Project 7: Data Visualization Tools: Social Media

Code Manual

Files:

templates:

index.html

facebook.html

Instagram.html

Setup:

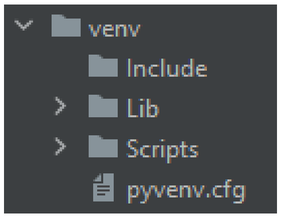
1. Clone the repository from GitHub
2. Install Flask and Download the virtual environment:
   macOS/Linux

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv
```

   Windows

```
> mkdir myproject
> cd myproject
> py -3 -m venv venv
```

This will download a virtual environment for you to run your flask app. You should see a venv folder in your directory:



To activate your environment, run these commands,

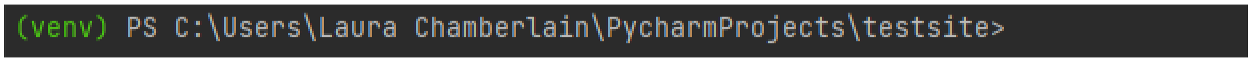
macOS/Linux

```
$ . venv/bin/activate
```

Windows

```
> venv\Scripts\activate
```

Once you run this you should see this:



```
(venv) PS C:\Users\Laura Chamberlain\PycharmProjects\testsite>
```

This signals that your virtual environment is activated.

While your virtual environment is activated, you need to install Flask and requests.

Run:

```
$ pip install Flask
```

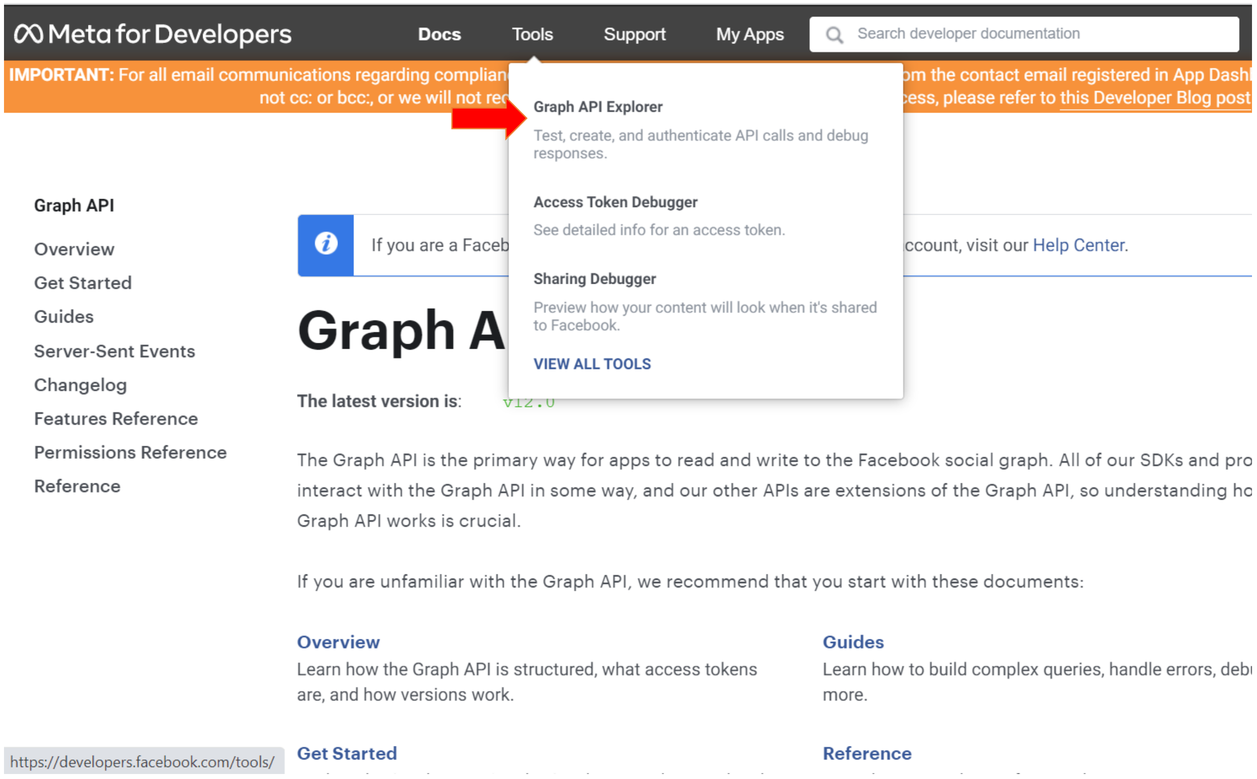
This will install Flask

```
$ python -m pip install requests
```

This will install requests.

If you get an ImportError when running the flask app, this may mean that you did not install these correctly.

3. Get Facebook Access Tokens
   Helpful Link: https://developers.facebook.com/docs/graph-api
   Login to your Facebook developer account and go to the Graph API Explorer Tool



IMPORTANT: For all email communications regarding complian… …om the contact email registered in App Dashb… not cc: or bcc:, or we will not rec… …cess, please refer to this Developer Blog post

If you are a Faceb… …ccount, visit our Help Center.

Graph A…

The latest version is: v12.0

The Graph API is the primary way for apps to read and write to the Facebook social graph. All of our SDKs and pro… interact with the Graph API in some way, and our other APIs are extensions of the Graph API, so understanding ho… Graph API works is crucial.

If you are unfamiliar with the Graph API, we recommend that you start with these documents:

Overview
Learn how the Graph API is structured, what access tokens are, and how versions work.

Guides
Learn how to build complex queries, handle errors, debu… more.

Get Started

Reference

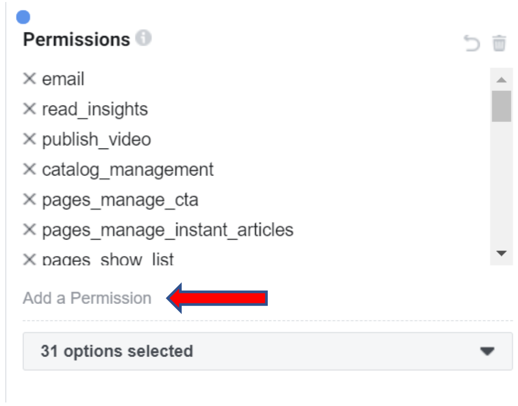
https://developers.facebook.com/tools/

Make sure you have all necessary permissions:

- email
- read_insights
- publish_video
- catalog_management
- pages_manage_cta
- pages_manage_instant_articles
- pages_show_list
- read_page_mailboxes
- ads_management
- ads_read
- business_management
- pages_messaging
- pages_messaging_phone_number
- pages_messaging_subscriptions
- instagram_basic
- instagram_manage_comments
- instagram_manage_insights
- instagram_content_publish
- publish_to_groups
- groups_access_member_info
- leads_retrieval
- whatsapp_business_management
- instagram_manage_messages

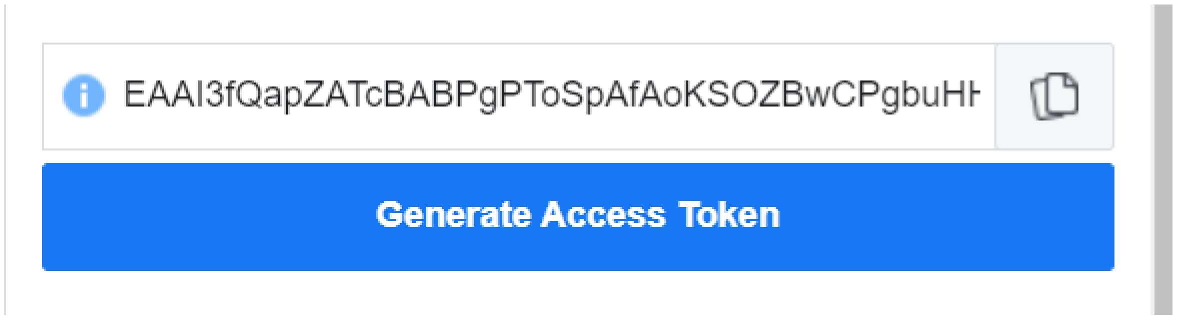
- attribution_read
- page_events
- pages_read_engagement
- pages_manage_metadata
- pages_read_user_content
- pages_manage_ads
- pages_manage_posts
- pages_manage_engagement

To add permissions, just go to the bottom right where it says permissions and choose add permission:



Under User or Page: Choose the Facebook Page that is connected to the Instagram Business Account

Press Generate Token and copy your access token



This will be a short lived access token, so it will expire in an hour.

Generate long term access token:

Put the short term access token into the defines.py, in getCreds under key page_access token.

Call the longLivedAccessToken() function and copy the returned access token into the same part of defines.py.

This access token will last three months. If you get a KeyError, that most likely means your access token is expired.

4. Run Flask app with following commands:

```
> set FLASK_APP=hello
> flask run
 * Running on http://127.0.0.1:5000/
```

The app should now run on your local server.

app.py

Description: Flask app

homepage():

```python
@app.route("/")
def homepage():
    info = dict()
    info['followers'] = follower_count() + igFollowers()
    info['f_percent'] = round(follower_count() / info['followers'] * 100, 1)
    info['i_percent'] = round(igFollowers() / info['followers'] * 100, 1)
    return render_template('index.html', info = info)
```

Description: Code for homepage of the website. It calls the index.html template and passes in a dictionary called info with information from the API calls

Variables:

info:

Keys:

followers: holds the sum of the followers across all platforms

f_percent: holds the percentage of followers facebook has of followers across all platforms

i_percent: holds the percentage of followers instagram has of followers across all platforms

facebook():

```python
@app.route("/facebook")
def facebook():
    info = dict()
    info['followers'] = follower_count()
    info['impressions'] = dict()
    impressions = getImpressions()
    info['impressions']['day'] = impressions['day']
    info['impressions']['week'] = impressions['week']
    info['impressions']['days_28'] = impressions['days_28']
    views = pageViews()
    info['views'] = dict()
    info['views']['day'] = views['day']
    info['views']['week'] = views['week']
    info['views']['days_28'] = views['days_28']
    users = engagedUsers()
    info['users'] = dict()
    info['users']['day'] = users['day']
    info['users']['week'] = users['week']
    info['users']['days_28'] = users['days_28']
    info['pimpressions'] = dict()
    pimpressions = getPostImpressions()
    info['pimpressions']['day'] = pimpressions['day']
    info['pimpressions']['week'] = pimpressions['week']
    info['pimpressions']['days_28'] = pimpressions['days_28']

    mInfo = postIds()
    info['likes'] = []
    info['avg_likes'] = 0
    for i in range(10):
        l = getPostLikes(mInfo[i]['id'])
        info['likes'] += [l]
        info['avg_likes'] += l
    info['avg_likes'] /= 10

    return render_template('facebook.html', info = info, mInfo = mInfo)
```

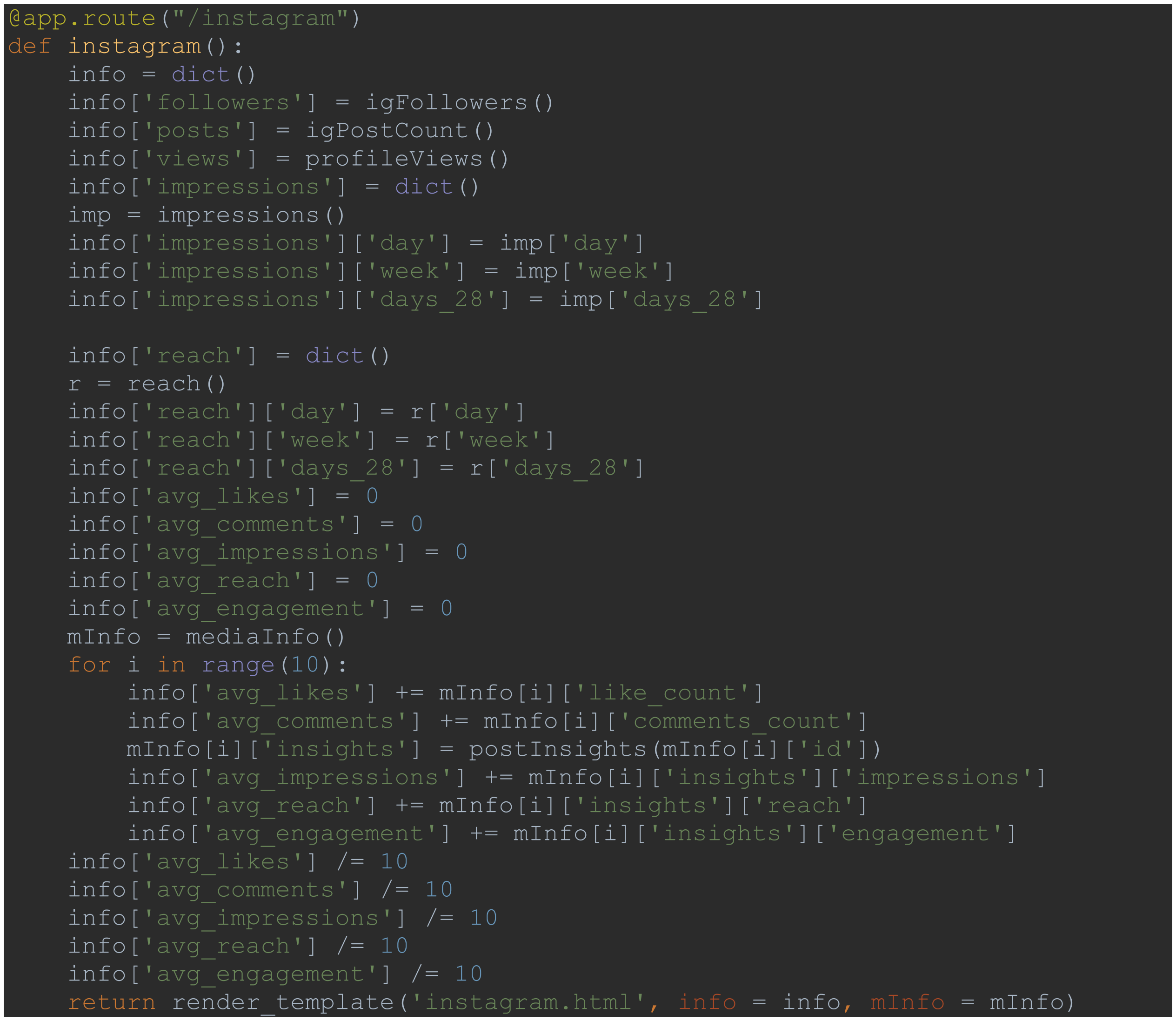
Description: Facebook Dashboard Page: passes in a dictionary and list into facebook.html

Variables:

info:
 Keys:
  followers: holds the number of facebook followers as an integer
  impressions: a dictionary with three keys hold the number of impressions over certain periods of time
   Keys:
    day: number impressions over the last day as an integer
    week: number impressions over the last week as an integer
    days_28: number impressions over the last 28 days as an integer
  views: a dictionary with three keys hold the number of page views over certain periods of time
   Keys:
    day: number page views over the last day as an integer
    week: number page views over the last week as an integer
    days_28: number page views over the last 28 days as an integer
  users: a dictionary with three keys hold the number of engaged users over certain periods of time
   Keys:
    day: number engaged users over the last day as an integer
    week: number engaged users over the last week as an integer
    days_28: number engaged users over the last 28 days as an integer

pimpressions: a dictionary with three keys hold the number of post impressions over certain periods of time
Keys:
day: number post impressions over the last day as an integer
week: number post impressions over the last week as an integer
days_28: number post impressions over the last 28 days as an integer
avg_likes: the average likes over the past 10 posts as an integer
likes: a list of like counts for the past 10 posts
mInfo: a list containing 10 dictionaries of information for the past 10 posts
Keys:
id: id for that post
timestamp: timestamp for that post

instagram():



```
@app.route("/instagram")
def instagram():
    info = dict()
    info['followers'] = igFollowers()
    info['posts'] = igPostCount()
    info['views'] = profileViews()
    info['impressions'] = dict()
    imp = impressions()
    info['impressions']['day'] = imp['day']
    info['impressions']['week'] = imp['week']
    info['impressions']['days_28'] = imp['days_28']

    info['reach'] = dict()
    r = reach()
    info['reach']['day'] = r['day']
    info['reach']['week'] = r['week']
    info['reach']['days_28'] = r['days_28']
    info['avg_likes'] = 0
    info['avg_comments'] = 0
    info['avg_impressions'] = 0
    info['avg_reach'] = 0
    info['avg_engagement'] = 0
    mInfo = mediaInfo()
    for i in range(10):
        info['avg_likes'] += mInfo[i]['like_count']
        info['avg_comments'] += mInfo[i]['comments_count']
        mInfo[i]['insights'] = postInsights(mInfo[i]['id'])
        info['avg_impressions'] += mInfo[i]['insights']['impressions']
        info['avg_reach'] += mInfo[i]['insights']['reach']
        info['avg_engagement'] += mInfo[i]['insights']['engagement']
    info['avg_likes'] /= 10
    info['avg_comments'] /= 10
    info['avg_impressions'] /= 10
    info['avg_reach'] /= 10
    info['avg_engagement'] /= 10
    return render_template('instagram.html', info = info, mInfo = mInfo)
```

Description: Instagram Dashboard Page. Passes in a dictionary and list into instagram.html

Variables:

info:
  Keys:
      followers: holds the number of instagram followers as an integer
      posts: holds the number of instagram posts as an integer
     views: holds the number of daily instagram profile views as an integer
     impressions: a dictionary with three keys hold the number of impressions over certain periods of time
              Keys:
                day: number impressions over the last day as an integer
                week: number impressions over the last week as an integer
                days_28: number impressions over the last 28 days as an integer
      views: a dictionary with three keys hold the reach over certain periods of time
              Keys:
                day: reach over the last day as an integer
                week: reach over the last week as an integer
                days_28: reach over the last 28 days as an integer
      avg_likes: the average likes over the past 10 posts as an integer
      avg_comments: the average comments over the past 10 posts as an integer
      avg_impressions: the average impressions over the past 10 posts as an integer
      avg_reach: the average reach over the past 10 posts as an integer
      avg_engagement: the average engagement over the past 10 posts as an integer
mInfo: a list containing 10 dictionaries of information for the past 10 posts
  Keys:
      like_count: number of likes for that post as an integer
      comments_count: number of comments for that post as an integer
      insights: a dictionary storing that posts insights
        Keys:
          impressions: the number of impressions that post has
          reach: reach that post has
          engagement: the amount of engagement that post has

facebook_interface.py

makeApiCall(url, endpointParams, debug = 'no')

```
def makeApiCall(url, endpointParams, debug='no'):
    data = requests.get(url, endpointParams)
    response = dict()
    response['url'] = url
    response['endpoint_params'] = json.dumps(endpointParams, indent=4)
    response['json_data'] = json.loads(data.content)
# printing this will help with visualizing json data
    response['json_data_pretty'] = json.dumps(response['json_data'], indent=4 )

    return response
```

Description: A function that helps make the facebook API calls

Input:

url: the beginning of the url before any endpoints

endpointParams: a dictionary with all of the endpoints as keys mapped to values. 'access_token' must be a key mapped to your access token

Output: dictionary with the json data returned by the API call

Keys:

'url' : url you passed in

'endpointParams': the dictionary of endpoint parameters you passed in

'json_data' : json data returned by the API call, used to fetch data

'json_data_pretty': json data returned by the API call in organized format, used for debugging

Example: If you wanted to get the API for the follower count you would make an API call to 'https://graph.facebook.com/{graph-api-version}/{pageid}?fields=followers_count'

```
Input:
url: https://graph.facebook.com/{graph-api-version}/{pageid}
endpointParams: endpointParams['access_token'] = {your-access-token} (this is always needed)
                endpointParams['fields'] = 'followers_count'
Output: response['json_data'] = {
                  "followers_count": 1206,
                  "id": "1503119803264962"
                 }
```

debugAccessToken()

```
def debugAccessToken():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['input_token'] = params['page_access_token']
    endpointParams['access_token'] = params['page_access_token']
    url = params['graph_domain'] + '/debug_token'
    response = makeApiCall(url, endpointParams, params['debug'])
    return response
```

Description: Makes an API call with the access token and returns a dictionary of information on our access token

Output: a dictionary with important information for our access token

Important Keys:

'expires_at': stores when the access token expires. This values can be translates into a readable date using datetime.datetime.fromtimestamp()

postIds()

```
def postIds():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    url = params['endpoint_base'] + 'me/posts'
    response = makeApiCall(url, endpointParams, params['debug'])
    ids = []
    for i in range(len(response['json_data']['data'])):
        d = dict()
        d['id'] = response['json_data']['data'][i]['id']
        d['timestamp'] = response['json_data']['data'][i]['created_time']
        ids += [d]
    return ids
```

Description: a list of dictionaries each containing the postid and timestampfrom most recent to least recent

Output:

id: list of dictionaries in order from most recent to least recent

Keys:

'id': holds the post id as a string

'timestamp': holds the timestamp of when the post was created

followers_count()

```
def follower_count():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['fields'] = 'followers_count'
    url = params['endpoint_base'] + params['page_id']
    response = makeApiCall(url, endpointParams, params['debug'])
    return response['json_data']['followers_count']
```

Description: Makes an API call and returns the follower count of this Facebook account as an integer

Output: an integer representing the number of followers

getImpressions()

```
def getImpressions():
  params = defines.getCreds()
  endpointParams = dict()
  endpointParams['metric'] = 'page_impressions'
  endpointParams['access_token'] = params['page_access_token']
  url = params['endpoint_base'] + params['page_id'] + '/insights'
  response = makeApiCall(url, endpointParams, params['debug'])
  impressions = dict()
  for i in range(len(response['json_data']['data'])):
    if response['json_data']['data'][i]['period'] == 'day':
      impressions['day'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'week':
      impressions['week'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'days_28':
      impressions['days_28'] = response['json_data']['data'][i]['values'][1]['value']
  return impressions
```

Description: Makes an API call and returns a dictionary of the number of times any content from your Page or about your Page entered a person's screen over three different time periods. This includes posts, stories, ads, as well other content or information on your Page.

Output: Output: a dictionary with three keys representing different periods of time

Keys:	'day' stores the impressions over the last day
	'week' stores the impressions over the last week
	'days_28' stores the impressions over the last 28 days

getUniqueImpressions()

```
def getUniqueImpressions():
  params = defines.getCreds()
  endpointParams = dict()
  endpointParams['metric'] = 'page_impressions_unique'
  endpointParams['access_token'] = params['page_access_token']
  url = params['endpoint_base'] + params['page_id'] + '/insights'
  response = makeApiCall(url, endpointParams, params['debug'])
  impressions = dict()
  for i in range(len(response['json_data']['data'])):
    if response['json_data']['data'][i]['period'] == 'day':
      impressions['day'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'week':
      impressions['week'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'days_28':
      impressions['days_28'] = response['json_data']['data'][i]['values'][1]['value']
  return impressions
```

Description: Makes an API call and returns a dictionary of the number of people who had any content from your Page or about your Page enter their screen over a three different time periods. This includes posts, stories, check-ins, ads, social information from people who interact with your Page and more.

Output: Output: a dictionary with three keys representing different periods of time

Keys: 'day' stores the unique impressions over the last day

'week' stores the unique impressions over the last week

'days_28' stores the unique impressions over the last 28 days

getPostImpressions()

```
def getPostImpressions():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['metric'] = 'page_posts_impressions'
    endpointParams['access_token'] = params['page_access_token']
    url = params['endpoint_base'] + params['page_id'] + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    impressions = dict()
    for i in range(len(response['json_data']['data'])):
        if response['json_data']['data'][i]['period'] == 'day':
            impressions['day'] = response['json_data']['data'][i]['values'][1]['value']
        if response['json_data']['data'][i]['period'] == 'week':
            impressions['week'] = response['json_data']['data'][i]['values'][1]['value']
        if response['json_data']['data'][i]['period'] == 'days_28':
            impressions['days_28'] = response['json_data']['data'][i]['values'][1]['value']
    return impressions
```

Description: Makes an API call and returns a dictionary of the number of times your Page's post entered a person's screen over a three different time periods. Posts include statuses, photos, links, videos and more.

Output: Output: a dictionary with three keys representing different periods of time

Keys: 'day' stores the post impressions over the last day
'week' stores the post impressions over the last week
'days_28' stores the post impressions over the last 28 days

getUniquePostImpressions()

```
def getUniquePostImpressions():
  params = defines.getCreds()
  endpointParams = dict()
  endpointParams['metric'] = 'page_posts_impressions_unique'
  endpointParams['access_token'] = params['page_access_token']
  url = params['endpoint_base'] + params['page_id'] + '/insights'
  response = makeApiCall(url, endpointParams, params['debug'])
  impressions = dict()
  for i in range(len(response['json_data']['data'])):
    if response['json_data']['data'][i]['period'] == 'day':
      impressions['day'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'week':
      impressions['week'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'days_28':
      impressions['days_28'] = response['json_data']['data'][i]['values'][1]['value']
  return impressions
```

Description: Makes an API call and returns a dictionary of the number of people who had your Page's post enter their screen over a three different time periods. Posts include statuses, photos, links, videos and more.

Output: a dictionary with three keys representing different periods of time

Keys: 'day' stores the unique post impressions over the last day
'week' stores the unique post impressions over the last week
'days_28' stores the unique post impressions over the last 28 days

engagedUsers()

```python
def engagedUsers():
  params = defines.getCreds()
  endpointParams = dict()
  endpointParams['metric'] = 'page_engaged_users'
  endpointParams['access_token'] = params['page_access_token']
  url = params['endpoint_base'] + params['page_id'] + '/insights'
  response = makeApiCall(url, endpointParams, params['debug'])
  users = dict()
  for i in range(len(response['json_data']['data'])):
    if response['json_data']['data'][i]['period'] == 'day':
      users['day'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'week':
      users['week'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'days_28':
      users['days_28'] = response['json_data']['data'][i]['values'][1]['value']
  return users
```

Description: Makes an API call and returns a dictionary of the number of people who engaged with your Page over a three different time periods. Engagement includes any click.

Output: a dictionary with three keys representing different periods of time

Keys: 'day' stores the number of engaged users over the last day
'week' stores the number of engaged users over the last week
'days_28' stores the number of engaged users over the last 28 days

pageViews()

```python
def pageViews():
  params = defines.getCreds()
  endpointParams = dict()
  endpointParams['metric'] = 'page_views_total'
  endpointParams['access_token'] = params['page_access_token']
  url = params['endpoint_base'] + params['page_id'] + '/insights'
  response = makeApiCall(url, endpointParams, params['debug'])
  views = dict()
  for i in range(len(response['json_data']['data'])):
    if response['json_data']['data'][i]['period'] == 'day':
      views['day'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'week':
      views['week'] = response['json_data']['data'][i]['values'][1]['value']
    if response['json_data']['data'][i]['period'] == 'days_28':
      views['days_28'] = response['json_data']['data'][i]['values'][1]['value']
  return views
```

Description: Makes an API call and returns a dictionary of the number of times a Page has been viewed by logged-in and logged-out people over a three different time periods.

Output: a dictionary with three keys representing different periods of time
Keys: 'day' stores the number of page views over the last day
'week' stores the number of page views over the last week
'days_28' stores the number of page views over the last 28 days

totalPostEngagement()

```
def totalPostEngagement():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['metric'] = 'page_post_engagements'
    endpointParams['access_token'] = params['page_access_token']
    url = params['endpoint_base'] + params['page_id'] + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    engagements = dict()
    for i in range(len(response['json_data']['data'])):
        if response['json_data']['data'][i]['period'] == 'day':
            engagements['day'] = response['json_data']['data'][i]['values'][1]['value']
        if response['json_data']['data'][i]['period'] == 'week':
            engagements['week'] = response['json_data']['data'][i]['values'][1]['value']
        if response['json_data']['data'][i]['period'] == 'days_28':
            engagements['days_28'] = response['json_data']['data'][i]['values'][1]['value']
    return engagements
```

Description: Makes an API call and returns a dictionary of the number of times people have engaged with your posts through reactions, comments, shares and more over a three different time periods.

Output: a dictionary with three keys representing different periods of time
Keys: 'day' stores the post engagement over the last day
'week' stores the post engagement over the last week
'days_28' stores the post engagement over the last 28 days

negativeFeedback()

```
def negativeFeedback():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['metric'] = 'page_negative_feedback'
    endpointParams['access_token'] = params['page_access_token']
    url = params['endpoint_base'] + params['page_id'] + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    neg = dict()
    for i in range(len(response['json_data']['data'])):
        if response['json_data']['data'][i]['period'] == 'day':
            neg['day'] = response['json_data']['data'][i]['values'][1]['value']
        if response['json_data']['data'][i]['period'] == 'week':
            neg['week'] = response['json_data']['data'][i]['values'][1]['value']
        if response['json_data']['data'][i]['period'] == 'days_28':
            neg['days_28'] = response['json_data']['data'][i]['values'][1]['value']
    return neg
```

Description: Makes an API call and returns a dictionary of the number of times people took a negative action (e.g., un-liked or hid a post) over a three different time periods.

Output: a dictionary with three keys representing different periods of time

Keys: 'day' stores the negative feedback over the last day
'week' stores the negative feedback over the last week
'days_28' stores the negative feedback over the last 28 days

getPostLikes(postId)

```
def getPostLikes( postId ):
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['metric'] = 'post_reactions_like_total,post_reactions_love_total,post_reactions_wow_total'
    endpointParams['access_token'] = params['page_access_token']
    url = params['endpoint_base'] + postId + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    likes = response['json_data']['data'][0]['values'][0]['value']
    loves = response['json_data']['data'][1]['values'][0]['value']
    wows = response['json_data']['data'][2]['values'][0]['value']
    return likes + loves + wows
```

Description: Post likes for a specific post given the post id. Likes for Facebook since they have many positive reactions will be like, love, and wow reactions.
Input:

postId: the id of the post id you want the like count for passed in as a string

Output: The number of positive reactions on that post as an integer

getMultiplePostLikes(num)

```
def getMultiplePostLikes( num ):
   ids = postIds()
   total = 0
   for id in ids[:num]:
      total += getPostLikes(id)
   return total
```

Description: Gets the sum of likes over the last specified number of posts. Likes for Facebook since they have many positive reactions will be like, love, and wow reactions.

Input:

num: num is the number of posts you would like to see the likes for. Facebook only saves post information over the past 2 years so num will be reset to the number of posts saved

Output: The sum of positive reactions on those posts as an integer

longLivedAccessToken()

```
def longLivedAccessToken():
   params = defines.getCreds()
   endpointParams = dict()
   endpointParams['grant_type'] = 'fb_exchange_token'  # tell facebook we want to exchange token
   endpointParams['client_id'] = params['client_id']  # client id from facebook app
   endpointParams['client_secret'] = params['client_secret']  # client secret from facebook app
   endpointParams['fb_exchange_token'] = params['page_access_token']  # access token to get exchange for a long
lived token
   url = params['endpoint_base'] + 'oauth/access_token'
   response = makeApiCall(url , endpointParams)
```

Description: Makes an API call that allows you to get a long-lived access token with a short-lived access token

Note: your access token in your defines file must be a short-lived access token

Output: a dictionary with two keys

Keys: 'access_token' stores the long-lived access token

'expires_in' stores when the access token expires. The value stored here can be converted to a readable time with datetime.datetime.fromtimestamp(value) Value is that number stored at 'expires_in'

Long lived access tokens expire in 3 months. To use, print response['json_data_pretty'] and copy the access token into the defines file.

instagram_interface.py

makeApiCall(url, endpointParams, debug = 'no')

```
def makeApiCall(url, endpointParams, debug='no'):
    data = requests.get(url, endpointParams)
    response = dict()
    response['url'] = url
    response['endpoint_params'] = json.dumps(endpointParams, indent=4)
    response['json_data'] = json.loads(data.content)
 #printing this will help with visualizing json data
    response['json_data_pretty'] = json.dumps(response['json_data'], indent=4 )

    return response
```

Description: A function that helps make the facebook API calls

Input:

url: the beginning of the url before any endpoints

endpointParams: a dictionary with all of the endpoints as keys mapped to values. 'access_token' must be a key mapped to your access token

Output: dictionary with the json data returned by the API call

Keys:

'url' : url you passed in

'endpointParams': the dictionary of endpoint parameters you passed in

'json_data' : json data returned by the API call, used to fetch data

'json_data_pretty': json data returned by the API call in organized format, used for debugging

Example: If you wanted to get the API for the follower count you would make an API call to
'https://graph.facebook.com/{graph-api-version}/{pageid}?fields=followers_count'

Input:

```
    url: https://graph.facebook.com/{graph-api-version}/{pageid}
    endpointParams: endpointParams['access_token'] = {your-access-token} (this is always needed)
                    endpointParams['fields'] = 'followers_count'
Output: response['json_data'] = {
                        "followers_count": 1206,
    "id": "1503119803264962"
}
```

instagramPageId()

```
def instagramPageId():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['fields'] = 'instagram_business_account'
    url = url = params['endpoint_base'] + params['page_id']
    response = makeApiCall(url, endpointParams, params['debug'])
    return response['json_data']['instagram_business_account']['id']
```

Description: Used to get the Instagram business account id. Not necessary to call more than once, This id is stored in the defines file. Stored in ig_page_id key in getCreds() dictionary

Output: Instagram business account id as a string.

igFollowers()

```
def igFollowers():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['fields'] = 'followers_count'
    endpointParams['access_token'] = params['page_access_token']
    url = params['endpoint_base'] + params['ig_page_id']
    response = makeApiCall(url, endpointParams, params['debug'])
    return response['json_data']['followers_count']
```

Description: Makes an API call gets the number of followers for the client's Instagram account

Output: Number of followers as an integer

igPostCount()

```
def igPostCount():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['fields'] = 'media_count'
    url = params['endpoint_base'] + params['ig_page_id']
    response = makeApiCall(url, endpointParams, params['debug'])
    return response['json_data']['media_count']
```

Description: Make an API call and gets the number of posts the client's Instagram account has

Output: The number of posts as an integer

impressions()

```
def impressions():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['metric'] = 'impressions'
    endpointParams['period'] = 'day,week,days_28'
    url = params['endpoint_base'] + params['ig_page_id'] + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    imp = dict()
    imp['day'] = response['json_data']['data'][0]['values'][0]['value']
    imp['week'] = response['json_data']['data'][1]['values'][0]['value']
    imp['days_28'] = response['json_data']['data'][2]['values'][0]['value']
    return imp
```

Description: Makes an API call and returns a dictionary of the total number of times the IG User's IG Media have been viewed over a three different time periods. Includes ad activity generated through the API, Facebook ads interfaces, and the Promote feature.

Output: a dictionary with the number of impressions over multiple time periods

Keys: 'day' for the number of impressions in the last day
'week' for the number of impressions in the last week
'days_28' for the number of impressions in the last 28 days

reach()

```
def reach():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['metric'] = 'reach'
    endpointParams['period'] = 'day,week,days_28'
    url = params['endpoint_base'] + params['ig_page_id'] + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    r = dict()
    r['day'] = response['json_data']['data'][0]['values'][0]['value']
    r['week'] = response['json_data']['data'][1]['values'][0]['value']
    r['days_28'] = response['json_data']['data'][2]['values'][0]['value']
    return r
```

Description: Makes an API call and returns a dictionary of the total number of unique users who have viewed at least one of the IG User's IG Media over a three different time periods. Repeat views and views across different IG Media owned by the IG User by the same user are only counted as a single view. Includes ad activity generated through the API, Facebook ads interfaces, and the Promote feature.

Output: a dictionary with the reach over multiple time periods

Keys: 'day' for the reach in the last day

'week' for the reach in the last week

'days_28' for the reach in the last 28 days

profileViews()

```
def profileViews():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['metric'] = 'profile_views'
    endpointParams['period'] = 'day'
    url = params['endpoint_base'] + params['ig_page_id'] + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    return response['json_data']['data'][0]['values'][0]['value']
```

Description: Makes an API and returns the total number of users who have viewed the IG User's profile within the last day.

Output: the number of profile views in the past day as an integer

mediaInfo()

```
def mediaInfo():
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['fields'] = 'business_discovery.username(' + params['ig_username'] +
'){media{comments_count,like_count,timestamp}}'
    url = params['endpoint_base'] + params['ig_page_id']
    response = makeApiCall(url, endpointParams, params['debug'])
    info = []
    for id in response['json_data']['business_discovery']['media']['data']:
        info += [id]
    return info
```

Description: Makes an API call and returns a list of dictionaries containing information for each post. Each element is for a different post going from most recent to least recent. Each dictionary contains keys for each piece of information.

Output: A list of dictionaries containing information for each post.

Keys: 'id' for the post id
'like_count': the number of likes a post has
'comments_count': the number of comments a post has

'timestamp': the timestamp of when the post was posted

postInsights(id)

```
def postInsights( id ):
    params = defines.getCreds()
    endpointParams = dict()
    endpointParams['access_token'] = params['page_access_token']
    endpointParams['metric'] = 'impressions,reach,engagement'
    url = params['endpoint_base'] + str(id) + '/insights'
    response = makeApiCall(url, endpointParams, params['debug'])
    insights = dict()
    insights['impressions'] = response['json_data']['data'][0]['values'][0]['value']
    insights['reach'] = response['json_data']['data'][1]['values'][0]['value']
    insights['engagement'] = response['json_data']['data'][2]['values'][0]['value']
    return insights
```

Description: Makes an API call and returns a dictionary with post insights where the id of the post is passed in as an argument.

Input:

id: an id of a post passed in as a string

Output: a dictionary of post insights for that post

Keys: 'impressions': holds the total number of times the IG Media object has been seen
'reach': holds the total number of unique Instagram accounts that have seen the IG Media object
'engagement': hold the sum of likes_count, comment_count and saved counts on the IG Media

defines.py

getCreds()

```
def getCreds():
   creds = dict()
   creds['page_access_token'] =
'EAAI3fQapZATcBACU37WCWJjRqE0sAf6MNAuL7UTPE3Qb94bDxO9DU3jjb25S2bnmCZBXQydUQmlbLNDGCZCcEio
ELPMvTMXAdBWsP38NCn7GHW1MTXiNFXZCDe7vLEOx4medAfx8xzZAnISQ6bv0ZBInBfpwO1phNf6ZAZABQqohK7l
CTUfuZAkm1'
   creds['client_id'] = '623960075625783'
   creds['client_secret'] = ''
   creds['ig_username'] = 'impactfinctr'

   creds['graph_domain'] = 'https://graph.facebook.com/'
   creds['graph_version'] = 'v12.0'
   creds['endpoint_base'] = creds['graph_domain'] + creds['graph_version'] + '/'
   creds['debug'] = 'no'
   creds['page_id'] = '1503119803264962'
   creds['ig_page_id'] = '17841404219581713'
   return creds
```

Description: Returns a dictionary with information to help make API calls for facebook and instagram. A place to store useful information needed to make many API calls.

Output:

Keys:

page_access_token: stores the access token that contains all of the permissions we need to access information on this facebook/instagram page this is a long-lived access token and needs to be replaced every 3 months

client_id: id for the facebook app we have created, can be found on the facebook developer account

client_secret: secret for the facebook app we have created, can be found on the facebook developer account

ig_username: instagram username for the client

graph_domain: every API call starts with 'https://graph.facebook.com/' so we stored it in graph_domain to reduce repetition

graph_version: there are different versions of the graph API and it also needs to be passed into the url for most API calls so we stored it in our dictionary to remain consistent

endpoint_base: combination of graph domain and graph version, this is used as the first part of the url for most API calls

debug: no for if you don't want to debug, yes if you do page_id: page id for facebook page, stored so we don't have to make that API call multiple times

ig_page_id: busniess account id for instagram account, stored so we don't have to make that API call multiple times