National Technical University of Athens

Interdisciplinary Master's Programme in

Data Science and Machine Learning

Second Lab in the course of
Geospatial Big Data and Analytics

Geospatial Services and Web Applications

Written by: Christos Nikou
ID: 03400146

email:christosnikou@mail.ntua.gr

April 28, 2022

Contents

Introduction

In this second lab exercise we shift from the online Platforms for big ***Earth Observation (EO) Data Management and Analysis*** (like *GEE*) that we made extensive use in the previous lab and we move onto the analysis of *EO* data at a local level. The biggest advantage of working in an online Platform is that we do not have to worry about finding the data. Usually, everything is on the Platform in use. The greatest challenges one has to face when leaving from the safety zone of a well organized platform is 1) where to look for the proper data, 2) how to download them and 3) how to combine them efficiently in order to extract valuable information about the areas of interest.

For all the aforementioned challenges there has been a longstanding effort to develop the necessary tools in order to take advantage of the big amount of data being around the web. Nowadays, we could definitely say without a doubt that almost every home user is able to carry out advanced geospatial surveys without the need of sophisticated technological equipment. Some of the most important reasons for this are: 1) The free availability of a big portion of the *EO* data, 2) the effective accessibility of the data through the ***Web Services*** and 3) the simplification of the processing procedure of the data through the development of high level programming languages as well as compatible *API*'s with these languages for geospatial analysis. Therefore, the main goal of this lab is the effective combination and usage of the free available data with these developments.

The two protocols of *Web Services* that we make an extensive use of throughout this lab are the ***Web Map Service (WMS)***, which returns a rendered map image, and ***Web Feature Service (WFS)***, which typically returns GML formats. The ***Geographic Markup Language (GML)*** standard is one of the oldest and most widely used *XML* - based geographic formats. *XML* formats often contain the geometry, attributes and rendering instructions such as color, styling, and symbology of the area of interest. For processing and visualizing the *EO* data we use *Python* programming language and the *OWSLib, Geopandas, rasterio, folium, leafmap* libraries.

The Tasks of the Lab

Task 1

1.1 Data Gathering and Data Description

As we have already stated before, one of the main goals of this lab is to gather and combine efficiently the vast amount of geospatial data being on the web. In this section we present all the sources of information that we used throughout this lab and we describe the purpose of each one of them. The references in this section are also cited in the subsequent tasks when we make use of the data provided by them. This part serves as a compacted reference point of the sources and the procedure that we used to gather our data.

In our analysis we focus our attention on more than one area, mainly located in Europe. In the second task for example, the area of interest is the wider region of *Kastoria.* The data for this region are

gathered from *https://pithos.okeanos.grnet.gr/public/fBSNLJeNxerluMj2MVDqF* (be careful - downloading on click!). The link provides *raster* data for the wider region of Kastoria in a *tiff* format. The size of the data is about *3GB* and its the only dataset that is stored locally in the computer machine where the *Jupyter notebook* of this lab was developed. The data consists of 24 Landsat satellite images with the following 10 bands: 1) Blue, 2) Green, 3) Red, 4) NIR, 5) SWIR1, 6) SWIR2, 7) NDVI, 8) MSAVI, 9) NDWI, 10) NDBI. The first image is on date 2016/01/25 and the last on 2016/12/30. Since the size of the images is large, to speed up the analysis we focus our attention on a sub-region of Kastoria. In Subsection 2.2 we use a technique to clip the images according to a predefined sub-polygon of our initial polygon. The *geojson.io* provides a clear cut way of extracting *vector* data in *GeoJSON* format for the purpose of clipping the images.

Another useful source of information is the meteorological portal of *NASA* that we gathered all of the meteorological data used in subsequent subsections. In addition, we make extensive use of the data provided from *https://geodata.gov.gr/*. Most of the geospatial queries in section 3 are about Greece. We use the *WFS* services of *geodata.gov.gr* to obtain *vector* data for the administrative region boundaries, airport locations, railroads, coastlines, lakes and national parks in Greece.

Most of the data pulled from the *Web Services* can be found in the search engine provided by *https://www.geoseer.net/*. Some of the *WMS/WFS* that we gathered from *geoseer* are listed below.

(i) The *WMS* from *https://www.geoseer.net/rl.php?ql=4b4a6ba584ec26c2&p=1&q=landsat%20europe#* for a nice *RGB* image of Europe from Landsat satellite.

(ii) The *WFS* from *https://www.geoseer.net/rl.php?ql=0e305deee537335b&p=6&q=corine%20land%20cover#* for the Corine Land Cover of Belgium for the year 2018.

(iii) The *WFS* from *https://www.geoseer.net/rl.php?ql=71eea20d99580c29&p=1&q=european%20countries* for the *vector* data consisting of the European countries.

(iv) The *WFS* from *https://www.geoseer.net/rl.php?ql=984cb6cfa675d6f8&p=1&q=rivers%20europe* for the *vector* data consisting of the European rivers.

(v) The *WMS* from *https://www.geoseer.net/rl.php?ql=0fdcc2c406c9c7e2&p=1&q=corine%20land%20cover* for the Corine Land Cover of Italy for the year 2018 used in section 4 for the interactive maps.

Task 2 - Kastoria

At this point we download into our local disk the data from Sentinel-2 for the region of Kastoria from the following *link*. The data comes with a *pdf* file containing information about the data. The *tiff* file contains 24 images of the region of Kastoria between 2016-01-25 and 2016-12-30. Each image contains the following bands: 1) Blue, 2) Green, 3) Red, 4) NIR, 5) SWIR1, 6), SWIR2, 7) NDVI, 8) MSAVI, 9) NDWI, 10) NDBI. To read and process the *tiff* file we use the *rasterio* library. *Rasterio* reads the data as *numpy arrays* and allows us to gain information about the data like number of bands, image size as well as information about the spatial location of the images and their coordinate system. The data type is *<class "rasterio.io.DatasetReader">*, the collection consists of 240 images of size 2281×2017. The coordinate system is the *EPSG: 32634* and the spatial bounding box is (left=509220.0, bottom=4472900.0, right=532030.0, top=4493070.0).

2.1 Kastoria RGB Visualizations

Reading the first three images that correspond to the Blue, Green and Red channel of the image captured on 2016-01-25 by the satellite we can see how the wider region of Kastoria looks in these three channels. In the figure below we see the result of these images.

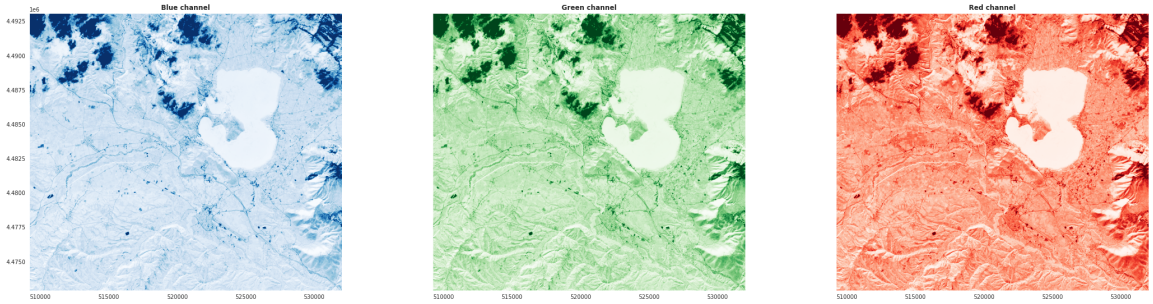

Figure 1: The Blue, Green and Red channels for the region of Kastoria.

Combining the three channels we obtain an *RGB* visualization of the first image.

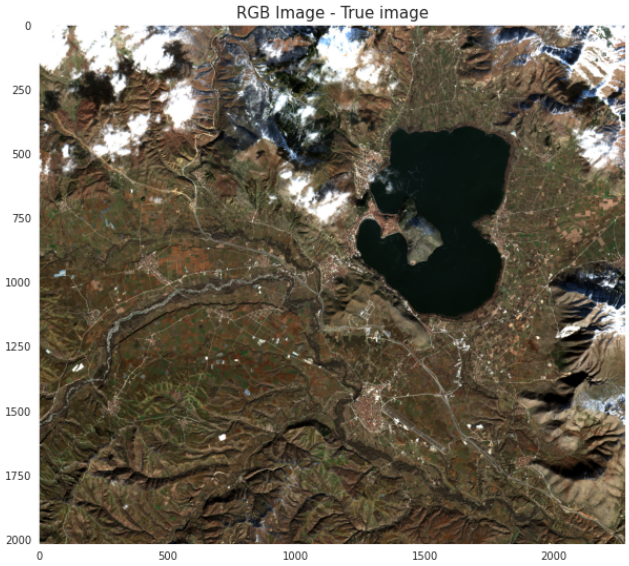

Figure 2: The *RGB* visualization of the first image.

2.2 Clipping the Images

At this point, using *geojson.io* we draw a polygon which is contained in the initial polygon representing the wider region of Kastoria. We download all the necessary *shp* files describing the sub - polygon and then using rasterio we convert the collection of images to same coordinate system with the *vector* data

and we keep the part of the images with boundaries described by the selected sub - polygon. In the figure below we can see the image in the initial coordinate system (*EPSG: 32634*) and the image obtained by the transformation to the *EPSG: 4326* coordinate reference system.

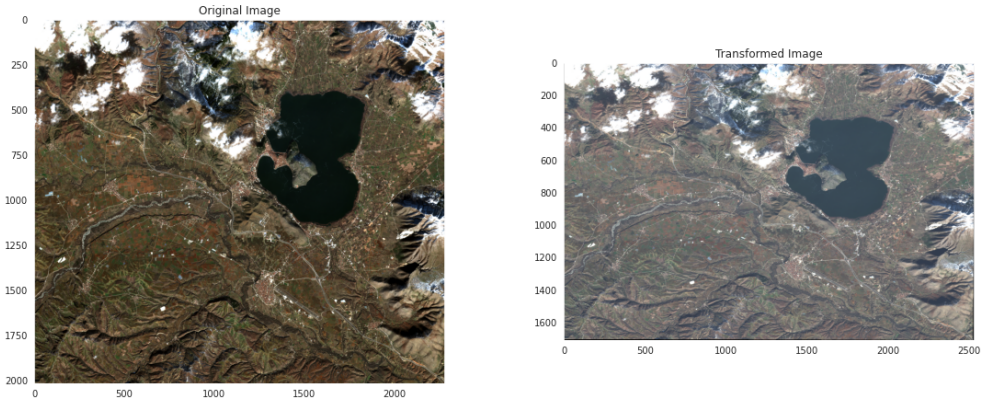

Figure 3: The original image (*EPSG: 32634*) and Transformed image (*EPSG: 4326*).

Below we see the transformed image next to the polygon of interest before clipping the images.

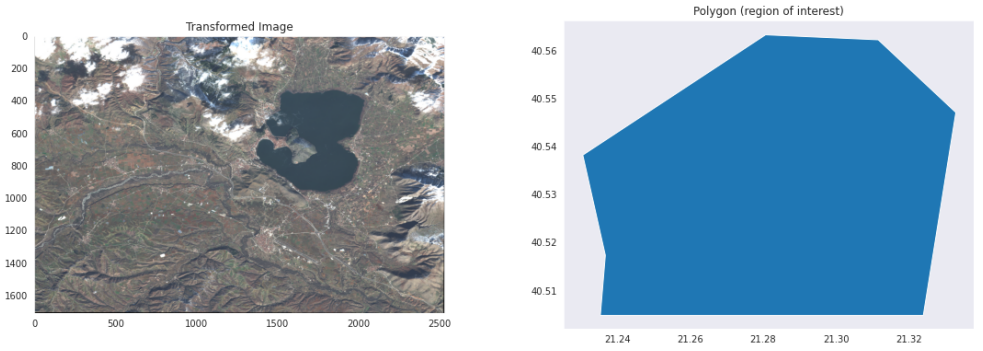

Figure 4: Before Clipping the images.

Clipping the images we get the following result.

Figure 5: The clipped image in *RGB* visualization.

2.3 NDVI Visualizations

Having the *NIR* and *Red* bands of the region for every image of the collection we create a visualization of the *Normalized Vegetation Index (NDVI)* which quantifies vegetation by measuring the difference between near-infrared (which vegetation strongly reflects) and red light (which vegetation absorbs). In precise mathematics, the *NDVI* is given by

$$NDVI = \frac{NIR - Red}{NIR + Red}. \tag{2.1}$$

NDVI always ranges from -1 to +1. But there is not a distinct boundary for each type of land cover. For example, when you have negative values, it's highly likely that it's water. On the other hand, if you have an *NDVI* value close to +1, there's a high possibility that it's dense green leaves. But when *NDVI* is close to zero, there is a sign of vegetation absence and it could even be an urbanized area. Healthy vegetation (chlorophyll) reflects more near-infrared (*NIR*) and green light compared to other wavelengths. But it absorbs more red and blue light.

This is why our eyes see vegetation as the color green. If we could see near-infrared, then it would be strong for vegetation too. Satellite sensors like Landsat and Sentinel-2 both have the necessary bands with *NIR* and *Red*. The calculation of the *NDVI* is pixel-wise with respect to the collection of images. This means that we apply the formula in (2.1) in each pixel of the image.

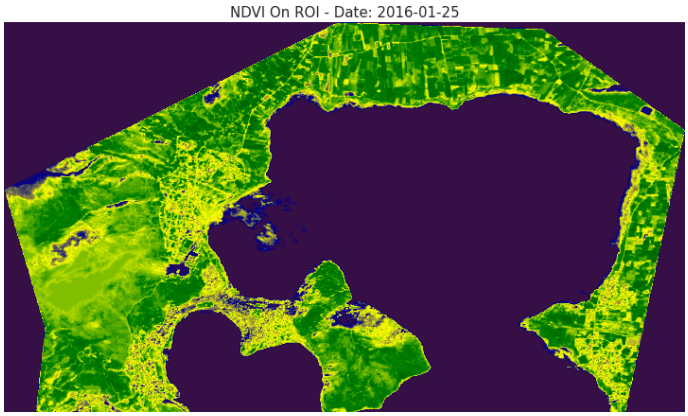

Figure 6: The *NDVI* visualization of the first image.

In Figure 6 the deep-purple colored pixels correspond to low values of the *NDVI*. As we can see in the middle of the picture corresponding to the lake has negative values or close to zero. Mid ranged values are represented by the blue colored pixels and for higher values we use yellow and green colors. Our next step is to obtain the *NDVI* visualization for every image of the collection. To achieve this, we construct a custom function named *index_gif* that generates a *gif* file with the *NDVI* images for the whole collection. In the following *link* you can see the result. Another approach to visualize the *NDVI* of the area is to calculate its mean value for each pixel separately. In this way, we can avoid extreme values which could be due to extreme phenomena, such as high cloud coverage for the day that the picture was captured. For example, we see in Figure 5 that the low-left part of the image is governed by clouds and hence, obscuring the information of this area. In the Figure below we can see more clearly what is happening in this part of the image.

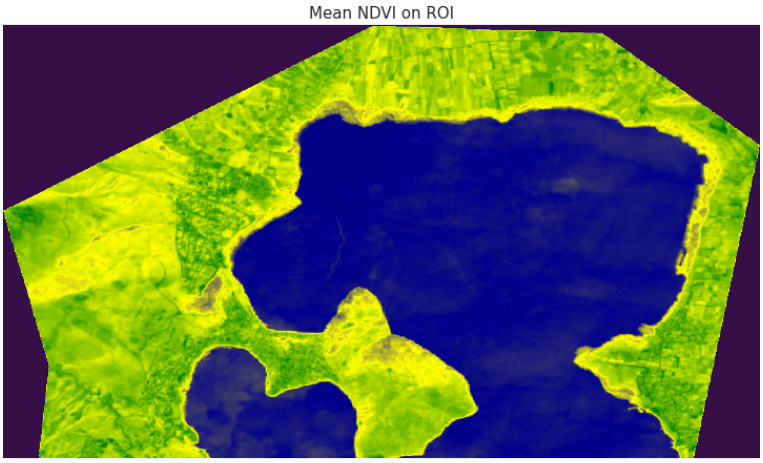

Figure 7: The mean *NDVI* value for the whole collection of images.

2.4 Time Series of the Spectral Indices

At this point we produce time series for the *NDVI, NDWI, NDBI* indices on the area of interest. In order to produce a time series chart we must use some kind of aggregation on our data or pick a specific point on the area and plot the series for this point. We proceed with the following two methods.

(i) We calculate the mean value on the whole area and plot a time series for every available date.

(ii) We pick the pixel with highest mean value with respect to each of the three indices and we produce a time series for this/these point(s).

(i) We first begin by evaluating the mean value of the whole ROI for the three aforementioned indices.

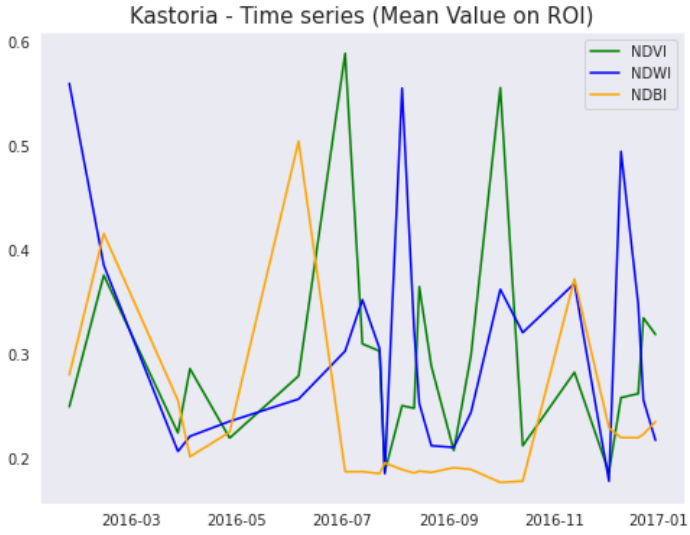

Figure 8

To handle the above calculation in *Python* we construct a utility function named *mean_value_on_roi* to carry out this computation. The function accepts the name of the index *(index)*, the *raster* data, (*raster_data*), the dictionary *(index_dict)* mapping the indices (e.g. *NDVI, NDWI*) to the number of the image in the collection of the images for the first date, and the total number of images *(num_images)*.

(ii) Now we find the pixels for the above three indices with the highest mean value and we plot a time series for these three pixels. In the Figure below we can see the exact location of these pixels for each index.

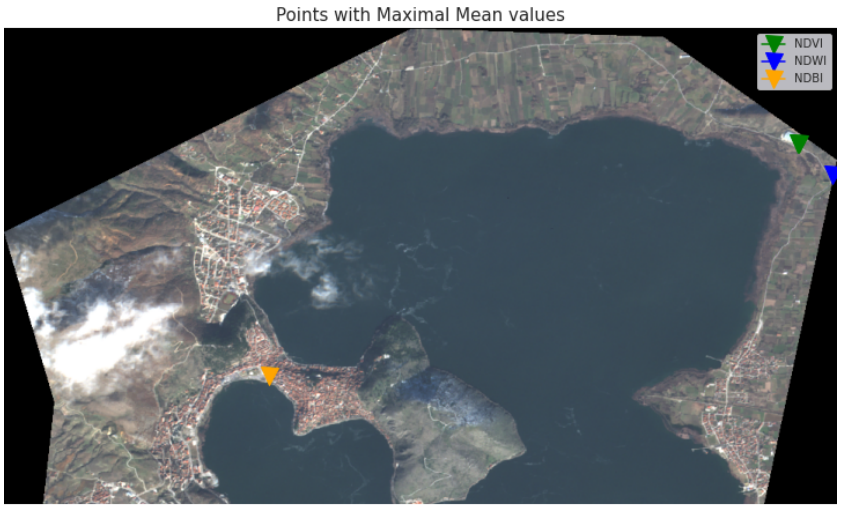

Figure 9: The points with maximal mean values for each index.

As we can see, the point with the highest *NDBI* value (yellow point) is located at the center of Kastoria town. The outcome is expected since the *Normalized Difference Built-up Index (NDBI)* uses the *NIR* and *SWIR* bands to emphasize manufactured built-up areas. It is ratio based to mitigate the effects of terrain illumination differences as well as atmospheric effects. Formally, the *NDBI* is given by

$$NDBI = \frac{SWIR - NIR}{SWIR + NIR}. \tag{2.2}$$

Negative values of *NDBI* represent water bodies whereas higher values represent build-up areas. *Normalized Difference Water Index (NDWI)* on the other hand, is used for the analysis of water bodies. The index uses *Green* and Near infra-red bands of remote sensing images. The *NDWI* can enhance water information efficiently in most cases. It is sensitive to build-up land and results in over-estimated water bodies. It is calculated by the formula

$$NDWI = \frac{NIR - SWIR}{NIR + SWIR}. \tag{2.3}$$

In the Figure below we can see the result for the second time series chart for the three indices.

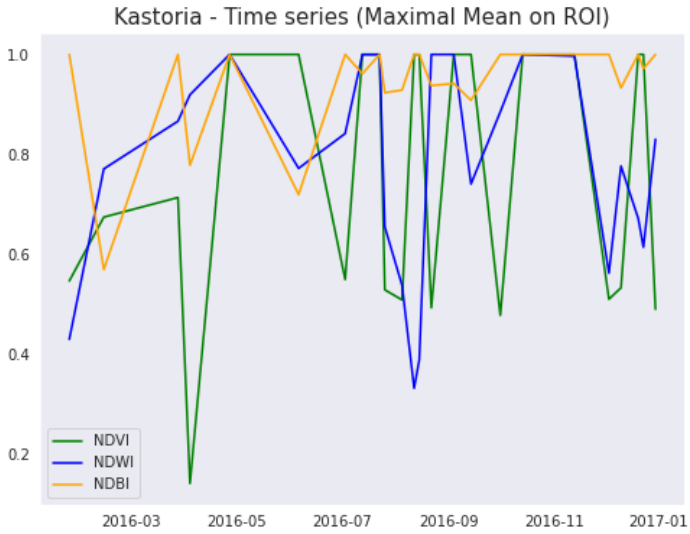

Figure 10

2.5 Meteorological Analysis - Kastoria

In this subsection we perform some basic analysis about the weather conditions in Kastoria. We gather our data from the meteorological portal of *NASA*. In detail we are interested in the following meteorological variables for a specific point at the center of Kastoria.

(i) *All Sky Insolation Clearness Index*: A fraction representing clearness of the atmosphere; the all sky insolation that is transmitted through the atmosphere to strike the surface of the earth divided by the average of top of the atmosphere total solar irradiance incident.

(ii) *All Sky Surface UVA Irradiance:* The ultraviolet A (UVA 315nm-400nm) irradiance under all sky conditions.

(iii) *All Sky Surface UVB Irradiance:* The ultraviolet B (UVB 280nm-315nm) irradiance under all sky conditions.

(iv) *Temperature at 2 Meters (in Celsius):* The average air (dry bulb) temperature at 2 meters above the surface of the earth

(v) *Relative Humidity at 2 Meters:* The ratio of actual partial pressure of water vapor to the partial pressure at saturation, expressed in percent.

(vi) *Wind Speed at 10 Meters (m/s):* The average wind speed at 10 meters above the surface of the earth.

The point that we pick for our analysis is located at the center of Kastoria and its *lat/lon* coordinates are (40.5213, 21.2618). By applying a simple linear transformation we calculate the pixel location of

that point on the clipped images. In the Figure below we can see the exact location of that point on the image.

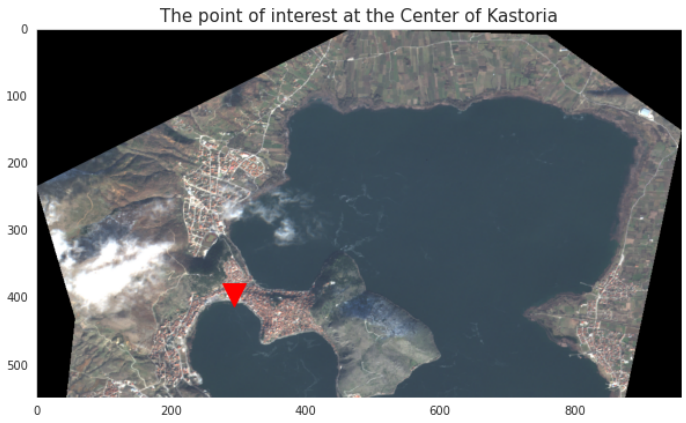

Figure 11: The point of meteorological interest at the Center of Kastoria.

The meteorological data that we collect expand for the whole year of 2021 for the point shown in Figure 11. In order to have a better interpretation, we compare the meteorological measurements in Kastoria with the same measurements for the point at the center of Athens shown in Figure 12.

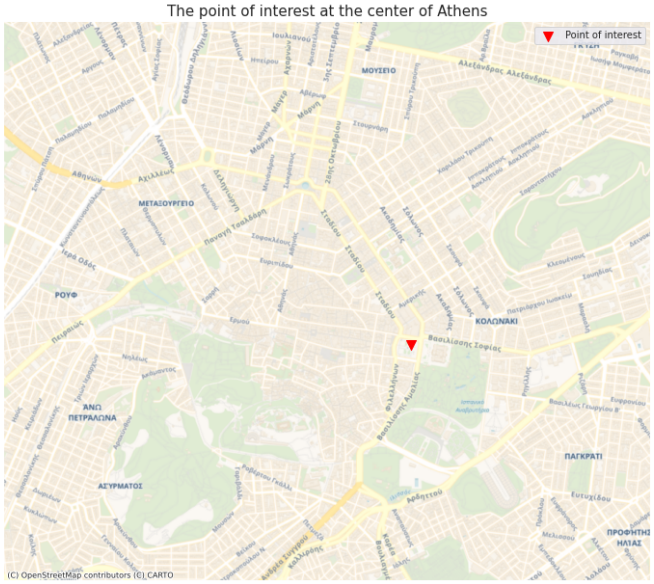

Figure 12: The point of meteorological interest at the Center of Athens.

Below we can see the time series with respect to the *All Sky Insolation Clearness Index* and the *All Sky Surface UVA Irradiance* variables.

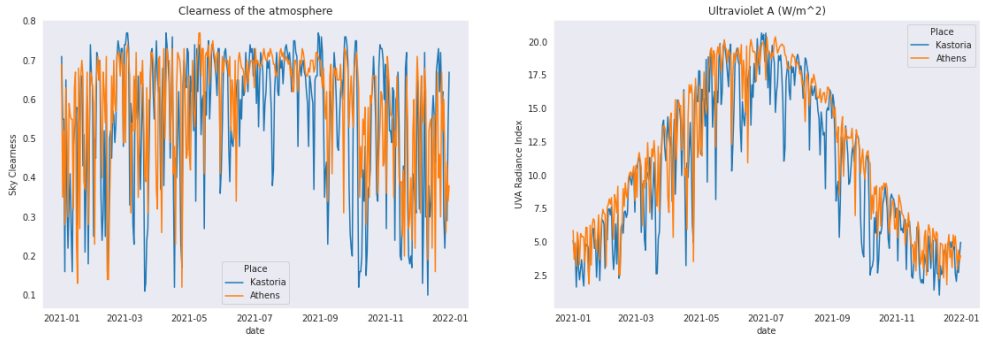

Figure 13: *Sky Clearness Index* and *UVA* irradiance in Kastoria.

As we can see, in both cases we observe similar behavior for the first two variables. Next, we compare the two cities with respect to the *UVB* irradiance and the temperature.

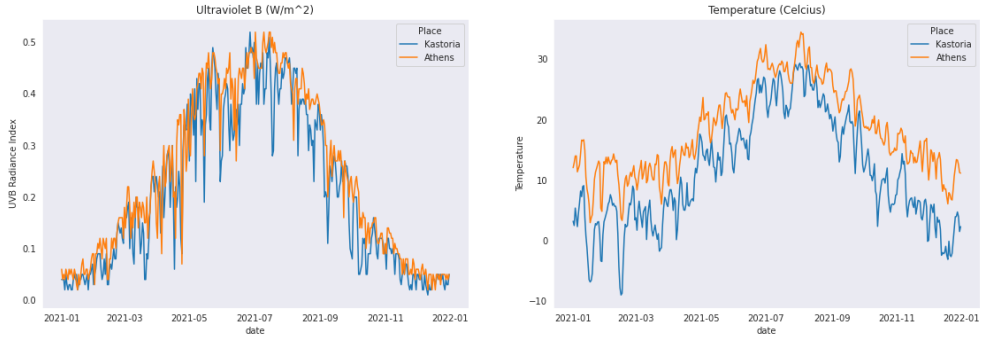

Figure 14: *UVB* irradiance and Temperature in Kastoria.

And finally we have the time series for the *Relative Humidity* and the *Wind speed*.

Figure 15: *Relative Humidity* and *Wind speed* in Kastoria.

Task 3 - Geospatial Queries and Web Services

In this section, as opposed to the previous one, we shift from the exclusive use of the *rasterio* library and work also with *Geopandas* and *OWSLib*. We make extensive use of the *WMS/WFS* services to download *raster* and *vector* data. *Geopandas* provides a clear cut way for performing geospatial queries by taking advantage only of the spatial location of the objects of interest.

3.1 Geospatial queries

For the queries we use the data provided by *https://geodata.gov.gr*. We pull *vector* data for the administrative region boundaries, airport locations, railroads, coastlines, lakes and national parks in Greece. In addition, we obtain a *raster RGB* image through the following WMS of Europe captured by Landsat which will serve the basemap for our figures.

3.1.1 Query 1: Administrative regions of Greece and Airports.

Combining the *WFS* of the administrative boundaries and the airport locations we can obtain the administrative region in which each airport belongs to. This query is performed in *Geopandas* through the *spatial join* method. Using as predicate the *intersection* operation, *Geopandas* joins the two *GeoDataFrames* by matching the rows that the two geometries have a non-empty intersection. In the table below we can see the first ten airports of Greece (including military airbases) and their corresponding administrative region they belong to.

Table 1: Airports and Administrative regions of Greece

Airport	Administrative Region
Δασκαλογιάννης CHQ Χανιά, Σούδα	Π. ΚΡΗΤΗΣ
Μάλεμε	Π. ΚΡΗΤΗΣ
Τυμπάκι LG54	Π. ΚΡΗΤΗΣ
Νίκος Καζαντζάκης LGIR HER Ηράκλειο	Π. ΚΡΗΤΗΣ
Καστέλι LGTL	Π. ΚΡΗΤΗΣ
Σητεία LGST JSH	Π. ΚΡΗΤΗΣ
Δωδεκανήσου-Κάσος- ΔΑΚΑ LGKS/KSJ	Π. ΝΟΤΙΟΥ ΑΙΓΑΙΟΥ
Δωδεκανήσου-Κάρπαθος-ΚΑΚΠ LGKP/AOK	Π. ΝΟΤΙΟΥ ΑΙΓΑΙΟΥ
Διαγόρας LGRP RHO Παραδείσι, Ρόδου	Π. ΝΟΤΙΟΥ ΑΙΓΑΙΟΥ
Μαρίτσα LGRD Ρόδος	Π. ΝΟΤΙΟΥ ΑΙΓΑΙΟΥ

Putting the *vector* data onto the *RGB* image of Europe and converting them into the same coordinate system we obtain an image with the exact location of each airport as well as the boundaries of the administrative regions.

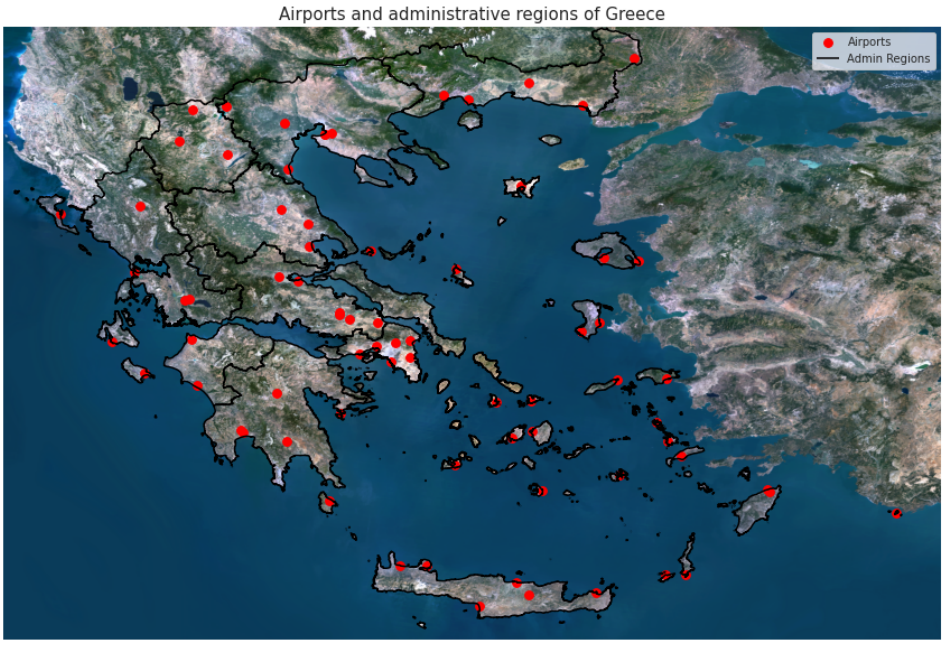

Figure 16: Airport and Administrative regions of Greece

3.1.2 Query 2: Settlements of Greece within 1km near some airport

In order to find the settlements we use the *buffer* method by *Geopandas*. First, we give a buffer of size 500m to every point of the two *Geopandas GeoDataFrames* corresponding to the airports and the settlements. Then, to find the settlements within 1km distance from some airport we perform the *spatial join* operation with the *intersection* predicate. In the table below we can see the first 10 settlements and the corresponding airports.

Table 2: Settlements within 1km distance near some Airport

Settlement	Municipality	Airport
Παραδείσιον	ΔΗΜΟΣ ΠΕΤΑΛΟΥΔΩΝ	Διαγόρας LGRP RHO Παραδείσι, Ρόδου
Βαγιές	ΔΗΜΟΣ ΠΕΤΑΛΟΥΔΩΝ	Διαγόρας LGRP RHO Παραδείσι, Ρόδου
Μάννα	ΔΗΜΟΣ ΕΡΜΟΥΠΟΛΕΩΣ	Σύρου LGSO JSY
Παρθένιον	ΔΗΜΟΣ ΛΕΡΟΥ	Λέρου LGLE LRS
Άργος	ΔΗΜΟΣ ΚΑΛΥΜΝΙΩΝ	Κάλυμνος LGKY JKL
Ανάληψις	ΔΗΜΟΣ ΑΣΤΥΠΑΛΑΙΑΣ	Αστυπάλαια LGPL JTV
Αερολιμήν	ΔΗΜΟΣ ΠΕΤΑΛΟΥΔΩΝ	Μαρίτσα LGRD Ρόδος
Αγία Παρασκευή	ΔΗΜΟΣ ΜΥΤΙΛΗΝΗΣ	Οδυσσέας Ελύτης LGMT MJT Μυτιλήνη
Έξω Φάρος	ΔΗΜΟΣ ΑΓΙΟΥ ΚΗΡΥΚΟΥ	Ικαρία LGIK JIK
Αγία Κυριακή	ΔΗΜΟΣ ΑΓΙΟΥ ΚΗΡΥΚΟΥ	Ικαρία LGIK JIK

3.1.3 Query 3: Transportation in Attica

In this query we find the number of transportation stations contained in the balls of radius 1.5km having as centers the settlements located in Attica. We combine the *GeoDataFrame* containing the locations of stations provided by *OASA* with the *GeoDataFrames* consisting of the administrative regions and settlements of Greece. We first keep only the settlements that are located in the administrative region of Attica. To perform this reduction we apply the *spatial join* operation using the *GeoDataFrames* containing the settlements and the administrative boundaries of Attica.

For visualization purposes, we keep only the settlements of Attica that are not "far away" from some transportation station. In detail, we calculate the *convex hull* of the points representing the transportation stations and by applying the *spatial join* operation we keep only the settlements contained in the *convex hull*. In the table below we can see the settlements with the most transportation stations within 1.5km distance from a reference point.

Table 3: Settlements with the most Transportation stations within 1.5 km radius.

No.	Settlement	Number of Stations
1.	Αθήνα	274
2.	Πειραιάς	221
3.	Κορυδαλλός	217
4.	Νίκαια	195
5.	Δάφνη	195
6.	Καισαριανή	193
7.	Περιστέρη	190
8.	Υμηττός	189
9.	Άγιος Δημήτριος	183
10.	Νέα Χαλκηδόνα	183
11.	Βύρωνας	180
12.	Νέα Σμύρνη	174
13.	Νέα Φιλαδέλφεια	169
14.	Ίλιον	165
15.	Άγιοι Ανάργυροι	163
16.	Κερατσίνι	159
17.	Νέα Ιωνία	159
18.	Καλλιθέα	153
19.	Ζωγράφος	150
20.	Αγία Βαρβάρα	149
21.	Γαλάτσι	146
22.	Άγιος Ιωάννης Ρέντης	144
23.	Πετρούπολη	135
24.	Χαλάνδρι	135
25.	Αιγάλεω	134

In the following figure we can see the transportation stations and the settlements that we used for the query.

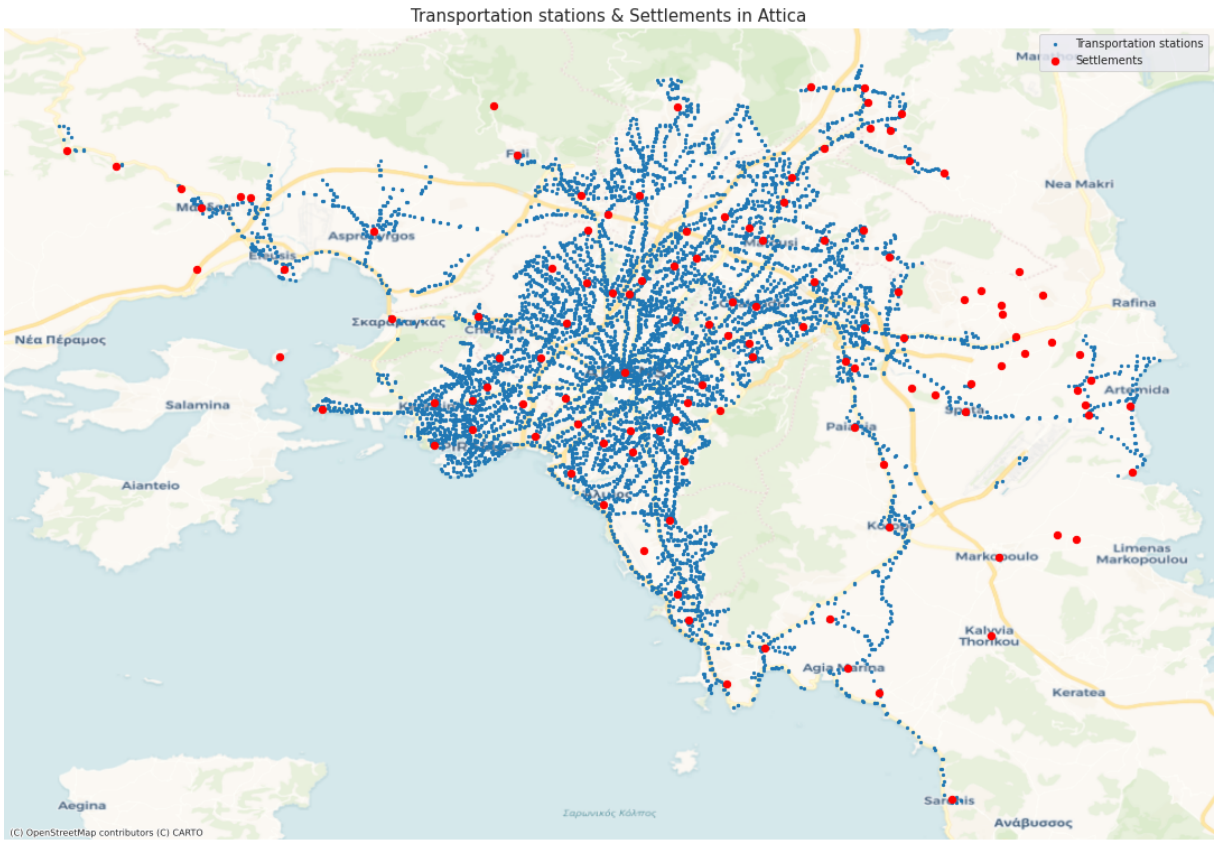

Figure 17: Transportation stations and settlements in Attica.

3.1.4 Query 4: Administrative regions and coastlines of Greece

In this query we calculate the total length of coastline contained in each administrative region. The geometry of the *Geopandas GeoDataFrame* containing the coastline of Greece consists of *LineString* and *MultiLineString* objects. To execute this query we proceed again with *spatial join*. We first create a new *GeoDataFrame* with the administrative regions and their corresponding coastlines, then we calculate the length of each line object and finally we aggregate the results with respect to each administrative region. In the following table we can see the results.

Table 4: Coastline length per Administrative region.

No.	Administrative Region	Length (km)
1.	Π. ΝΟΤΙΟΥ ΑΙΓΑΙΟΥ	2001.12
2.	Π. ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ	816.38
3.	Π. ΒΟΡΕΙΟΥ ΑΙΓΑΙΟΥ	770.83
4.	Π. ΠΕΛΟΠΟΝΝΗΣΟΥ	678.35
5.	Π. ΚΡΗΤΗΣ	644.66
6.	Π. ΙΟΝΙΩΝ ΝΗΣΩΝ	631.82
7.	Π. ΑΤΤΙΚΗΣ	588.15
8.	Π. ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ	504.03
9.	Π. ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ	489.90
10.	Π. ΘΕΣΣΑΛΙΑΣ	417.39
11.	Π. ΗΠΕΙΡΟΥ	253.77
12.	Π. ΑΝΑΤΟΛΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ - ΘΡΑΚΗΣ	238.94

3.1.5 Query 5: Administrative regions of Greece and railways

Using the same operation procedure as in the previous query but with the railway lines this time we obtain the total railway length line of each administrative region.

Table 5: Length of railway network per Administrative region.

No.	Administrative Region	Length (km)
1.	Π. ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ	552.69
2.	Π. ΠΕΛΟΠΟΝΝΗΣΟΥ	383.34
3.	Π. ΑΝΑΤΟΛΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ - ΘΡΑΚΗΣ	366.67
4.	Π. ΘΕΣΣΑΛΙΑΣ	303.61
5.	Π. ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ	271.29
6.	Π. ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ	242.12
7.	Π. ΑΤΤΙΚΗΣ	216.40
8.	Π. ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ	121.29

Combining the *vector* data containing information about the railway lines and the administrative regions with the *raster RGB* image restricted onto the bounding box of Greece we obtain a visualization of the railway network in Greece.

Figure 18: The railway network of Greece.

3.1.6 Query 6: European countries and Rivers

At this point we count the number of the European countries that each river flows through. We use the *WFS* service to pull the *vector* data about the *European Rivers* and the boundaries for the *European countries*. In the following table we see the first ten rivers that flow through most European countries.

Table 6: Number of countries per river

River	Countries	Number of Countries
Sûre	Luxembourg, Belgium, Germany	3
Raab	Hungary, Slovakia, Austria	3
Latorytsya	Hungary, Ukraine, Slovakia	3
Someşul Cald	Hungary, Romania	2
Merkys	Belarus, Lithuania	2
Ljubljanica	Croatia, Slovenia	2
Slana	Hungary, Slovakia	2
Crisul Alb	Hungary, Romania	2
Mesta	Greece, Bulgaria	2
Drau	Croatia, Hungary	2

To execute the above query we first apply a *spatial join* to the *GeoDataFrames* corresponding to the European rivers and European countries and then for every river we count the number of rows that appear in the merged DataFrame. In a similar manner, we count the number of rivers per country. Below we see the top 15 European countries with the most rivers.

Table 7: Number of rivers per country

No.	Country	Number of Rivers
1.	France	27
2.	Germany	25
3.	Spain	24
4.	Belarus	21
5.	Sweden	18
6.	Romania	18
7.	Norway	17
8.	Ukraine	17
9.	Poland	17
10.	Finland	16
11.	Hungary	12
12.	Italy	12
13.	Bulgaria	9
14.	Slovakia	9
15.	Austria	8

3.2 Geospatial Analysis on Belgium

In this subsection we perform a short geospatial analysis on Belgium and in the capital of Belgium, Brussels. We create static maps for the various land uses in Belgium for the year 2018 and we perform the same meteorological analysis as in the case of Kastoria in subsection 2.5 with the point of interest being at the center of Brussels this time.

3.2.1 Corine Land Cover - 2018

We start by creating a static map of the Corine Land Cover for the year 2018. The sources that were utilized to create the map is the *WFS* containing the *vector* data for the various land uses and the *WMS* of Europe's *RGB* image by Landsat. From the following *link* we extracted the information about the color codes of each area. In order to get the color codes we used the *pandas.read_html* method and through a request we pulled the second table from the previous link. After creating a mapping of the code areas to the color codes we mapped the colors onto our *vector* data containing the boundaries of each land use. Below you can see the result.

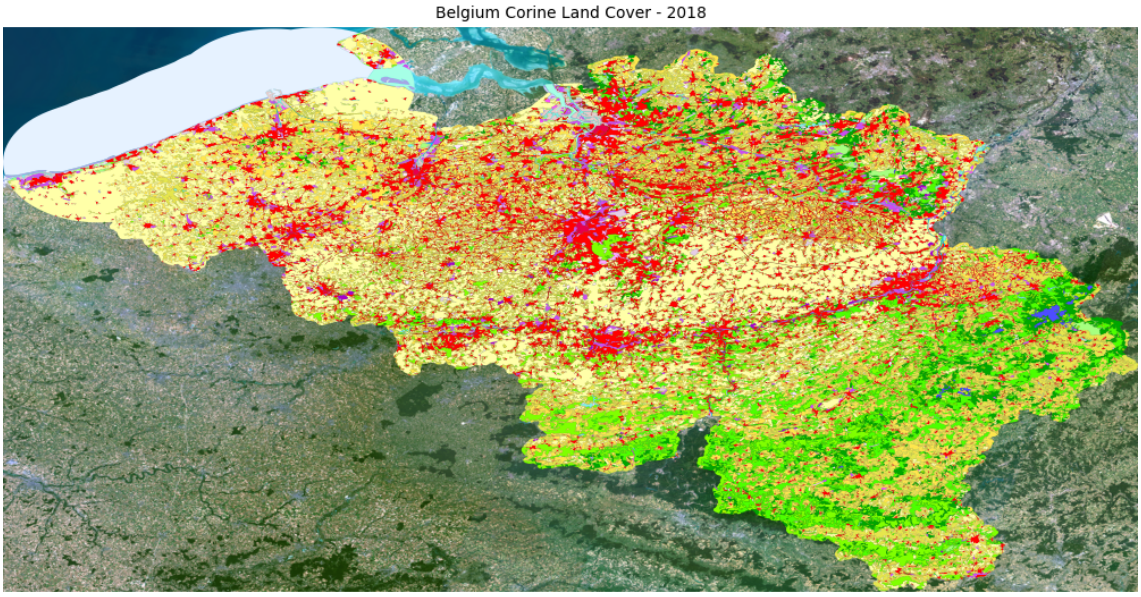

Figure 19: Corine Land Cover 2018 - Belgium.

3.2.2 Meteorological Analysis on Brussels

Now we focus our attention on the capital of Belgium, Brussels. As in the case with Kastoria we choose a random point at the center of Brussels and we collect meteorological measurements for this point from the portal of NASA. Below we can see the coordinates in the *EPSG: 4326* system of the point at the center of Brussels.

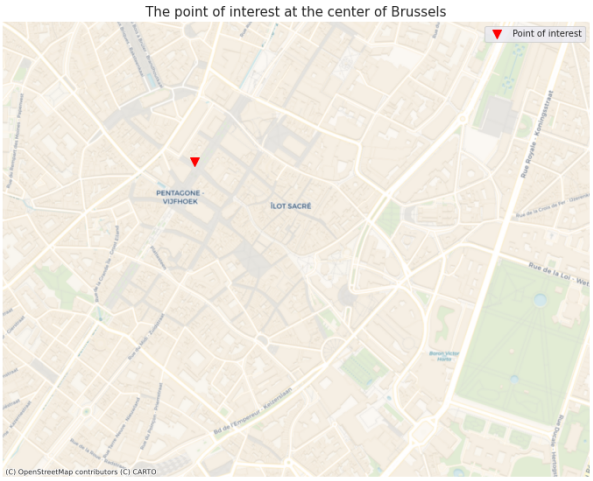

Figure 20: The point of meteorological interest at the center of Brussels.

We pull the measurements for the above point for the variables described in subsection 2.5 and we compare the corresponding measurements for the point at the center of Athens shown in Figure 12. Below we can see the comparison for the *Sky Clearness Index* and the *UVA irradiance*.

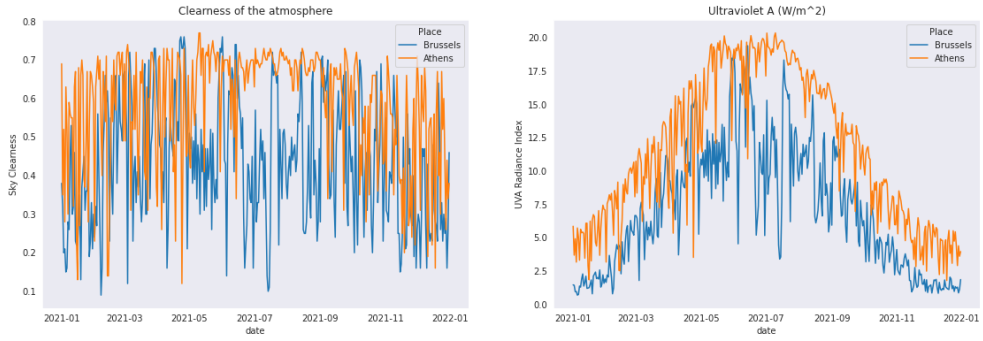

Figure 21: *Sky Clearness Index* and *UVA* irradiance in Brussels.

As we can see, in both cases Athens has higher values than Brussels, especially in the case of the *UVA* radiation. Below we see the comparison of the two cities with respect to the *UVB* irradiance and the temperature.

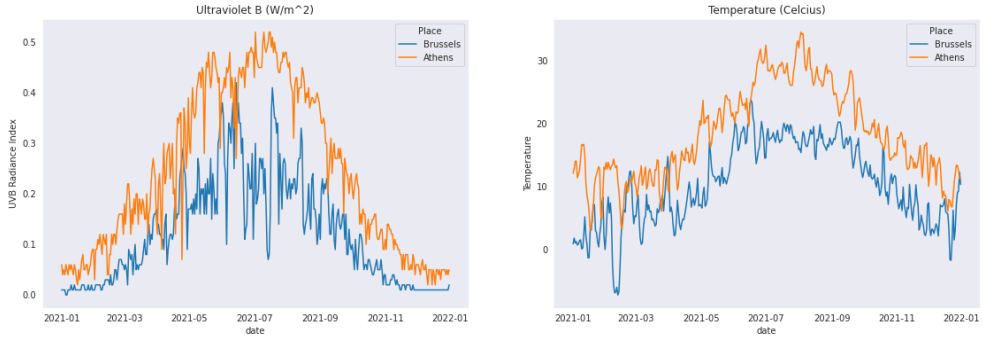

Figure 22: *UVB* irradiance and Temperature in Brussels.

An interesting observation from the right figure corresponding to the temperature is that in August the difference between the two cities exceeds the value of 15 degrees Celsius in some cases. Also, we see that in Brussels the temperature is lower than 20 degrees almost the whole year. Finally, in the following figure we see the measurements for the *Relative Humidity* and *Wind speed.*

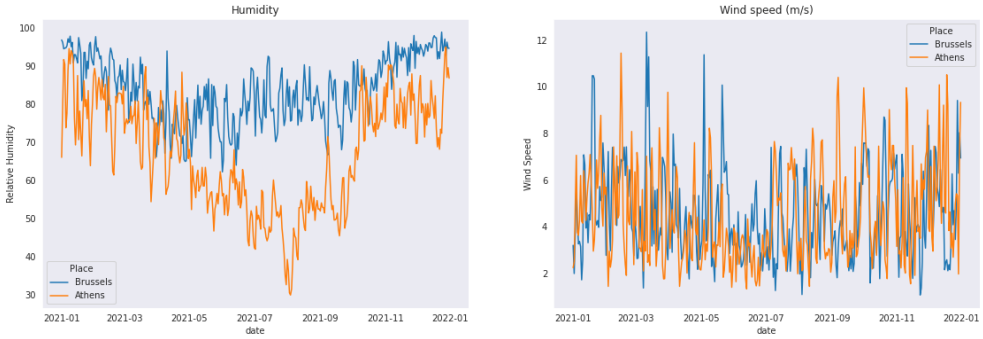

Figure 23: *Relative Humidity* and *Wind speed* in Brussels.

Task 4 - Interactive Maps

In this section using *Folium* and *Leafmap* libraries we construct interactive maps. We combine the *raster* and *vector* data that we gathered in the previous sections and add them onto the base maps provided by the two libraries. These two libraries provide an elegant way of creating interactive maps by taking advantage of the simplicity of *Python* language.

4.2.1 An interactive map for Greece

In the first interactive map we use all of the data that we gathered from *https://geodata.gov.gr/*. We use the *vector* data consisting of the administrative region boundaries, airport locations, railroads, coastlines, lakes and national parks in Greece.

We generate this map using *folium*. *Folium* provides an easy way of adding *raster* images as layers on the background of the map. By its method *raster_layers.WmsTileLayer* by passing the url of the *WMS* and picking the layer of interest we can easily add the layer on our map. Utilizing this method, we use the *RGB* image of Europe captured by Landsat as a background layer to our map. Furthermore, in order for our map to be more informative we add markers for every location. The small problem we have to face here is the fact that each national park may consist of several polygons, hence to avoid multiple markers on the same national park we first calculate the centroid of these polygons and choose the centroid of these centroids to be our marker. This operation is executed by a custom function named *centroid_of_centroids*. The *html* file of this map can be downloaded from the following *link*.

4.2.2 Corine Land Cover Interactive map

In the second interactive map we use the *WMS* to obtain *raster* data from *https://www.geoseer.net/* for the Corine land cover. We display the various land uses in Italy, Switzerland, Croatia and Slovenia for the year 2018 (the latest free available data of the Corine Land cover). A problem that had to be tackled was the addition of the legend showing the correspondence between the colors and the descriptions of the different land uses. To solve this problem, we used the table that we downloaded for the same purposes in subsection 3.2.1. In the following *link* you can access this map.

References

[1] Vitor CF Gomes, Gilberto R Queiroz, and Karine R Ferreira. “An overview of platforms for big earth observation data management and analysis”. In: *Remote Sensing* 12.8 (2020), p. 1253.

[2] Joel Lawhead. *Learning Geospatial Analysis with Python: Understand GIS fundamentals and perform remote sensing data analysis using Python 3.7*. Packt Publishing Ltd, 2019.