Najeem Muhammed

3 Followers About Follow

Analyzing my Google Photos library with python and pandas

Najeem Muhammed Jul 28, 2020 · 6 min read

I have been backing up almost all of my images in Google Photos for almost 10 years now. I have always been wondering, what kind of interesting information I can unearth if I can download the EXIF data of all those images into a data analysis tool like a Pandas Dataframe.

After few hours of research and experimenting, I achieved more than what I initially planned. This post is me trying to write it down for myself and anyone who is interested in doing the same.

Intended Audience

A Brief Overview

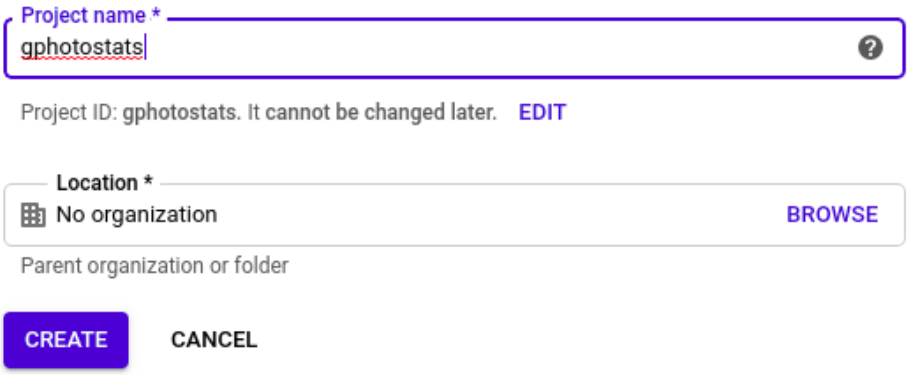
I figured that I need to create an app in Google Cloud Console, let this app access my Google Photos data and download and analyze this data. Below are the main steps which I took.

1. Create an app in Google Cloud Platform and enable Google Photos API in the app.
2. Set up and download credentials of the newly created app.
3. Use the above credentials in python code to run the app and connect to my Google Photos account.
4. Download the data.
5. Load the data into a Pandas Dataframe.
6. Analyze the data.

Details of each step as follows

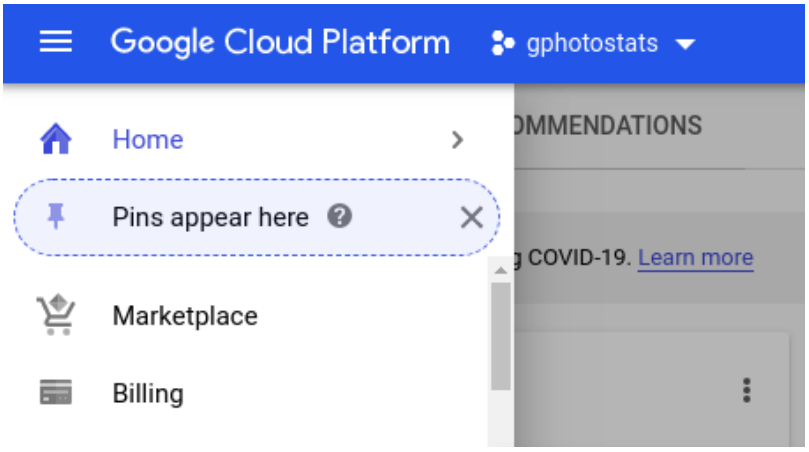
Step 1: Creating an app in Google cloud Platform

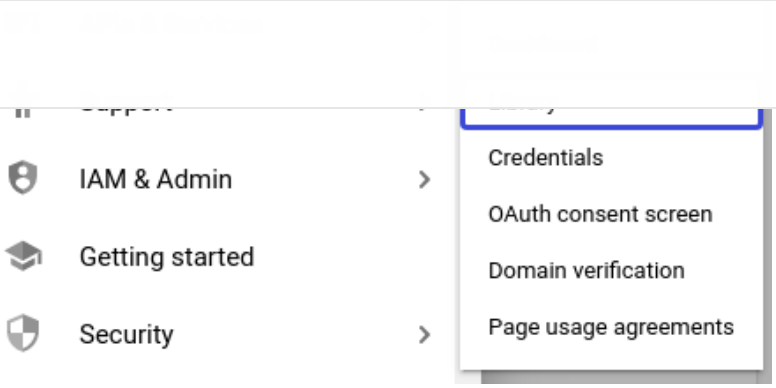
Head to Google Cloud Console dashboard and click *Create Project* button and create a new project.



Create a project

Now you need to enable Google Photos API in this project. Use the navigation menu to browse to *APIs & Services* and *Library*.





Navigate to API Library

In the new page, search for *Photos Library API* and click *Enable*. You may get a warning saying to use this API, you may need credentials and you need to create one. That's our next step.

Step 2: Create and download credentials

There is one more step before creating the credential. That is to configure a consent screen for our app when it tries to access our (or someone else's) data. You may be familiar with this page if you have authenticated any app to access your data in Google account.

Navigate to *APIs & Services > OAuth Consent Screen*.

Select *External* and click *Create*. The only info we need to fill in the new page is the 'Application Name'. The rest we can leave blank for now.

Now it's time to create the credentials. Navigate to *APIs & Services > Credentials* and click *Create Credentials* at the top of the screen and select *OAuth client ID*. Select *Desktop app* and provide a name.

Once you've created an OAuth credential, you can download a json file with the OAuth Client ID. Save it as `credentials.json`. We will use this file in our python script.

Step 3: Let's write some python code

I have created a github repo with all the code used in this article. I'd recommend to create a virtual env to install the dependencies. Use the `requirements.txt` in the github repo to install the dependencies. I wrote the code in a Jupyter Lab. This makes the interactive data analysis all the more convenient.

The first part of the code is connecting to the API

```
import pickle
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request

```

```python
creds = None
# The file token.pickle stores the user's access and refresh tokens, and is
# created automatically when the authorization flow completes for the first
# time.
if os.path.exists('token.pickle'):
    with open('token.pickle', 'rb') as token:
        creds = pickle.load(token)
# If there are no (valid) credentials available, let the user log in.
if not creds or not creds.valid:
    if creds and creds.expired and creds.refresh_token:
        creds.refresh(Request())
    else:
        flow = InstalledAppFlow.from_client_secrets_file(
            'credentials.json', SCOPES)
        creds = flow.run_local_server(port=0)
    # Save the credentials for the next run
    with open('token.pickle', 'wb') as token:
        pickle.dump(creds, token)

google_photos = build('photoslibrary', 'v1', credentials=creds)
```

gphotos_stats_connect.py hosted with ♡ by GitHub view raw

Running this code for the first time will pop up a browser window which will ask your permission to grant access to your Google Photos data to the app you just created. It will warn you that this app has not been verified by Google. and proceed only if you trust the developer. I guess you can trust yourselves and grant access to the app. Once you grant this access, the access token will be saved in a file `token.pickle` . You can reuse this file to connect to the service the next time.

Ultimately, this code will create a resource (`google_photos`) for interacting with the API. In the next step, we'll use this resource to extract info from the API.

Step 4: Download the data

Now that we have connected python in our system and our Google Photos data through the API, we're all set to download the data. Below code will iteratively download `mediaItem` (photo/video) information using the API. You can find more about the API here.

```python
items = []
nextpagetoken = None
# The default number of media items to return at a time is 25. The maximum pageSize is 100.
while nextpagetoken != '':
    print(f"Number of items processed:{len(items)}", end='\r')
    results = google_photos.mediaItems().list(pageSize=100, pageToken=nextpagetoken).execute()
    items += results.get('mediaItems', [])
```

It may take a few minutes to query the whole library. All the data will be stored in the list `items`. The data will be in a nested dict format. A typical data item will look as below.

```
{
    'id': 'xxxx',
    'productUrl': 'https://photos.google.com/lr/photo/xxxx',
    'baseUrl': 'https://lh3.googleusercontent.com/xxxx',
    'mimeType': 'image/jpeg',
    'mediaMetadata': {
        'creationTime': '2020-07-12T22:56:54Z',
        'width': '3265',
        'height': '4898',
        'photo': {
            'cameraMake': 'Canon',
            'cameraModel': 'Canon EOS 550D',
            'focalLength': 154,
            'apertureFNumber': 5.6,
            'isoEquivalent': 160
        }
    },
    'filename': 'IMG_1229.JPG'
}
```

Now, let's convert this into a pandas dataframe.

Step 5: Loading Data into Pandas

We can directly convert the `items` list into a dataframe. However, since some of the data is nested, we need to further split the column into more columns. The following code will achieve this.

```
import pandas as pd
# Convert the list of dict into a dataframe.
df = pd.DataFrame(items)

# Taking the column mediaMetadata and splitting it into individual columns
dfmeta = df.mediaMetadata.apply(pd.Series)

# Combining all the different columns into one final dataframe
photos = pd.concat(
    [
        df.drop('mediaMetadata', axis=1),
        dfmeta.drop('photo', axis=1),
        dfmeta.photo.apply(pd.Series)
    ], axis=1
)
```

gphotos_stats_pandas.py hosted with ♡ by GitHub view raw

If you check the `dtypes` of this dataframe, you can see that some of the data are not in the format we would like them to be. If you check `photos.dtypes` here's how the output will look like.

```
id                  object
productUrl          object
baseUrl             object
mimeType            object
filename            object
creationTime        object
width               object
height              object
video               object
apertureFNumber    float64
cameraMake          object
cameraModel         object
focalLength        float64
isoEquivalent      float64
dtype: object
```

As you can see, some of the data types are not in the format we want them to be. Let's fix them.

```
# Convert the creation time to a datetime dtype
photos.creationTime = pd.to_datetime(photos.creationTime)

# Convert other numeric data into numeric dtypes
for c in ['width', 'height', 'apertureFNumber', 'focalLength', 'isoEquivalent']:
    photos[c] = pd.to_numeric(photos[c])
```

gphotos_stats_dtypefix.py hosted with ♡ by GitHub view raw

Since we've done so much work to get this data, it will be wise to save it to the disk in some form. I used `to_hdf` function of pandas dataframe to save it to the disk.

```
photos.to_hdf('google_photo_data.hdf', key='photos')
```

Now let's analyze the data!

Step 6: Data Analysis

Finally! We're ready for some fun!

Let's get some info out of this dataframe. What I'm doing here may not be what you're looking for, however, the following code snippets will give you some idea on what can be achieved.

```
In [2]: photos = pd.read_hdf('google_photo_data.hdf')
```

Total number of media items

The length of the dataframe will give us an idea of how many media items are there.

```
In [3]: len(photos)

Out[3]: 53517
```

Different types of media items

There may be different media types in the library. `mimeType` column will give us an idea about that.

```
In [4]: photos.mimeType.value_counts()

Out[4]: image/jpeg    52641
        video/mp4       508
        image/gif       231
```

GooglePhotosStats_analyze_1.ipynb hosted with ♡ by GitHub view raw

If you have photos from different time zones, you may have to do some investigations here as the data may not be showing the local time and instead showing the time from your current time zone.

Displaying image from Google Photos in the Jupyter Notebook

You can also display any image from your library inside a Jupyter notebook using the `IPython.display.Image` notebook function. We will have to get the `baseUrl` of the image that we want to display. However, the `baseUrl` becomes outdated after 1 hour. So we need to get the updated `baseUrl` every hour. I wrote a small helper function to get the updated `baseUrl` and display the image.

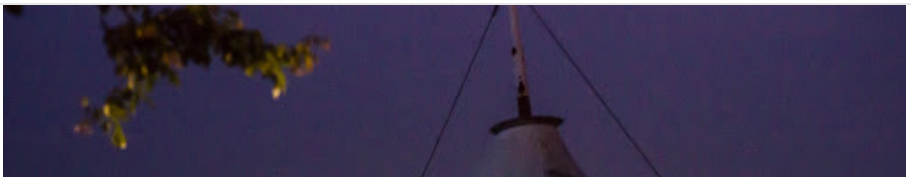
```
In [1]: import pickle

        import pandas as pd

        from IPython.display import Image
        from googleapiclient.discovery import build
```

```
In [2]: with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)
        google_photos = build('photoslibrary', 'v1', credentials=creds)
```

```
In [3]: def display_image(id):
            img = google_photos.mediaItems().get(mediaItemId=id).execute()
            return Image(img['baseUrl'], format='jpg')
```

```
In [4]: photos = pd.read_hdf('google_photo_data.hdf')
```

Out[5]:



GooglePhotosStats_display_image.ipynb hosted with ♡ by GitHub view raw

Since every call to the API will eat away from your quota for the day, it's wiser to limit the number of calls.

More analysis

In [1]:
```
import pandas as pd
```

In [2]:
```
photos = pd.read_hdf('google_photo_data.hdf')
```

Orientation of the image

If width is more than height, it's landscape else portrait. If they're equal, it's a square!

In [3]:
```
def orientation(row):
    if row['width'] > row['height']:
        return "Landscape"
    elif row['height'] > row['width']:
        return "Portrait"
    return "Square"
photos['orientation'] = photos.apply(orientation, axis=1)
```

In [4]:
```
photos.groupby('orientation').id.count().plot(kind='barh')
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe1392e7130>

GooglePhotosStats_analyze_2.ipynb hosted with ♡ by GitHub view raw

There is a lot more that can be done, but I hope you get the idea!

Closing thoughts

A lot more analysis can be done with the data that we have extracted. Since Google does not store the whole EXIF data, a lot of info is unavailable. If you have maintained your own archive of images, you can use tools like Exiftool to extract the EXIF data and create a pandas dataframe out of that. The rest of the process will be similar to what is shown here.

I'm curious to know what kind of interesting stats you unearthed from your Google Photos collection.

Python Pandas Google Data Science Photography