Obtinerea si prelucrarea datelor geospatiale

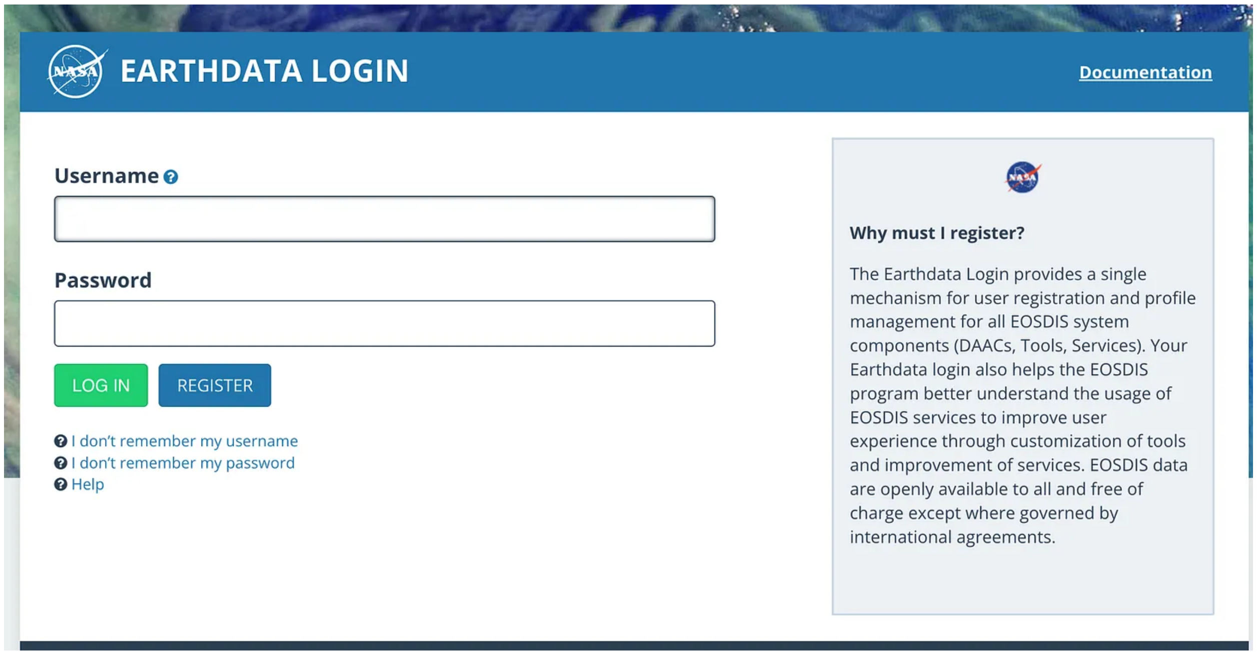
Stefan Nicov

2024-06-27

We will use the Black Marble data from NASA (link).

The advantage is that the data is regularly updated and the luminosity values could be adjusted for snow and satellite angle.

To obtain the data, visit the https://blackmarble.gsfc.nasa.gov website and register a free account by filling the requested information.



Why must I register?

The Earthdata Login provides a single mechanism for user registration and profile management for all EOSDIS system components (DAACs, Tools, Services). Your Earthdata login also helps the EOSDIS program better understand the usage of EOSDIS services to improve user experience through customization of tools and improvement of services. EOSDIS data are openly available to all and free of charge except where governed by international agreements.

Figure 1: registration

After that, visit the NASA's LAADS DAAC website by this link https://ladsweb.modaps.eosdis.nasa.gov/ and click the 'login' drop down on the top right.

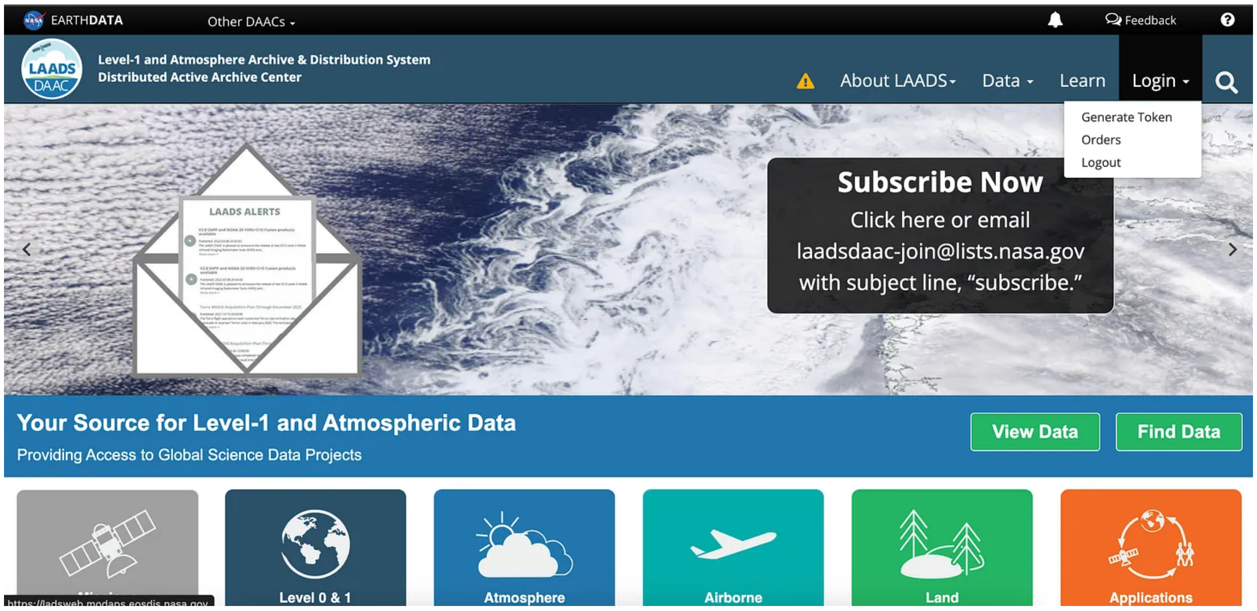

EARTHDATA
Other DAACs
Feedback
Level-1 and Atmosphere Archive & Distribution System
Distributed Active Archive Center
About LAADS
Data
Learn
Login
Generate Token
Orders
Logout
Subscribe Now
Click here or email laadsdaac-join@lists.nasa.gov with subject line, "subscribe."
Your Source for Level-1 and Atmospheric Data
Providing Access to Global Science Data Projects
View Data
Find Data
Level 0 & 1
Atmosphere
Airborne
Land
Applications

Figure 2: drop down menu

Then, in the same drop down, click 'Generate Token' and copy the token text. This text is called your 'bearer'. We'll be using it to connect to Black Marble's database via API.

Before accessing, downloading and processing the data, we need to make sure that we have the necessary libraries. First of all, install the `sf`, `terra`, `tmap`, `raster`, `exactextractr`, `lubridate`, `geodata`, `blackmarbler`, `ggplot2`, `dplyr`, `tidytera`, `magick`, `ggforce` packages. After installimg, make sure to load the necessary libraries.

Before downloading, I recommend to put the `options(timeout = 600)` command in the console. Since the packages are quite big and downloading them requires a significant amount of time, you may easily heat the timeout limit.

```r
# Load necessary libraries

library(sf)
library(terra)
library(tmap)
library(blackmarbler)
library(raster)
library(exactextractr)
library(lubridate)
library(geodata)
library(ggplot2)
library(dplyr)
library(lubridate)
library(tidyterra)
library(magick)
library(ggforce)
```

Then, paste the copied token into the bearer variable.

```r
# Set your Bearer token

bearer <- "insert your bearer token"
```

Then we need to define our Region of Interest.

```r
# Obtaining region of interest (ROI) as a simple feature (sf) object
# for Moldova at the first administrative level

sf_md <- gadm(country = "MDA", level = 1, path = tempdir()) %>%
  st_as_sf()
```

`gadm`: This function is used to download administrative boundary data from the GADM (Global Administrative Areas) database.

`country = "MDA"`: Specifies the country code for Moldova. The GADM database uses ISO 3166-1 alpha-3 country codes.

`level = 1`: Specifies the administrative level. 0 for the country level, 1 for the first administrative level (e.g., states or provinces), 2 for the second administrative level (e.g., counties or districts).

`path = tempdir()`: Specifies the directory path where the downloaded shapefiles will be stored. `tempdir()` creates a temporary directory for this purpose.

`%>%`: the pipe operator from the `dplyr` package, used to chain commands together, passing the output of one function directly into the next function.

`st_as_sf()`: This function is used to convert the object (in this case, the downloaded shapefile) into an sf (simple features) object, which is a standardized way to represent spatial vector data in R.

Now we will download our Black Marble Data for our region of interest and in the specified date range. Be aware that this might take quite a long time.

Here we download the data as a `SpatRaster` object which is used to then plot the values on the map of the region of interest.

```r
# Downloading Black Marble data for the ROI over a specified date range
md_spat_raster_monthly <- bm_raster(roi_sf = sf_md, #the region of interest
                              product_id = "VNP46A3", #monthly luminosity data
                              date = seq.Date(from = ymd("2021-01-01"),
                              to = ymd("2023-12-01"), by = "month"),
                              bearer = bearer, # token
                              variable = c("AllAngle_Composite_Snow_Free"),
                              quality_flag_rm = c(255, 2),
                              quiet = TRUE,
                              h5_dir = "data/monthly")
```

The attributes of the `bm_raster()` function are:

`roi_sf`: region of interest, must be a sf polygon.

`product_id`: One of the following: "VNP46A1" for Daily (raw), "VNP46A2" for Daily (corrected), "VNP46A3" for Monthly, and "VNP46A4" for Annual data

`date`: Date of raster data. Entering one date will produce a SpatRaster object. Entering multiple dates will produce a SpatRaster object with multiple bands; one band per date. For "VNP46A1" and "VNP46A2" must be a date (eg. "2021-10-03"), for "VNP46A3" can be a date (the day will be ignored) or year-month, and for "VNP46A4" can be a date (the day and month will be ignored) or year.

`bearer`: NASA bearer token

`variable`: Variable to used to create raster (default: `NULL`). If `NULL`, uses the default variables. For a full list of possibl values of variables, click here

`quality_flag_rm`: Quality flag values to use to set values to `NA`. Each pixel has a quality flag value, where low quality values can be removed. Values are set to `NA` for each value in the `quality_flag_rm vector`. (Default: `NULL`).

`interpol_na`: When data for more than one date is downloaded, whether to interpolate NA values using the terra::approximate function. Additional arguments for the terra::approximate function can also be passed into bm_raster (eg, method, rule, f, ties, z, NA_rule). (`Default: FALSE`).

`quiet`: Supress the console messages about the downloaded data, error messages, etc. (`Default: FALSE`).

`h5_dir`: Black Marble data are originally downloaded as h5 files. If `h5_dir = NULL`, the function downloads to a temporary directory then deletes the directory. If `h5_dir` is set to a path, h5 files are saved to that directory and not deleted. The function will then check if the needed h5 file already exists in the directory; if it exists, the function will not re-download the h5 file.

Here we download the data as `Raster` object which is used to plot values over time for different regions.

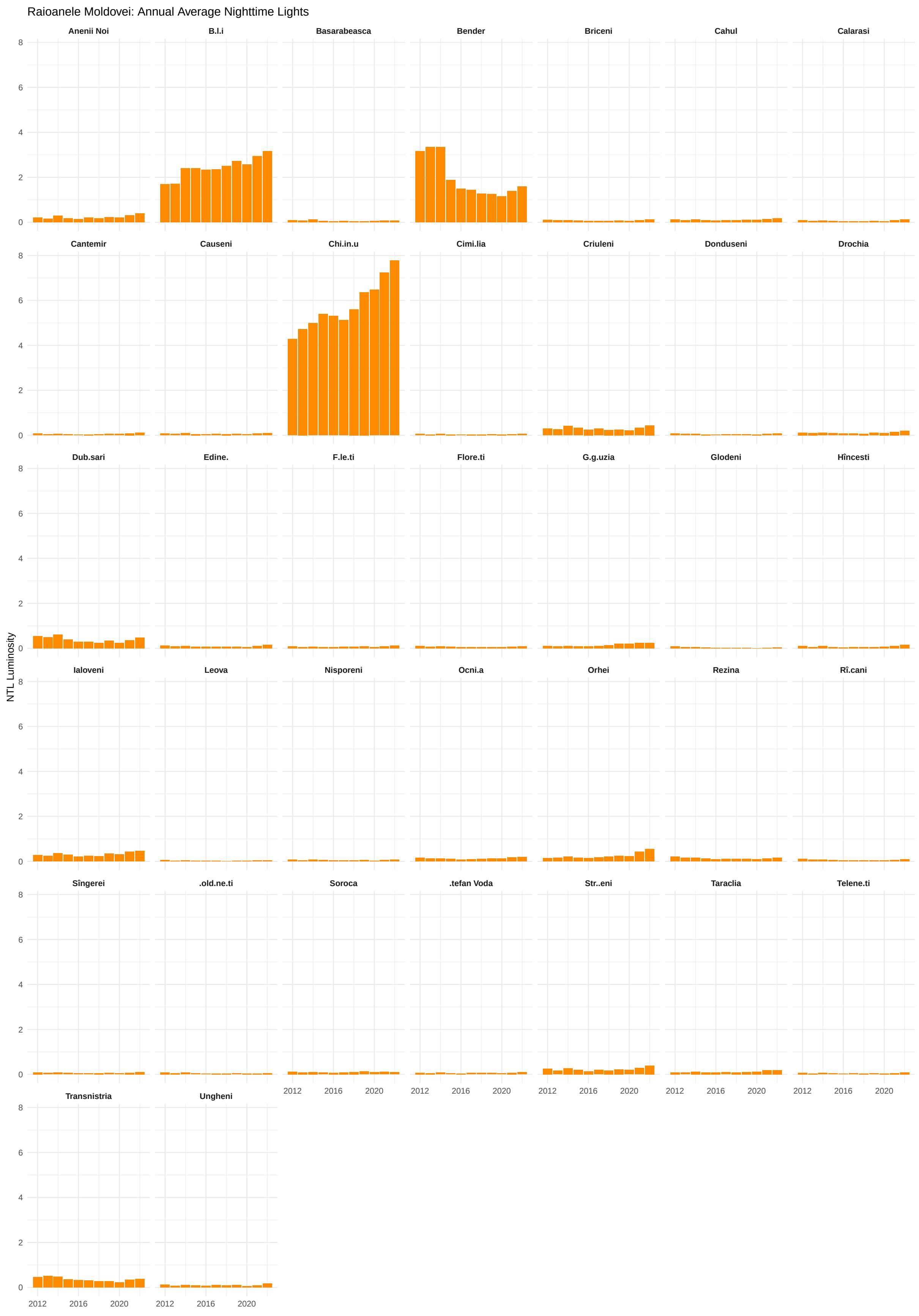
```r
#### Extract annual data

md_raster_yearly <- bm_extract(roi_sf = sf_md,
                     product_id = "VNP46A4",
                     date = 2012:2022,
                     bearer = bearer,
                     variable = c("AllAngle_Composite_Snow_Free"),
                     quality_flag_rm = c(255, 2),
                     quiet = TRUE,
                     h5_dir = "data/yearly")
```

The attributes of the `bm_extract()` function or similar to the `bm_raster()` function. For more details, check the official package reference manual here.

Now we will plot panels for every region of the country of the Night Time Luminosity (NTL) over time. In this panel, all the regions' luminosity will be graphed based on the maximum value of NTL of the most luminous region.

```r
#### Trends over time
md_raster_yearly |>
  ggplot() +
  geom_col(aes(x = date,
  y = ntl_mean),
  fill = "darkorange") + #adjust graph colour
  facet_wrap(~NAME_1) +
  labs(x = NULL,
       y = "NTL Luminosity",
       title = "Raioanele Moldovei: Annual Average Nighttime Lights") +
  scale_x_continuous(labels = seq(2012, 2022, 4),
                     breaks = seq(2012, 2022, 4)) +
  theme_minimal() +
  theme(strip.text = element_text(face = "bold"))
```



Raioanele Moldovei: Annual Average Nighttime Lights
NTL_Luminosity
Anenii Noi
B.l.i
Basarabeasca
Bender
Briceni
Cahul
Calarasi
Cantemir
Causeni
Chi.in.u
Cimi.lia
Criuleni
Donduseni
Drochia
Dub.sari
Edine.
F.le.ti
Flore.ti
G.g.uzia
Glodeni
Hîncesti
Ialoveni
Leova
Nisporeni
Ocni.a
Orhei
Rezina
Rî.cani
Sîngerei
.old.ne.ti
Soroca
.tefan Voda
Str..eni
Taraclia
Telene.ti
Transnistria
Ungheni
0
2
4
6
8
2012
2016
2020

```r
  #save the plot to .png file
  ggsave(filename = paste0("NTL by region v1.png"), width = 14, height = 21, path = "output")
```

In this graph, every region's plot will have a `ylim` based on this region's maximum luminosity.

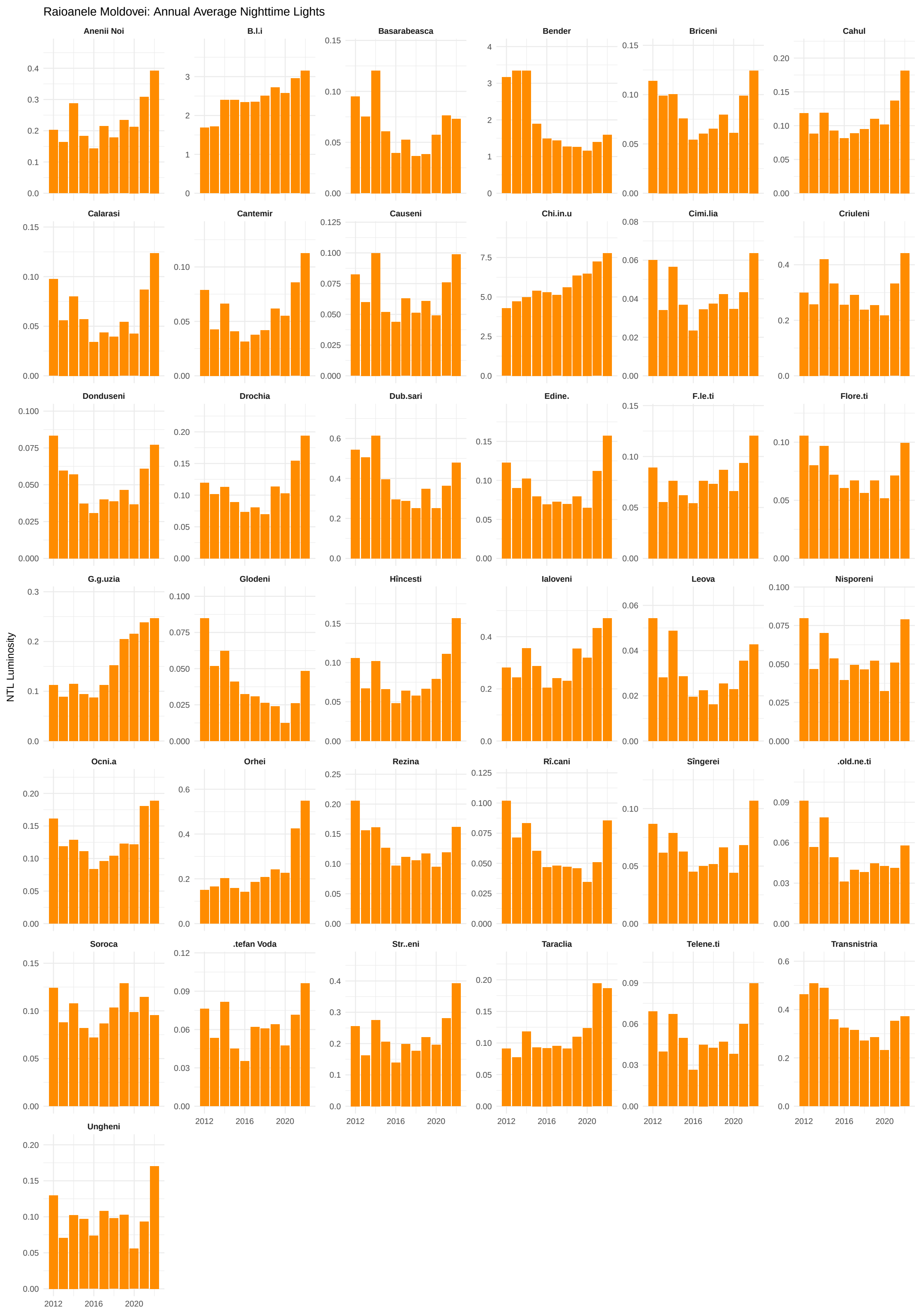
```r
# Calculate the custom ylim for each NAME_1 category
custom_ylim <- md_raster_yearly %>%
  group_by(NAME_1) %>%
  summarize(max_ntl_mean = max(ntl_mean, na.rm = TRUE)) %>%
  mutate(ylim = 1.2 * max_ntl_mean)

# Merge the custom ylim back to the original data frame
md_raster_yearly <- md_raster_yearly %>%
  left_join(custom_ylim, by = "NAME_1")

# Create the plot
p <- ggplot(md_raster_yearly, aes(x = date, y = ntl_mean)) +
  geom_col(fill = "darkorange") +
  facet_wrap_paginate(~NAME_1, scales = "free_y") +
  labs(x = NULL, y = "NTL Luminosity", title = "Raioanele Moldovei: Annual Average Nighttime Lights") +
  scale_x_continuous(labels = seq(2012, 2022, 4), breaks = seq(2012, 2022, 4)) +
  theme_minimal() +
  theme(strip.text = element_text(face = "bold"))

# Customize ylim for each facet
p <- p + ggforce::facet_wrap_paginate(
  ~NAME_1,
  ncol = 6,
  nrow = 7,
  scales = "free_y",
  strip.position = "top"
) +
  geom_blank(data = md_raster_yearly %>%
               mutate(ntl_mean = ylim))

# Print the plot
print(p)
```


Raioanele Moldovei: Annual Average Nighttime Lights
NTL Luminosity
Anenii Noi
B.l.i
Basarabeasca
Bender
Briceni
Cahul
Calarasi
Cantemir
Causeni
Chi.in.u
Cimi.lia
Criuleni
Donduseni
Drochia
Dub.sari
Edine.
F.le.ti
Flore.ti
G.g.uzia
Glodeni
Hîncesti
Ialoveni
Leova
Nisporeni
Ocni.a
Orhei
Rezina
Rî.cani
Sîngerei
.old.ne.ti
Soroca
.tefan Voda
Str..eni
Taraclia
Telene.ti
Transnistria
Ungheni
2012
2016
2020

```r
#save the plot to .png file
ggsave(filename = paste0("NTL by region v2.png"), plot = p, width = 14, height = 21, path = "output")
```

Since our `SpatRaster` object has different layers (corresponding to different months) we will first check the names of the layers and then plot only the layer we are interested in.

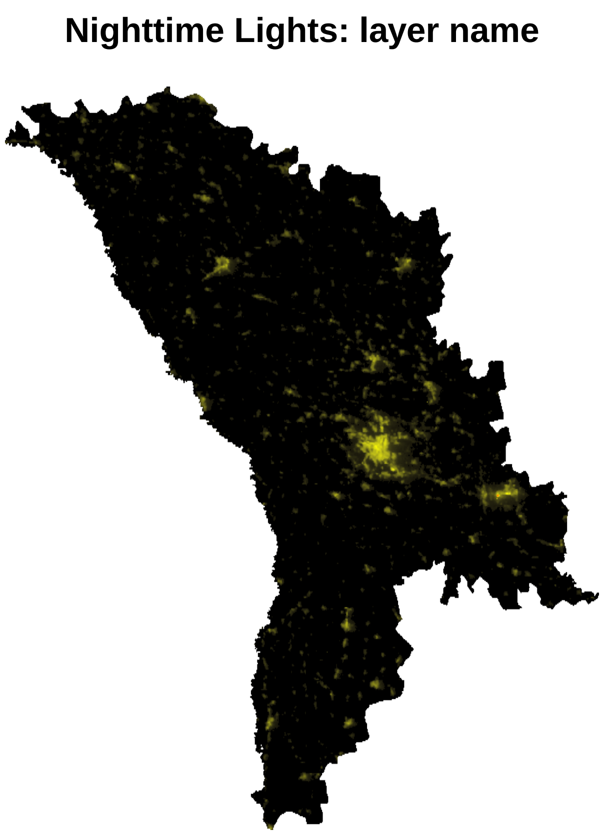
```r
r <- md_spat_raster_monthly
r <- r |> terra::mask(vect(sf_md))
layer_names <- names(r)

#r <- r[["layer name"]]
```

Now we will create a map of the ROI with the luminosity values represented by pixels of different colors. We will plot the log of these values, since the distribution is skewed.

```r
## Distribution is skewed, so log
r[] <- log(r[] + 1)

##### Map
ggplot() +
  geom_spatraster(data = r) +
  scale_fill_gradient2(low = "black",
                       mid = "yellow",
                       high = "red",
                       midpoint = 4.5,
                       na.value = "transparent") +
  labs(title = "Nighttime Lights: layer name") +
  coord_sf() +
  theme_void() +
  theme(plot.title = element_text(face = "bold", hjust = 0.5),
  legend.position = "none")
```



Here we will create such an image for all the existing layers (dates)

```r
# Define your SpatRaster object
r <- md_spat_raster_monthly
r <- r |> terra::mask(vect(sf_md))

# Log transform the data
r[] <- log(r[] + 1)

# Define the time period
start_date <- as.Date("2021-01-01")
end_date <- as.Date("2023-12-31")
dates <- seq.Date(start_date, end_date, by = "month")

# Iterate through each month
for (i in layer_names) {


  # Plot the raster data for the current layer
  p <- ggplot() +
    geom_spatraster(data = r[[i]]) +
    scale_fill_gradient2(low = "black",
                         mid = "yellow",
                         high = "red",
                         midpoint = 4.5,
                         na.value = "transparent") +
    labs(title = paste("Nighttime Lights:", i)) +
    coord_sf() +
    theme_void() +
    theme(plot.title = element_text(face = "bold", hjust = 0.5),
          legend.position = "none")

  # Save the plot as a PNG file
  ggsave(filename = paste0("Nighttime_Lights_", i, ".png"), plot = p, width = 8, height = 6, path = "ou
}
```

Here we combine all the created images into a .gif file to visualize the change in NTL.

```
# Set the directory containing the PNG files
png_dir <- "output"

# Get the list of PNG files
png_files <- list.files(png_dir, pattern = "*.png", full.names = TRUE)

# Ensure the files are sorted chronologically
png_files <- sort(png_files)

# Read the images into a magick image object
images <- image_read(png_files)

# Create the GIF
gif <- image_animate(images, fps = 2) # Adjust fps as needed

# Save the GIF
image_write(gif, path = "output/Nighttime_Lights_2021-2023.gif")
```