapa con el nuevo color de puntos

### IV\_AnalisisSentimientos.py

En este script se predice si los tweets son positivos o negativos, mediante machine learning, para posteriormente ser graficados.

* ***Paquetes necesarios.***

Entrada:

import PIL

import osmviz

from PIL import Image

from PIL import ImageDraw, ImageFont

from IPython.display import Image

from IPython import get\_ipython

import numpy as np

import imageio

from colour import Color

from copy import deepcopy

import pandas as pd

import nltk

nltk.download("stopwords")

from nltk.corpus import stopwords

from string import punctuation

import sklearn

from sklearn.feature\_extraction.text import CountVectorizer

from nltk.stem import SnowballStemmer

from nltk.tokenize import word\_tokenize

from sklearn.model\_selection import cross\_val\_score

from sklearn.svm import LinearSVC

from sklearn.pipeline import Pipeline

import joblib

import langid

from langdetect import detect

import textblob

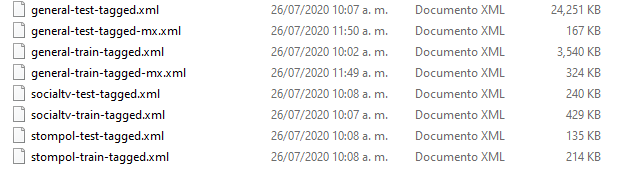
* ***Descargar los dataset que se usarán para predecir los sentimientos de los tweets.***

The Spanish Society for Natural Language Processing (SEPLN) proporciona varios dataset para el análisis de sentimientos de forma gratuita para propósitos académicos.

Para poder descargar los dataset debe registrarse en el siguiente enlace:

<http://tass.sepln.org/tass_data/download.php>

De cualquier forma, en la carpeta TASS-Dataset ya se encuentran descargados algunos archivos para el análisis de sentimientos.



* ***Convertir cada dataset descargado a un archivo CSV y a un data frame***

Entrada:

pd.set\_option('max\_colwidth',1000)

try:

    general\_tweets\_corpus\_train = pd.read\_csv('TASS-Dataset/general-train-tagged.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/general-train-tagged.xml', encoding="UTF-8"))

    root = xml.getroot()

    general\_tweets\_corpus\_train = pd.DataFrame(columns=('content', 'polarity', 'agreement'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [tweet.content.text, tweet.sentiments.polarity.value.text, tweet.sentiments.polarity.type.text]))

        row\_s = pd.Series(row)

        row\_s.name = i

        general\_tweets\_corpus\_train = general\_tweets\_corpus\_train.append(row\_s)

    general\_tweets\_corpus\_train.to\_csv('TASS-Dataset/general-train-tagged.csv', index=*False*, encoding='utf-8')

try:

    general\_tweets\_corpus\_test = pd.read\_csv('TASS-Dataset/general-test-tagged.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/general-test-tagged.xml', encoding="UTF-8"))

    root = xml.getroot()

    general\_tweets\_corpus\_test = pd.DataFrame(columns=('content', 'polarity'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [tweet.content.text, tweet.sentiments.polarity.value.text]))

        row\_s = pd.Series(row)

        row\_s.name = i

        general\_tweets\_corpus\_test = general\_tweets\_corpus\_test.append(row\_s)

    general\_tweets\_corpus\_test.to\_csv('TASS-Dataset/general-test-tagged.csv', index=*False*, encoding='utf-8')

try:

    stompol\_tweets\_corpus\_train = pd.read\_csv('TASS-Dataset/stompol-train-tagged.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/stompol-train-tagged.xml', encoding = "UTF-8"))

    root = xml.getroot()

    stompol\_tweets\_corpus\_train = pd.DataFrame(columns=('content', 'polarity'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [' '.join(list(tweet.itertext())), tweet.sentiment.get('polarity')]))

        row\_s = pd.Series(row)

        row\_s.name = i

        stompol\_tweets\_corpus\_train = stompol\_tweets\_corpus\_train.append(row\_s)

    stompol\_tweets\_corpus\_train.to\_csv('TASS-Dataset/stompol-train-tagged.csv', index=*False*, encoding='utf-8')

try:

    stompol\_tweets\_corpus\_test = pd.read\_csv('TASS-Dataset/stompol-test-tagged.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/stompol-test-tagged.xml', encoding = "UTF-8"))

    root = xml.getroot()

    stompol\_tweets\_corpus\_test = pd.DataFrame(columns=('content', 'polarity'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [' '.join(list(tweet.itertext())), tweet.sentiment.get('polarity')]))

        row\_s = pd.Series(row)

        row\_s.name = i

        stompol\_tweets\_corpus\_test = stompol\_tweets\_corpus\_test.append(row\_s)

    stompol\_tweets\_corpus\_test.to\_csv('TASS-Dataset/stompol-test-tagged.csv', index=*False*, encoding='utf-8')

try:

    social\_tweets\_corpus\_test = pd.read\_csv('TASS-Dataset/socialtv-test-tagged.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/socialtv-test-tagged.xml', encoding = "UTF-8"))

    root = xml.getroot()

    social\_tweets\_corpus\_test = pd.DataFrame(columns=('content', 'polarity'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [' '.join(list(tweet.itertext())), tweet.sentiment.get('polarity')]))

        row\_s = pd.Series(row)

        row\_s.name = i

        social\_tweets\_corpus\_test = social\_tweets\_corpus\_test.append(row\_s)

    social\_tweets\_corpus\_test.to\_csv('TASS-Dataset/socialtv-test-tagged.csv', index=*False*, encoding='utf-8')

try:

    social\_tweets\_corpus\_train = pd.read\_csv('TASS-Dataset/socialtv-train-tagged.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/socialtv-train-tagged.xml', encoding = "UTF-8"))

    root = xml.getroot()

    social\_tweets\_corpus\_train = pd.DataFrame(columns=('content', 'polarity'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [' '.join(list(tweet.itertext())), tweet.sentiment.get('polarity')]))

        row\_s = pd.Series(row)

        row\_s.name = i

        social\_tweets\_corpus\_train = social\_tweets\_corpus\_train.append(row\_s)

    social\_tweets\_corpus\_train.to\_csv('TASS-Dataset/socialtv-train-tagged.csv', index=*False*, encoding='utf-8')

try:

    general\_tweets\_corpus\_train\_mx = pd.read\_csv('TASS-Dataset/general-train-tagged-mx.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/general-train-tagged-mx.xml', encoding="UTF-8"))

    #sample tweet object

    root = xml.getroot()

    general\_tweets\_corpus\_train\_mx = pd.DataFrame(columns=('content', 'polarity'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [tweet.content.text, tweet.sentiment.polarity.value.text]))

        row\_s = pd.Series(row)

        row\_s.name = i

        general\_tweets\_corpus\_train\_mx = general\_tweets\_corpus\_train\_mx.append(row\_s)

    general\_tweets\_corpus\_train\_mx.to\_csv('TASS-Dataset/general-train-tagged-mx.csv', index=*False*, encoding='utf-8')

try:

    general\_tweets\_corpus\_test\_mx = pd.read\_csv('TASS-Dataset/general-test-tagged-mx.csv', encoding='utf-8')

except:

    from lxml import objectify

    xml = objectify.parse(open('TASS-Dataset/general-test-tagged-mx.xml', encoding="UTF-8"))

    root = xml.getroot()

    general\_tweets\_corpus\_test\_mx = pd.DataFrame(columns=('content', 'polarity'))

    tweets = root.getchildren()

    for i in range(0,len(tweets)):

        tweet = tweets[i]

        row = dict(zip(['content', 'polarity', 'agreement'], [tweet.content.text, tweet.sentiment.polarity.value.text]))

        row\_s = pd.Series(row)

        row\_s.name = i

        general\_tweets\_corpus\_test\_mx = general\_tweets\_corpus\_test\_mx.append(row\_s)

    general\_tweets\_corpus\_test\_mx.to\_csv('TASS-Dataset/general-test-tagged-mx.csv', index=*False*, encoding='utf-8')

* ***Concatenar los dataset anteriores en un solo data frame corpus.***

Entrada:

tweets\_corpus = pd.concat([

        social\_tweets\_corpus\_train,

        social\_tweets\_corpus\_test,

        stompol\_tweets\_corpus\_test,

        stompol\_tweets\_corpus\_train,

        general\_tweets\_corpus\_test,

        general\_tweets\_corpus\_train,

        general\_tweets\_corpus\_test\_mx,

        general\_tweets\_corpus\_train\_mx

    ])

* ***Modificar el dataframe corpus (tweets\_corpus) para que descarte los tweets con polaridad neutral.***

El data frame corpus (tweets\_corpus) tiene un nuevo campo, por tweet, llamado ‘agreement’, el cual solo puede tener dos posibles valores ‘AGREEMENT’ Y ‘DISAGREEMENT’. El valor ‘DISAGREEMENT’ corresponde a tweets con polaridad neutral.

Entrada:

tweets\_corpus = tweets\_corpus.query('agreement != "DISAGREEMENT" and polarity != "NONE"')

* ***Definir las funciones para la tokenización y stemming.***

Entrada:

#Eliminar enlaces

tweets\_corpus = tweets\_corpus[-tweets\_corpus.content.str.contains('^http.\*$')]

spanish\_stopwords = stopwords.words('spanish')

non\_words = list(punctuation)

non\_words.extend(['¿', '¡'])

non\_words.extend(map(str,range(10)))

stemmer = SnowballStemmer('spanish')

*def* stem\_tokens(tokens, stemmer):

    stemmed = []

    for item in tokens:

        stemmed.append(stemmer.stem(item))

    return stemmed

*def* tokenize(text):

    # Eliminar caracteres que no sean letras

    text = ''.join([c for c in text if c not in non\_words])

    # tokenize

    tokens =  word\_tokenize(text)

    # stem

    try:

        stems = stem\_tokens(tokens, stemmer)

    except Exception as e:

        print(e)

        print(text)

        stems = ['']

    return stems

* ***Preparar el modelo que evaluará los tweets.***

Agregar una nueva columna al data frame corpus (tweets\_corpus) en donde las polaridades de los tweets serán representadas de forma binaria.

Entrada:

tweets\_corpus = tweets\_corpus[tweets\_corpus.polarity != 'NEU']

tweets\_corpus['polarity\_bin'] = 0

tweets\_corpus.polarity\_bin[tweets\_corpus.polarity.isin(['P', 'P+'])] = 1

tweets\_corpus.polarity\_bin.value\_counts(normalize=*True*)

* ***Definir el modelo de evaluación, realizar un GridSearch para encontrar los hiperparámetros óptimos y guardar el modelo.***

Entrada:

vectorizer = CountVectorizer(

                analyzer = 'word',

                tokenizer = tokenize,

                lowercase = *True*,

                stop\_words = spanish\_stopwords)

pipeline = Pipeline([

    ('vect', vectorizer),

    ('cls', LinearSVC()),

])

parameters = {

    'vect\_\_max\_df': (0.5, 1.9),

    'vect\_\_min\_df': (10, 20,50),

    'vect\_\_max\_features': (500, 1000),

    'vect\_\_ngram\_range': ((1, 1), (1, 2)),  # unigrams or bigrams

    'cls\_\_C': (0.2, 0.5, 0.7),

    'cls\_\_loss': ('hinge', 'squared\_hinge'),

    'cls\_\_max\_iter': (500, 1000)

}

grid\_search = sklearn.model\_selection.GridSearchCV(pipeline, parameters, n\_jobs=-1 , scoring='roc\_auc')

grid\_search.fit(tweets\_corpus.content, tweets\_corpus.polarity\_bin)

grid\_search.best\_params\_

#Guardar el modelo

joblib.dump(grid\_search, 'grid\_search.pkl')

Salida:

El modelo

\* Nota: Debido a la enorme cantidad de datos en el data frame corpus (tweets\_corpus), el modelo puede tomar mucho tiempo en terminar. En un equipo con 6gb de RAM y con un procesador de 4 nucleos a 2.2GHz, finalizó tras 12 horas.

Si quiere omitir el proceso anterior, el modelo ya se encuentra en la carpeta del proyecto. Solo debe cargarlo de la siguiente manera:

grid\_search = joblib.load('grid\_search.pkl')

* ***Hacer una validación cruzada para mostrar el rendimiento del modelo.***

Entrada:

model = LinearSVC(C=.2, loss='squared\_hinge',max\_iter=1000,multi\_class='ovr',

              random\_state=*None*,

              penalty='l2',

              tol=0.0001

)

vectorizer = CountVectorizer(

    analyzer = 'word',

    tokenizer = tokenize,

    lowercase = *True*,

    stop\_words = spanish\_stopwords,

    min\_df = 50,

    max\_df = 1.9,

    ngram\_range=(1, 1),

    max\_features=1000

)

corpus\_data\_features = vectorizer.fit\_transform(tweets\_corpus.content)

corpus\_data\_features\_nd = corpus\_data\_features.toarray()

scores = cross\_val\_score(

    model,

    corpus\_data\_features\_nd[0:len(tweets\_corpus)],

    y=tweets\_corpus.polarity\_bin,

    scoring='roc\_auc',

    cv=5

    )

scores.mean()

Salida:

AUC

0.9168214861317802

* ***Convertir el archivo CSV con los tweets (tweets\_tampico\_madero.csv) en un dataframe y seleccionar aquellos que se encuentren dentro del área que nos interesa.***

Entrada:

tweets = pd.read\_csv('tweets/tweets\_tampico\_madero.csv', names = ["timestamp", "user", "tweet", "lat", "lon"],encoding='utf-8')

tweets = tweets[tweets.tweet.str.len() < 150]

tweets.lat = pd.to\_numeric(tweets.lat, errors='coerce')

tweets = tweets[tweets.lat.notnull()]

min\_lon = -97.99

min\_lat = 22.20

max\_lon = -96.0

max\_lat = 23.30

tweets = tweets[(tweets.lat.notnull()) & (tweets.lon.notnull())]

tweets = tweets[(tweets.lon > min\_lon) & (tweets.lon < max\_lon) & (tweets.lat > min\_lat) & (tweets.lat < max\_lat)]

* ***Detectar el idioma de los tweets, para posteriormente guardar aquellos que estén en español.***

Entrada:

*def* langid\_safe(tweet):

    try:

        return langid.classify(tweet)[0]

    except Exception as e:

        pass

*def* langdetect\_safe(tweet):

    try:

        return detect(tweet)

    except Exception as e:

        pass

*def* textblob\_safe(tweet):

    try:

        return textblob.TextBlob(tweet).detect\_language()

    except Exception as e:

        pass

#Puede tomar un tiempo en terminar

tweets['lang\_langid'] = tweets.tweet.apply(langid\_safe)

tweets['lang\_langdetect'] = tweets.tweet.apply(langdetect\_safe)

tweets['lang\_textblob'] = tweets.tweet.apply(textblob\_safe)

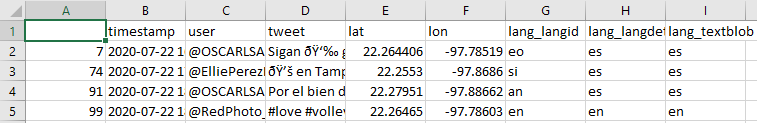
tweets['lang\_textblob'] = tweets.tweet.apply(textblob\_safe)

* ***Exportar el data frame de los tweets con el campo de identificación de idioma a un archivo CSV (tweets\_tampico\_madero\_2.csv).***

Entrada:

tweets.to\_csv('heatmap-files/tweets\_tampico\_madero\_2.csv', encoding='utf-8')

Salida:

Un archivo CSV con los tweets y su identificador de idioma



* ***Modificar el data frame de los tweets para solo guardar aquellos que estén en idioma español.***

Entrada:

tweets = tweets.query(''' lang\_langdetect == 'es' or lang\_langid == 'es' or lang\_textblob == 'es'  ''')

* ***Utilizar el modelo entrenado para predecir los sentimientos de lo tweets y exportar el data frame resultante a un archivo CSV (tweets\_polarity\_bin.csv) y a un archivo de texto (tweets\_heatmap\_polarity\_binary) para ser graficado posteriormente.***

Entrada:

pipeline = Pipeline([

    ('vect', CountVectorizer(

            analyzer = 'word',

            tokenizer = tokenize,

            lowercase = *True*,

            stop\_words = spanish\_stopwords,

            min\_df = 50,

            max\_df = 1.9,

            ngram\_range=(1, 1),

            max\_features=1000

            )),

    ('cls', LinearSVC(C=.2, loss='squared\_hinge',max\_iter=1000,multi\_class='ovr',

             random\_state=*None*,

             penalty='l2',

             tol=0.0001

             )),

])

pipeline.fit(tweets\_corpus.content, tweets\_corpus.polarity\_bin)

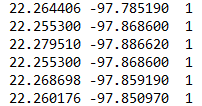
tweets['polarity'] = pipeline.predict(tweets.tweet)

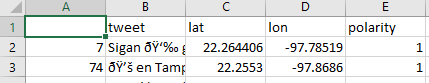
tweets[['tweet', 'lat', 'lon', 'polarity']].to\_csv('heatmap-files/tweets\_polarity\_bin.csv', encoding='utf-8')

with open('heatmap-files/tweets\_heatmap\_polarity\_binary','w') as file:

    file.write(tweets[['lat','lon', 'polarity']].to\_string(header=*False*, index=*False*))

Salida:





* ***Definir dos nuevos gradientes de color para identificar a los tweets positivos (verde) y los negativos (rojo).***

Entrada:

# Verde = tweets positivos

hsva\_min = Color()

hsva\_min.hex\_l = '#24b736'

hsva\_max = Color()

hsva\_max.hex\_l = '#24b736'

color\_gradient = list(hsva\_max.range\_to(hsva\_min,256))

alpha = np.arange(0,256)[::-1]

gradient = []

for i, color\_point in enumerate(color\_gradient):

    rgb = list(color\_point.get\_rgb())

    rgb = [int(e \* 255) for e in rgb]

    rgb.append(alpha[i])

    gradient.append([rgb])

color\_gradient = np.array(gradient)

width = 43

color\_gradient\_row = deepcopy(color\_gradient)

for col in range(width-1):

    color\_gradient = np.hstack((color\_gradient, color\_gradient\_row))

imageio.imwrite('gradients/gradient\_green.png', color\_gradient)

# Rojo = tweets negativos

hsva\_min = Color()

hsva\_min.hex\_l = '#ff3639'

hsva\_max = Color()

hsva\_max.hex\_l = '#ff3639'

color\_gradient = list(hsva\_max.range\_to(hsva\_min,256))

alpha = np.arange(0,256)[::-1]

gradient = []

for i, color\_point in enumerate(color\_gradient):

    rgb = list(color\_point.get\_rgb())

    rgb = [int(e \* 255) for e in rgb]

    rgb.append(alpha[i])

    gradient.append([rgb])

color\_gradient = np.array(gradient)

width = 43

color\_gradient\_row = deepcopy(color\_gradient)

for col in range(width-1):

    color\_gradient = np.hstack((color\_gradient, color\_gradient\_row))

imageio.imwrite('gradients/gradient\_red.png', color\_gradient)

 Salida:

* ***Convertir el archivo de texto de las coordenadas y polaridades (tweets\_heatmap\_polarity\_binary) en un data frame.***

Entrada:

tweets = pd.read\_table('heatmap-files/tweets\_heatmap\_polarity\_binary', encoding='utf-8', sep=' ', header=*None*)

del tweets[tweets.columns[0]]

del tweets[tweets.columns[2]]

tweets.columns = ['lat', 'lon', 'polarity']

* ***Exportar las coordenadas con polaridad negativa a un archivo de texto.***

Entrada:

with open('heatmap-files/tweets\_with\_polarity\_negative','w') as file:

    file.write(tweets[['lat','lon']][tweets.polarity==0].to\_string(header=*False*, index=*False*))

Salida:

* ***Exportar las coordenadas con polaridad positiva a un archivo de texto.***

Entrada:

with open('tweets\_with\_polarity\_positive','w') as file:

    file.write(tweets[['lat','lon']][tweets.polarity==1].to\_string(header=*False*, index=*False*))

Salida:

* ***Graficar la capa de tweets negativos.***

**OPCIÓN 1:** Usando el símbolo del sistema

Abrir el símbolo del sistema y navegar hasta el directorio de nuestro proyecto y ejecutar el siguiente comando.

Entrada:

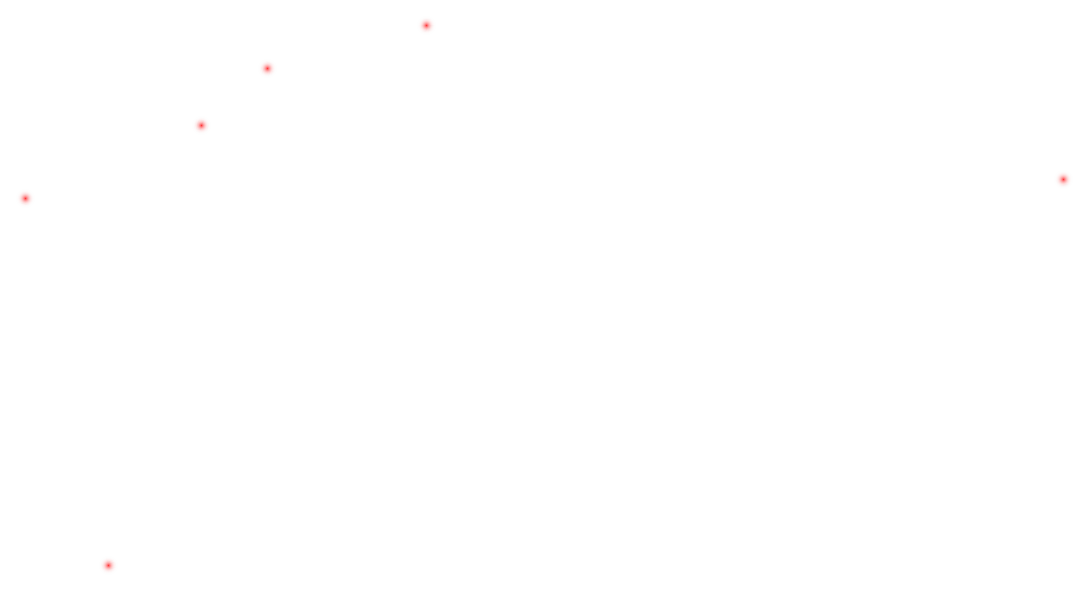
python heatmap.py -o maps/tampico\_madero\_tweets\_polarity\_negative.png --width 1920 -p heatmap-files/tweets\_with\_polarity\_negative -P equirectangular --osm --osm\_base https://basemaps.cartocdn.com/rastertiles/light\_all --decay 0.8 -r 10 --zoom 0 --margin 15 -G gradients/gradient\_red.png --layer

**OPCION 2:** Usando el IDE Spyder, Jupiter Qt console o Jupiter notebook

Simplemente ejecutar la siguiente línea

Entrada:

get\_ipython().system('python heatmap.py -o maps/tampico\_madero\_tweets\_polarity\_negative.png --width 1920 -p heatmap-files/tweets\_with\_polarity\_negative -P equirectangular --osm --osm\_base https://basemaps.cartocdn.com/rastertiles/light\_all --decay 0.8 -r 10 --zoom 0 --margin 15 -G gradients/gradient\_red.png --layer')

Salida:

* ***Graficar la capa de tweets positivos.***

**OPCIÓN 1:** Usando el símbolo del sistema

Abrir el símbolo del sistema y navegar hasta el directorio de nuestro proyecto y ejecutar el siguiente comando.

Entrada:

python heatmap.py -o maps/tampico\_madero\_tweets\_polarity\_positive.png --width 1920 -p heatmap-files/tweets\_with\_polarity\_positive -P equirectangular --osm --osm\_base https://basemaps.cartocdn.com/rastertiles/light\_all --decay 0.8 -r 10 --zoom 0 --margin 15 -G gradients/gradient\_green.png

**OPCION 2:** Usando el IDE Spyder, Jupiter Qt console o Jupiter notebook

Simplemente ejecutar la siguiente línea

Entrada:

get\_ipython().system('python heatmap.py -o maps/tampico\_madero\_tweets\_polarity\_positive.png --width 1920 -p heatmap-files/tweets\_with\_polarity\_positive -P equirectangular --osm --osm\_base https://basemaps.cartocdn.com/rastertiles/light\_all --decay 0.8 -r 10 --zoom 0 --margin 15 -G gradients/gradient\_green.png')

Salida:

* ***Combinar las imágenes de los tweets postitvos y negativos en un solo archivo.***

Entrada:

background = PIL.Image.open('maps/tampico\_madero\_tweets\_polarity\_positive.png')

foreground = PIL.Image.open('maps/tampico\_madero\_tweets\_polarity\_negative.png')

background.paste(foreground, (0, 0), foreground)

draw = ImageDraw.Draw(background)

font = ImageFont.truetype("fonts/ProductSans.ttf", 14)

draw.text((1020, 600),"Tampico-Madero, Tam., Mex.", fill = "black", font=font)

draw.text((1020, 620),"Tweets positivos y negativos", fill = "black", font=font)

background.save('maps/tampico\_madero\_tweets\_polarity.png')

Salida:

