Measuring Social Influence at Instagram

Project 1.1

Data Engineering 9A

Ing. Fis. Didier Gamboa Angulo

Trends in Data Science

Student:

Adrian Roberto Carmona Rodriguez
Erik Hernandez Hernandez
Isabel Camara Montalvo
Adriana Maribel Ku Huchim
Luis Fernando Koh Avila
Alfredo Alexander Paz Martinez

Universidad Politécnica de Yucatán

22th September 2022

Introduction

Nowadays, networks play an important role in our daily lives. The impact of instagram in the world is much stronger than we imagined, this is invading all sectors, and for those who started as a network for photography lovers and wanted to avoid text, it is now clear that it is a portal for brands and people to get a status either in their personal life or a large-scale digital marketing.

The accelerated growth of this platform increases the interest on the marketing strategies carried out by the companies that make themselves known on instagram.

Analyzing the profile of instagram users and their behavior on the application, will allow an approximation of the users that make up the platform, as well as analyzing the impact on them generated by the advertising that appears in a relatively trusting environment for the users appears in an environment of relative trust for the users according to the number of likes, publications, etc.

Objectives

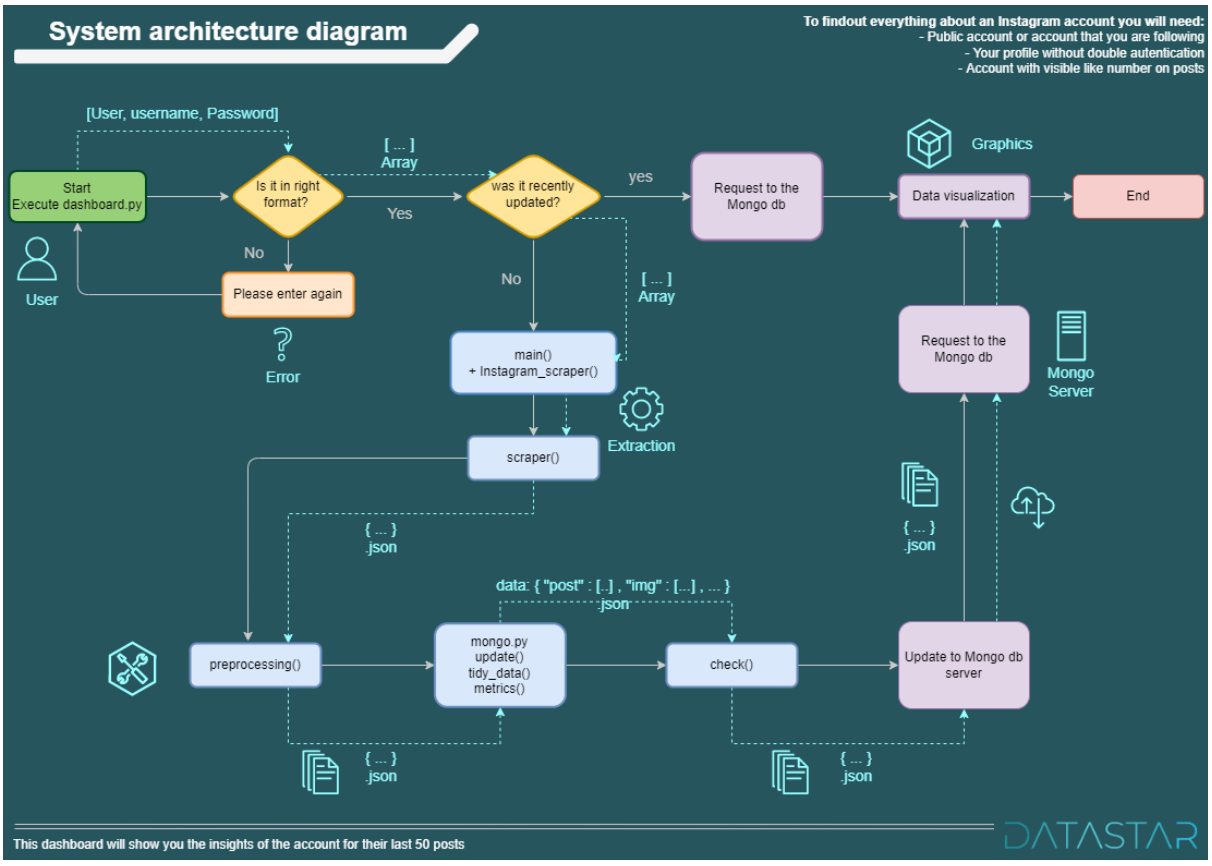
This project will be confirmed by the design of a web application which will have certain qualities to be able to extract information about any instagram user, and with this, use it to measure their content with respect to their followers, publications and reach that may come to have.

The features of this application will perform the following actions:

- Receive user account to be analyzed.
- Retrieve information from the given user account (Web Scraping).
- Store information in a database.
- Show social influence metrics into an application.

In the same way this project will help us to develop our skills corresponding to topics that we have seen throughout the career as data cleaning, database connection and at the end a good projection regarding visualization which will give us a focus to be able to deduce analysis with respect to that extracted information.

System Architecture


System architecture diagram
To findout everything about an Instagram account you will need:
- Public account or account that you are following
- Your profile without double autentication
- Account with visible like number on posts
[User, username, Password]
Start
Execute dashboard.py
User
Is it in right format?
No
Please enter again
Error
[...]
Array
Yes
was it recently updated?
yes
No
[...]
Array
Request to the Mongo db
Graphics
Data visualization
End
Request to the Mongo db
Mongo Server
main()
+ Instagram_scraper()
Extraction
scraper()
{ ... }
json
data: { "post" : [..] , "img" : [...] , ... }
json
preprocessing()
mongo.py
update()
tidy_data()
metrics()
check()
Update to Mongo db server
{ ... }
json
{ ... }
json
{ ... }
json
This dashboard will show you the insights of the account for their last 50 posts
DATASTAR

In the first part we have the dashboard entry, in which it validates the user and password which, if correct, continues the flow, otherwise it returns to the starting point. From there it checks in the mongo storage cache if the user has been called before, shows the display of the same and the process ends. If the user has not been called before, it enters the main.py, and they begin to work the scraper function to extract the information of the user defined previously; after, started the function of the preprocessing that makes cleaning of the data, makes the metrics to show, and makes the shipment of the data already prepared, and makes an entrance of the information to the mongoDB uploading the data prepared to be used by our method of visualization that is the streamlite to show the metrics, graphs and the information broken to analyze the information.

Tecnologies

Selenium

Selenium is an open source tool for test automation of web browsers, Selenium provides a tool with which we can record and / or playback, edit and debug test cases, which will allow us to run the tests repeatedly as many times as necessary.

Selenium IDE (Integrated Development Environment), also known as Integrated Development Environment, this component is an automation tool that allows us to record, edit and debug tests, it is also known as Selenium Recorder.

This tool allows developers to save time and effort when they have to resolve an issue and have had to generate a new version, so the test automation will allow them to perform specific tests on the development and see that what previously worked has not broken.

Selenium Webdriver: it is a tool that allows to automate UI (User Interface) testing of Web applications but it is based on a more modern and stable approach than the Selenium RC version, so Webdriver unlike RC does not use middleware but controls the browser communicating directly with it, some of the languages that are supported are:

- Java
- C#
- Python
- Ruby
- PHP
- JavaScript

MongoDB

MongoDB is an open source document-oriented NoSQL database system written in C++, which instead of storing data in tables stores it in BSON data structures (similar to JSON) with a dynamic schema.

MongoDB would undoubtedly be the speed, which achieves a perfect balance between performance and functionality thanks to its content query system. But its main features are not limited to this, MongoDB also has other features that make it the preferred choice of many developers.

Main features:

- Ad Hoc Queries
- Indexing
- Replication
- Load Balancing
- File Storage
- Server-side Javascript execution

Python

Python is a high-level programming language used to develop applications of all kinds. Unlike other languages such as Java or .NET, it is an interpreted language, that is, it is not necessary to compile it to run applications written in Python, but they are executed directly by the computer using a program called interpreter, so it is not necessary to "translate" it to machine language.

Results

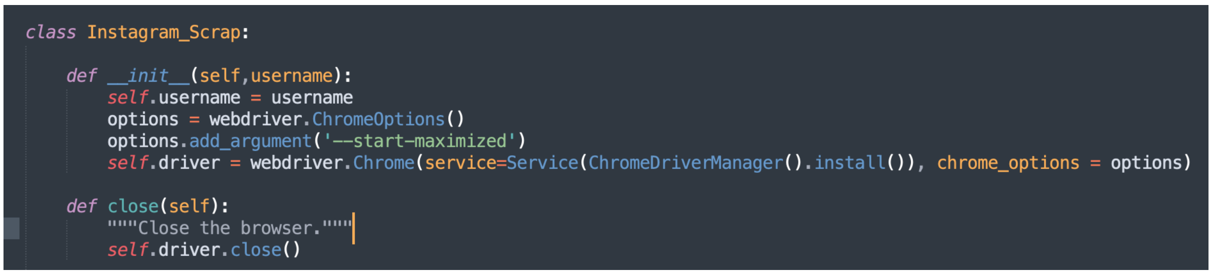
Web Scraping Functions

For the instagram web scraper we follow the following structure where we call different functions with an specific task to perform the following:

Class Instagram_Scrap: Class that contains all the functions needed to scrap an instagram profile.

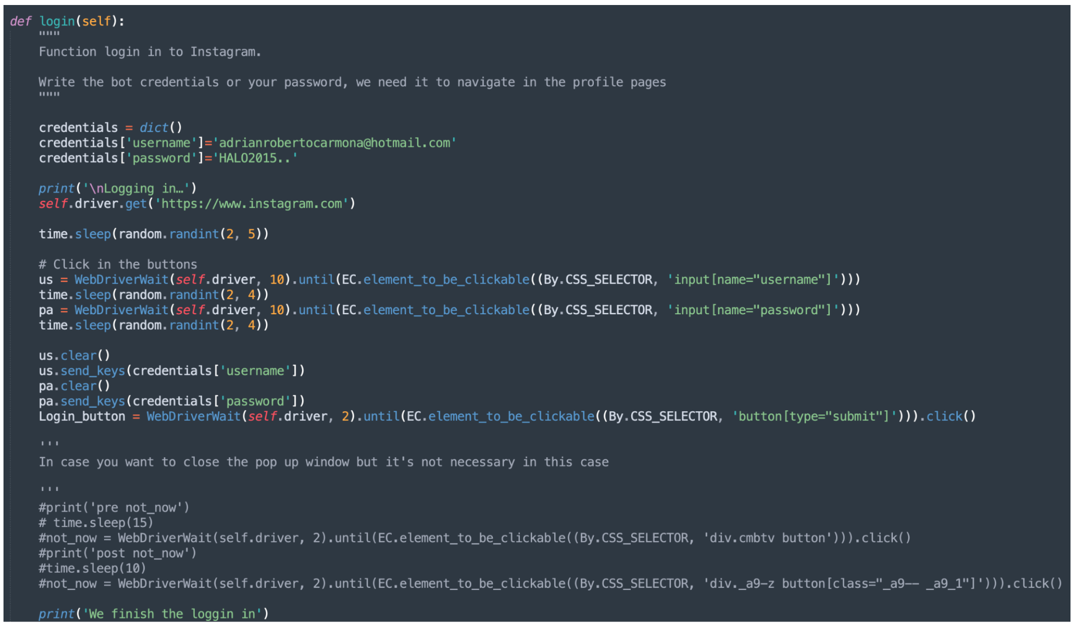
- **Init**
  This function is the first one to execute since it is the one who downloads the chromedriver that will let us open the Chrome navigator. After install it executes the driver opening the browser.

- **close**
  This function closes the current navigator that has been used to scrap the profile.



```python
class Instagram_Scrap:

    def __init__(self,username):
        self.username = username
        options = webdriver.ChromeOptions()
        options.add_argument('--start-maximized')
        self.driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), chrome_options = options)

    def close(self):
        """Close the browser."""
        self.driver.close()
```

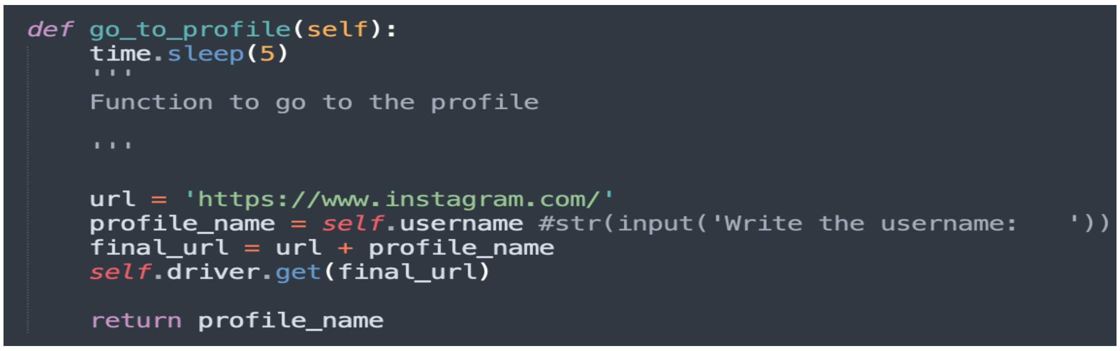
- **login**
  This function will receive the input credentials from the user in the dashboard, it will store the password and username so the scrapper can do its processes with that account.
  Given that we are into the main google page, it will be necessary to open the instagram website through the website to put the credentials. Once there the next step will be to select the selectors and input the credentials, after that the login button will be pushed.



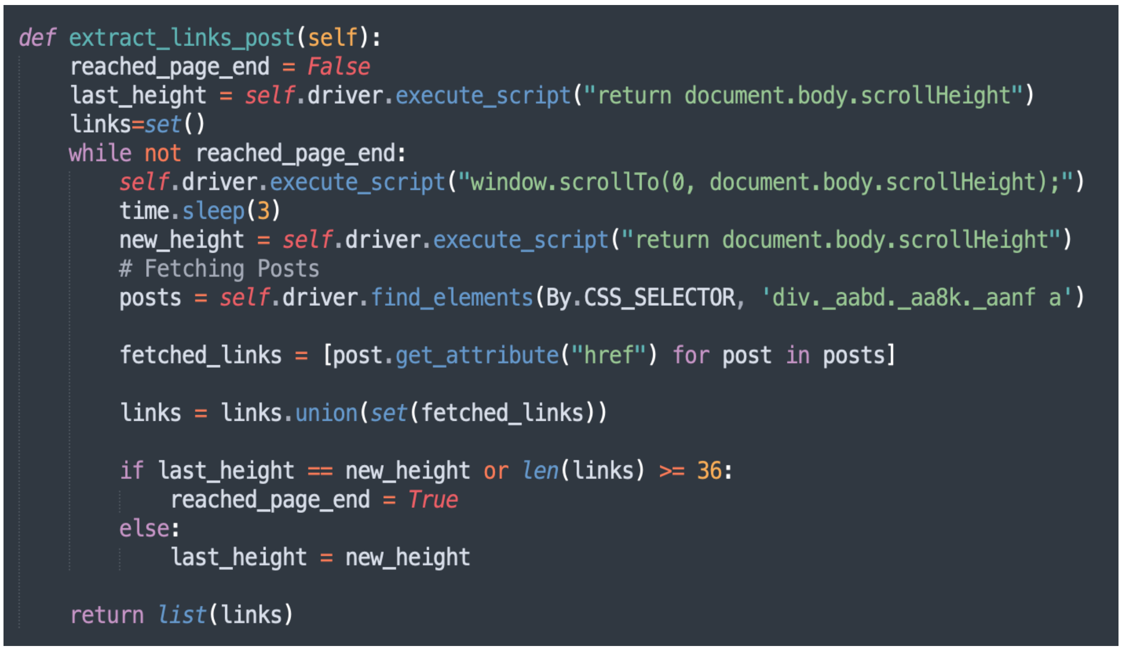
```python
def login(self):
    """
    Function login in to Instagram.

    Write the bot credentials or your password, we need it to navigate in the profile pages
    """

    credentials = dict()
    credentials['username']='adrianrobertocarmona@hotmail.com'
    credentials['password']='HALO2015..'

    print('\nLogging in…')
    self.driver.get('https://www.instagram.com')

    time.sleep(random.randint(2, 5))

    # Click in the buttons
    us = WebDriverWait(self.driver, 10).until(EC.element_to_be_clickable((By.CSS_SELECTOR, 'input[name="username"]')))
    time.sleep(random.randint(2, 4))
    pa = WebDriverWait(self.driver, 10).until(EC.element_to_be_clickable((By.CSS_SELECTOR, 'input[name="password"]')))
    time.sleep(random.randint(2, 4))

    us.clear()
    us.send_keys(credentials['username'])
    pa.clear()
    pa.send_keys(credentials['password'])
    Login_button = WebDriverWait(self.driver, 2).until(EC.element_to_be_clickable((By.CSS_SELECTOR, 'button[type="submit"]'))).click()

    '''
    In case you want to close the pop up window but it's not necessary in this case

    '''
    #print('pre not_now')
    # time.sleep(15)
    #not_now = WebDriverWait(self.driver, 2).until(EC.element_to_be_clickable((By.CSS_SELECTOR, 'div.cmbtv button'))).click()
    #print('post not_now')
    #time.sleep(10)
    #not_now = WebDriverWait(self.driver, 2).until(EC.element_to_be_clickable((By.CSS_SELECTOR, 'div._a9-z button[class="_a9-- _a9_1"]'))).click()

    print('We finish the loggin in')
```

- **go_to_profile**
  The purpose of this function is to redirect the scrapper to the profile we want to scrap, for that reason it will take the argument inputted by the user in the

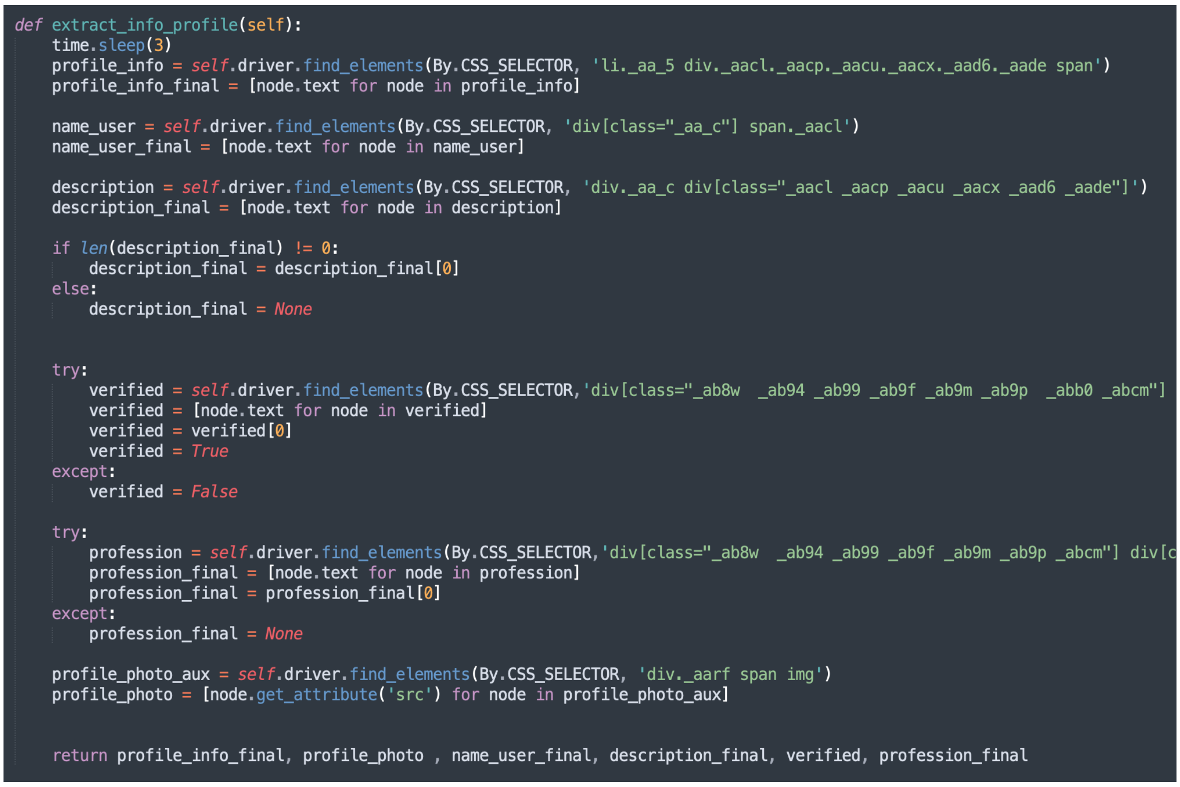
dashboard the function will sum the strings ‘https://www.instagram.com/’ + ‘username’ and through the driver it will go to that page, in case the user does not exists it will not do anything until a valid username is typed.


```python
def go_to_profile(self):
    time.sleep(5)
    '''
    Function to go to the profile

    '''

    url = 'https://www.instagram.com/'
    profile_name = self.username #str(input('Write the username:   '))
    final_url = url + profile_name
    self.driver.get(final_url)

    return profile_name
```

- **extract_links_post**
  This function will get the current structure of the page when opening and it will define its height on a variable called last height. The purpose of this is to scroll down until there are no more changes on the page knowing that it has reached the end. The function has two stop conditions, the first one is to check that the page does not change anymore which will mean that we have reached the limit, on the other hand the second one is that we have reached the defined number of links (of photos) we want to scrap.

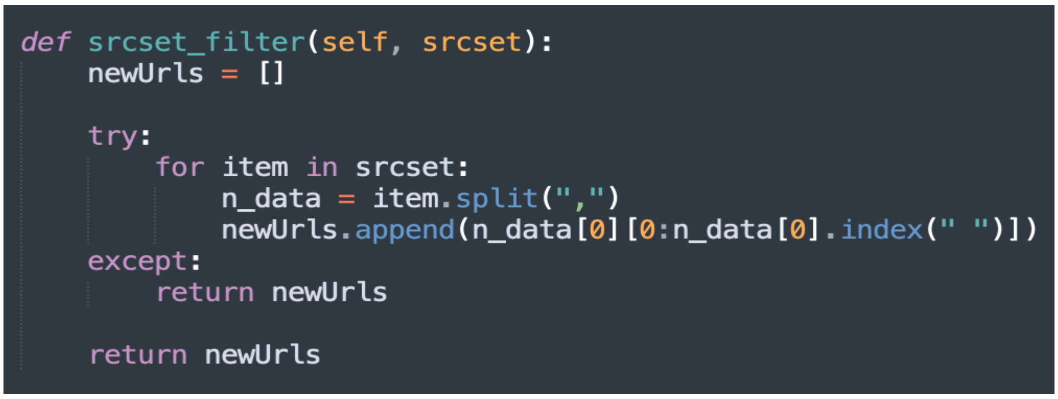

```python
def extract_links_post(self):
    reached_page_end = False
    last_height = self.driver.execute_script("return document.body.scrollHeight")
    links=set()
    while not reached_page_end:
        self.driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(3)
        new_height = self.driver.execute_script("return document.body.scrollHeight")
        # Fetching Posts
        posts = self.driver.find_elements(By.CSS_SELECTOR, 'div._aabd._aa8k._aanf a')

        fetched_links = [post.get_attribute("href") for post in posts]

        links = links.union(set(fetched_links))

        if last_height == new_height or len(links) >= 36:
            reached_page_end = True
        else:
            last_height = new_height

    return list(links)
```

- **extract_info_profile**
  This function will take all the elements of the main page of the instagram profile we want to scrap. It will start scrapping the profile info which will include the number of publications, followers, followings, also the user name, description and in case that the profile will have it, the verification and profession (will be done inside a try and except in order to not get error in case the profile does not have it). It is important to highlight that all this information will be obtained by using different selectors which will be scraped through the find_elements and get_attribute selenium methods.



```python
def extract_info_profile(self):
    time.sleep(3)
    profile_info = self.driver.find_elements(By.CSS_SELECTOR, 'li._aa_5 div._aacl._aacp._aacu._aacx._aad6._aade span')
    profile_info_final = [node.text for node in profile_info]

    name_user = self.driver.find_elements(By.CSS_SELECTOR, 'div[class="_aa_c"] span._aacl')
    name_user_final = [node.text for node in name_user]

    description = self.driver.find_elements(By.CSS_SELECTOR, 'div._aa_c div[class="_aacl _aacp _aacu _aacx _aad6 _aade"]')
    description_final = [node.text for node in description]

    if len(description_final) != 0:
        description_final = description_final[0]
    else:
        description_final = None


    try:
        verified = self.driver.find_elements(By.CSS_SELECTOR,'div[class="_ab8w  _ab94 _ab99 _ab9f _ab9m _ab9p  _abb0 _abcm"]
        verified = [node.text for node in verified]
        verified = verified[0]
        verified = True
    except:
        verified = False

    try:
        profession = self.driver.find_elements(By.CSS_SELECTOR,'div[class="_ab8w  _ab94 _ab99 _ab9f _ab9m _ab9p _abcm"] div[c
        profession_final = [node.text for node in profession]
        profession_final = profession_final[0]
    except:
        profession_final = None

    profile_photo_aux = self.driver.find_elements(By.CSS_SELECTOR, 'div._aarf span img')
    profile_photo = [node.get_attribute('src') for node in profile_photo_aux]


    return profile_info_final, profile_photo , name_user_final, description_final, verified, profession_final
```

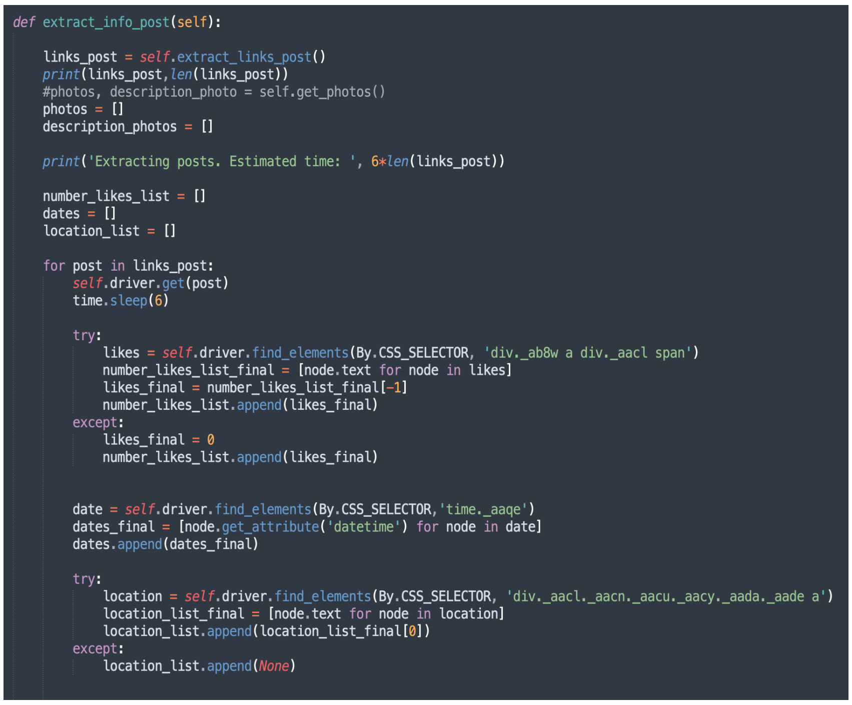
- **srcset_filter**
  This function will be used only for the photos that were taken through the attribute srcset, in this way we are able to extract the photo with the highest quality. This will be done with slicing and will return the url on a list. In case there is no photo (video) it will return an empty list.



```python
def srcset_filter(self, srcset):
    newUrls = []

    try:
        for item in srcset:
            n_data = item.split(",")
            newUrls.append(n_data[0][0:n_data[0].index(" ")])
    except:
        return newUrls

    return newUrls
```

- **extract_info_post**

  This function will extract the elements for each post from the extracted links, so it will iterate over the list of links getting the images in the posts, the number of likes, dates, photo description, link of the post and the location where the post was taken. It is important to mention that due to some instagram restrictions it was necessary to try with different selector to do the same purpose since in most of the cases the selector changes. Besides that in case that some elements are not found it will return a none as a value. Finally all those extractions will include a manage of exceptions since as it was mentioned may have the selectors or not.



```python
def extract_info_post(self):

    links_post = self.extract_links_post()
    print(links_post,len(links_post))
    #photos, description_photo = self.get_photos()
    photos = []
    description_photos = []

    print('Extracting posts. Estimated time: ', 6*len(links_post))

    number_likes_list = []
    dates = []
    location_list = []

    for post in links_post:
        self.driver.get(post)
        time.sleep(6)

        try:
            likes = self.driver.find_elements(By.CSS_SELECTOR, 'div._ab8w a div._aacl span')
            number_likes_list_final = [node.text for node in likes]
            likes_final = number_likes_list_final[-1]
            number_likes_list.append(likes_final)
        except:
            likes_final = 0
            number_likes_list.append(likes_final)


        date = self.driver.find_elements(By.CSS_SELECTOR,'time._aaqe')
        dates_final = [node.get_attribute('datetime') for node in date]
        dates.append(dates_final)

        try:
            location = self.driver.find_elements(By.CSS_SELECTOR, 'div._aacl._aacn._aacu._aacy._aada._aade a')
            location_list_final = [node.text for node in location]
            location_list.append(location_list_final[0])
        except:
            location_list.append(None)
```

Calling functions

Import the class to use their methods

```python
from scrapper import Instagram_Scrap
import json
import os
```

Log in

```python
#Logging in
web = Instagram_Scrap()
web.login()


#Go to the desired profile
username = web.go_to_profile()
```

In this part we initialize the class and use its method log in. Then we will go to the profile we want to scrap (this will return the username to use later in the json file).

Extracting the information

```python
# Extract number of posts, followers and followings
info, profile_photo ,name, description,verified, profession_final = web.extract_info_profile()


# Extract general info of the profile: photos, number of likes, date
number_likes_list, dates, photos, description_photo, links_post, location_list, = web.extract_info_post()

#It closes the browser
web.close()
```

This section of code will call the method extract_info_profile to get the main information from the profile and extract_info_post. It will store it into their respective variables.
Once we get that information we will close the browser.

Saving the information

```python
user = dict()


user['username'] = username
user['number_post'] = info[0]
user['number_followers'] = info[1]
user['number_followings'] = info[2]
user['photo_profile'] = profile_photo
user['real_name'] = name
user['description'] = description
user['verified'] = verified
user['profession'] = profession_final
```

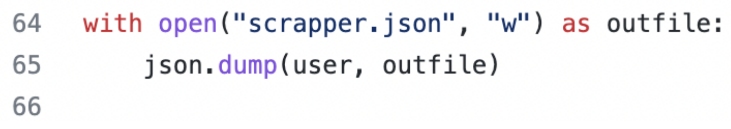
We generate a dictionary that will store all the scrapped information through the methods above.

Saving the information for each post

```python
user_posts = dict()
print(len(number_likes_list) , len(photos))

for i in location_list:
    print(i)

for i in range(len(photos)):


    user_posts[f'user_post{i}'] = {}

    user_posts[f'user_post{i}']['id'] = i
    user_posts[f'user_post{i}']['likes'] = (number_likes_list[i])
    user_posts[f'user_post{i}']['date'] = dates[i]
    user_posts[f'user_post{i}']['photo'] = photos[i]
    user_posts[f'user_post{i}']['description_photo'] = description_photo[i]
    user_posts[f'user_post{i}']['url_post'] = links_post[i]
    user_posts[f'user_post{i}']['location'] = location_list[i]

user['posts_info'] = user_posts
```

In this section of code we create a dictionary where we will store the elements of each post in another dictionary inside of it. The value of the main dictionary will be the user posts which will also store a dictionary's respective information.

Saving the json file



```
with open("scrapper.json", "w") as outfile:
    json.dump(user, outfile)

```

This section saves the resulting json

Preprocessing

Analyzing the information

After downloading information using the scraper, the returned .JSON file, it which contained the general information of the account and the individual data of each post, analyzing the downloaded data, we realized that there was a lot of information with dirty data, either that it had extra characters, or irrelevant information for our visualization process, so we proceeded to clean the data as follows.

The information was separated into two sections:

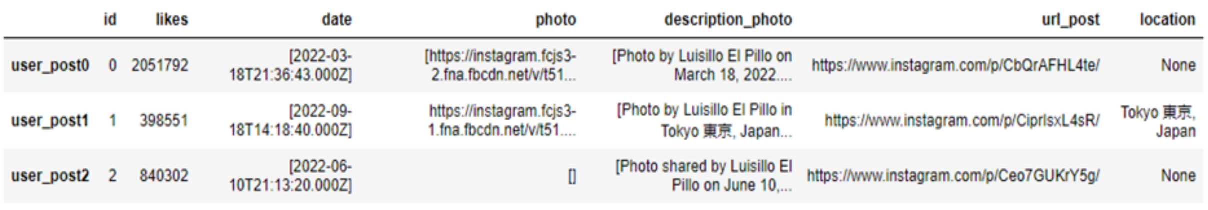
- General data.
- Data per post.

General data

Cleaning was not necessary in this section, since all the data was clean from the beginning, so this section was reserved in the cleaning process.

Data per post

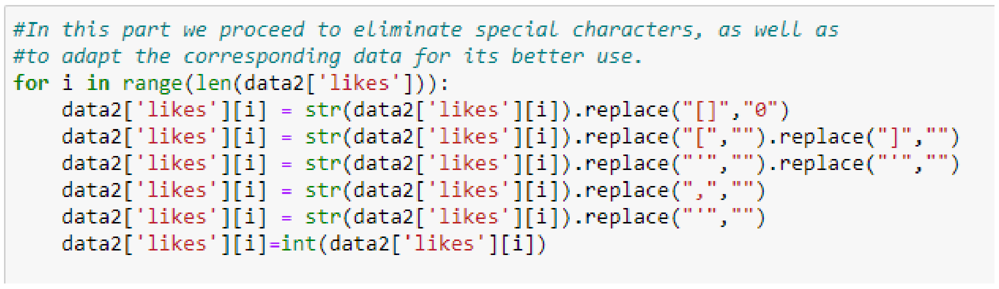
Unlike the general data, this section was the one that contained extra characters, for example in the dates as you can see in the image contains not only the year-month-day format, but also contains information of the local time, which to us in our project was not useful.



	id	likes	date	photo	description_photo	url_post	location
user_post0	0	2051792	[2022-03-18T21:36:43.000Z]	[https://instagram.fcjs3-2.fna.fbcdn.net/v/t51....	[Photo by Luisillo El Pillo on March 18, 2022....	https://www.instagram.com/p/CbQrAFHL4te/	None
user_post1	1	398551	[2022-09-18T14:18:40.000Z]	https://instagram.fcjs3-1.fna.fbcdn.net/v/t51....	[Photo by Luisillo El Pillo in Tokyo 東京, Japan...	https://www.instagram.com/p/CiprlsxL4sR/	Tokyo 東京, Japan
user_post2	2	840302	[2022-06-10T21:13:20.000Z]	[]	[Photo shared by Luisillo El Pillo on June 10,...	https://www.instagram.com/p/Ceo7GUKrY5g/	None

Clean Likes

The column of "likes", we had to work it to keep an integer, because they would be used later to obtain different metrics, in this part, the extracted data maintained extra characters such as "[], ', "" or commas ",, so as you can see in the following figure a search was made to find these characters and be replaced, in the end the already clean data was saved in an integer.

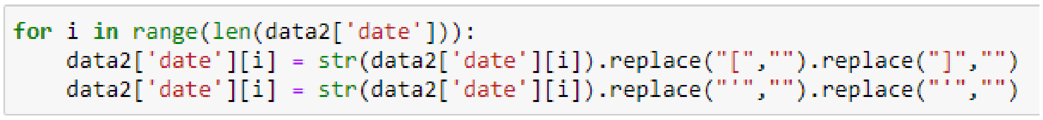


```python
#In this part we proceed to eliminate special characters, as well as
#to adapt the corresponding data for its better use.
for i in range(len(data2['likes'])):
    data2['likes'][i] = str(data2['likes'][i]).replace("[]","0")
    data2['likes'][i] = str(data2['likes'][i]).replace("[","").replace("]","")
    data2['likes'][i] = str(data2['likes'][i]).replace("'","").replace("'","")
    data2['likes'][i] = str(data2['likes'][i]).replace(",","")
    data2['likes'][i] = str(data2['likes'][i]).replace("'","")
    data2['likes'][i]=int(data2['likes'][i])
```

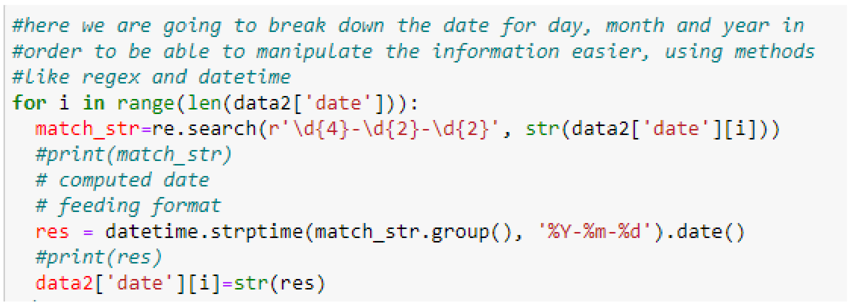
.

Clean date

Here, taking the information from the "date" column, we proceed with the cleaning of special characters. Subsequently, we use the RE library and format the date value data. This was necessary because we need to extract day, month and year.



```python
for i in range(len(data2['date'])):
    data2['date'][i] = str(data2['date'][i]).replace("[","").replace("]","")
    data2['date'][i] = str(data2['date'][i]).replace("'","").replace("'","")
```



```python
#here we are going to break down the date for day, month and year in
#order to be able to manipulate the information easier, using methods
#like regex and datetime
for i in range(len(data2['date'])):
  match_str=re.search(r'\d{4}-\d{2}-\d{2}', str(data2['date'][i]))
  #print(match_str)
  # computed date
  # feeding format
  res = datetime.strptime(match_str.group(), '%Y-%m-%d').date()
  #print(res)
  data2['date'][i]=str(res)
```

```
year=[]
month=[]
day=[]

date_time=pd.to_datetime(data2['date'])

for i in range(len(data2['date'])):
    year.append(date_time.dt.year[i])
    month.append(date_time.dt.month[i])
    day.append(date_time.dt.day[i])
```

Clean Photo

In this part we work the information about the data with the photos, which are first extracted from the special characters (“[], ”) to have the data of the locations that links each post with the photo where it is referenced.

```
#In this part we proceed to eliminate special characters, as well as
#to adapt the corresponding data for its better use.
for i in range(len(data2['photo'])):
    data2['photo'][i] = str(data2['photo'][i]).replace("[]","Others Media")
    data2['photo'][i] = str(data2['photo'][i]).replace("[","").replace("]","")
    data2['photo'][i] = str(data2['photo'][i]).replace("'","").replace("'","")
```

Location

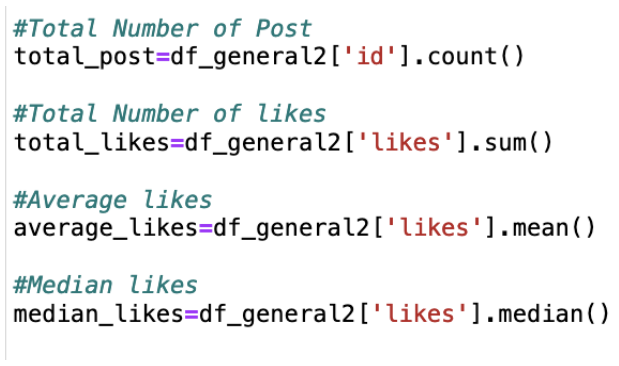
This section helps us by locating the location extracted from the photos, we can geolocate them with their names thanks to a python library called geopy.

```
for i in df_general2["location"]:
    loc = Nominatim(user_agent="GetLoc")
    getLoc = loc.geocode(i)
    if i!=None:
        if(getLoc==None):
            print("no se encontro")
        else:
            print(getLoc.latitude)
            location2.append(str(i))
            long=getLoc.longitude
            lati=getLoc.latitude
            latitude.append(lati)
            longitude.append(long)
```

Metrics

In this section we adjusted some metrics referenced to the likes to show the characteristics of the data.

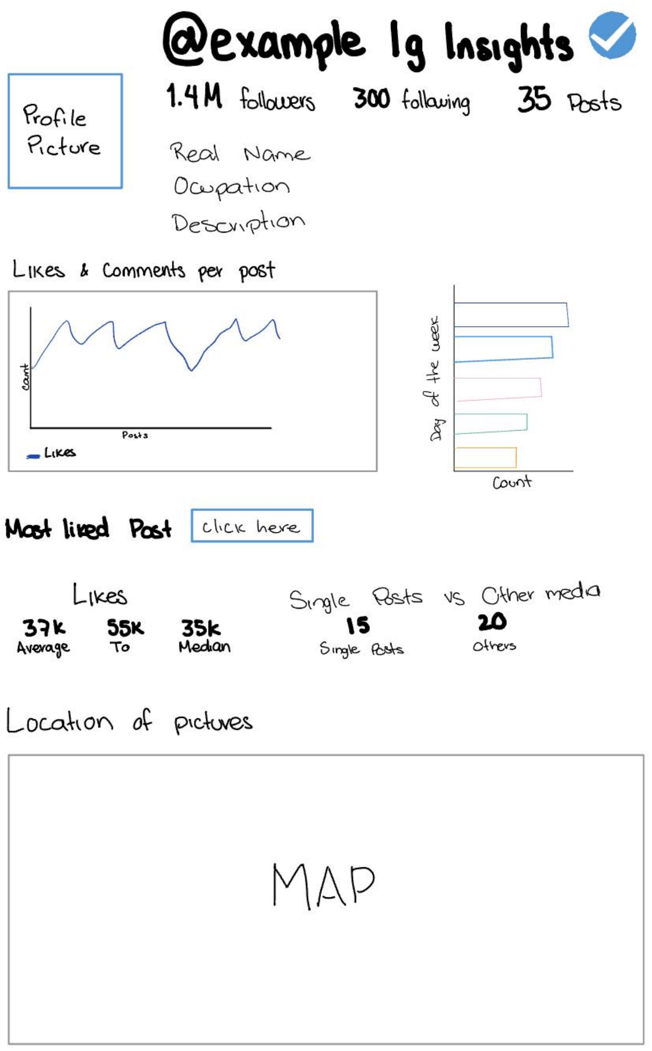
- Total numbers of post
- Total likes of the account
- Average likes
- Median likes



```python
#Total Number of Post
total_post=df_general2['id'].count()

#Total Number of likes
total_likes=df_general2['likes'].sum()

#Average likes
average_likes=df_general2['likes'].mean()

#Median likes
median_likes=df_general2['likes'].median()
```

Dashboard

Design



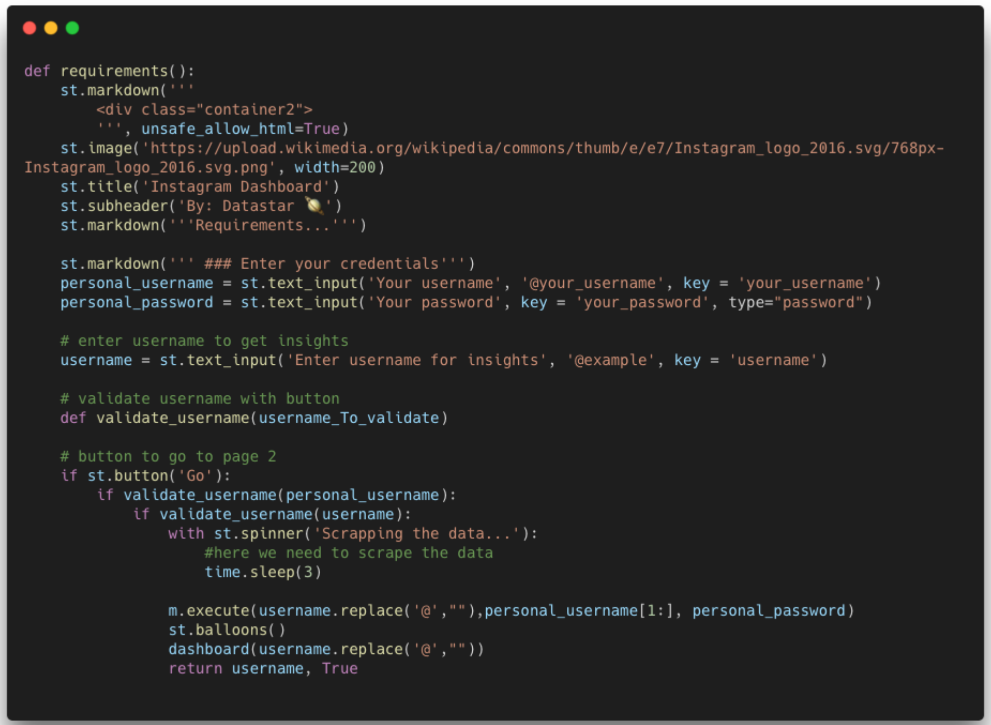
From the data we have, we decided to add 3 main plots: Likes per post, Likes per day of the week and a map of the pictures that include locations on them.
For the colors and the general design, we went with the Instagram style, to make it look like an extensión you can access directly from instagram in its web version.

The elements in the Dashboard can be listed as follows:
- Title containing the username of the profile used for insights.
- Verified icon (if the account is verified).
- Profile picture.

- Number of followers.
- Number of following.
- Total posts in the account.
- Real name of the profile.
- Profession (if set by the user).
- Biography.
- Likes per post: It shows the amount of likes as the y axis and the number of post in the x axis.
- Post per day: Where the x axis has the days of the week and the y axis a count of how many posts you have on said day.
- Link to the most liked post.
- Insights about likes: Total, Average, Median.
- Comparison of posts with one picture and other media, which refers to posts with multiple photos or videos.
- Map with locations set in the posts.
- Dataframe containing the names of the places in the map.

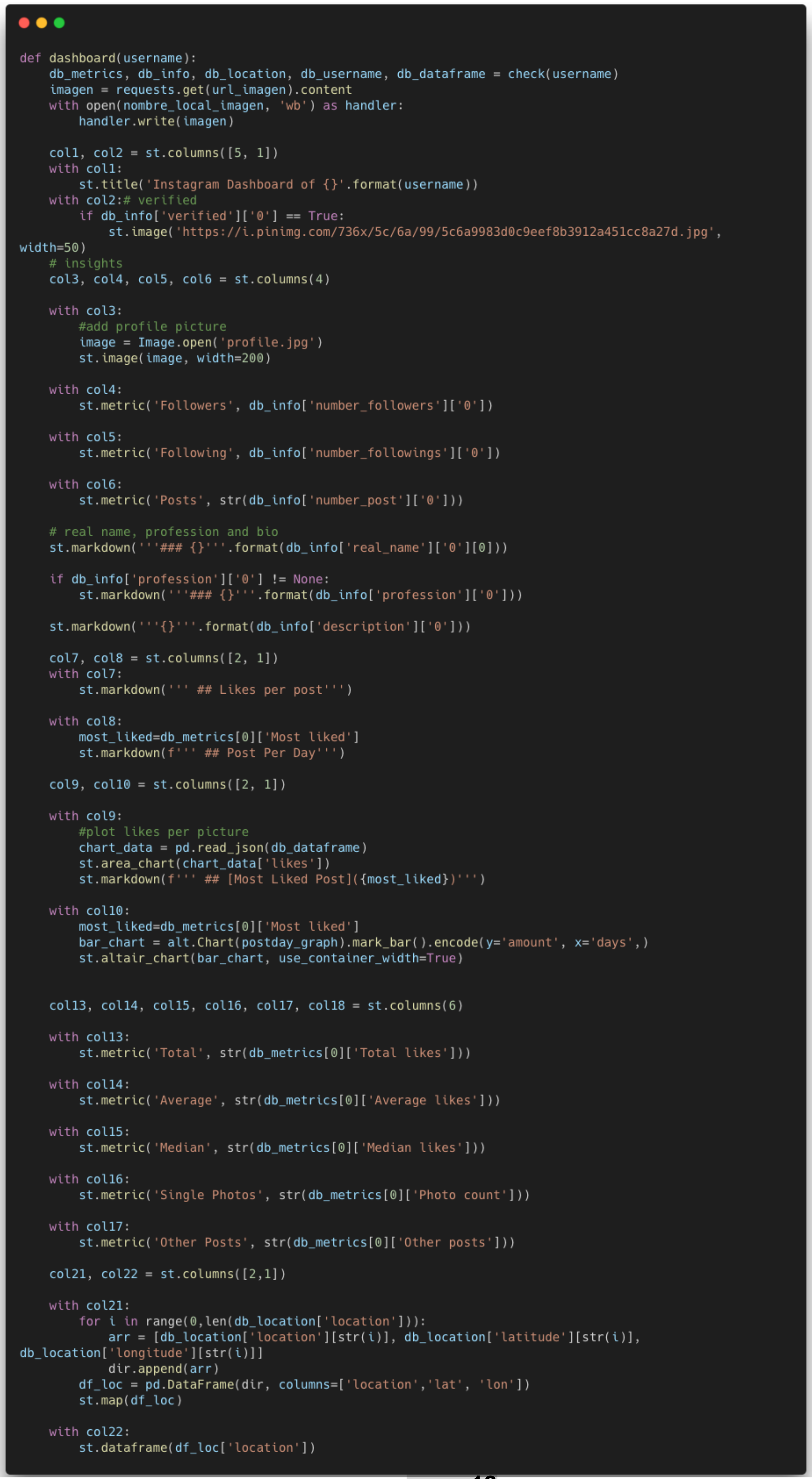
Development

The code for the dashboard was divided in two ‘pages’. The first one, which appears as you run the code, this page contains the requirements and where you input usernames and a password to do the scrapping.



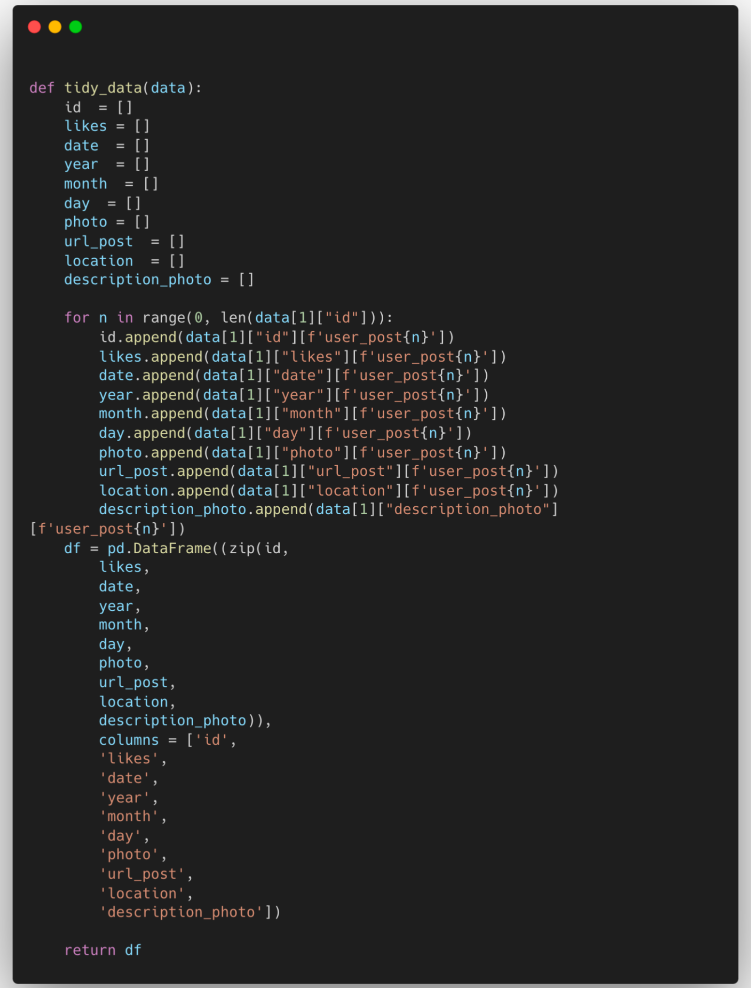
```
def requirements():
    st.markdown('''
        <div class="container2">
        ''', unsafe_allow_html=True)
    st.image('https://upload.wikimedia.org/wikipedia/commons/thumb/e/e7/Instagram_logo_2016.svg/768px-
Instagram_logo_2016.svg.png', width=200)
    st.title('Instagram Dashboard')
    st.subheader('By: Datastar 🪐')
    st.markdown('''Requirements...''')

    st.markdown(''' ### Enter your credentials''')
    personal_username = st.text_input('Your username', '@your_username', key = 'your_username')
    personal_password = st.text_input('Your password', key = 'your_password', type="password")

    # enter username to get insights
    username = st.text_input('Enter username for insights', '@example', key = 'username')

    # validate username with button
    def validate_username(username_To_validate)

    # button to go to page 2
    if st.button('Go'):
        if validate_username(personal_username):
            if validate_username(username):
                with st.spinner('Scrapping the data...'):
                    #here we need to scrape the data
                    time.sleep(3)

                m.execute(username.replace('@',""),personal_username[1:], personal_password)
                st.balloons()
                dashboard(username.replace('@',""))
                return username, True
```

The second page is shown if the first one has the usernames validated, this second page is the display of the dashboard. Additionally to the 3 main plots, we have multiple metrics distributed through the graph. Some elements are conditioned to only appear if they exist on the profile, like the verification icon or the profession. Here we have a verified profile with profession, and another without those elements to compare.



```
def dashboard(username):
    db_metrics, db_info, db_location, db_username, db_dataframe = check(username)
    imagen = requests.get(url_imagen).content
    with open(nombre_local_imagen, 'wb') as handler:
        handler.write(imagen)

    col1, col2 = st.columns([5, 1])
    with col1:
        st.title('Instagram Dashboard of {}'.format(username))
    with col2:# verified
        if db_info['verified']['0'] == True:
            st.image('https://i.pinimg.com/736x/5c/6a/99/5c6a9983d0c9eef8b3912a451cc8a27d.jpg',
width=50)
    # insights
    col3, col4, col5, col6 = st.columns(4)

    with col3:
        #add profile picture
        image = Image.open('profile.jpg')
        st.image(image, width=200)

    with col4:
        st.metric('Followers', db_info['number_followers']['0'])

    with col5:
        st.metric('Following', db_info['number_followings']['0'])

    with col6:
        st.metric('Posts', str(db_info['number_post']['0']))

    # real name, profession and bio
    st.markdown('''### {}'''.format(db_info['real_name']['0'][0]))

    if db_info['profession']['0'] != None:
        st.markdown('''### {}'''.format(db_info['profession']['0']))

    st.markdown('''{}'''.format(db_info['description']['0']))

    col7, col8 = st.columns([2, 1])
    with col7:
        st.markdown(''' ## Likes per post''')

    with col8:
        most_liked=db_metrics[0]['Most liked']
        st.markdown(f''' ## Post Per Day''')

    col9, col10 = st.columns([2, 1])

    with col9:
        #plot likes per picture
        chart_data = pd.read_json(db_dataframe)
        st.area_chart(chart_data['likes'])
        st.markdown(f''' ## [Most Liked Post]({most_liked})''')

    with col10:
        most_liked=db_metrics[0]['Most liked']
        bar_chart = alt.Chart(postday_graph).mark_bar().encode(y='amount', x='days',)
        st.altair_chart(bar_chart, use_container_width=True)


    col13, col14, col15, col16, col17, col18 = st.columns(6)

    with col13:
        st.metric('Total', str(db_metrics[0]['Total likes']))

    with col14:
        st.metric('Average', str(db_metrics[0]['Average likes']))

    with col15:
        st.metric('Median', str(db_metrics[0]['Median likes']))

    with col16:
        st.metric('Single Photos', str(db_metrics[0]['Photo count']))

    with col17:
        st.metric('Other Posts', str(db_metrics[0]['Other posts']))

    col21, col22 = st.columns([2,1])

    with col21:
        for i in range(0,len(db_location['location'])):
            arr = [db_location['location'][str(i)], db_location['latitude'][str(i)],
db_location['longitude'][str(i)]]
            dir.append(arr)
        df_loc = pd.DataFrame(dir, columns=['location','lat', 'lon'])
        st.map(df_loc)

    with col22:
        st.dataframe(df_loc['location'])
```

Connection of front-end and back-end
(Mongo DB)



```
def tidy_data(data):
    id  = []
    likes = []
    date  = []
    year  = []
    month  = []
    day  = []
    photo = []
    url_post  = []
    location  = []
    description_photo = []

    for n in range(0, len(data[1]["id"])):
        id.append(data[1]["id"][f'user_post{n}'])
        likes.append(data[1]["likes"][f'user_post{n}'])
        date.append(data[1]["date"][f'user_post{n}'])
        year.append(data[1]["year"][f'user_post{n}'])
        month.append(data[1]["month"][f'user_post{n}'])
        day.append(data[1]["day"][f'user_post{n}'])
        photo.append(data[1]["photo"][f'user_post{n}'])
        url_post.append(data[1]["url_post"][f'user_post{n}'])
        location.append(data[1]["location"][f'user_post{n}'])
        description_photo.append(data[1]["description_photo"]
[f'user_post{n}'])
    df = pd.DataFrame((zip(id,
        likes,
        date,
        year,
        month,
        day,
        photo,
        url_post,
        location,
        description_photo)),
        columns = ['id',
        'likes',
        'date',
        'year',
        'month',
        'day',
        'photo',
        'url_post',
        'location',
        'description_photo'])

    return df
```

Tidy Data & Secondary Metrics

This function is in charge of extracting the information from the json cleaned by the preprocessing, to later generate a dataframe, which can be manipulated as desired.

The objective is to be able to store in the best possible format to be able to be extracted and read in the best possible format, for any future request to the database in the mongo server. Also, by giving it a dataframe format, we can generate metrics in a much easier way.

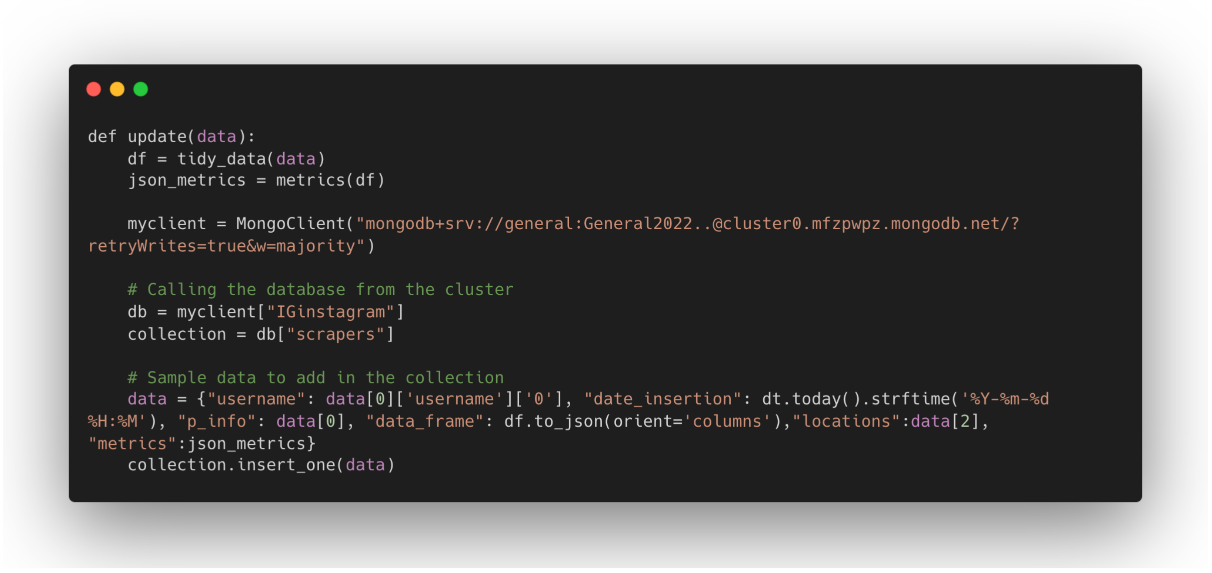
Secondary metrics

As mentioned above, new metrics were calculated thanks to the practical reformatting generated with the tidy data, so they were able to measure things like :

- number of posts per day
- number of likes per post

Which allows us to have a vision of which days we can generate more likes, as well as which days we are active.

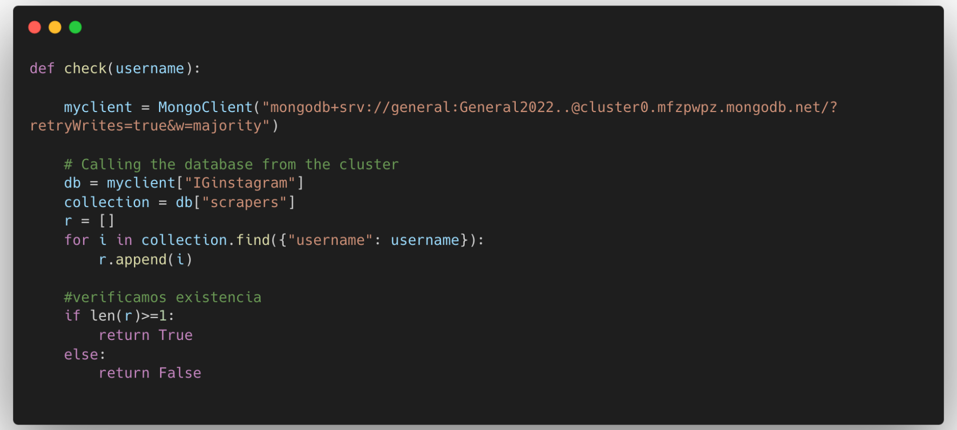
Database Connection



```
def update(data):
    df = tidy_data(data)
    json_metrics = metrics(df)

    myclient = MongoClient("mongodb+srv://general:General2022..@cluster0.mfzpwpz.mongodb.net/?
retryWrites=true&w=majority")

    # Calling the database from the cluster
    db = myclient["IGinstagram"]
    collection = db["scrapers"]

    # Sample data to add in the collection
    data = {"username": data[0]['username']['0'], "date_insertion": dt.today().strftime('%Y-%m-%d
%H:%M'), "p_info": data[0], "data_frame": df.to_json(orient='columns'),"locations":data[2],
"metrics":json_metrics}
    collection.insert_one(data)
```

Once the data is ready in the correct format, we want to be able to make a push to the database, so by means of a cursor, we can generate a push of the documents in json format, both of the metrics, the personal information, the user and the calculated metrics, so as not to recalculate them or repeat the scraping each time a request is made.

Requesting user’s data

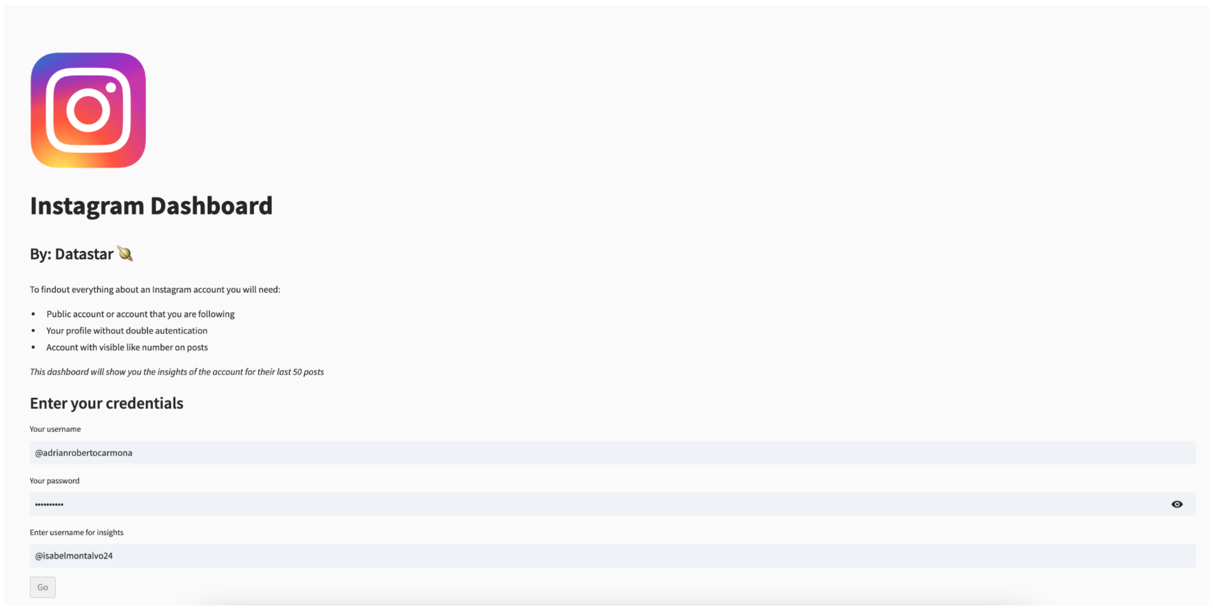

```
def check(username):

    myclient = MongoClient("mongodb+srv://general:General2022..@cluster0.mfzpwpz.mongodb.net/?
retryWrites=true&w=majority")

    # Calling the database from the cluster
    db = myclient["IGinstagram"]
    collection = db["scrapers"]
    r = []
    for i in collection.find({"username": username}):
        r.append(i)

    #verificamos existencia
    if len(r)>=1:
        return True
    else:
        return False
```

Once the data is already in the database, a request is made where the dataset can locate the user, to return if the user exists or the user is not in the database.

After returning the true or false, we can decide by the main function, which will be the next action, since it can be decided to two different actions. The first one is to scrape the user until its visualization and the second one is to make request of the user data, so it is not necessary to have to scrape everything again.

One thing to take into account, is that in generating, you check how many days ago the scraper of the publication was made, to decide whether to scrape again or make a simple request.

Results



Instagram Dashboard

By: Datastar 🪐

To findout everything about an Instagram account you will need:

- Public account or account that you are following
- Your profile without double autentication
- Account with visible like number on posts

This dashboard will show you the insights of the account for their last 50 posts

Enter your credentials

Your username

@adrianrobertocarmona

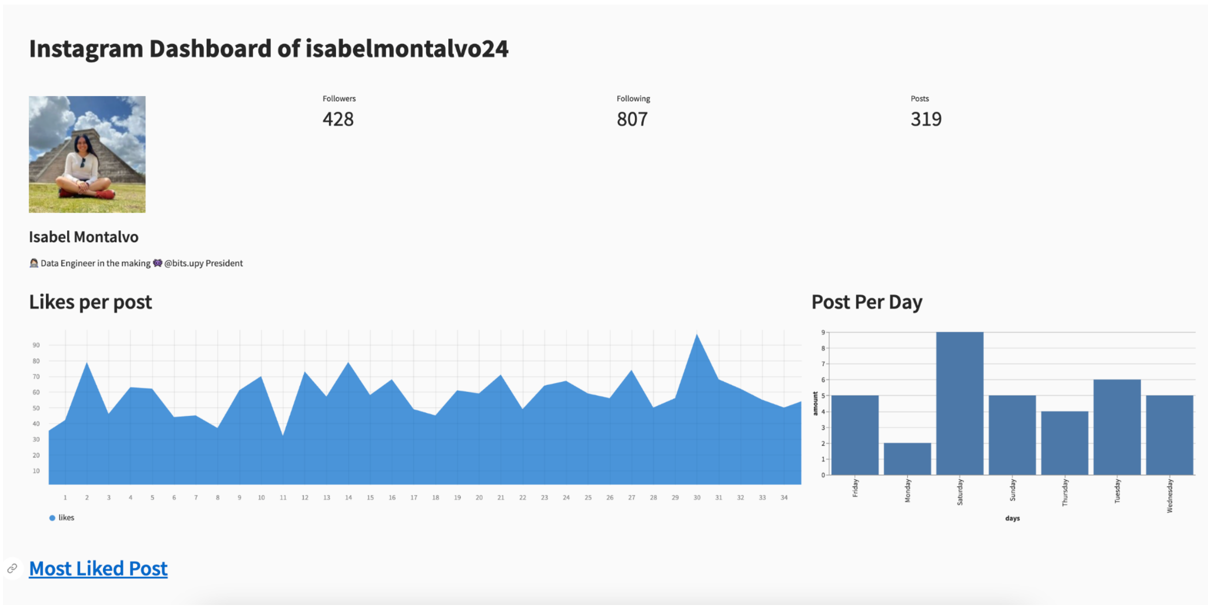
Your password

••••••••••

Enter username for insights

@isabelmontalvo24

This is the main window displaying the web page interface where the user will be capable of adding the information of the credentials in order to use the account to perform the scraping process. On the other hand it will also be requested to ask the user profile it is planned to check. As it can be seen this window shows briefly the instructions to the user and the company name.



Instagram Dashboard of isabelmontalvo24

Followers	Following	Posts
428	807	319

Isabel Montalvo

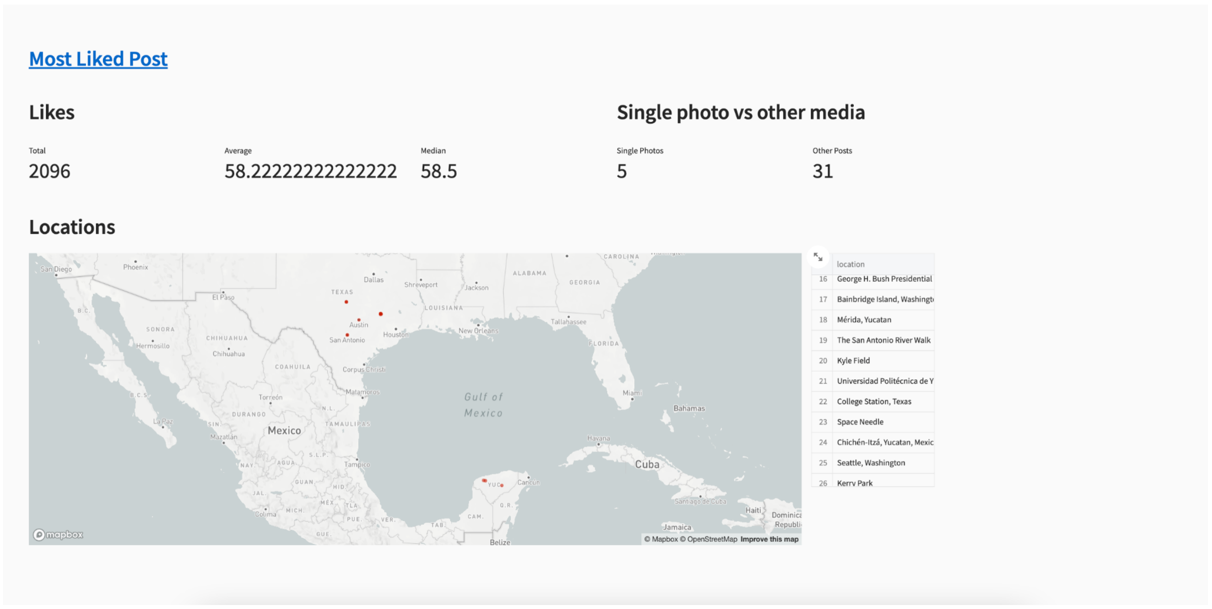
👩‍💻 Data Engineer in the making 👾 @bits.upy President

Likes per post

Post Per Day

Most Liked Post

Here we can see the profile photo, total number of followers, followings and posts, and the behavior of all those posts with their likes on a chart. Besides that we can see the number of posts per day.



Most Liked Post

Likes

Total	Average	Median
2096	58.22222222222222	58.5

Single photo vs other media

Single Photos	Other Posts
5	31

Locations

	location
16	George H. Bush Presidential
17	Bainbridge Island, Washingt
18	Mérida, Yucatan
19	The San Antonio River Walk
20	Kyle Field
21	Universidad Politécnica de Y
22	College Station, Texas
23	Space Needle
24	Chichén-Itzá, Yucatan, Mexic
25	Seattle, Washington
26	Kerry Park

As you can see, if the person went to a trip, a map will be shown with all the locations that was visited by the person in red points and also, a table will be shown to the right side on the screen

Project Value Estimation

Name	Field	Hours worked	Pay rate	Total
Erick Hernandez Hernandez	Web scraping	10	180/hour	1800
Luis Fernando Koh Avila	Web scraping	10	180/hour	1800
Adriana Maribel Ku Huchim	Preprocessing and Documental technical	14	160/hour	2240
Alfredo Alexander Paz Martínez	Preprocessing and Documental technical	14	160/hour	2240
Team Lead Adrian Roberto Carmona Rodriguez	Database connection and dashboard - Project Management	16	220/hour	3520
Isabel Cámara Montalvo	Dashboard design and front-end	16	140/hour	1680
Total cost			20,000	

Conclusion

During the development of this project we understand the real process the business and teams have to do in order to achieve a final product. During this process we learn and go through many obstacles, starting from the web scraper, due to our background in programming we did not have enough experience extracting information from websites for that we had to understand how the selectors work as well as libraries such as Selenium letting us control a web browser through programmed instructions, besides that due to Instagram is a massive corporation where the data its is main active it was such a difficult tasks to take information from there, facing many problems such as variations in many selectors, shadow bans, and even temporarily blocks in our accounts. In order to solve this we had to limit the number of posts and scrolls in the scraping process.

On the other hand, when it comes to the preprocessing process there were many challenges too, due to the way information was extracted, most of the information was gathering as text format and for that it was necessary to apply many techniques and functions such as slicing, transformations for the date format and replacing statements. Given that the purpose of the preprocessing was to get the information clean and ready for the database, many datasets were transformed as json format and unified into a single file.

For the web page we had to learn how to program a front end interface where the user is able to interact with for that we learn the HTML as well as CSS to program the classes and objects to beautify the site. It was a rough process even though our formation is not as software developers.

As a final observation we found the project interesting and helpful to prepare ourselves for the business sector as well as understand the methodologies and processes a good team has to overcome. It was challenging to work with people we have never worked with before but through communication and dialogue it was possible to achieve the final objectives at time.