Using the climateR and AOI package

Kenneth Pomeyie, Scout Jarman, Paul Gaona-Partida

Introduction

A climatic data element is a measured parameter that describes the climate of a specific location or region, such as precipitation, temperature, wind speed, or humidity. A raster file is one way of storing climate data, where the values are stored as pixels. The size of these raster files tends to get large when the area increases, time is included, or when multiple variables or models are included. In most cases, only a small amount of information is needed from these files. The hard way to extract site-specific climate information from raster files is to download all of the files, load all the data, then extract the small part you need. However, this data extraction method is inefficient and memory-limited because R loads the entire raster into RAM. There are two packages that can help to extract climate data easily, they are `climateR` and `AOI`. The `AOI` package also offers two plotting functions to help with quick region exploration. This vignette shows how `climateR` and `AOI` helps us work with climate rasters the easy way.

Introduction to AOI and climateR packages

There are two packages we will be demonstrating which help us obtain boundaries, climate rasters, and make plots. These two packages are small, simple, and work well with each other and with common spatial classes like `sf`, `sfc`, `raster`, `bbox`.

`climateR` Package

The `climateR` package was made to help simplify the steps to get climate data into R[1]. It provides functions to get climate raster data from 11 dataset including PRISM, GridMET, and TerraClimate. All of the datasets contain historical observations, the GridMET, PRISM, CHIRPS, and EDDI datasets have observations up to the current year, month, and yesterdays, and the MACA, LOCA, BCCA, and BCSD can provide forecasted climate data up to 2100. The full list of datasets can be found on the github page at [1].

`AOI` Package

The `AOI` package seeks to provide reproducible, programmatic boundaries for mapping analyses[2]. This package provides two main functions for easily getting points, polygons, and bounding boxes for continents, countries, regions, states, counties, cities, zipcodes, as well as addresses, events, and much more. It also provides functions for quickly plotting and exploring regions, as well as providing the possibility of creating your own polygons.

These two packages were made by Mike Johnson, and work well together. But they are not on CRAN, so you will have to install them from Github (example below). Be sure that you have R 4.1 or newer or you wont be able to install them.

Advantages of `AOI` and `climateR` package

A summary of the advantages to using `AOI` and `climateR` are:

1. `climateR` offers support for `sf`, `sfc`, and `bbox` objects.
2. `climateR` provides access to 11 climate resources centralized under a common access pattern (e.g. getPRISM, getDaymet).
3. `climateR` allows the user to get the climate data they need without downloading the full dataset.
4. `AOI` allows users to program traditional and non-traditional boundaries/geometries for analysis and mapping.

Coding Examples: `AOI`

The following code demonstrates how to use the most common functions in `AOI`, with an explanation of how to code works under the hood.

Load libraries

The following shows how to install `AOI` and `climateR` from github, and load the packages that will be used in the examples.

```
remotes::install_github("mikejohnson51/AOI",
  upgrade = "never"
) # Only for compiling document
remotes::install_github("mikejohnson51/climateR",
  upgrade = "never"
) # Only for compiling document

# Load libraries
library(climateR)
library(AOI)
library(raster)
library(sf)
library(ggplot2)
library(leaflet.extras)
```

`AOI::Geocode` to get point geometries

The first common function is `geocode`, which takes in a location, zipcode or event as a string, and returns the associated point as a dataframe. Make sure `pt = TRUE` so that it will return the point as an `sf` object. All the other parameters were set to 'NULL' or 'FALSE, as an example to specify parameters or location for accurate results.

```
# Get point geometry of Logan
logan <- geocode(
  location = "Logan",
  zipcode  = NULL,
  event    = NULL,
  pt       = TRUE,
  bb       = FALSE,
  all      = FALSE,
  full     = FALSE
)
```

The way that `geocode` works is different depending on if you are using the zipcode, location, or event parameters. If using `zipcode`, the package references a pre-made list of zipcode point geometries found as the centroid of the zipcode boundary. When using `location`, it passes your argument to an OpenStreetMap search URL[3]. This allows the function to be very flexible, as anything you can search on OpenStreetMap you can get using this function. Similarly, when using the `events` parameter, it formats a Wikipedia search url, and returns the first result which has a latitude/longitude associated with the Wikipedia page. This allows the function to be flexible, as anything you can find in Wikipedia that has a latitude/longitude you can load into R. But this means you have to be careful and verify your results are the locations you think they are.

`AOI::aoi_map` for easy plotting

Luckily `AOI` has a map function to easily plot and check our `AOI` objects. Be sure to use `returnMap = TRUE` to get the map back. This function creates an interactive `leaflet` map where you can explore the region of your `AOI` and beyond.

```
# Check if the location is right
aoi_map(
  AOI = logan,
  returnMap = TRUE
)
```

As we can see this is not the Logan we were hoping for. Here is a correction to the previous example, this time with the correct specifications for the area of interest.

```
# Use correct location name
logan <- geocode(
  location = "Logan UT",
  pt = TRUE
)
```

Again, we visually check to make sure the `AOI` is the one we think it is.

```
# Check if the location is right
aoi_map(
  AOI = logan,
  returnMap = TRUE
)
```

`AOI::aoi_draw` for polygon creation

Another useful function that `AOI` has to offer is the `aoi_draw` function. This opens an interactive `shiny` app which allows the user draw a custom polygon, and save it as an `sf` object.

There does appear to be a bug, where you cannot save the polygon directly to your global environment. Instead, save it as file, then use the following commands to load in the polygon.

```r
# Interactively draw an Area of Interest
aoi_draw()

test <- sf::st_read("aoi_draw/test.gpkg") # Where test is the name we used

aoi_map(test,
  returnMap = TRUE
)
```

AOI::aoi_get for boundaries

The `aoi_get` function is similar to `geocode`, but it is primarily for getting polygons of boundaries and boarders for states and countries. The following shows how to get the polygon for Cache county in Utah, as well as the other parameters se to their defaults.

```r
# Get Cache county boundaries
cache_aoi <- aoi_get(
  x       = NULL,
  country = NULL,
  state   = "UT",
  county  = "Cache",
  fip     = NULL,
  km      = FALSE,
  union   = FALSE
)
cache_aoi
```

```
## Simple feature collection with 1 feature and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -112.1658 ymin: 41.36882 xmax: -111.4016 ymax: 41.99955
## Geodetic CRS:  WGS 84
##   state_region state_division feature_code state_name state_abbr  name
## 1            4              8      1448017       Utah         UT Cache
##   fip_class tiger_class combined_area_code metropolitan_area_code
## 1        H1       G4020                 NA                   <NA>
##   functional_status  land_area water_area fip_code
## 1                 A 3016627500   21097698    49005
##                         geometry
## 1 MULTIPOLYGON (((-111.5078 4...
```

```r
# Map area of interest
aoi_map(
  AOI = cache_aoi,
  returnMap = TRUE
)
```

We can also easily get all the counties in a state by using `county = "all"`.

```r
# Get Utah state boundaries
ut_aoi <- aoi_get(state = "UT", county = "all")
```

```r
# Map area of interest
aoi_map(AOI = ut_aoi, returnMap = TRUE)
```

The way that this function works depends on what region you are searching for. If looking for international boundaries like continents or countries, this function uses the `rnaturalearth` package to get the geometry. Or if you are search for a boundary in the US, it converts the parameters into a FIPS code, then uses the `fipio` package to get the geometry.

Coding example: `climateR`

Point Based Time Series

In this example, we will show how to download a point based time series with a single climate dataset for a specified location. For this example, we want to get the minimum temperature of Logan, UT from the PRISM climate source. PRISM is a dataset of daily high-spatial resolution (4-km) surface meteorological data covering the contiguous US from 1981-present. Using the `getPRISM` function, we specify our area of interest (Logan, UT), the start and end time, and the minimum temperature as the parameter.

```r
# get minimum temperature from PRSIM
logan_temp <- getPRISM(
  AOI       = geocode(location = "Logan UT", pt = TRUE),
  param     = "tmin",
  startDate = "2021-01-01",
  endDate   = "2021-12-31"
)
```

```
## Spherical geometry (s2) switched off

## Spherical geometry (s2) switched on
```

```r
head(logan_temp)
```

```
##   source   lat       lon       date              tmin
## 1  prism 41.75 -111.8333 2021-01-01 -7.53800010681152
## 2  prism 41.75 -111.8333 2021-01-02 -6.80399990081787
## 3  prism 41.75 -111.8333 2021-01-03 -8.00100040435791
## 4  prism 41.75 -111.8333 2021-01-04  -8.0649995803833
## 5  prism 41.75 -111.8333 2021-01-05 -7.94000005722046
## 6  prism 41.75 -111.8333 2021-01-06   -11.28600025177
```

Notice that we did not have to download the full PRISM dataset, and this ran fast. This is because `climateR` uses the OPeNDAP api to access the datasets[4]. OPeNDAP is a free, open source, non-profit organization which hosts large datasets from the scientific community[5]. It allows very specific queries from users, which is how we can specify a specific location, time, and variable, and only get what we need without downloading the entire dataset. The function is also paralleized, so that multiple downloads can be conducted for more speed.

Because `climateR` offers access to 11 datasets, it has 11 main functions, all formatted as `getDATASETNAME`, where DATASETNAME is which ever dataset you want, such as PRISM. All the functions have the same base number of parameters being `AOI` (any `sf` or `sp` polygon), `param` (the variable you are interested in), `startDate` (formatted YYYY-MM-DD), and `endDate`. Some datasets will accept additional parameters, but the are the minimum.

Multi Point Time Series Example

In this example, we will show how to download multi point times series of climate data. We will download precipitation and minimum temperature for Logan, Salt Lake, and Brigham in Utah from the GridMET climate source. GridMET is a dataset of daily high-spatial resolution (4-km) surface meteorological data covering the contiguous US from 1979-yesterday. First, we will use the `geocode` function to create an sf object of point geometry for the specified locations. Next, we will feed the `getGridMET` with our defined sf object and specify our climate type along with the date range. The `getGridMET` function will return a list of two raster files representing precipitation and minimum temperature. Finally, we use the `extract_sites` to extract the climate data for the locations, which returns a list.

```r
# get point of 3 sites
sites <- AOI::geocode(location = c(
  "Logan UT", "Salt Lake UT", "Brigham UT"
), pt = TRUE)


# get precipitation and min temp from GridMET
ut_ppt_tmin <- getGridMET(
  AOI = sites,
  param = c("prcp", "tmin"), startDate = "2021-01-01",
  endDate = "2021-01-31"
)
```

```
## Spherical geometry (s2) switched off
```

```
## Spherical geometry (s2) switched on
```

```r
# extract precipitation and temperature for specified sites from the raster stack
site_extract <- extract_sites(ut_ppt_tmin, sites, "request")

head(site_extract$gridmet_prcp, n = 5)
```

```
##         date site_Logan UT site_Salt Lake UT site_Brigham UT
## 1 2021-01-01             0                 0               0
## 2 2021-01-02             0                 0               0
## 3 2021-01-03           1.2                 0             1.6
## 4 2021-01-04             3               0.4             1.8
## 5 2021-01-05             0                 0             2.6
```

```r
head(site_extract$gridmet_tmin, n = 5)
```

```
##         date site_Logan UT site_Salt Lake UT site_Brigham UT
## 1 2021-01-01         265.6             266.8           263.2
## 2 2021-01-02         264.4             268.1           265.1
## 3 2021-01-03         264.3             269.4           268.2
## 4 2021-01-04         264.7             269.6           269.7
## 5 2021-01-05         261.7               267             267
```

Multi-Layer Raster Example

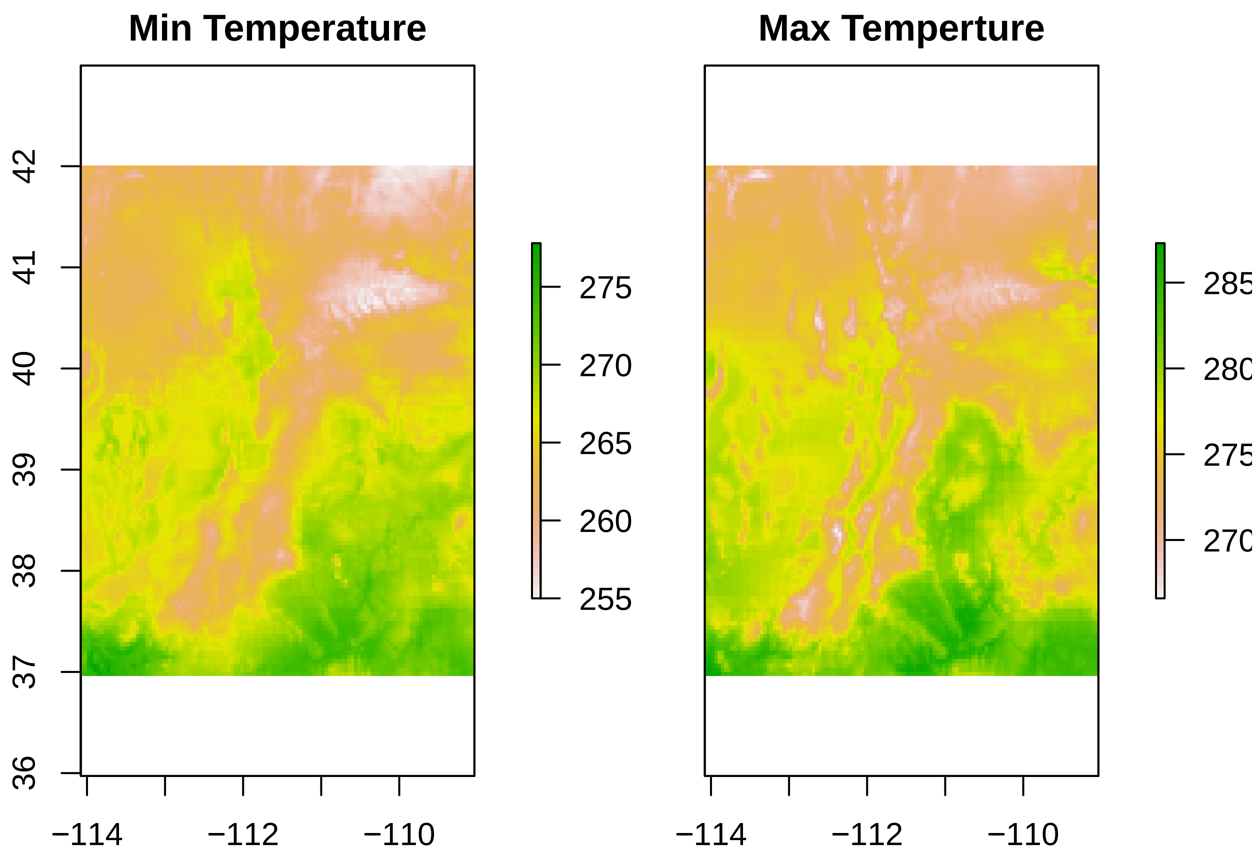
In this example, we show how to download a multi-layer raster file from the GridMET climate source. Using `getGridMET` function we will define the area of interest, the climate parameter and the date range. The difference example 4 and this example is the type of sf object fed to the `getGridMET`. In this example, since we want a raster file we defined the area of interest as a polygon. Below is the plot of the rasters of minimum and maximum temperature for the state of Utah for January 1, 2000.

```r
# get max and min temp  rasters from GridMET
grids <- getGridMET(aoi_get(state = "UT"),
  param = c("tmax", "tmin"),
  startDate = "2000-01-01"
)
```

```
## Spherical geometry (s2) switched off
```

```
## Spherical geometry (s2) switched on
```

```r
raster::plot(raster::stack(grids), main = c("Min Temperature", "Max Temperture"))
```



References

1. climateR Github Page

2. AOI Github Page
3. OpenStreetMap Page
4. OPeNDAP Home Page
5. OPeNDAP Resource