Location Analysis Through My Gallery

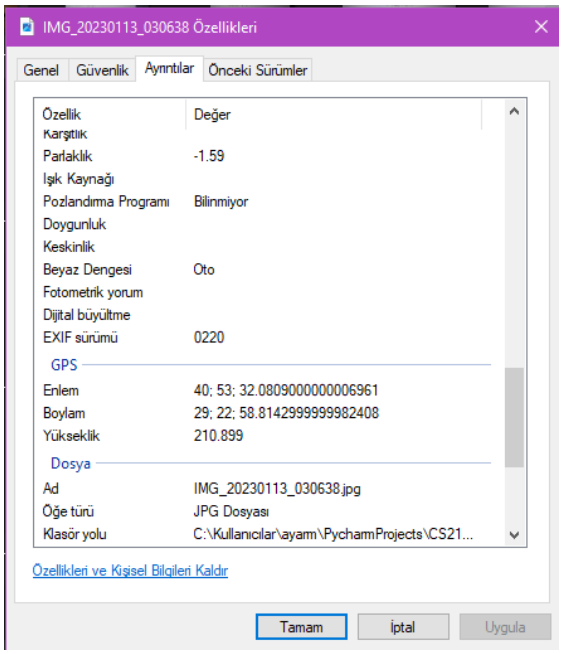
Motivation

I can say that this project was born when I started to use Google Maps' API and realized that there was a toll fee, so I drew a new route for myself. There is no end to the information we can obtain using the photographs in our gallery and the metadata embedded in these photographs

Among all this information, the one that caught my attention the most was, of course, GPS Info. I had the opportunity to test and prove many of my hypotheses in this adventure, which I started with only location, time and file name. In the following parts of my report, I will tell you about the method I used and the results I got. Good reading!

Data Source

My data source is my gallery itself. I am not interested in any types of photographing or something. I am a casual man who has a phone. I man only take photos when necessary (I have 64GB storage, so I mean it!) Still I got nearly 2000 photos in last 2 years. (Here at left my simple metadata and at right it is a simple JSON data which I get after I process my entire gallery with main.py)



main.py – MetaData to JSON file

```
import os
from PIL import Image
from PIL.ExifTags import TAGS, GPSTAGS
import json

folder_path = r"resources\images"
output_path = r"sources\files"
gps_info_dict = {}

for filename in os.listdir(folder_path):
    if filename.lower().endswith('.jpg'):
        img_path = os.path.join(folder_path, filename)
        image = Image.open(img_path)
        exif = {}

# Check if EXIF data is available
if image._getexif():
    for tag, value in image._getexif().items():
        if tag in TAGS:
            tag_name = TAGS[tag]
            if tag_name == 'DateTimeOriginal' or tag_name ==
'DateTimeDigitized':
                exif['DateTime'] = value
            elif tag == 34853:  # GPSInfo tag
                gps_info = {}
                for gps_tag, gps_value in value.items():
                    gps_tag_name = GPSTAGS.get(gps_tag, gps_tag)
                    if gps_tag_name == 'GPSDateStamp':
                        gps_info[gps_tag_name] = gps_value
                    elif gps_tag_name == 'GPSTimeStamp':
                        gps_info[gps_tag_name] = tuple(map(float,
gps_value))
                exif['GPSInfo'] = gps_info

    if exif:
        gps_info_dict[filename] = exif

json_output_file = os.path.join(output_path, 'gps_info.json')
with open(json_output_file, 'w') as json_file:
    json.dump(gps_info_dict, json_file, indent=2, default=str)
```

It simply iterating all images in the resources\images folder. And if EXIF data of the image exist then save it as json object to later export it out as gps_info.json. I only have .jps ‘s so it didn’t make any problem but the code can be upgraded to be able to work with other types of images too.

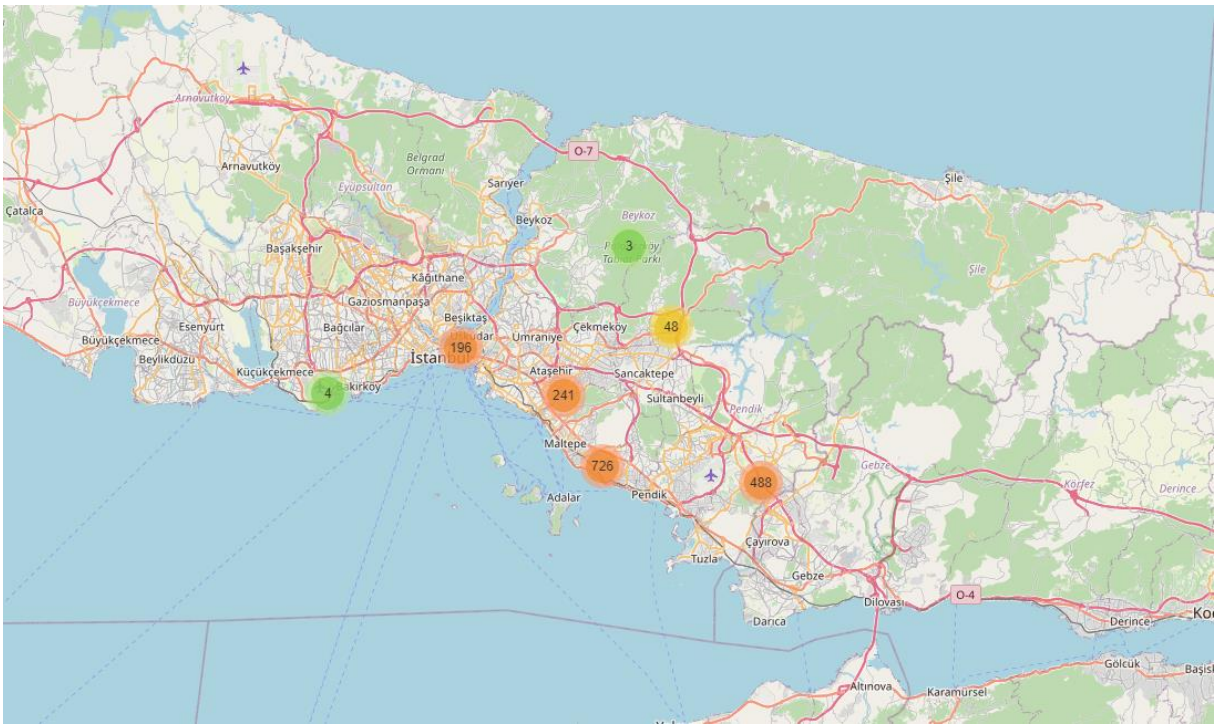
locator.py – JSON to Visualise on Map

```
import folium
from folium.plugins import MarkerCluster
import os
import json

# Assuming the locator.py file is in the project folder (CS210_PROJ)
folder_path = os.path.dirname(os.path.realpath(__file__))

# Corrected path to the images folder
image_folder_path = os.path.join(folder_path, 'resources', 'images')
json_file_path = os.path.join(folder_path, 'sources', 'files',
'gps_info.json')

with open(json_file_path, 'r') as json_file:
    gps_info_dict = json.load(json_file)

# create a map object
turkey_center = [39.9334, 32.8597]
mymap = folium.Map(location=turkey_center, zoom_start=6)

# Create Marker Cluster
marker_cluster = MarkerCluster(name="Markers Demo").add_to(mymap)

for filename, info in gps_info_dict.items():
    if 'GPSInfo' in info and '2' in info['GPSInfo'] and '4' in
info['GPSInfo']:
        lat_deg, lat_min, lat_sec = info['GPSInfo']['2']
        lon_deg, lon_min, lon_sec = info['GPSInfo']['4']

        # Check if the latitude and longitude information is present
        if lat_deg and lon_deg:
            # Convert degrees, minutes, seconds to decimal degrees
            lat = lat_deg[0] + lat_min[0] / 60.0 + lat_sec[0] / lat_sec[1]
/ 3600.0
            lon = lon_deg[0] + lon_min[0] / 60.0 + lon_sec[0] / lon_sec[1]
/ 3600.0

            # Adjust coordinates based on direction (N/S, E/W)
            if info['GPSInfo']['1'] == 'S':
                lat = -lat
            if info['GPSInfo']['3'] == 'W':
                lon = -lon

            # Create a Marker for each GPS location
            photo_path = "resources/images/" + filename

            folder_path = os.path.dirname(os.path.realpath(__file__))

            image_folder_path = os.path.join(folder_path, 'resources',
'images').replace("\\", "/")
            current_image_path = os.path.join(image_folder_path,
filename).replace("\\", "/")
            #print("XXXXXXXXXXXXXXXXXXXXXXX:", current_image_path)
            if os.path.exists(photo_path):
                #print(filename, " ", photo_path)
                popup_content = f"<b>Filename:</b> {filename}<br>" \
                                f"<b>DateTime:</b> {info.get('DateTime',
'')}<br>" \
```

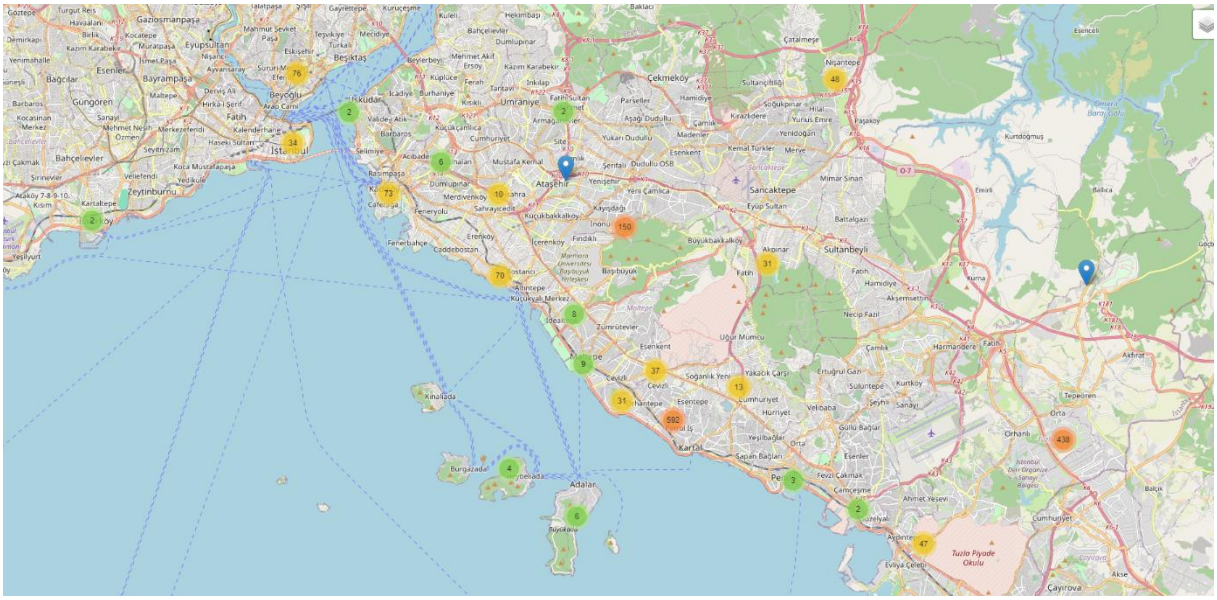
```
                                f"<img loading='lazy'
src='{current_image_path}' width='200'></a>"

                folium.Marker(location=[lat, lon],
                              popup=folium.Popup(popup_content,
max_width=300),
                              tooltip=f"DateTime: {info.get('DateTime',
'')}").add_to(marker_cluster)

# Layer control for turning Marker Cluster on/off
folium.LayerControl().add_to(mymap)

# Save the map object as html
output_html = os.path.join(folder_path, 'sources', 'files',
'gps_visualization_with_cluster_and_photo_popup.html')
mymap.save(output_html)

print(f' {output_html}')
print(image_folder_path)
```

Folium is a simple python library which made my job a lot easier than I thought. I simple create a map with real world size. And zoom in to turkey only. And for each JSON object which has a correct and implementable GPS and Date informations, we put a marker on the map. And in the last part I clustered these markers to get even a better looking map. Here Turkey:

And here is İstanbul:

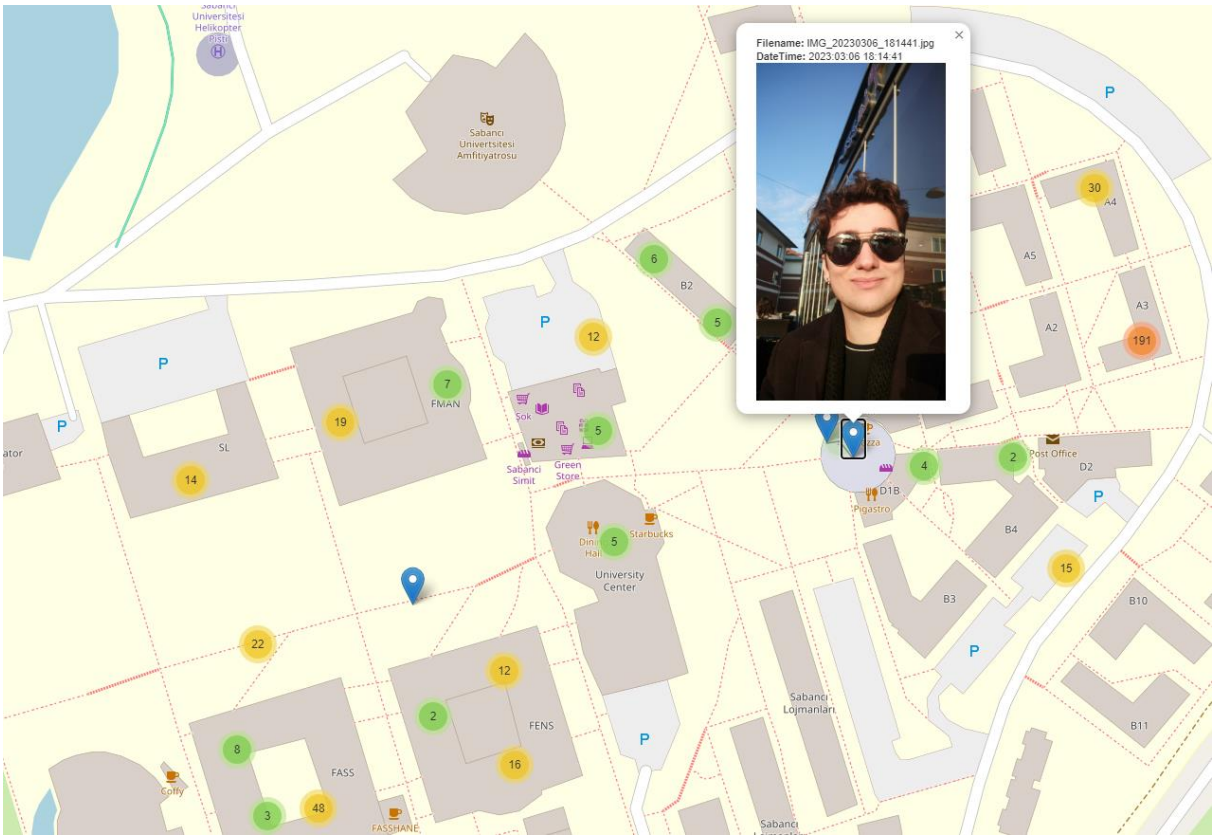


If we continue to zoom in those clusters, we get more clear and accurate locations. Here is Anatolia:



I was deeply impressed when I first saw these results. It's fascinating how precise the location information stored in our photographs is! Here is Sabancı University:

We get a popup when we click on a marker which shows a preview.



analysis.ipynb – whole analysis process from JSON

Here is my dataFrame which I will be working with for the next parts:

```
display(gps_data)
```

	Filename	DateTime	GPSInfo
0	IMG_20200106_130625.jpg	2020-01-06 13:06:25	{'7': [[10, 1], [6, 1], [21, 1]], '29': '2020:01:06'}
1	IMG_20210726_140940.jpg	2021-07-26 14:09:39	{'7': [[11, 1], [9, 1], [34, 1]], '29': '2021:07:26'}
2	IMG_20210729_124314.jpg	2021-07-29 12:43:14	{'7': [[9, 1], [43, 1], [9, 1]], '29': '2021:07:29'}
3	IMG_20220704_231605.jpg	2022-07-04 23:16:05	{'1': 'N', '2': [[39, 1], [56, 1], [61760, 10000]], '3': 'E', '4': [[32, 1], [51, 1], [409942, 10000]], '5': 0, '6': [941000, 1000], '7': [[20, 1], [15, 1], [50, 1]], '27': 'ASCIINETWORK', '29': '2022:07:04'}
4	IMG_20220704_231618.jpg	2022-07-04 23:16:18	{'1': 'N', '2': [[39, 1], [56, 1], [57746, 10000]], '3': 'E', '4': [[32, 1], [51, 1], [415576, 10000]], '5': 0, '6': [943300, 1000], '7': [[20, 1], [16, 1], [11, 1]], '27': 'ASCIINETWORK', '29': '2022:07:04'}
...	...	...	...
2055	IMG_20231025_153412.jpg	2023-10-25 15:34:12	{'1': 'N', '2': [[40, 1], [53, 1], [286735, 10000]], '3': 'E', '4': [[29, 1], [22, 1], [571958, 10000]], '5': 0, '6': [200599, 1000], '7': [[12, 1], [33, 1], [56, 1]], '27': 'ASCIINETWORK', '29': '2023:10:25'}
2056	IMG_20231025_162456.jpg	2023-10-25 16:24:56	{'1': 'N', '2': [[40, 1], [53, 1], [284391, 10000]], '3': 'E', '4': [[29, 1], [22, 1], [572120, 10000]], '5': 0, '6': [201199, 1000], '7': [[13, 1], [24, 1], [52, 1]], '27': 'ASCIINETWORK', '29': '2023:10:25'}
2057	IMG_20231025_162501.jpg	2023-10-25 16:25:01	{'1': 'N', '2': [[40, 1], [53, 1], [284391, 10000]], '3': 'E', '4': [[29, 1], [22, 1], [572120, 10000]], '5': 0, '6': [201199, 1000], '7': [[13, 1], [24, 1], [52, 1]], '27': 'ASCIINETWORK', '29': '2023:10:25'}
2058	IMG_20231025_162501_1.jpg	2023-10-25 16:25:01	{'1': 'N', '2': [[40, 1], [53, 1], [284391, 10000]], '3': 'E', '4': [[29, 1], [22, 1], [572120, 10000]], '5': 0, '6': [201199, 1000], '7': [[13, 1], [24, 1], [52, 1]], '27': 'ASCIINETWORK', '29': '2023:10:25'}
2059	IMG_20231025_162504.jpg	2023-10-25 16:25:04	{'1': 'N', '2': [[40, 1], [53, 1], [284391, 10000]], '3': 'E', '4': [[29, 1], [22, 1], [572120, 10000]], '5': 0, '6': [201199, 1000], '7': [[13, 1], [24, 1], [52, 1]], '27': 'ASCIINETWORK', '29': '2023:10:25'}

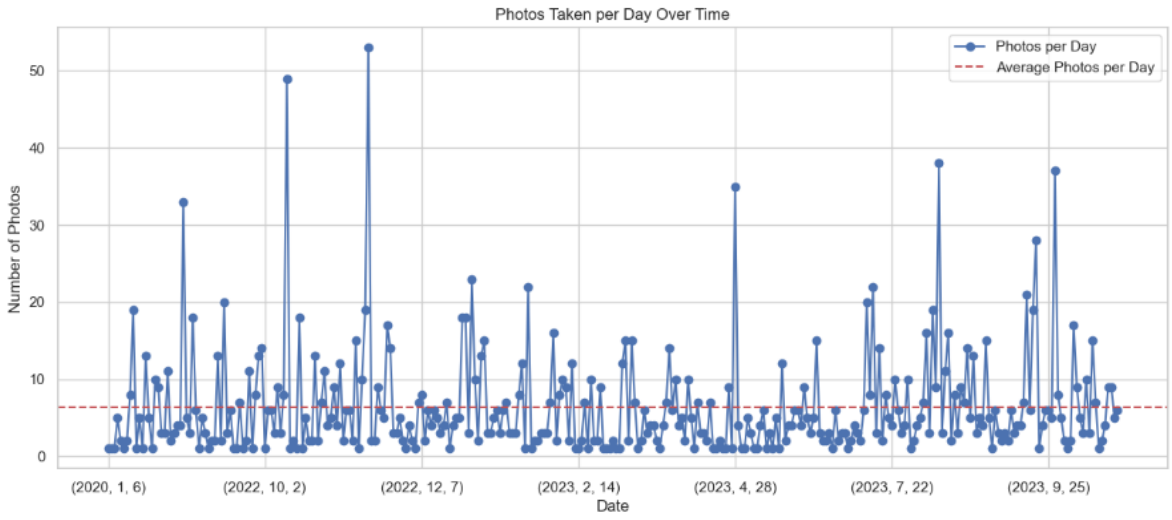
I will seperate the DateTime column into Day, Month and Year to make computations easier just for me. As you can see we have total of 2059 JSON objects. Some of them include a detailed GPS Info within their metadata, but some of them don't. We can check if a object location is within a spesific range to understand if the object is close to that point. For example:

```
for filename, info in json_data.items():
    if 'GPSInfo' in info and '2' in info['GPSInfo'] and '4' in info['GPSInfo']:
        lat_deg, lat_min, lat_sec = info['GPSInfo']['2']
        lon_deg, lon_min, lon_sec = info['GPSInfo']['4']

        # Check if the latitude and longitude information is present
        if lat_deg and lon_deg:
            # Convert degrees, minutes, seconds to decimal degrees
            lat = lat_deg[0] + lat_min[0] / 60.0 + lat_sec[0] / lat_sec[1] / 3600.0
            lon = lon_deg[0] + lon_min[0] / 60.0 + lon_sec[0] / lon_sec[1] / 3600.0

            # Adjust coordinates based on direction (N/S, E/W)
            if info['GPSInfo']['1'] == 'S':
                lat = -lat
            if info['GPSInfo']['3'] == 'W':
                lon = -lon

            # Check if the photo is in Istanbul
            if 40.8 <= lat <= 41.3 and 28.4 <= lon <= 30.2:
                photos_in_istanbul += 1

            # Check if the photo is in Ankara
            elif 39.7 <= lat <= 40.2 and 32.6 <= lon <= 33.1:
                photos_in_ankara += 1

            # Check if the photo is in Muğla
            elif 36.3 <= lat <= 37.5 and 27.0 <= lon <= 29.5:
                photos_in_mugla += 1

# Print the results
print(f"Number of photos taken in Istanbul: {photos_in_istanbul}")
print(f"Number of photos taken in Ankara: {photos_in_ankara}")
print(f"Number of photos taken in Muğla: {photos_in_mugla}")
```

```
Number of photos taken in Istanbul: 1706
Number of photos taken in Ankara: 48
Number of photos taken in Muğla: 77
```

Here is a graph that shows the average numbers of photos taken per day from 2020 to 2023:

```
gps_data['DateTime'] = pd.to_datetime(gps_data['DateTime'], format='%Y:%m:%d %H:%M:%S', errors='coerce')

# Drop rows with missing values
gps_data = gps_data.dropna()

# Extract day, month, and year
gps_data['Day'] = gps_data['DateTime'].dt.day
gps_data['Month'] = gps_data['DateTime'].dt.month
gps_data['Year'] = gps_data['DateTime'].dt.year

# Plotting
sns.set(style="whitegrid")
plt.figure(figsize=(15, 6))
# Use the desired column for plotting, for example, 'GPSInfo' represents the entire 'GPSInfo' dictionary
photos_per_day = gps_data.groupby(['Year', 'Month', 'Day'])['GPSInfo'].count()
average_photos_per_day = photos_per_day.mean()

photos_per_day.plot(marker='o', linestyle='-', label='Photos per Day')
plt.axhline(average_photos_per_day, color='r', linestyle='--', label='Average Photos per Day')
plt.title('Photos Taken per Day Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Photos')
plt.legend()
plt.show()
```



It strikes me that some days I take more photos than others, and I immediately investigate. I wonder if it has any connection with special days? The days which I took over 30 photos are: 8 August, the day me and my family went to an picnic with our friends; 12 October, the dak we went to Özyeğin University as archery club; 19 November, the day we took a lot of Interior photography for home restoration; 28 April, Basilica Cistern travel for SPS homework; 14 August, first day of my long term Turkcell internship so I went to HQ and took a lot of office and document photos; 27 September, we went to the orientation program in out campus as archery club.

```python
# Count the number of photos for each date
photo_counts = gps_data.groupby(['Year', 'Month', 'Day']).size().reset_index(name='NumPhotos')

# Filter dates with more than 30 photos
high_photo_dates = photo_counts[photo_counts['NumPhotos'] > 30]

# Display the dates with over 30 photos
print(high_photo_dates.to_string(index=False))
```

```
 Year  Month  Day  NumPhotos
 2022      8    8         33
 2022     10   12         49
 2022     11   19         53
 2023      4   28         35
 2023      8   14         38
 2023      9   27         37
```

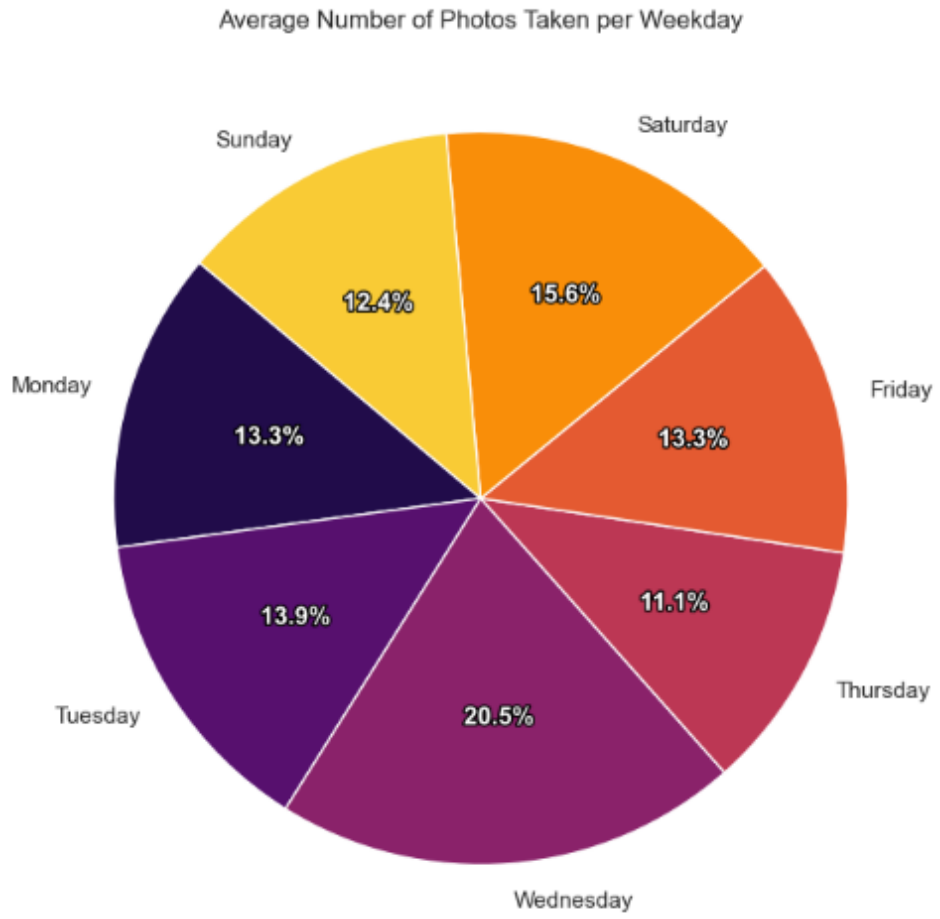
I will be using the points Yeditepe, Sabancı Universities and my home which is in Atalar/ Kartal. Why Yeditepe? Because we do our study dates generally at Caffee's near Yeditepe University:

In [20]:
```python
from geopy.distance import geodesic

# Function to calculate distance using Haversine formula
def haversine_distance(coord1, coord2):
    return geodesic(coord1, coord2).km

# Define the location of Yeditepe University
yeditepe_coords = (40.97588391234294, 29.15217540960381)

# Extract latitude and longitude from 'GPSInfo' column
gps_data['Latitude'] = gps_data['GPSInfo'].apply(lambda x: x.get('7', [[0, 0], [0, 0], [0, 0]])[0][0] if isinstance(x, dict) else
gps_data['Longitude'] = gps_data['GPSInfo'].apply(lambda x: x.get('7', [[0, 0], [0, 0], [0, 0]])[1][0] if isinstance(x, dict) el

# Recalculate the distance to Yeditepe University
gps_data['Distance_to_Yeditepe'] = gps_data.apply(lambda row: ((row['Latitude'] - yeditepe_coords[0])**2 + (row['Longitude'] - y

# Filter photos close to Yeditepe University
photos_near_yeditepe = gps_data[gps_data['Distance_to_Yeditepe'] < 22.5]

# Print the results
print(f"Number of photos taken close to Yeditepe University: {len(photos_near_yeditepe)}")

# Define the location of Sabancı University
sabanci_coords = (40.89126917789958, 29.378318763258005)

# Extract latitude and longitude from 'GPSInfo' column
gps_data['Latitude_Sabanci'] = gps_data['GPSInfo'].apply(lambda x: x.get('7', [[0, 0], [0, 0], [0, 0]])[0][0] if isinstance(x, d
gps_data['Longitude_Sabanci'] = gps_data['GPSInfo'].apply(lambda x: x.get('7', [[0, 0], [0, 0], [0, 0]])[1][0] if isinstance(x, 

# Recalculate the distance to Sabancı University
gps_data['Distance_to_Sabanci'] = gps_data.apply(lambda row: ((row['Latitude_Sabanci'] - sabanci_coords[0])**2 + (row['Longitude_

# Filter photos close to Sabancı University
photos_near_sabanci = gps_data[gps_data['Distance_to_Sabanci'] < 26.5]

# Print the results
print(f"Number of photos taken close to Sabancı University: {len(photos_near_sabanci)}")

# Define the location of your home
home_coords = (40.89920301192123, 29.17745313241991)

# Extract latitude and longitude from 'GPSInfo' column for your home
gps_data['Latitude_Home'] = gps_data['GPSInfo'].apply(lambda x: x.get('7', [[0, 0], [0, 0], [0, 0]])[0][0] if isinstance(x, dict
gps_data['Longitude_Home'] = gps_data['GPSInfo'].apply(lambda x: x.get('7', [[0, 0], [0, 0], [0, 0]])[1][0] if isinstance(x, dic

# Recalculate the distance to your home
gps_data['Distance_to_Home'] = gps_data.apply(lambda row: ((row['Latitude_Home'] - home_coords[0])**2 + (row['Longitude_Home'] -

# Filter photos close to your home
photos_near_home = gps_data[gps_data['Distance_to_Home'] < 28]  # Adjust the distance threshold as needed

# Print the results
print(f"Number of photos taken close to your home: {len(photos_near_home)}")
```

```
Number of photos taken close to Yeditepe University: 152
Number of photos taken close to Sabancı University: 415
Number of photos taken close to your home: 563
```

Here is another interesting statistic about me:

```
# Set Seaborn style and color palette
sns.set(style="whitegrid")
sns.set_palette("inferno", 7)


# Group by weekday and calculate the average number of photos
average_photos_per_weekday = gps_data.groupby(gps_data['DateTime'].dt.weekday)['GPSInfo'].count().mean()

# Count the number of photos per weekday
photos_per_weekday = gps_data.groupby(gps_data['DateTime'].dt.weekday)['GPSInfo'].count()

# Map weekday numerical values to their names
weekday_names = [calendar.day_name[i] for i in range(7)]

# Plotting a pie chart with custom text color
plt.figure(figsize=(8, 8))
pie = plt.pie(photos_per_weekday, labels=weekday_names, autopct='%1.1f%%', startangle=140)

# Customize text color
for text in pie[2]:
    text.set_color('white')
    text.set_fontsize(12)
    text.set_fontweight('bold')
    text.set_path_effects([withStroke(linewidth=2, foreground='black')])

plt.title('Average Number of Photos Taken per Weekday')
plt.show()
```

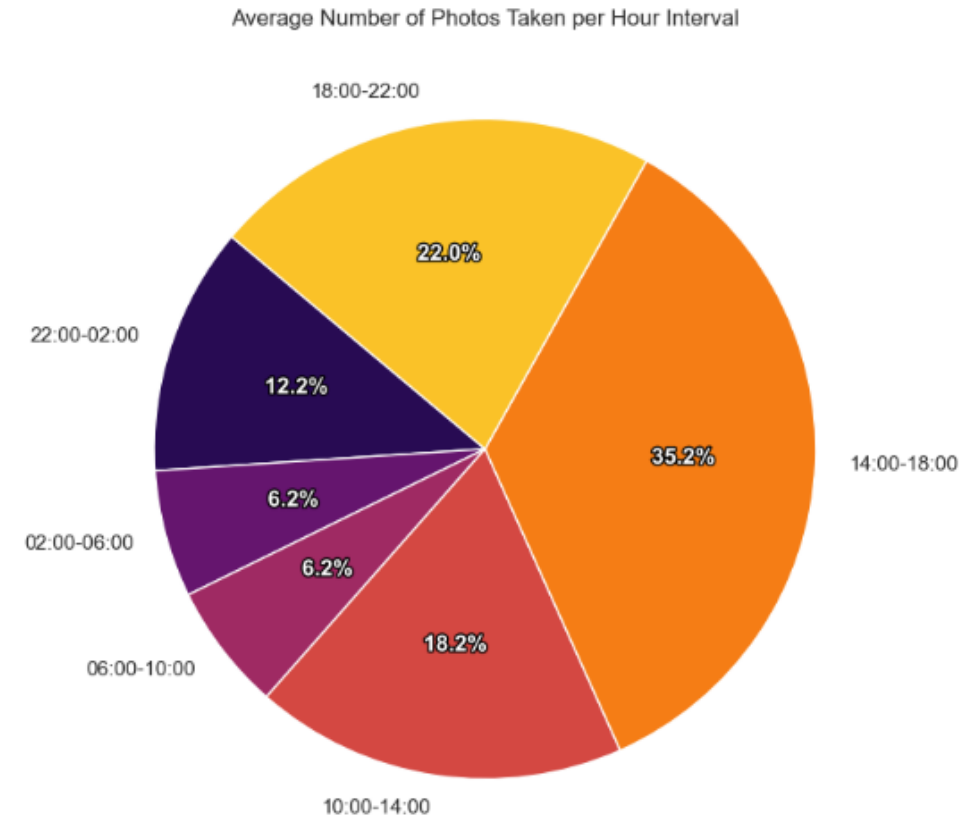


I thought I woul be takşin much more photos at weekends but just the opposite. Me and my gf generally meet at Yeditepe in Wednesdays, so it

is still suprising but interesting information. I set a borderline to numbers, and used inferno sa palette.

Here is another interesting fact about me:

```
gps_data['HourInterval'] = pd.cut(gps_data['Hour'], bins=[interval[0] for interval in time_intervals], labels=['00-02', '0

# Set Seaborn style and color palette
sns.set(style="whitegrid", palette="inferno")

# Group by hour interval and calculate the average number of photos
average_photos_per_hour_interval = gps_data.groupby('HourInterval')['GPSInfo'].count().mean()

# Count the number of photos per hour interval
photos_per_hour_interval = gps_data.groupby('HourInterval')['GPSInfo'].count()

# Map hour intervals to their names
hour_interval_names = ['22:00-02:00', '02:00-06:00', '06:00-10:00', '10:00-14:00', '14:00-18:00', '18:00-22:00']

# Plotting a pie chart with custom text color
plt.figure(figsize=(8, 8))
pie = plt.pie(photos_per_hour_interval, labels=hour_interval_names, autopct='%1.1f%%', startangle=140)

# Customize text color
for text in pie[2]:
    text.set_color('white')
    text.set_fontsize(12)
    text.set_fontweight('bold')
    text.set_path_effects([withStroke(linewidth=2, foreground='black')])

plt.title('Average Number of Photos Taken per Hour Interval')
plt.show()
```

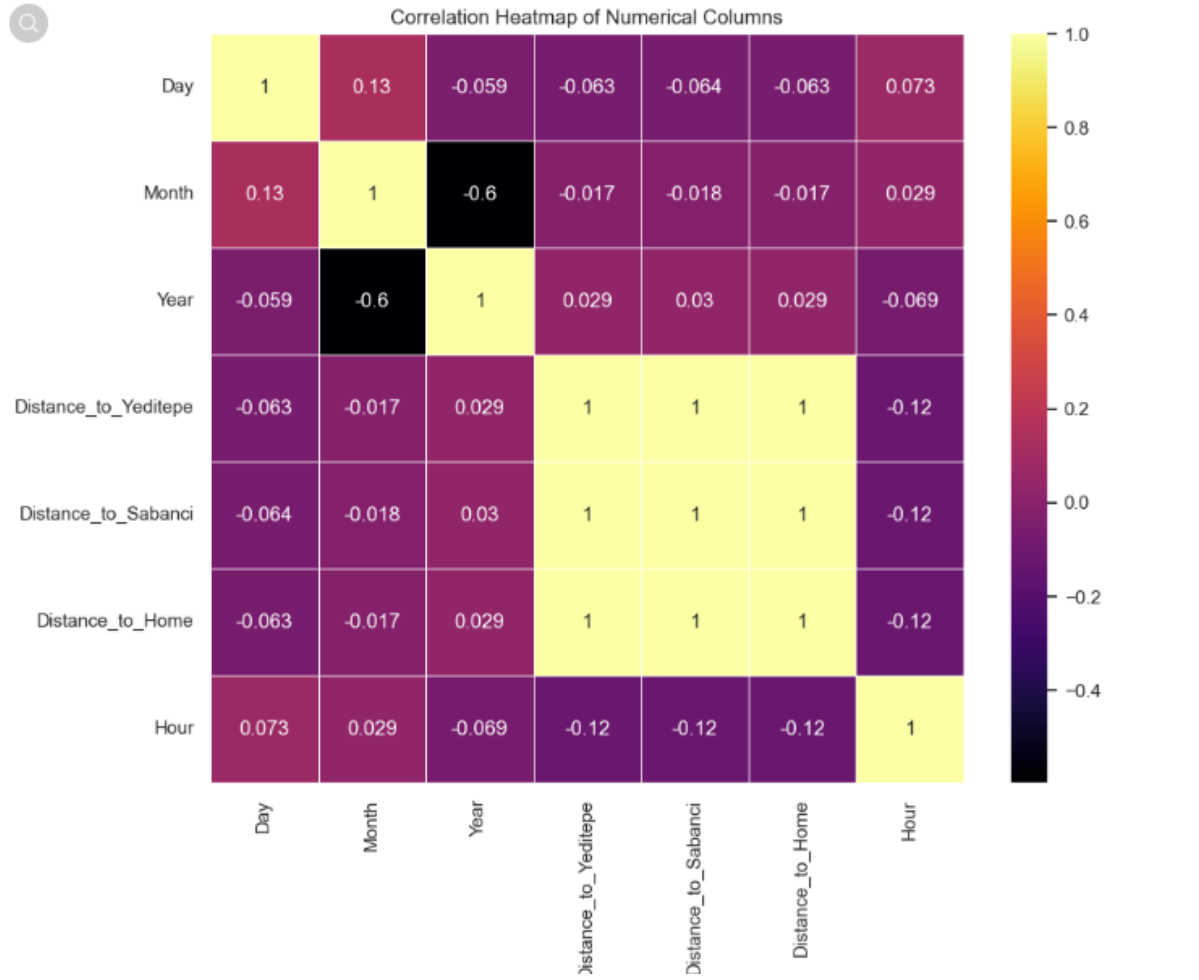


I divided day into 6 parts with 4 hours intervals each. The amout of photos I took ar 22 to 02 and 10 to 14 are really suprised me. If my phone wasn't 7 years old and it could save exposure settings in the

metadata too, I could prove that I was taking photos of my laptop screen at that time of night and create a hypothesis.

Here is a heatmap which shows us the correlation between numerical columns.

```
columns_to_remove = ["Latitude", "Longitude", "Latitude_Sabanci", "Longitude_Sabanci", "Latitude_Home", "Longitude_Home"]
gps_data.drop(columns=columns_to_remove, inplace=True)

numerical_columns = gps_data.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numerical_columns.corr()

# Set Seaborn style and color palette
sns.set(style="white")
sns.set_palette("inferno")

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="inferno", linewidths=.5)
plt.title('Correlation Heatmap of Numerical Columns')
plt.show()
```



As you can see I added Distance_toYeditepe/Home/Sabancı columns to easily calculate the necessary computations. And as we can see there is a high correlation between Distance_to_Yeditepe and Hour, it tells us that I was at Yeditepe University at more similar hours.

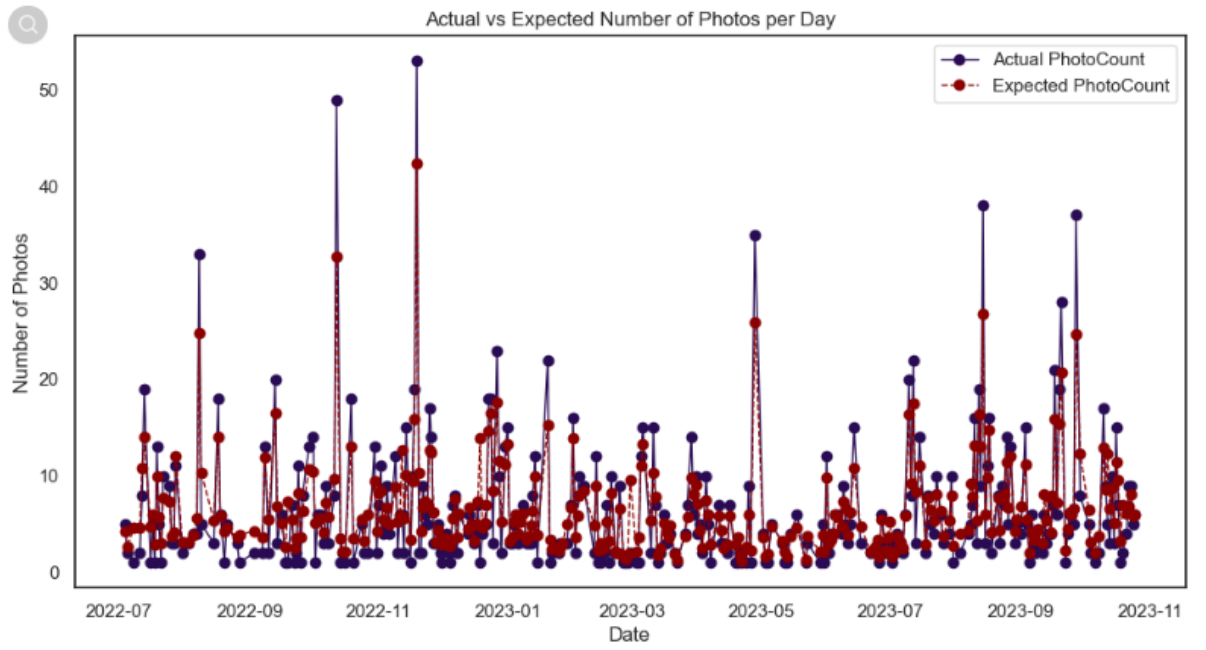
```
# Assuming 'exam_dates_list' is a list of exam dates in the format 'mm dd'
exam_dates_list = ['10 25', '08 15', '09 01', '09 15', '10 01', '10 15', '11 01', '11 15']

# Convert exam dates to datetime format
exam_dates = pd.to_datetime(['2023 ' + date for date in exam_dates_list], format='%Y %m %d')
print(exam_dates)

# Assuming you have a DataFrame named 'gps_data'
# Replace 'your_exam_dates_column' with the actual column name

# Step 1: Ensure 'Date' column is in datetime format
gps_data['Date'] = pd.to_datetime(gps_data['DateTime'].dt.date)

# Step 2: Extract hour information
gps_data['Hour'] = gps_data['DateTime'].dt.hour

# Step 3: Create a new column indicating whether the date corresponds to an exam date
gps_data['IsExamDate'] = gps_data['Date'].isin(exam_dates)

# Filter data for the last 4 months
last_4_months_data = gps_data[gps_data['DateTime'] >= gps_data['DateTime'].max() - pd.DateOffset(months=4)]

# Step 4: Plot histogram with two sides
plt.figure(figsize=(12, 6))

sns.kdeplot(last_4_months_data[last_4_months_data['IsExamDate'] == False]['Hour'], color='blue', label='Non Exam Days', fill=True
sns.kdeplot(last_4_months_data[last_4_months_data['IsExamDate'] == True]['Hour'], color='orange', label='Exam Days', fill=True)

plt.xlabel('Hour of the Day')
plt.ylabel('Density')
plt.title('Hourly Distribution of Photos Taken (Last 4 Months)')
plt.legend()

# Set x-axis limits to the actual range of hours
plt.xlim(0, 24)

plt.show()
```

```
DatetimeIndex(['2023-10-25', '2023-08-15', '2023-09-01', '2023-09-15',
               '2023-10-01', '2023-10-15', '2023-11-01', '2023-11-15'],
              dtype='datetime64[ns]', freq=None)
```

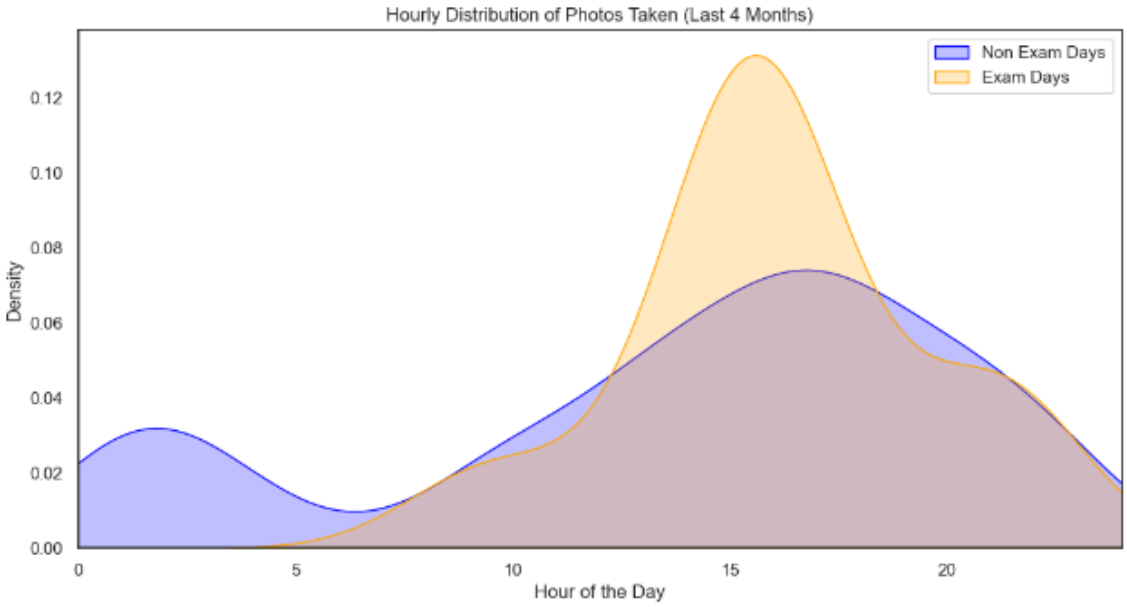
```
# Assuming 'data' is your DataFrame
# Convert 'DateTime' column to datetime format
new_gps_data = pd.DataFrame(data)

new_gps_data['DateTime'] = pd.to_datetime(new_gps_data['DateTime'], format='%Y:%m:%d %H:%M:%S')

# Extract features from 'DateTime'
new_gps_data['Year'] = new_gps_data['DateTime'].dt.year
new_gps_data['Month'] = new_gps_data['DateTime'].dt.month
new_gps_data['Day'] = new_gps_data['DateTime'].dt.day

# Count the number of photos for each date
actual_photo_count = new_gps_data.groupby(['Year', 'Month', 'Day']).size().reset_index(name='ActualPhotoCount')

# Extract features from 'GPSInfo' if needed and include them in X
# For simplicity, I'm using only 'Year', 'Month', 'Day' as features
X = new_gps_data[['Year', 'Month', 'Day']].drop_duplicates()

# Train a model (Random Forest Regressor as an example)
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, actual_photo_count['ActualPhotoCount'])

# Predict expected PhotoCount for the entire dataset
expected_photo_count = pd.Series(model.predict(X), name='ExpectedPhotoCount')

# Merge actual and expected counts based on the date
result['Date'] = pd.to_datetime(result[['Year', 'Month', 'Day']])
result = result[result['Date'] >= '2022-07-01'].sort_values(by='Date')

# Plot the actual and expected numbers
plt.figure(figsize=(12, 6))
plt.plot(result['Date'], result['ActualPhotoCount'], label='Actual PhotoCount', marker='o', linewidth=1)
plt.plot(result['Date'], result['ExpectedPhotoCount'], label='Expected PhotoCount', linestyle='--', marker='o', color="darkred",
plt.xlabel('Date')
plt.ylabel('Number of Photos')
plt.title('Actual vs Expected Number of Photos per Day')
plt.legend()
plt.show()
```

In the machine learning part I wanted to draw a regression line which allow me to compute the expected value for a point x. I used sklearn library, to train the program test the results with the actual datas. And

you can see the comparison between the Actial and the expected Number of Photos taken in Day:



And in the last part I wanted to comparise my numbers of photos taken and exam dates. I wanted to see in what hours did I take photos in both exam and non-exam days. (generally in exam days, I take too much photos to take notes etc.) Here is the results:



In non-exam days there is a normal distribution amoung number of photos taken per hour but in the exam days, nearly all of my photos are taken in a sinple scode of time.