Solar Radiation in New York City

Salvatore Porcheddu

23 3 2021

Contents

Introduction

In this project we will explore solar radiation data coming from New York City.

We will be using data extracted from the Data.gov API, focusing on the following variables:

- **Average Direct Normal Irradiance**: the amount of solar radiation received by a surface per unit area;
- **Average Tilt at Latitude**: the amount of radiation received by a surface per unit area that does not arrive on a direct path from the sun;
- **Average Global Horizontal Irradiance**: the total amount of shortwave radiation that a surface horizontal to the ground receives from above.

Goal of this project is to build a dataframe with the solar radiation data and analyze it.

Data Extraction

In order to query the Data.gov API we need an API key, obtained by creating an account. We also need latitude and longitude of New York City.

Here's a list containing all of the parameters that we will use (please note that the API key will be stored into a variable and will never explicitly appear for privacy reasons):

```r
parameters_list <- list(api_key = api_key, lat = 41, lon = -75)

# The Solar Resource Data can be queried by means of the following URL:
URL <- "https://developer.nrel.gov/api/solar/solar_resource/v1.json"
```

Let's query the API and get the data in JSON format (please note that the necessary packages, **httr** and **jsonlite** have already been loaded):

```r
response <- GET(URL, query = parameters_list)

# Let's see if our request was successful
status <- status_code(response)

# Content extraction
content <- content(response, "text")

status
```

```
## [1] 200
```

```r
content
```

```
## [1] "{\"version\":\"1.0.0\",\"warnings\":[],\"errors\":[],\"metadata\":{\"sources\":[\"Perez-SUNY/NRE
```

The status code **200** informs us that everything went correctly.

As we can see, the data that we extracted is not easily suitable for manipulation and analysis, thus we will need to convert it into a list thanks to the **jsonlite** package:

```r
json_lists <- fromJSON(content)

str(json_lists)
```

```
## List of 6
##  $ version : chr "1.0.0"
##  $ warnings: list()
##  $ errors  : list()
##  $ metadata:List of 1
##   ..$ sources: chr "Perez-SUNY/NREL, 2012"
##  $ inputs  :List of 2
##   ..$ lat: chr "41"
##   ..$ lon: chr "-75"
##  $ outputs :List of 3
##   ..$ avg_dni     :List of 2
##   .. ..$ annual : num 3.69
##   .. ..$ monthly:List of 12
##   .. .. ..$ jan: num 3.12
##   .. .. ..$ feb: num 3.36
##   .. .. ..$ mar: num 4.1
##   .. .. ..$ apr: num 4.07
##   .. .. ..$ may: num 4.15
##   .. .. ..$ jun: num 4.17
##   .. .. ..$ jul: num 4.6
##   .. .. ..$ aug: num 4.14
##   .. .. ..$ sep: num 4.02
##   .. .. ..$ oct: num 3.26
##   .. .. ..$ nov: num 2.58
##   .. .. ..$ dec: num 2.72
##   ..$ avg_ghi     :List of 2
##   .. ..$ annual : num 3.87
##   .. ..$ monthly:List of 12
##   .. .. ..$ jan: num 1.97
##   .. .. ..$ feb: num 2.69
##   .. .. ..$ mar: num 3.86
##   .. .. ..$ apr: num 4.7
##   .. .. ..$ may: num 5.45
##   .. .. ..$ jun: num 5.78
##   .. .. ..$ jul: num 5.98
##   .. .. ..$ aug: num 5.14
##   .. .. ..$ sep: num 4.23
##   .. .. ..$ oct: num 2.94
##   .. .. ..$ nov: num 1.99
##   .. .. ..$ dec: num 1.67
##   ..$ avg_lat_tilt:List of 2
##   .. ..$ annual : num 4.52
##   .. ..$ monthly:List of 12
##   .. .. ..$ jan: num 3.55
##   .. .. ..$ feb: num 4.04
##   .. .. ..$ mar: num 4.86
##   .. .. ..$ apr: num 4.97
##   .. .. ..$ may: num 5.18
##   .. .. ..$ jun: num 5.24
##   .. .. ..$ jul: num 5.58
##   .. .. ..$ aug: num 5.24
##   .. .. ..$ sep: num 5
##   .. .. ..$ oct: num 4.11
##   .. .. ..$ nov: num 3.26
##   .. .. ..$ dec: num 3.13
```

From this list of 6 elements, we only really need the `outputs` element, which is itself a list with components named `avg_dni`, `avg_ghi` and `avg_lat_tilt`: each of these components represents one of the variables for which we wanted to extract data.

Note that `avg_dni`, `avg_ghi` and `avg_lat_tilt` are also lists: each of them contains an `annual` component with a single value and a `monthly` component with a value for each month of the year.

In the following we will be building a dataframe for the `monthly` nested list of each of the average variables. This can be done using two different approaches, which we will now explore.

With the **first approach** we extract each component that we are interested in from the initial complex list and combine them into a dataframe (the necessary packages have been loaded):

```r
outputs <- json_lists[[6]]

avg_dni <- unlist(outputs$avg_dni$monthly)
avg_ghi <- unlist(outputs$avg_ghi$monthly)
avg_lat_tilt <- unlist(outputs$avg_lat_tilt$monthly)
```

```
months <- month.abb

dataframe <- tibble(month = months, avg_dni = avg_dni, avg_ghi = avg_ghi,
                    avg_lat_tilt = avg_lat_tilt)

dataframe
```

```
## # A tibble: 12 x 4
##    month avg_dni avg_ghi avg_lat_tilt
##    <chr>   <dbl>   <dbl>        <dbl>
##  1 Jan      3.12    1.97         3.55
##  2 Feb      3.36    2.69         4.04
##  3 Mar      4.1     3.86         4.86
##  4 Apr      4.07    4.7          4.97
##  5 May      4.15    5.45         5.18
##  6 Jun      4.17    5.78         5.24
##  7 Jul      4.6     5.98         5.58
##  8 Aug      4.14    5.14         5.24
##  9 Sep      4.02    4.23         5
## 10 Oct      3.26    2.94         4.11
## 11 Nov      2.58    1.99         3.26
## 12 Dec      2.72    1.67         3.13
```

With the **second approach** we build the dataframe by simplifying the `outputs` list and restructuring it using a matrix and then converting it into a dataframe:

```
unlisted_outputs <- unlist(outputs)

matrix <- matrix(unlisted_outputs, nrow = 13, ncol = 3)

# The first row of the matrix refers to the annual value so we have to remove it
matrix <- matrix[-1,]

dataframe2 <- as_tibble(matrix)

# Changing column names
colnames(dataframe2) <- c("avg_dni", "avg_ghi", "avg_lat_tilt")

# Adding in the months
dataframe2 <- dataframe2 %>%
  mutate(month = months)

# Rearranging the columns so that `month` becomes the first columns
dataframe2 <- dataframe2[,c(4,1,2,3)]

dataframe2
```

```
## # A tibble: 12 x 4
##    month avg_dni avg_ghi avg_lat_tilt
##    <chr>   <dbl>   <dbl>        <dbl>
##  1 Jan      3.12    1.97         3.55
##  2 Feb      3.36    2.69         4.04
##  3 Mar      4.1     3.86         4.86
##  4 Apr      4.07    4.7          4.97
##  5 May      4.15    5.45         5.18
##  6 Jun      4.17    5.78         5.24
##  7 Jul      4.6     5.98         5.58
##  8 Aug      4.14    5.14         5.24
##  9 Sep      4.02    4.23         5
## 10 Oct      3.26    2.94         4.11
## 11 Nov      2.58    1.99         3.26
## 12 Dec      2.72    1.67         3.13
```

Creating a function to extract the data

Before finalising our project by visualizing the data that we extracted, we would like to provide a custom function to speed up the process of querying the API and converting the resulting data into a dataframe. The function will comprise all the steps that we have previously done plus an automatic error detection system:

```r
nrel_api_json_get_df <- function(endpoint, queries = list()) {
  # Loading necessary libraries
  library(httr)
  library(jsonlite)
  library(tibble)

  # Preparing the URL
  URL <- modify_url("https://developer.nrel.gov/", path = endpoint)

  # API requests
  response <- GET(URL, query = queries)

  # Tracking errors
  if (http_error(response)){
    print(status_code(response))
    print(http_status(response))
    stop("Something went wrong.", call. = FALSE)
  }

  if (http_type(response) != "application/json") {
    stop("API did not return json", call. = FALSE)
  }

  # Extracting content
  json_text <- content(response, "text")

  # Converting content into a complex list
  json_lists <- jsonlite::fromJSON(json_text)

  # Extracting the nested list containing the variables
  outputs <- json_lists[[6]]

  # Preparing the columns
  avg_dni <- unlist(outputs$avg_dni$monthly)
  avg_ghi <- unlist(outputs$avg_ghi$monthly)
  avg_lat_tilt <- unlist(outputs$avg_lat_tilt$monthly)
  months <- month.abb

  # Building the dataframe
  dataframe <- tibble(month = months, avg_dni = avg_dni, avg_ghi = avg_ghi,
                      avg_lat_tilt = avg_lat_tilt)

  # Returning the dataframe
  return(dataframe)
}

# Let's try our new function
solar_resource_df <- nrel_api_json_get_df("api/solar/solar_resource/v1.json",
                                          parameters_list)

solar_resource_df
```

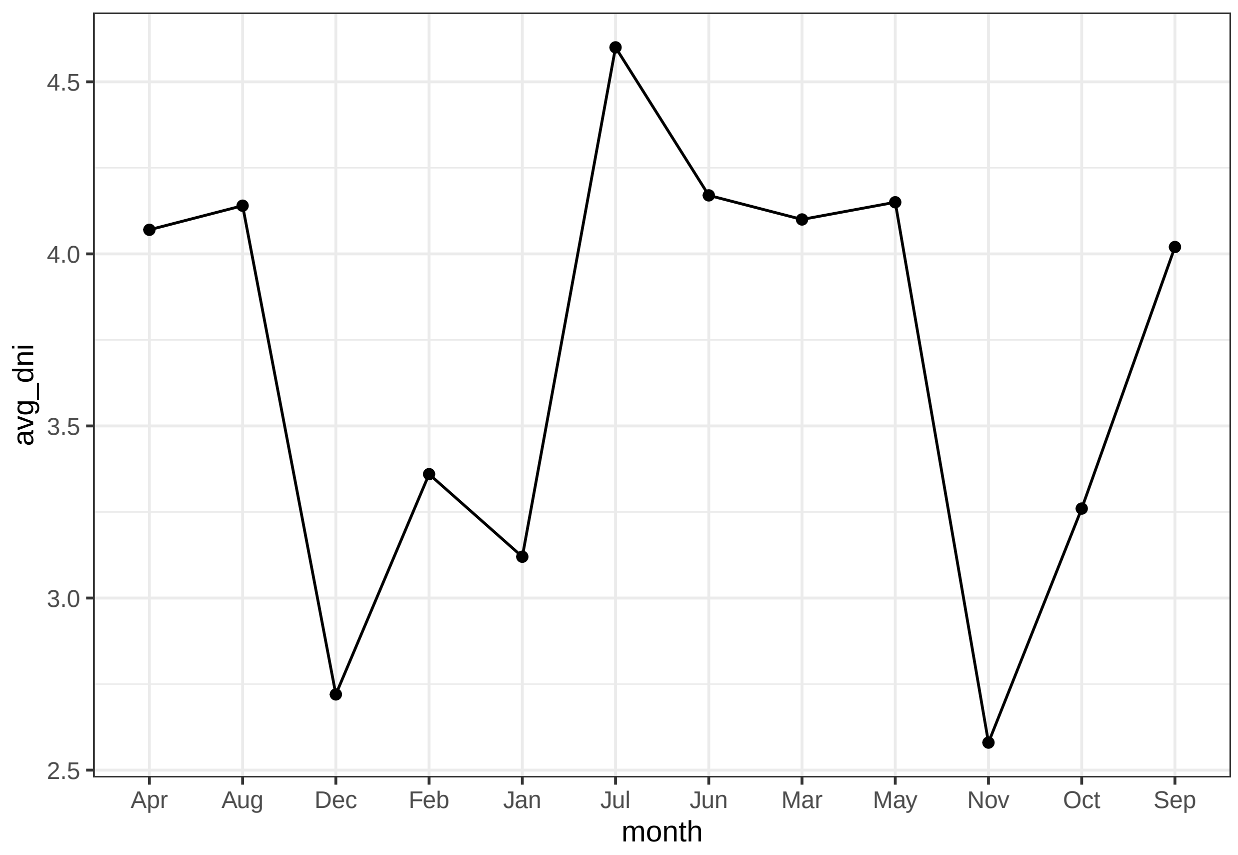
```
## # A tibble: 12 x 4
##    month avg_dni avg_ghi avg_lat_tilt
##    <chr>   <dbl>   <dbl>        <dbl>
##  1 Jan      3.12    1.97         3.55
##  2 Feb      3.36    2.69         4.04
##  3 Mar      4.1     3.86         4.86
##  4 Apr      4.07    4.7          4.97
##  5 May      4.15    5.45         5.18
##  6 Jun      4.17    5.78         5.24
##  7 Jul      4.6     5.98         5.58
##  8 Aug      4.14    5.14         5.24
##  9 Sep      4.02    4.23         5
## 10 Oct      3.26    2.94         4.11
## 11 Nov      2.58    1.99         3.26
## 12 Dec      2.72    1.67         3.13
```

Visualizing the data

The final step in our project is to plot the data that we have extracted to see the variation in the solar radiation during the months of the year.

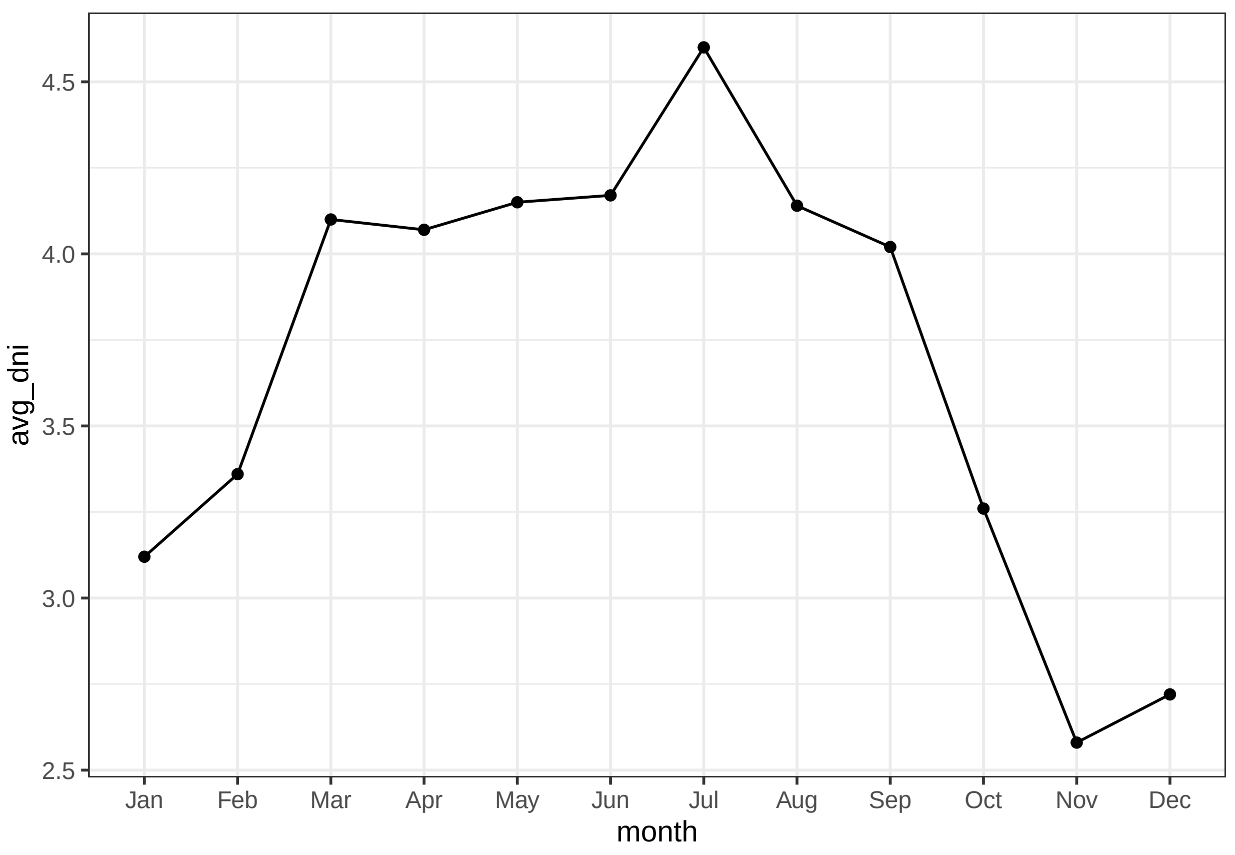
Of the available variables, we will only plot the average direct normal irradiance `avg_dni` and we will do it twice: the first time with the data as it is, the second time after converting the `month` column to a factor.

```r
# ggplot2 has already been loaded

# first plot
ggplot(solar_resource_df, aes(x = month, y = avg_dni, group = 1)) +
  geom_line() +
  geom_point() +
  theme_bw()
```



As we can see, if we leave the `month` column as it is the months will be arranged alphabetically. If we want to arrange them chronologically, we will have transform the column with the `factor` function:

```
solar_resource_df <- solar_resource_df %>%
  mutate(month = factor(month, levels = month.abb))

# second plot
ggplot(solar_resource_df, aes(x = month, y = avg_dni, group = 1)) +
  geom_line() +
  geom_point() +
  theme_bw()
```



As we can see, the average solar radiation increases in spring and summer and is the highest in July.

Conclusion

In this project we have worked with the US **Data.gov API** to extract and visualize data about the New York City average solar radiation.

After extracting the data with a GET request in JSON format, we have converted it into a R list object and pulled out the information about the monthly average solar radiation variables.

We have then transformed the data into a dataframe and added a month column, before visualizing it using a line plot with point markers.

In addition to our main analysis we have also provided a custom function to allow for a faster data extraction from the **Data.gov API**.