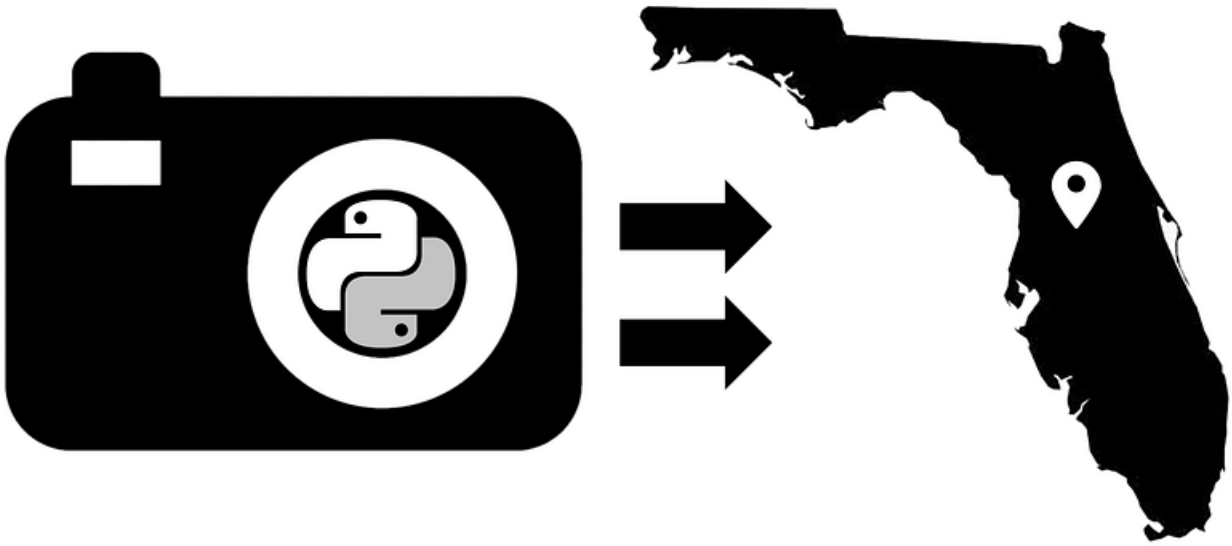
Data Science with Python! Extracting Metadata from Images



Hi everyone! In this post we will learn how to extract metadata from images.

“Metadata is structured information that describes, explains, locates or otherwise makes it easier to retrieve, use, or manage an information resource. Metadata is often called data about data or information about information”[1]. The metadata for digital images can contain the location where a photograph was taken, the type of camera used and the the time the photograph was taken. The metadata can be accessed using Python and Pillow, a module to work with images.

Packages Utilized in the Article

The following packages were utilized to display, extract and plot information from images: IPython, Pillow and ipyleaflet. To code along, it is recommended to set up a Python virtual environment to install the

packages or use a web IDE such as Google Colab. Please note to use Google Colab you must be signed into a Google account.

Extracting Metadata with Pillow

An Exif (Exchangeable Image File Format) file stores metadata for digital photographs. Many phones and cameras create these data files each time a picture is taken[2]. For this article we will extract metadata from the photo below labeled climb_photo.jpg.

Pillow can be used to access the data, the function below was created by a user on a Stack Overflow post to extract Exif data[3]. The function takes a single parameter, the path containing the image. An empty dictionary

is created which will store the Exif metadata. The photograph is read into Pillow's Image class and the information is extracted using the built-in _getexif() function. Some photos do not contain metadata, if the Exif file is blank then the function stops running. If the photograph does contain an Exif file, the function decodes the Exif tags and saves the data into the exif_data dictionary.

```
def get_exif(filename):
    exif_data = {}
    image = Image.open(filename)
    #save the metadata in the picture into a variable
    info = image._getexif()
    if info:
      #match the numerical tags to the ExifTags dictionary
      for tag, value in info.items():
          decoded = ExifTags.TAGS.get(tag, tag)
          #nested dictionary containing location data under 'GPSInfo'
          if decoded == "GPSInfo":
              gps_data = {}
              for gps_tag in value:
                  sub_decoded = ExifTags.GPSTAGS.get(gps_tag, gps_tag)
                  gps_data[sub_decoded] = value[gps_tag]
              exif_data[decoded] = gps_data
          else:
              exif_data[decoded] = value

    return exif_data
```

For our photo we will run the function and save the data into the following variable: exif_data = get_exif('climb_photo.jpg').The following keys return some of the most relevant metrics: Make (the manufacturer of the phone), Model (model of the phone), DateTime (the date and time the photo was taken), ImageWidth & ImageHeight (dimensions of the pho) and GPSInfo (dictionary containing GPS data). Accessing exif_data[DateTime] we find that the photo was taken on February 22nd 2023 at 12:37 pm.

Plotting the Location Where the Photograph Was Taken

Within the exif_data dictionary there is another dictionary containing

GPS data labeled GPSInfo. We can write another function to extract the location where the photo was taken. The function will take the Exif data in dictionary format as the input parameter. The function saves the GPS data into a variable called gps_metadata. The location can be expressed in decimal degree (DD) format, where latitude and longitude geographic coordinates are displayed as decimal fractions of a degree[4]. A few calculations need to be run to return the DD format.

The first block of code is used to determine if the longitude and latitude reference values are negative or positive. The latitude value is positive if it is north of the equator while the longitude is positive if it is east of the prime meridian. The latitude is negative if it is south of the equator and the longitude is negative if it is west of the prime meridian.

Next the degrees, minutes and seconds are calculated. The coordinates need to be converted from a Pillow data type to a floating point number using list comprehension. From there the minutes and seconds can be divided by their respective values then the degrees, minutes and seconds are summed together and multiplied by the latitude reference number. The function return the decimal degree values for the latitude and longitude in tuple form.

$$D_{dec} = \left(D + \frac{M}{60} + \frac{S}{3600}\right)(Ref)$$

Where: D_{dec} is the decimal degree, are the degrees, M are the minutes, S are the seconds and Ref the longitude or latitude reference

```
def gps_extract(exif_dict):
  gps_metadata = exif_dict['GPSInfo']
```

```
  #latitudinal information
  #positive latitudes are north of the equator, negative latitudes are south of the equator
  lat_ref_num = 0
  if gps_metadata['GPSLatitudeRef'] == 'N':
    lat_ref_num += 1
  if gps_metadata['GPSLatitudeRef'] == 'S':
    lat_ref_num -= 1

  lat_list = [float(num) for num in gps_metadata['GPSLatitude']]
  lat_coordiante = (lat_list[0]+lat_list[1]/60+lat_list[2]/3600) * lat_ref_num

  #longitudinal information
  #positive longitudes are east of the prime meridian, negative longitudes are west of the
prime meridian
  long_ref_num = 0
  if gps_metadata['GPSLongitudeRef'] == 'E':
    long_ref_num += 1
  if gps_metadata['GPSLongitudeRef'] == 'W':
    long_ref_num -= 1

  long_list = [float(num) for num in gps_metadata['GPSLongitude']]
  long_coordiante = (long_list[0]+long_list[1]/60+long_list[2]/3600) * long_ref_num


  #return the latitude and longitude as a tuple
  return (lat_coordiante,long_coordiante)
```

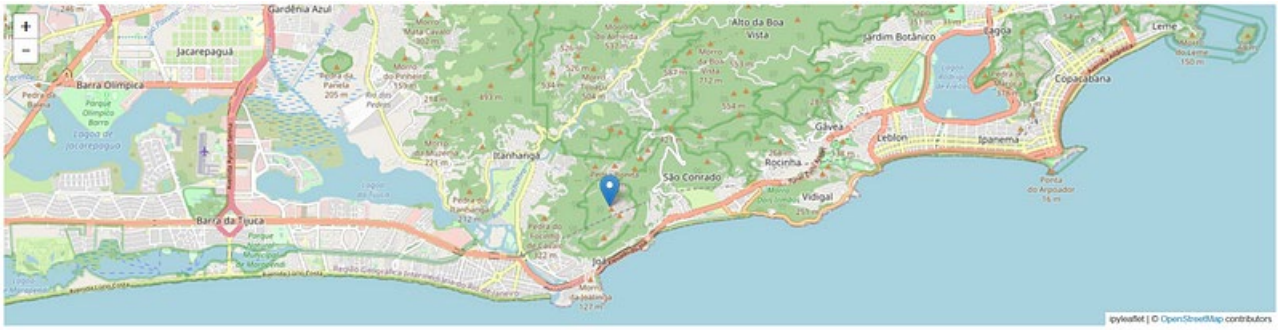
Using ipyleaflet, we can plot the location of the photo on a map. The tuple containing the latitude and longitude of where the photo was taken is used as an input into ipyleaflet's Map class. A marker can be added to the map using the same location. The map can then be displayed, we can see that the photo was taken in Rio de Janeiro on the Pedra da Gávea trail.

```
center = gps_extract(exif_data)

#create a map using ipyleaflet's "Map" class
m = Map(center=center, zoom=15)

#set a marker on a map
marker = Marker(location=center, draggable=False)
m.add_layer(marker);

m
```



Rio de Janeiro, Brazil

Removing Metadata From a Photograph

For privacy reasons, a person may want to delete metadata contained in a photograph before they share it with other people. Reading in the original climb_photo into the Pillow Image class, the original photo can be overwritten or a copy can be saved which contains no metadata. Running the get_exif on the copy, the output returns a blank dictionary meaning there is no metadata for the new image.

```
img = Image.open('climb_photo.jpg')
img.save('climb_no_meta.jpg')
get_exif('climb_no_meta.jpg')
```

References and Additional Information

The Python notebook containing the full code can be accessed on GitHub. A video tutorial accompanying this article can be viewed on YouTube.

Connect

Feel free to connect with Adrian on YouTube, LinkedIn, Twitter, GitHub and Odysee. Happy coding!

References

1. National Information Standards Organization. (2004). *Understanding Metadata*. Bethesda, MD: NISO Press.

2. Adobe. (2023). *EXIF Files*. Retrieved from adobe.com: https://www.adobe.com/creativecloud/file-types/image/raster/exif-file.html
3. Quintas, T. (2022, September 6). *Extract GPS data using python and PIL is failing*. Retrieved from stackoverflow.com: https://stackoverflow.com/questions/72530975/extract-gps-data-using-python-and-pil-is-failing
4. Wikipedia. (2023). *Decimal degrees*. Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Decimal_degrees

By Adrian Dolinay on March 13, 2023.

Canonical link

Exported from Medium on March 20, 2023.