Tutorial: Regional mapping of climate variables from point samples

Data preparation

D G Rossiter
Cornell University, Section of Soil & Crop Sciences
ISRIC–World Soil Information

March 5, 2024

Contents

三人行，必有我师
"Three people walking together, one must be my teacher"

This tutorial shows how to prepare the dataset used in the analysis tutorial "Regional mapping of climate variables from point samples". In that tutorial we use **agricultural climate** as an example of a spatial points dataset, and attempt to model its spatial distribution over a four-(USA)State region

The Northeast Regional Climate Center[1] has kindly provided a set of point ESRI shapefiles with various variables related to agricultural climate measured from 1971-2000. This dataset was devloped by the Unifed Climate Access Network (UVAN) network, and covers the entire United States of America and its dependencies. It consists of several variables relevant for agricultural climate: mean monthly and annual precipitation, mean monthly and annual temperature, annual freeze free period base 32°F and 28°F[2], annual extreme minimum temperature, and monthly and annual growing degree-days , base 50°F (relevant for C4 crops) and base 40°F (relevant for C3 crops)[3].

To model the spatial distribution of agricultural climate, we will use (1) the coördinates of the weather stations; (2) their elevation; (3) several environmental covariates which may be related to climate: terrain and population density.

In this tutorial we show how to set up the dataset for modelling.

1 Setup

Task 1 : Load R packages used in this Tutorial, with the `library` function.

•

```
library(sf)     # Simple Features spatial representation
library(terra)  # raster and vector data processing
library(sp)     # mostly obsolete but has some useful functions
library(ggplot2)  # Grammar of Graphics graphics
```

2 Climate stations

In this section we load files with the weather information, limit the geographic area to a study area, make both a spatial and non-spatial versions of the dataset and re-project the spatial version.

2.1 ESRI shapefiles for Northeast climate

The ESRI shapefiles are provided in a compressed file named `weather_stn_sums_1971_2000.zip`.

[1] http://www.nrcc.cornell.edu
[2] -2.2 and 0°C
[3] 4.44 and 10°C, respectively

Task 2 : Locate this file, unpack it in a working directory named `NEweather`, start R (preferably via RStudio) and connect to that directory. •

If using RStudio, you can navigate to the directory where you've stored the climate files with the "Files" window pane, and then select menu command `Session | Set Working Directory…| To Files Pane Location`.

If entering an R command at the command prompt, within the RStudio console window or directly in R, use the `setwd` function with the path name as its argument.

In this example – that is, on my system – I have set up a directory named `NEweather`, under a subdirectory for datasets that I call `ds`, this under my home directory on a Unix-like system (including Mac OS X), symbolized as `~`. You can choose any location on your file system.

For example, if the `NEweather` directory is inside a `ds` 'datasets' directory under my home directory, the following command will connect to it on a Unix-like system:

```
setwd("~/ds/NEweather")
```

On a Windows system, you will need to specify a full path, e.g.:

```
setwd("M://css6200/dgr2/datasets/NEweather")
```

The location of these files on your system will almost surely be different, so adjust this command accordingly.

If working in RStudio, the most reliable way to set the directory is to set up an R Studio project, and then use the `Session | Set Working Directory | To Project Directory` menu command.

Task 3 : List the shapefiles in this directory. •

The `list.files` function lists the files to the console output; you can also look at the directory in a file manager.

- Its first argument is the directory in which to look.
- The optional `pattern` argument gives a *regular expression* to match the file names. Here we just want the base shape files; there are other "helper" files with the same name but different file extensions, so we specify a pattern of the `shp` extension, at the end of the string, as symbolized by the special `$` regular expression character.

If the files are in the directory where you are connected[4], you can just specify `"."`, which is a Unix and R abbreviation for "the current directory". If they are in a subdirectory, you need to name that, using the forward slash `"/"` character to show you are descending the directory tree.

So, find the unpacked files like this:

[4] use `getwd` to find this

```
list.files(".", pattern=".shp$")
```

```
## [1] "extmin_7100j.shp" "frz28_7100j.shp"  "frz32_7100j.shp"
## [4] "gdd40_7100j.shp"  "gdd50_7100j.shp"  "maat7100.shp"
## [7] "map7100.shp"
```

Each of these shapefiles has associated metadata, with extension `.xml` which can be read in several viewers.

Task 4 : Load the shapefile for the all-USA GDD50 and examine its structure. •

The `st_read` function of the `sf` "Simple Features spatial objects" package can read many formats into `sf` objects implemented by the `sf` package; one of the formats is the ESRI shapefile.

- The first argument `sf` is the "data set name", which in ESRI terminology is not a data set, but rather the directory where the shapefile is located.
- The second argument `layer` is the layer name, *without* an extension – this is because there are several files, with different extensions, associated with one shapefile.
- There are many optional arguments, with defaults, that control the input, see `?st_read` for details. One that is needed on **64-bit operating systems**[5], but which will not cause any problems on other systems, is `int64_as_string`. By default it is set to convert integers into string, to avoid data loss of large integers (needing more than 32 bits). The numbers in most datasets, including this one, are much smaller. So we set this argument to `FALSE`; this will read integers in correctly as numbers.

If you have set the working directory to the location of the dataset, the data set name is written as `"."`, meaning "the currently connected directory". The `dsn` argument should be the same as the directory argument to `list.files`, above.

Load the dataset:

```
usa <- st_read(dsn=".", layer="gdd50_7100j",
               int64_as_string = FALSE, quiet = TRUE)
print(usa)
```

```
## Simple feature collection with 5556 features and 26 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -177.35 ymin: -14.33 xmax: 174.1 ymax: 71.28
## Geodetic CRS:  NAD27
## First 10 features:
##    STATION_ID STATE            STATION_NA LATITUDE_D LONGITUDE_
## 1       10160    AL       ALEXANDER CITY      32.95     -85.95
## 2       10178    AL           ALICEVILLE      33.13     -88.13
## 3       10252    AL         ANDALUSIA 3 W      31.30     -86.52
## 4       10272    AL     ANNISTON METRO AP      33.58     -85.85
## 5       10369    AL          ASHLAND 3 SE      33.23     -85.82
```

[5] for example, Microsoft Windows 10

```
## 6       10395    AL              ATHENS 2      34.80     -86.98
## 7       10402    AL  ATMORE STATE NURSERY      31.17     -87.43
## 8       10430    AL  AUBURN AGRONOMY FARM      32.60     -85.50
## 9       10505    AL BANKHEAD LOCK AND DAM      33.45     -87.35
## 10      10583    AL           BAY MINETTE      30.88     -87.78
##    ELEVATION_ OID_ COOP_ID STATE_1              STN_NAME LAT_DD
## 1         640    0   10160      AL        ALEXANDER CITY     33
## 2         240    1   10178      AL             ALICEVILLE     33
## 3         250    2   10252      AL          ANDALUSIA 3 W     31
## 4         594    3   10272      AL     ANNISTON METRO AP     34
## 5        1010    4   10369      AL           ASHLAND 3 SE     33
## 6         720    5   10395      AL               ATHENS 2     35
## 7         300    6   10402      AL  ATMORE STATE NURSERY     31
## 8         652    7   10430      AL  AUBURN AGRONOMY FARM     33
## 9         280    8   10505      AL BANKHEAD LOCK AND DAM     33
## 10        271    9   10583      AL            BAY MINETTE     31
##    LONG_DD ELEV_FT JAN_GDD50 FEB_GDD50 MAR_GDD50 APR_GDD50 MAY_GDD50
## 1      -86     640        66        49       185       335       585
## 2      -88     240        61        61       206       350       620
## 3      -87     250       132       115       278       435       685
## 4      -86     594        69        62       198       353       601
## 5      -86    1010        66        61       187       330       567
## 6      -87     720        40        29       125       284       542
## 7      -87     300       123       106       268       432       688
## 8      -86     652        85        72       212       376       638
## 9      -87     280        45        44       151       277       536
## 10     -88     271       132       113       272       456       697
##    JUN_GDD50 JUL_GDD50 AUG_GDD50 SEP_GDD50 OCT_GDD50 NOV_GDD50
## 1        780       914       886       702       401       172
## 2        822       957       923       729       408       169
## 3        858       967       951       804       507       260
## 4        798       933       914       714       405       166
## 5        750       880       858       684       392       179
## 6        741       883       855       636       323       111
## 7        843       945       942       792       519       266
## 8        816       926       911       741       451       204
## 9        747       883       861       666       348       131
## 10       843       930       920       768       509       262
##    DEC_GDD50 ANN_GDD50             geometry
## 1         78      5153 POINT (-85.95 32.95)
## 2         76      5382 POINT (-88.13 33.13)
## 3        146      6138  POINT (-86.52 31.3)
## 4         80      5293 POINT (-85.85 33.58)
## 5         79      5033 POINT (-85.82 33.23)
## 6         38      4607  POINT (-86.98 34.8)
## 7        136      6060 POINT (-87.43 31.17)
## 8         98      5530   POINT (-85.5 32.6)
## 9         57      4746 POINT (-87.35 33.45)
## 10       138      6040 POINT (-87.78 30.88)
```

Geographic objects defined in the `sf` package all have a coördinate reference system (CRS), which defines how the coördinates for the object relate to the Earth's surface. This can be `NA` "not available" or a list of CRS parameters.

Task 5 : Show the CRS for the climate stations shapefile. •

We use the `st_crs` function of the `sf` package to show the CRS for the **usa** object.

```
st_crs(usa)

## Coordinate Reference System:
##   User input: NAD27
##   wkt:
## GEOGCRS["NAD27",
##     DATUM["North American Datum 1927",
```

```
##         ELLIPSOID["Clarke 1866",6378206.4,294.978698213898,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4267]]

st_bbox(usa)

##    xmin    ymin    xmax    ymax
## -177.35  -14.33  174.10   71.28
```

This uses the "Well Known Text"(WKT) form of the CRS. A simpler view is with the `proj4` format:

```
st_crs(usa)$proj4string

## [1] "+proj=longlat +datum=NAD27 +no_defs"
```

The ESRI shapefile is correctly imported as a spatial object, with the correct coördinate reference system (CRS), here geographic coördinates on the NAD27 datum, which uses the Clarke 1866 ellipsoid; this is EPSG[6] code 4267.

The bounding box covers almost the whole world from W to E. This is because the USA controls possessions in the western Pacific ocean; also, the Aleutian Islands of Alaska cross the international date line.

Task 6 : Show the field names and their data types. •

```
names(usa)

##  [1] "STATION_ID" "STATE"      "STATION_NA" "LATITUDE_D" "LONGITUDE_"
##  [6] "ELEVATION_" "OID_"       "COOP_ID"    "STATE_1"    "STN_NAME"
## [11] "LAT_DD"     "LONG_DD"    "ELEV_FT"    "JAN_GDD50"  "FEB_GDD50"
## [16] "MAR_GDD50"  "APR_GDD50"  "MAY_GDD50"  "JUN_GDD50"  "JUL_GDD50"
## [21] "AUG_GDD50"  "SEP_GDD50"  "OCT_GDD50"  "NOV_GDD50"  "DEC_GDD50"
## [26] "ANN_GDD50"  "geometry"
```

The metadata lists the field names, but without any explanation; fortunately they are fairly self-explanatory, for example `STATE` for the US State responsible for the weather station, `STATION_NA` for the station name, `ELEVATION_` for the elevation, etc. We see fields for GDD50 for each of 12 months (e.g., `JAN_GDD50`), as well as annual (`ANN_GDD50`.

For completeness, we change some fields that were read as numeric to character strings using the `as.character` function. These fields are not numbers in the mathematical sense. Rather, they are identification codes, with no analytical meaning other than to identify locations.

```
str(usa)

## Classes 'sf' and 'data.frame': 5556 obs. of  27 variables:
```

[6] https://epsg.org/home.html

```
##  $ STATION_ID: int  10160 10178 10252 10272 10369 10395 10402 10430 10505 10583 ...
##  $ STATE     : chr  "AL" "AL" "AL" "AL" ...
##  $ STATION_NA: chr  "ALEXANDER CITY" "ALICEVILLE" "ANDALUSIA 3 W" "ANNISTON METRO AP" ...
##  $ LATITUDE_D: num  33 33.1 31.3 33.6 33.2 ...
##  $ LONGITUDE_: num  -86 -88.1 -86.5 -85.8 -85.8 ...
##  $ ELEVATION_: int  640 240 250 594 1010 720 300 652 280 271 ...
##  $ OID_      : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ COOP_ID   : int  10160 10178 10252 10272 10369 10395 10402 10430 10505 10583 ...
##  $ STATE_1   : chr  "AL" "AL" "AL" "AL" ...
##  $ STN_NAME  : chr  "ALEXANDER CITY" "ALICEVILLE" "ANDALUSIA 3 W" "ANNISTON METRO AP" ...
##  $ LAT_DD    : int  33 33 31 34 33 35 31 33 33 31 ...
##  $ LONG_DD   : int  -86 -88 -87 -86 -86 -87 -87 -86 -87 -88 ...
##  $ ELEV_FT   : int  640 240 250 594 1010 720 300 652 280 271 ...
##  $ JAN_GDD50 : num  66 61 132 69 66 40 123 85 45 132 ...
##  $ FEB_GDD50 : num  49 61 115 62 61 29 106 72 44 113 ...
##  $ MAR_GDD50 : num  185 206 278 198 187 125 268 212 151 272 ...
##  $ APR_GDD50 : num  335 350 435 353 330 284 432 376 277 456 ...
##  $ MAY_GDD50 : num  585 620 685 601 567 542 688 638 536 697 ...
##  $ JUN_GDD50 : num  780 822 858 798 750 741 843 816 747 843 ...
##  $ JUL_GDD50 : num  914 957 967 933 880 883 945 926 883 930 ...
##  $ AUG_GDD50 : num  886 923 951 914 858 855 942 911 861 920 ...
##  $ SEP_GDD50 : num  702 729 804 714 684 636 792 741 666 768 ...
##  $ OCT_GDD50 : num  401 408 507 405 392 323 519 451 348 509 ...
##  $ NOV_GDD50 : num  172 169 260 166 179 111 266 204 131 262 ...
##  $ DEC_GDD50 : num  78 76 146 80 79 38 136 98 57 138 ...
##  $ ANN_GDD50 : num  5153 5382 6138 5293 5033 ...
##  $ geometry  :sfc_POINT of length 5556; first list element:  'XY' num  -86 33
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA NA ...
##   ..- attr(*, "names")= chr [1:26] "STATION_ID" "STATE" "STATION_NA" "LATITUDE_D" ...
```

```
usa$STATION_ID <- as.character(usa$STATION_ID)
usa$STATION_NA <- as.character(usa$STATION_NA)
usa$STN_NAME <- as.character(usa$STN_NAME)
usa$OID_<- as.character(usa$OID_)
usa$COOP_ID <- as.character(usa$COOP_ID)
summary(usa)
```

```
##   STATION_ID           STATE            STATION_NA
##  Length:5556        Length:5556        Length:5556
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##    LATITUDE_D       LONGITUDE_        ELEVATION_
##  Min.   :-14.33   Min.   :-177.35   Min.   : -194
##  1st Qu.: 35.17   1st Qu.:-110.09   1st Qu.:  465
##  Median : 39.56   Median : -96.52   Median : 1030
##  Mean   : 39.32   Mean   : -97.55   Mean   : 1866
##  3rd Qu.: 43.44   3rd Qu.: -85.37   3rd Qu.: 2710
##  Max.   : 71.28   Max.   : 174.10   Max.   :11320
##      OID_             COOP_ID            STATE_1
##  Length:5556        Length:5556        Length:5556
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##    STN_NAME             LAT_DD          LONG_DD          ELEV_FT
##  Length:5556        Min.   :-14.00   Min.   :-177.0   Min.   : -194
##  Class :character   1st Qu.: 35.00   1st Qu.:-110.0   1st Qu.:  460
##  Mode  :character   Median : 40.00   Median : -96.0   Median : 1025
##                     Mean   : 39.12   Mean   : -96.9   Mean   : 1863
##                     3rd Qu.: 43.00   3rd Qu.: -85.0   3rd Qu.: 2705
##                     Max.   : 71.00   Max.   : 174.0   Max.   :11320
##    JAN_GDD50        FEB_GDD50        MAR_GDD50      APR_GDD50
##  Min.   :   0.0   Min.   :  0.00   Min.   :   0   Min.   :   0.0
##  1st Qu.:   0.0   1st Qu.:  0.00   1st Qu.:   7   1st Qu.:  44.0
```

```
##  Median :   9.0   Median : 13.00   Median :  23   Median :  99.0
##  Mean   :  43.3   Mean   : 49.09   Mean   :  91   Mean   : 176.5
##  3rd Qu.:  29.0   3rd Qu.: 42.00   3rd Qu.: 118   3rd Qu.: 269.0
##  Max.   :1013.0   Max.   :918.00   Max.   :1026   Max.   :1038.0
##    MAY_GDD50        JUN_GDD50        JUL_GDD50        AUG_GDD50
##  Min.   :   0.0   Min.   :   0.0   Min.   :   0.0   Min.   :   0.0
##  1st Qu.: 198.0   1st Qu.: 405.0   1st Qu.: 579.0   1st Qu.: 539.0
##  Median : 324.0   Median : 567.0   Median : 734.0   Median : 682.0
##  Mean   : 367.6   Mean   : 568.4   Mean   : 724.7   Mean   : 685.3
##  3rd Qu.: 517.0   3rd Qu.: 744.0   3rd Qu.: 902.0   3rd Qu.: 855.0
##  Max.   :1078.0   Max.   :1335.0   Max.   :1568.0   Max.   :1512.0
##    SEP_GDD50        OCT_GDD50        NOV_GDD50        DEC_GDD50
##  Min.   :   0.0   Min.   :   0.0   Min.   :  0.0   Min.   :   0.00
##  1st Qu.: 277.8   1st Qu.:  61.0   1st Qu.:  8.0   1st Qu.:   0.00
##  Median : 430.0   Median : 154.0   Median : 24.0   Median :  10.00
##  Mean   : 461.0   Mean   : 224.7   Mean   : 87.3   Mean   :  47.68
##  3rd Qu.: 636.0   3rd Qu.: 341.0   3rd Qu.:108.0   3rd Qu.:  35.00
##  Max.   :1203.0   Max.   :1023.0   Max.   :996.0   Max.   :1023.00
##    ANN_GDD50              geometry
##  Min.   :    0   POINT        :5556
##  1st Qu.: 2155   epsg:4267    :   0
##  Median : 3060   +proj=long...:   0
##  Mean   : 3527
##  3rd Qu.: 4584
##  Max.   :11985
```

The records are from the 48 USA states, several territories (e.g., `PR` for Puerto Rico), and one unspecified, code `??`.

```
length(unique(usa$STATE))

## [1] 54

unique(usa$STATE)

##  [1] "AL" "AZ" "AR" "CA" "CO" "CT" "DE" "FL" "GA" "ID" "IL" "IN" "IA"
## [14] "KS" "KY" "LA" "ME" "MD" "MA" "MI" "MN" "MS" "MO" "MT" "NE" "NV"
## [27] "NH" "NJ" "NM" "NY" "NC" "ND" "OH" "OK" "OR" "PA" "RI" "SC" "SD"
## [40] "TN" "TX" "UT" "VT" "VA" "WA" "WV" "WI" "WY" "AK" "HI" "PR" "VI"
## [53] "??" "PI"
```

Challenge: What does code `??` represent? Hint: use the selection technique of the next section to examine these.

2.2 Subsetting to four States

Task 7 : Select the records reported by NY State and its neighbours VT, NJ and PA. •

Both the `%in%` set operator and the `[]` matrix selection operator find matches between strings, and return a list of matrix row numbers. This us the `ix` vector, whose entries are either `TRUE` or `FALSE`, i.e., a vector of **logical** type. The `sum` function takes the `TRUE` values as 1 and the `FALSE` as 0, so the sum is the number of `TRUE` values, i.e., those that met the logical condition. The `which` function is shorthand for "which of the elements in the logical vector are `TRUE`?" It returns a vector of row numbers for which the logical expression, in this case, that the record is from one of the four states, is `TRUE`. The syntax `[ix,]` then selects only the records (dataframe rows) where the value is `TRUE`.

```
dim(usa)[1]  # number of rows in the full dataset

## [1] 5556

ix <- (usa$STATE %in% c("NY","VT","NJ","PA"))  # logical expression
str(ix)      # each row is either TRUE or FALSE

##  logi [1:5556] FALSE FALSE FALSE FALSE FALSE FALSE ...

sum(ix)      # number of rows selected

## [1] 305

head(which(ix))  # first six row numbers of stations in these states

## [1] 2852 2853 2854 2855 2856 2857

ne <- usa[ix,]  # select only these states' records
dim(ne)[1]   # number of rows in the restricted dataset

## [1] 305
```

The `dim` function returns the dimensions of a data frame as a two-element vector: number of rows (records) and columns (fields). The first dimension is the number of rows; we see how this reduces from 5556 in the full dataset to 305 in the four-state dataset.

Even after selection the `STATE` factor has all the original levels for the whole USA (i.e. all state abbreviations), even though most have no records.

Therefore we renumber the levels of the `STATE` factor, with the `factor` function, since we only have four states in this dataset. Although `STATE` was already an R factor (categorical variable), re-applying the `factor` function discards the unused levels (states we did not select) and re-numbers the factor levels.

```
class(ne$STATE)

## [1] "character"

levels(ne$STATE)

## NULL

ne$STATE <- factor(ne$STATE)
levels(ne$STATE)

## [1] "NJ" "NY" "PA" "VT"
```

Task 8 : Display the bounding box. •

```
st_bbox(ne)

##   xmin   ymin   xmax   ymax
## -80.47  38.95 -71.98  44.93
```

The bounding box is now much smaller than for the original dataset; it is just big enough to enclose the stations from the four selected states.

2.3 * Checking for duplicate locations

Obviously there can not be two climate stations at the exact same location. However, since the database records station coördinates in geographic units with a limited resolution, it may be that two close-by stations may be recorded with the same coördinates. That will cause various geostatistical procedures in analysis to fail, because these procedures do not know how to understand co-located points with different attribute values, e.g., different GDD50.

Task 9 : Check for co-located points. •

The `zerodist` function of the `sp` package checks for this, however it requires its argument to be an `sp` object – an older representation of spatial data than `sf` objects.

```
(d <- sp::zerodist(as_Spatial(ne)))

##      [,1] [,2]
```

> **Note:** The `package::function` syntax explicitly names the package where the named function is found. Here we use it to avoid loading the `sp` package – we just need to use this one function. Of course, the package must be installed on the system.

In this case there are no co-located points, so for the purposes of this exercise you can **skip to the next section** (§2.4).

However, elsewhere in this dataset there are some duplicates. If you take up the challenge to work on other areas of the USA, you may need to deal with these.

```
(d <- sp::zerodist(as_Spatial(usa)))

##      [,1] [,2]
## [1,] 1280 1311
## [2,] 3201 3202
## [3,] 5538 5553

usa[d, ]

## Simple feature collection with 6 features and 26 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -85.28 ymin: 7.45 xmax: 151.83 ymax: 40.43
## Geodetic CRS:  NAD27
##      STATION_ID STATE          STATION_NA LATITUDE_D LONGITUDE_
## 1280     123777    IN HARTFORD CITY 4 ESE      40.43     -85.28
## 3201     314260    NC         HOT SPRINGS      35.90     -82.83
## 5538     914111    PI            CHUUK AP       7.45     151.83
## 1311     127069    IN       PORTLAND 1 SW      40.43     -85.28
## 3202     314265    NC       HOT SPRINGS 2      35.90     -82.83
## 5553     914851    PI             TRUK AP       7.45     151.83
##      ELEVATION_ OID_ COOP_ID STATE_1            STN_NAME LAT_DD
## 1280        942 1278  123777      IN HARTFORD CITY 4 ESE     40
## 3201       1396 3199  314260      NC         HOT SPRINGS     36
## 5538          5 5501  914111      PI            CHUUK AP      7
## 1311        910 1309  127069      IN       PORTLAND 1 SW     40
## 3202       1480 3200  314265      NC       HOT SPRINGS 2     36
## 5553          5 5516  914851      PI             TRUK AP      7
##      LONG_DD ELEV_FT JAN_GDD50 FEB_GDD50 MAR_GDD50 APR_GDD50
```

```
## 1280     -85     942         9        14        36       113
## 3201     -83    1396        29        15        78       202
## 5538     152       5       976       876       979       954
## 1311     -85     910         1         9        21        80
## 3202     -83    1480        37        23        95       218
## 5553     152       5       979       890       992       966
##      MAY_GDD50 JUN_GDD50 JUL_GDD50 AUG_GDD50 SEP_GDD50 OCT_GDD50
## 1280       371       615       734       682       459       194
## 3201       448       651       799       768       576       273
## 5538       988       948       973       957       939       973
## 1311       329       579       716       638       415       146
## 3202       453       657       802       781       582       296
## 5553      1004       963       985       985       960       992
##      NOV_GDD50 DEC_GDD50 ANN_GDD50            geometry
## 1280        34        16      3277 POINT (-85.28 40.43)
## 3201        91        24      3954  POINT (-82.83 35.9)
## 5538       963       976     11502  POINT (151.83 7.45)
## 1311        21        13      2968 POINT (-85.28 40.43)
## 3202       106        35      4085  POINT (-82.83 35.9)
## 5553       963       992     11671  POINT (151.83 7.45)
```

We see there are three duplicates. In the GDD50 file `gdd50_7100j.shp` there are one pair of duplicates in Indiana, one in North Carolina, and one in the Pacific Islands.

In case there are co-located points, there are two solutions:

1. Examine the record and compare with other information, e.g., an internet map (Google, Bing, OpenStreet …), to see if one named station is obviously mis-located. If there is strong evidence for this, change the coördinates for one of the stations.

   For example, of the duplicate points in Indiana, we confirm from Google Earth that the reported coördinates are indeed about 4 miles ESE of Hartford City for record FALSE. The same coördinates are given for 1 mile SW of Portland, record FALSE. But Portland is quite a bit to the E, the centre of the town is at $(40.43, -84.98)$. The longitude differs by about 0.3° from the reported value. Looking at the Google Earth image, we measure 1 mile SW from the centre of Portland, which lies in a large cemetary. Looking closely, we see the administration building, which is the likely location of the weather station. Clicking on this point gives a corrected coördinates of $(40.42, -84.99)$.

   To adjust the incorrect point, we should change both the the coördinates in the `sf` object, and also the coördinates given as attributes in the `data.frame` object.

```
st_coordinates(usa)[d[1],]; st_coordinates(usa)[d[4],]

##      X      Y
## -85.28  40.43
##      X      Y
## -85.28  40.43

usa[d[1], c("STATE", "STATION_NA")]; usa[d[4], c("STATE", "STATION_NA")]

## Simple feature collection with 1 feature and 2 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -85.28 ymin: 40.43 xmax: -85.28 ymax: 40.43
## Geodetic CRS:  NAD27
```

```
##      STATE          STATION_NA             geometry
## 1280    IN HARTFORD CITY 4 ESE POINT (-85.28 40.43)
## Simple feature collection with 1 feature and 2 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -85.28 ymin: 40.43 xmax: -85.28 ymax: 40.43
## Geodetic CRS:  NAD27
##      STATE    STATION_NA             geometry
## 1311    IN PORTLAND 1 SW POINT (-85.28 40.43)

st_geometry(usa)[d[4]] <- st_sfc(st_point(c(-84.99, 40.42)))
usa[d[1],4:5] <- c(40.42, -84.99)  # this is the order in the dataframe
```

2. Delete one of the co-located points. The choice on which to delete could come from examining the whole record to see which seems more reliable, for example, which has a reported elevation which matches best the location given by the coördinates. Or, one record has a longer time-series than the other, so is presumably more reliable for long-term averages. For example, if the fourth point in the list of duplicates should be eliminated, the code would look like this:

```
usa <- (usa[-d[4],])
```

   The `-` operator applied to the first matrix subscript means all rows (records) except those in the list.

2.4 Projecting from geographic to metric CRS

This shapefile is in geographic coördinates. For geostatistical analysis it's necessary to convert it to metric (projected) coördinates, i.e., where distances between points can be accurately measured. The UTM CRS is not suitable, because the of this shapefile bounding box is about 8.5° wide, and a UTM zone only covers 6°. Further, the region is oriented more or less E–W and not N–S.

> **Note:** Selecting an appropriate CRS (projection and coördinate system) is not simple. If the study area is in a single political entity, e.g., a USA State, it's common to select the CRS that is used for that entity's own mapping. For example, many USA States have a State Plane Coördinate System. Otherwise we have to consider the shape of the area: E–W areas often use conic projections, N–S areas often use transverse Mercator (Gauss-Krüger) projections. All projections distort; for geostatistics and trend surfaces we want minimal distortion of distances. Several comprehensive references [1, 4, 5] explain projections, their properties, and typical uses.

A reasonable choice for this study area is an Albers Equal-area conic projection, optimized for the northeastern USA, and on the WGS84 ellipsoid. Although this is an equal-area projection, scale is not distorted along the two standard parallels which define it, and if these are carefully chosen the error in measuring distances (which is what we want) is quite small [4, §14].

The `proj.4` "Cartographic Projections, version 4" project[7] provides standard Unix functions to define and transform projections. This is used in QGIS and in the R `sf` package. Classes defined in the `sf` package, such as the climate stations dataset, include a `geometry` attribute, which includes the CRS according to the `proj.4` definitions.

[7] https://proj.org/en/9.3/

For example, the climate stations were imported from an ESRI shapefile, with the CRS defined; we can see this with the `st_crs` function of the `sf` package:

```
st_crs(ne)

## Coordinate Reference System:
##   User input: NAD27
##   wkt:
## GEOGCRS["NAD27",
##     DATUM["North American Datum 1927",
##         ELLIPSOID["Clarke 1866",6378206.4,294.978698213898,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4267]]

st_crs(ne)$proj4string

## [1] "+proj=longlat +datum=NAD27 +no_defs"
```

These are geographic coördinates, as we can see from the `GEOGCRS` header, not projected on a flat surface with metric coördinates.

The `st_transform` function of the `sf` package can transform among CRS. These can be defined with `proj.4` syntax, but commonly are specified by their "well-known identifier" (WKID), a numeric code that can be used to identify a projection, either from EPSG or ESRI. However here we want to define our custom projection, so the WKID approach will not work.

We need to find the parameters of a suitable metric CRS, and defined a string for it in `proj.4` format; this can then be used in the `st_transform` function to define the projection from geographic to metric coördinates. Projections and their parameters are found on the `proj.4` project webpage[8], under the "Projections" category. Another source is the projections transform list at the GeoTiff site[9]. Here[10] we find the required parmeters and how they are represented in the `proj.4` string:

```
+proj=aea   +lat_1=Latitude of first standard parallel
            +lat_2=Latitude of second standard parallel
            +lat_0=Latitude of false origin
            +lon_0=Longitude of false origin
            +x_0=Easting of false origin
            +y_0=Northing of false origin
```

The "false origin" is the $(0, 0)$ of the coördinates. Since this is a metric CRS, where distances are measured, we also need to specify units of measure with the `+units=` substring.

[8] http://proj4.org/
[9] http://geotiff.maptools.org/proj_list/
[10] http://geotiff.maptools.org/proj_list/albers_equal_area_conic.html

We choose parallels (parameters `+lat_1` and `+lat_2`) about 1° of latitude inside the N and S edges of the four states, and a meridian (parameter `+lon_0`) near the middle. For convenience we use integer meridians and parallels, estimated with the `round`, `floor` ad `ceiling` functions. There is no mathematical reason for this.

```
st_bbox(ne)

##   xmin   ymin   xmax   ymax
## -80.47  38.95 -71.98  44.93

round(median(st_bbox(ne)[c("xmin","xmax")]),1)

## [1] -76.2

round(median(st_bbox(ne)[c("ymin","ymax")]),1)

## [1] 41.9

floor(st_bbox(ne)["ymin"]+1)

## ymin
##   39

ceiling(st_bbox(ne)["ymin"]-1)

## ymin
##   38
```

We set the false origin (0, 0) (parameters `+lat_0` and `+lon_0`) near the centre of the map, at (76°W, 42.5°N); so the projected coördinates will be negative to the south and west of this point. This is an arbitrary choice, any point could be used for the origin of the projected coördinates.

Task 10 : Reproject the shapefile to an Albers equal-area projection with standard parallels 39 and 44°N, central meridian 76°W, origin of coördinate (in meters) at this meridian and 42.5°N (the centre of the two standard parallels), on the WGS84 ellipsoid.

Display the bounding box in the transformed coördinates. •

> **Note:** These choices minimize the scale distortion across the mapped area. The choice of origin near the centre keeps the coördinates as small as possible; this is just for convenience and has no effect on the calculations.

The `st_transform` method of the `sf` package performs the transformation, when given an `sf` object and a new CRS, which is specified with the `st_crs` function, or (as in this case) extracted from the CRS of another object.

```
(ne.crs <-
  st_crs("+proj=aea +lat_0=42.5 +lat_1=39
         +lat_2=44 +lon_0=-76 +ellps=WGS84 +units=m"))$proj4string

## [1] "+proj=aea +lat_0=42.5 +lon_0=-76 +lat_1=39 +lat_2=44 +x_0=0 +y_0=0 +ellps=WGS84 +units=m +no_d

ne.m <- st_transform(ne, ne.crs)
```

We also need a data frame version of this object – where the coördinates are just fields and not treated specially as they are in `sf` classes – for some analysis functions.

Task 11 : Make a data frame from this `sf` object. •

This requires three steps: (1) convert the geometry column of the `sf` object (i.e., the coördinates) as a separate object; (2) remove the geometry column from the `sf` object, i.e., convert to a `data.frame`; (3) add the coördinates to this data frame with the `rbind` function.

```
class(ne.m)

## [1] "sf"         "data.frame"

class(ne.coords <- st_coordinates(ne.m))

## [1] "matrix" "array"

class(ne.df <- st_drop_geometry(ne.m))

## [1] "data.frame"

ne.df <- cbind(ne.df, ne.coords)
names(ne.df)

##  [1] "STATION_ID" "STATE"      "STATION_NA" "LATITUDE_D" "LONGITUDE_"
##  [6] "ELEVATION_" "OID_"       "COOP_ID"    "STATE_1"    "STN_NAME"
## [11] "LAT_DD"     "LONG_DD"    "ELEV_FT"    "JAN_GDD50"  "FEB_GDD50"
## [16] "MAR_GDD50"  "APR_GDD50"  "MAY_GDD50"  "JUN_GDD50"  "JUL_GDD50"
## [21] "AUG_GDD50"  "SEP_GDD50"  "OCT_GDD50"  "NOV_GDD50"  "DEC_GDD50"
## [26] "ANN_GDD50"  "X"          "Y"

rm(ne.coords)
```

Task 12 : Change the coördinate names to show that these are metric Eastings and Northings. •

We find the column numbers with the current coördinate names with the `which` function.

```
names(ne.df)[which(names(ne.df) == "X")] <- "E"
names(ne.df)[which(names(ne.df) == "Y")] <- "N"
names(ne.df)

##  [1] "STATION_ID" "STATE"      "STATION_NA" "LATITUDE_D" "LONGITUDE_"
##  [6] "ELEVATION_" "OID_"       "COOP_ID"    "STATE_1"    "STN_NAME"
## [11] "LAT_DD"     "LONG_DD"    "ELEV_FT"    "JAN_GDD50"  "FEB_GDD50"
## [16] "MAR_GDD50"  "APR_GDD50"  "MAY_GDD50"  "JUN_GDD50"  "JUL_GDD50"
## [21] "AUG_GDD50"  "SEP_GDD50"  "OCT_GDD50"  "NOV_GDD50"  "DEC_GDD50"
## [26] "ANN_GDD50"  "E"          "N"
```

3 State boundaries

For display purposes we'd like to see the State boundaries of the four selected states. These are not used in the analysis, only in map display, so this section can be omitted if you are only interested in the analysis.

The US Census Bureau provides various cartographic boundary shapefiles[11] at different generalized scales to be used in small-scale mapping. This in-

[11] https://www.census.gov/geographies/mapping-files.html

cludes State boundaries[12]. Since we are only mapping four States, we select the largest available scale, namely 1:500 000.

Task 13 : Download the State boundaries shapefile for the USA from the US Census Bureau, save in your working directory, and unpack the compressed file into a subdirectory. The latest version (as of December 2023) is from 2022, in various resolutions. We choose the finest resolution, 1:500k design scale. •

The downloaded compressed file is named `cb_2022_us_state_500k.zip`.

Task 14 : Read the State boundaries into R as an `sf` object and select only the four states of interest in this analysis, •

Assuming you've put the State boundary file into the same directory as the NE weather stations and unpacked it, the shapefile is read in using `st_read`:

```
state <- st_read(dsn="./cb_2022_us_state_500k", "cb_2022_us_state_500k"
                 , quiet = TRUE)
print(state)
```

```
## Simple feature collection with 56 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -179.1467 ymin: -14.5487 xmax: 179.7785 ymax: 71.38782
## Geodetic CRS:  NAD83
## First 10 features:
##    STATEFP  STATENS    AFFGEOID GEOID STUSPS         NAME LSAD
## 1       35 00897535 0400000US35    35     NM   New Mexico   00
## 2       46 01785534 0400000US46    46     SD South Dakota   00
## 3       06 01779778 0400000US06    06     CA   California   00
## 4       21 01779786 0400000US21    21     KY     Kentucky   00
## 5       01 01779775 0400000US01    01     AL      Alabama   00
## 6       13 01705317 0400000US13    13     GA      Georgia   00
## 7       05 00068085 0400000US05    05     AR     Arkansas   00
## 8       42 01779798 0400000US42    42     PA Pennsylvania   00
## 9       29 01779791 0400000US29    29     MO     Missouri   00
## 10      08 01779779 0400000US08    08     CO     Colorado   00
##           ALAND      AWATER                       geometry
## 1  314198573403   726463825 MULTIPOLYGON (((-109.0502 3...
## 2  196341552329  3387681983 MULTIPOLYGON (((-104.0579 4...
## 3  403673617862 20291712025 MULTIPOLYGON (((-118.6044 3...
## 4  102266581101  2384240769 MULTIPOLYGON (((-89.40565 3...
## 5  131185042550  4582333181 MULTIPOLYGON (((-88.05338 3...
## 6  149486268417  4418716153 MULTIPOLYGON (((-81.27939 3...
## 7  134660743067  3121974727 MULTIPOLYGON (((-94.61792 3...
## 8  115882119641  3397575101 MULTIPOLYGON (((-80.51989 4...
## 9  178052253239  2487526202 MULTIPOLYGON (((-95.77355 4...
## 10 268418736192  1185778676 MULTIPOLYGON (((-109.0603 3...
```

The summary shows that this file already has a CRS defined, it came with the shapefile. The bounding box covers pretty much the whole world E–W (note the USA crosses the international date line in the Aleutian Islands, Alaska), and from about 14°S to about 71°N.

We see that the `STUSPS` field contains the State abbreviations, so we use this to select the States of interest, using the `%in%` set membership operator.

[12] https://www.census.gov/geographies/mapping-files/time-series/geo/cartographic-boundary.html

```
dim(state)

## [1] 56 10

state.ne <- state[state$STUSPS %in% c('NY','NJ', 'PA', 'VT'),]
dim(state.ne)

## [1]  4 10

st_bbox(state.ne)

##      xmin      ymin      xmax      ymax
## -80.51989  38.92852 -71.46456  45.01666
```

The bounding box has been restricted to surround the extreme coördinates of these four States.

Task 15 : Project into the Albers projection used in this project. •

We transform to this project's CRS using `st_transform`. To ensure consistency, we obtain the CRS from the object we projected in §2.4, above, with the `st_crs` function.

```
state.ne.m <- st_transform(state.ne, st_crs(ne.m))
str(state.ne.m)

## Classes 'sf' and 'data.frame': 4 obs. of  10 variables:
##  $ STATEFP : chr  "42" "36" "50" "34"
##  $ STATENS : chr  "01779798" "01779796" "01779802" "01779795"
##  $ AFFGEOID: chr  "0400000US42" "0400000US36" "0400000US50" "0400000US34"
##  $ GEOID   : chr  "42" "36" "50" "34"
##  $ STUSPS  : chr  "PA" "NY" "VT" "NJ"
##  $ NAME    : chr  "Pennsylvania" "New York" "Vermont" "New Jersey"
##  $ LSAD    : chr  "00" "00" "00" "00"
##  $ ALAND   : num  1.16e+11 1.22e+11 2.39e+10 1.90e+10
##  $ AWATER  : num  3.40e+09 1.93e+10 1.03e+09 3.53e+09
##  $ geometry:sfc_MULTIPOLYGON of length 4; first list element: List of 1
##   ..$ :List of 1
##   .. ..$ : num [1:1841, 1:2] -380307 -380258 -380157 -379965 -379965 ...
##   ..- attr(*, "class")= chr [1:3] "XY" "MULTIPOLYGON" "sfg"
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA
##   ..- attr(*, "names")= chr [1:9] "STATEFP" "STATENS" "AFFGEOID" "GEOID" ...

st_bbox(state.ne.m)

##      xmin      ymin      xmax      ymax
## -387135.2 -396311.1  357707.2  288684.4
```

4 Prediction grids

A natural question is how target variables vary over the entire study area, not just at the observation points – this is the interpolation (or extrapolation) problem. For regional studies we want to predict and visualize over the entire area; i.e., we want to produce a **map** of the target variable. For this we need the predictors used in the models at a set of **grid cells** covering the **whole study area**. In this tutorial these predictors are the metric coördinates (E and N) in our defined CRS, and elevation. We could create an empty grid with coördinates with the `expand.grid` or `st_sample` commands but we also need the elevation for the predictive models which use this as a

predictor.

4.1 Digital Elevation Model (DEM)

For the elevation, we can use the digital elevation model (DEM) produced by the Shuttle Radar Topography mission[13]. This mission has mapped most of the Earth at a nominal 1 arc-second (≈30 m) resolution. Of course, for regional climate this fine resolution is not necessary, a 4 km resolution is sufficient.

I have processed the ASCII format file to restrict the area to just the lower 48 USA states, downscaled it to 4 km resolution, created a `SpatRaster` object defined by the `terra` package, and saved it to the data directory as a GeoTIFF[14] file.

> **Note:** The `terra` package[15] is a replacement for the `raster` package from the same author (Robert Hijmans[16]). Its main advantage is that raster files are not read into memory as a whole, but only as needed. It also has vector data types.
>
> Hijmans has good online tutorials on Spatial Data Science[17].

Task 16 : Load the `SpatRaster` grid object `dem.ne.m` from the provided GeoTIFF file. •

The `rast` function from the `terra` package is used to load `SpatRaster` objects from files.

```
dem.ne.m <- terra::rast("./dem_ne_4km.tif")
class(dem.ne.m)
str(dem.ne.m)
summary(values(dem.ne.m))
```

```
## [1] "SpatRaster"
## attr(,"package")
## [1] "terra"
##   srtm_1km_48
##  Min.   :-7.046e+09
##  1st Qu.: 3.050e+02
##  Median : 7.880e+02
##  Mean   :-8.152e+08
##  3rd Qu.: 1.374e+03
##  Max.   : 4.156e+03
##  NA's   :3376
```

Some elevations are negative, which seems unlikely for the northeastern USA; these are likely the result of the global algorithm. These must be non-negative, otherwise the square root operation, as applied to the elevation in some models, fails.

The `pmax` "parallel maximum" function applies `max` element-wise along a vector, such as the values of a `SpatRaster`.

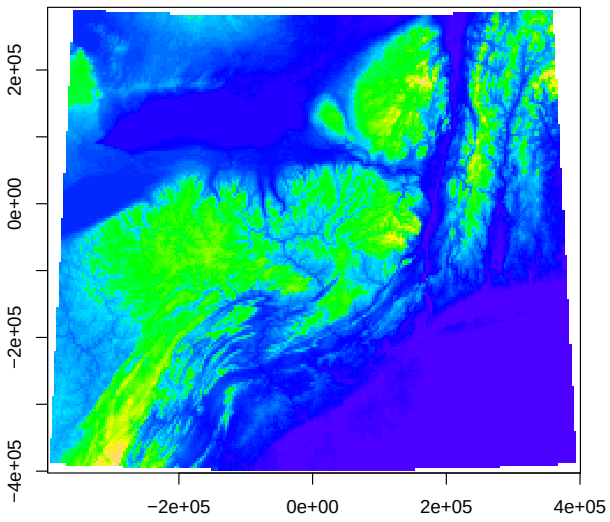
[13] http://www2.jpl.nasa.gov/srtm/
[14] https://www.ogc.org/standard/geotiff/
[15] https://rspatial.github.io/terra/index.html
[16] https://desp.ucdavis.edu/people/robert-hijmans
[17] https://rspatial.org/index.html

```
values(dem.ne.m) <- pmax(0, values(dem.ne.m))
summary(values(dem.ne.m))
```

```
##   srtm_1km_48
##  Min.   :   0.0
##  1st Qu.: 304.6
##  Median : 788.0
##  Mean   : 878.4
##  3rd Qu.:1374.0
##  Max.   :4156.3
##  NA's   :3376
```

Display the grid, using the `topo.colors` "topographic colours" palette often applied to elevations.

```
terra::image(dem.ne.m, col = topo.colors(64))
```



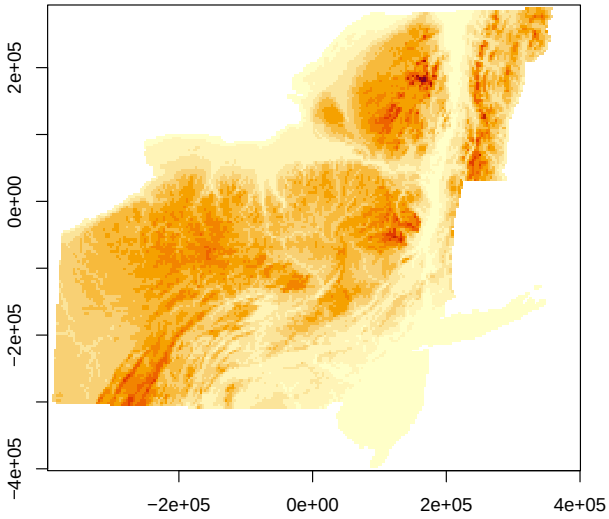
4.2 Masking to study area

The imported tile covers the entire bounding box of the four States, and includes areas of ocean, neighbouring States, and our friends of the Great White North. It can be interesting to see how our predictive models extrapolate to those areas, but we really should only predict over the area where we have observations, i.e., the four States. So, we **mask** the DEM to have valid elevation values only in that area.

Task 17 : Mask the DEM to just the four States with observations. •

For this, we use the States polygons as a mask.

```
dem.ne4.m <- terra::mask(dem.ne.m, state.ne.m)
terra::image(dem.ne4.m)
```



4.3 DEM as spatial points

For some geostatistical operations we need to represent the grid as a set of points, not pixels.

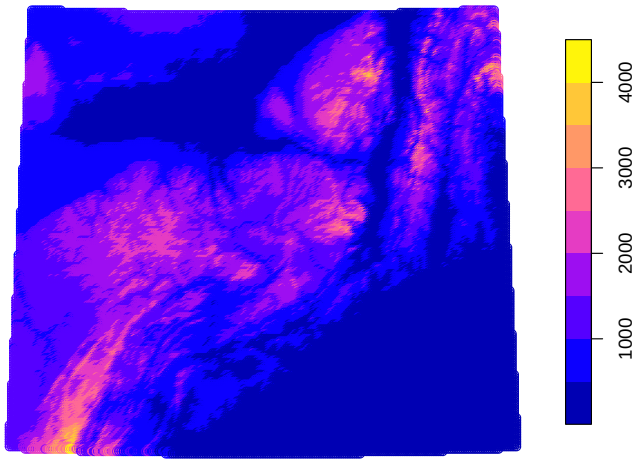
Task 18 : Convert the two grids to `sf` objects with the appropriate names. •

The `st_as_sf` method makes this conversion, but the `SpatRaster` must first be converted in to a "vector" geometry in `terra`, using the `vector` function. This is because `terra` distinguished between vector and raster geometries, and `st_as_sf` can only work on vector geometries. For this first conversion we use `terra::as.points`

```
# convert to `sf` with coordinate geometry
class(dem.ne.m)

## [1] "SpatRaster"
## attr(,"package")
## [1] "terra"

dim(dem.ne.m)

## [1] 188 231   1

v <- as.points(dem.ne.m)
class(v)

## [1] "SpatVector"
## attr(,"package")
## [1] "terra"

dem.ne.m.sf <- st_as_sf(v)
plot(dem.ne.m.sf, main = "4km DEM, feet a.s.l.")
dim(dem.ne.m.sf)
```

```
## [1] 40052     2

class(dem.ne.m.sf)

## [1] "sf"         "data.frame"

names(dem.ne.m.sf)[1] <- "ELEVATION_" # to match with dataframe
summary(dem.ne.m.sf)

##    ELEVATION_              geometry
##  Min.   :   0.0   POINT        :40052
##  1st Qu.: 304.6   epsg:NA      :    0
##  Median : 788.0   +proj=aea ...:    0
##  Mean   : 878.4
##  3rd Qu.:1374.0
##  Max.   :4156.3

rm(v)
```

4km DEM, feet a.s.l.



Note that this already has the correct CRS, inherited from the `SpatRaster`.

4.4 DEM as dataframe

To predict over this grid with the results of some models (e.g., OLS trend surface) we need it to be an R data frame object, with fields for the factors used in the models, namely Northing, Easting and elevation. These must have the same names as in the model formulations.

Task 19 : Convert the two grids to dataframes with the appropriate names. •

The `as.data.frame` method applied to a `SpatRaster` makes this conversion. The `xy` argument adds the coördinates of each grid cell's centre.

```
# convert to `sf` with coordinate geometry
dem.ne.m.df <- as.data.frame(dem.ne.m, xy = TRUE)
dem.ne4.m.df <- as.data.frame(dem.ne4.m, xy = TRUE)
str(dem.ne.m.df)

## 'data.frame': 40052 obs. of  3 variables:
##  $ x          : num  -356674 -353224 -349774 -346324 -342874 ...
##  $ y          : num  288086 288086 288086 288086 288086 ...
##  $ srtm_1km_48: num  574 574 574 574 574 ...

# make the field names compatible with the point dataset
names(dem.ne.m.df) <- c("E", "N", "ELEVATION_")
names(dem.ne4.m.df) <- c("E", "N", "ELEVATION_")
```

5 Environmental covariates

5.1 Distance to water bodies

Water bodies can influence local climate due to the high heat capacity of water and the temperature of water coming from upstream or out in the ocean.

5.1.1 Distance to the Great Lakes

The two Great Lakes in this region (Erie and Ontario) may have a local climate effect, because of the high heat capacity of the lake water.

A Great Lakes shapefile has been created by Natural Earth[18] "a public domain map dataset available at 1:10m, 1:50m, and 1:110 million scales". We selected and downloaded the 1:10m product[19] as `ne_10m_lakes.zip`, imported into QGIS 3, selected lakes Erie and Ontario, and exported as an OGC Geopackage[20].

Task 20 : Load this geopackage, reproject into the project CRS. •

```
lakes <- st_read("./LakesOntarioErie.gpkg")
```

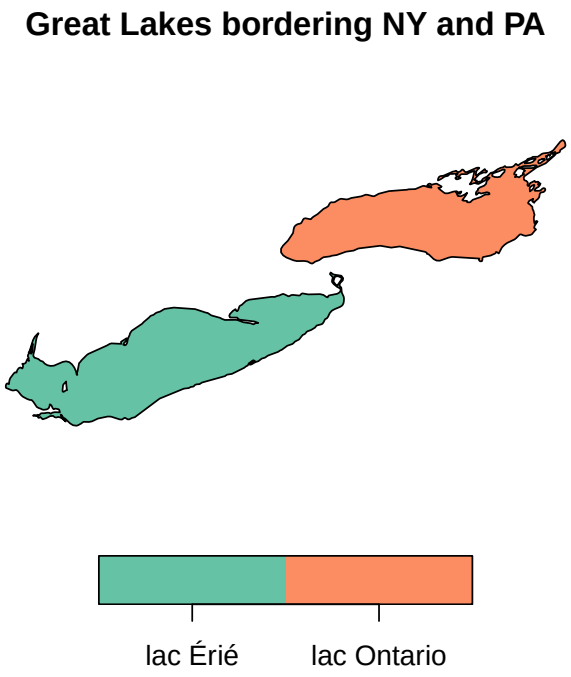
```
## Reading layer `LakesOntarioErie' from data source
##   `/Users/rossiter/Library/Mobile Documents/com~apple~CloudDocs/Documents/data/edu/MappingRegionalC
##   using driver `GPKG'
## Simple feature collection with 2 features and 37 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -83.46621 ymin: 41.3936 xmax: -75.77021 ymax: 44.50444
## Geodetic CRS:  WGS 84
```

```
st_crs(lakes)$proj4string

## [1] "+proj=longlat +datum=WGS84 +no_defs"

lakes <- st_transform(lakes, crs=st_crs(ne.m))
plot(lakes["name_fr"], main = "Great Lakes bordering NY and PA")
```

[18] http://www.naturalearthdata.com

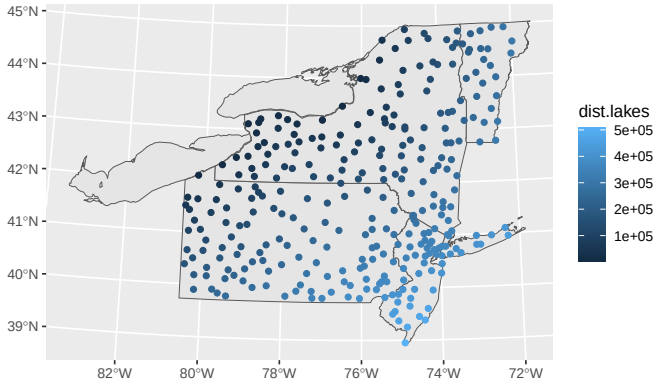
[19] https://www.naturalearthdata.com/http//www.naturalearthdata.com/download/10m/physical/ne_10m_lakes.zip

[20] http://www.geopackage.org



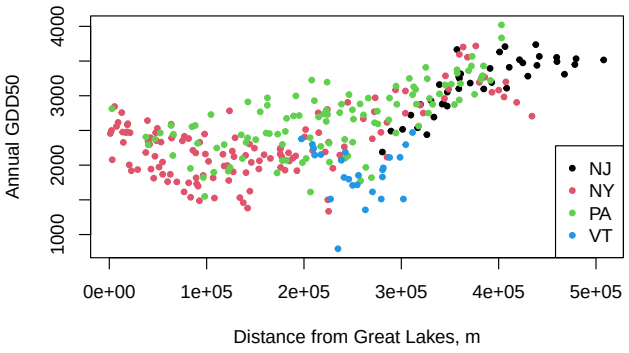
Task 21 : Calculate the distance between the lakes and the stations, add this attribute to the points, and show the spatial distribution. •

```
dd <- sf::st_distance(lakes, ne.m)
dim(dd) # distance to each lake

## [1]   2 305

ne.m$dist.lakes <- apply(dd, 2, min)
summary(ne.m$dist.lakes)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1079  118052  223043  219294  316853  507907

ne.df$dist.lakes <- ne.m$dist.lakes
ggplot(ne.m) +
  geom_sf(data=lakes) +
  geom_sf(data=state.ne.m) +
  geom_sf(aes(color=dist.lakes))
```



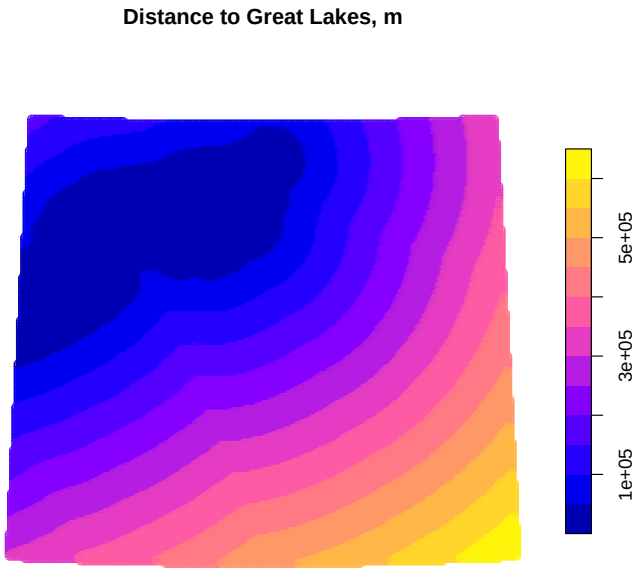
Plot the relation between the target variable and this covariable:

```
plot(ne.m$dist.lakes, ne.m$ANN_GDD50, col=ne.m$STATE, pch=20,
     xlab="Distance from Great Lakes, m", ylab="Annual GDD50")
legend("bottomright", legend=levels(ne.df$STATE), pch=20, col=1:4)
```



Task 22 : Calculate the distance between the lakes and the grid, and add this attribute to the grid. •

```
dd <- st_distance(lakes, dem.ne.m.sf)
dem.ne.m.sf$dist.lakes <- apply(dd, 2, min)
dem.ne.m.df$dist.lakes <- dem.ne.m.sf$dist.lakes
plot(dem.ne.m.sf["dist.lakes"],
     main = "Distance to Great Lakes, m")
```



5.1.2 Distance to the Atlantic Ocean

The Atlantic Ocean has a local cooling effect along the shore, especially on Long Island.

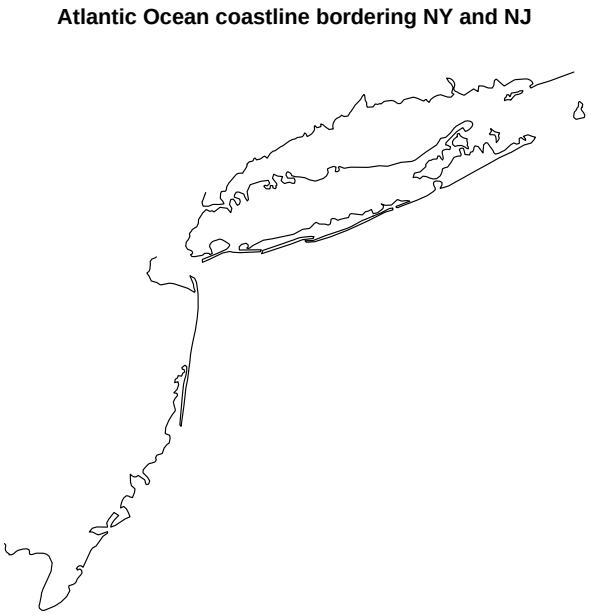
A US shoreline shapefile has been created by Natural Earth . We selected and downloaded the 1:10m product[21] as `ne_10m_coastline.zip`, imported into QGIS 3, manually split the coastline and selected the portion bordering our area, and exported as an OGC Geopackage.

Task 23 : Load this geopackage, reproject into the project CRS. •

```
coast <- st_read("./AtlanticCoastLine.gpkg", quiet = TRUE)
```

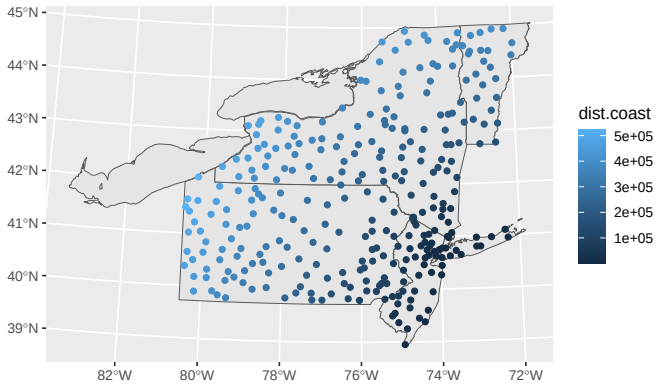
```
names(coast)

## [1] "featurecla" "scalerank"  "min_zoom"   "geom"

coast <- st_transform(coast, ne.crs)
plot(coast["featurecla"], key.pos = 1,
     col = "black",
     main = "Atlantic Ocean coastline bordering NY and NJ")
```

[21] https://www.naturalearthdata.com/http//www.naturalearthdata.com/download/10m/physical/ne_10m_coastline.zip



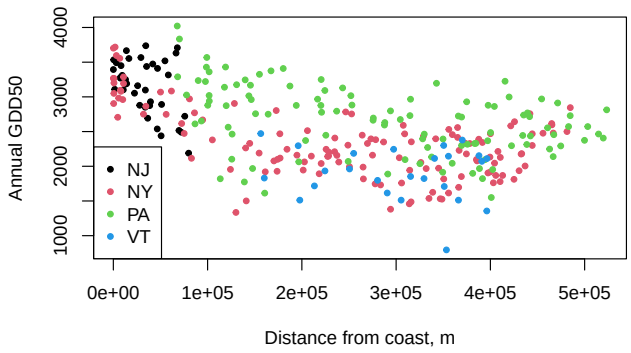
Task 24 : Compute distance to stations and grid cells, save in objects, and show the spatial distribution. •

```
dd <- st_distance(coast, ne.m)
ne.m$dist.coast <- apply(dd, 2, min)
ne.df$dist.coast <- ne.m$dist.coast
ggplot(ne.m) +
  geom_sf(data=lakes) +
  geom_sf(data=state.ne.m) +
  geom_sf(aes(color=dist.coast))
```



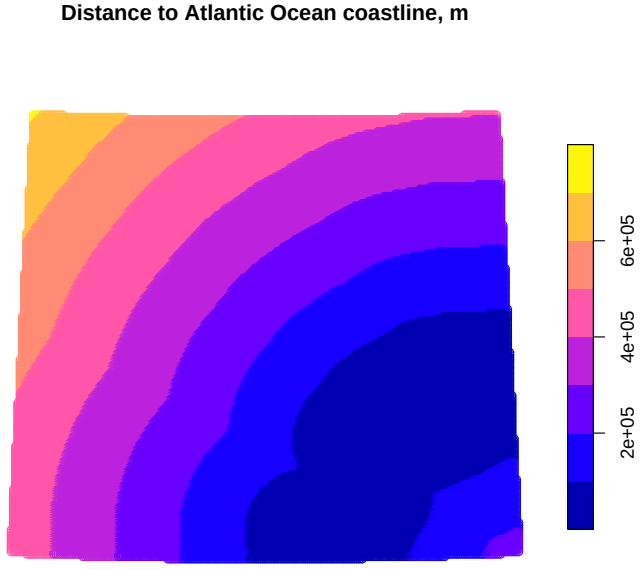
Plot the relation between the target variable and this covariable.

```
plot(ne.m$dist.coast, ne.m$ANN_GDD50, col=ne.m$STATE, pch=20,
     xlab="Distance from coast, m", ylab="Annual GDD50")
legend("bottomleft", legend=levels(ne.df$STATE), pch=20, col=1:4)
```



Task 25 : Calculate the distance between the coast and the grid, and add this attribute to the grid. •

```
dd <- st_distance(coast, dem.ne.m.sf)
dem.ne.m.sf$dist.coast <- apply(dd, 2, min)
dem.ne.m.df$dist.coast <- dem.ne.m.sf$dist.coast
plot(dem.ne.m.sf["dist.coast"],
     main = "Distance to Atlantic Ocean coastline, m")
```



5.2 Terrain

Local terrain may influence climate. For example, narrow valleys in the Finger Lakes regions are known as "frost pockets" and often have morning ground fogs in spring and early summer.

For a good introduction to terrain analysis, see Wilson and Gallant [6].

We have selected two terrain indices that have a relation with local topography, which is where we expect any local effects on climate.

Terrain covariates are all derived from the base DEM, in this example a fairly coarse $\approx 4 \times 4$ km grid. Among the programs to compute terrain covariates, SAGA GIS[22] is one of the most comprehensive. The SAGA algorithms are also available in QGIS 3[23].

Task 26 : Import these coverages into R. •

We specify the full file names for the grid data, and the associated files are automatically consulted as needed.

5.2.1 Multiresolution Index of Valley Bottom Flatness (MRVBF)

The **multiresolution index of valley bottom flatness** (MRVBF) [2] identifies valley bottoms based on their topographic signature as flat low-lying areas, at increasingly-broad scales, and combines these into a single index.

One parameter for MRVBF was adjusted from defaults: initial threshold for slope 4% (not 16%) because of the coarse resolution of the DEM. The others were default: lowness threshold 0.4, upness threshold 0.35, shape for slope 4, shape for elevation 3, maximum resolution 100%; not classified (i.e., retain the continuous value). The result was saved as a SAGA file set; these have extension `.sdat`for the data (raster), `.sgrd` for the grid parameters, `.prj` for the projection information, and `.mgrd` for the grid metadata, i.e., the processing steps. This latter is useful to confirm how the raster was created.

Fortunately, the `rast` function understands SAGA files.

Load the MRVBF map:

```
mrvbf <- rast("./mrvbf_ne_4km.sdat")
print(mrvbf)
```

```
## class       : SpatRaster
## dimensions  : 202, 231, 1  (nrow, ncol, nlyr)
## resolution  : 3450, 3450  (x, y)
## extent      : -396349.3, 400600.7, -402583.4, 294316.6  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=aea +lat_0=42.5 +lon_0=-76 +lat_1=39 +lat_2=44 +x_0=0 +y_0=0 +ellps=WGS84 +units
## source      : mrvbf_ne_4km.sdat
## name        : mrvbf_ne_4km
## min value   :     0.000000
## max value   :     3.991617
```

```
plot(mrvbf, asp=1, col=sp::bpy.colors(32), main = "MRVBF")
st_crs(mrvbf)$proj4string == st_crs(ne.m)$proj4string

## [1] TRUE
```

[22] http://saga-gis.org
[23] https://www.qgis.org

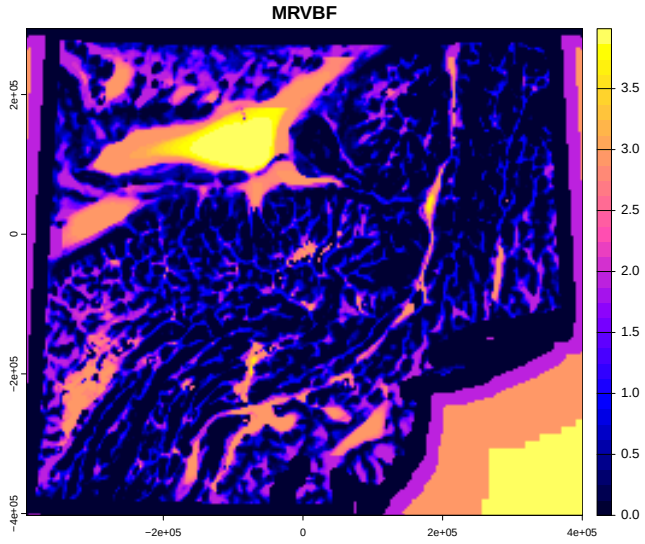


Figure 1: Multiresolution Index of Valley Bottom Flatness

5.2.2 Terrain Ruggedness Index (TRI)

The **terrain ruggedness index** (TRI) [3] expresses heterogeneity. It is a simple calculation: the sum change in ele‘vation between a grid cell and its eight neighbours. Both SAGA and QGIS implement this. The SAGA version is modified to allow a radius of more than one neighbour, and optionally to weight the elevation changes by inverse distance.

Since we are interested in medium-range effects, we chose a radius of 3 cells (3450 m $\times$ 3 $\approx$ 10 km) and inverse-distance square weighting. These are empirical choices, and you could experiment with others.

Load the terrain ruggedness map.

```
tri3 <- rast("./dem_ne_4km_TRI3_IDW2.sdat")
print(tri3)
```

```
## class       : SpatRaster
## dimensions  : 202, 231, 1  (nrow, ncol, nlyr)
## resolution  : 3450, 3450  (x, y)
## extent      : -396349.3, 400600.7, -402583.4, 294316.6  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=aea +lat_0=42.5 +lon_0=-76 +lat_1=39 +lat_2=44 +x_0=0 +y_0=0 +ellps=WGS84 +units
## source      : dem_ne_4km_TRI3_IDW2.sdat
## name        : dem_ne_4km_TRI3_IDW2
## min value   :               0.000
## max value   :             371.698
```

```
# replace NA with 0's
length(ix <- which(is.na(values(tri3))))

## [1] 8155

if (length(ix) > 0) { values(tri3)[ix] <- 0 }
plot(tri3, asp=1, col=topo.colors(16), main = "TRI")
st_crs(tri3)$proj4string == st_crs(ne.m)$proj4string
```

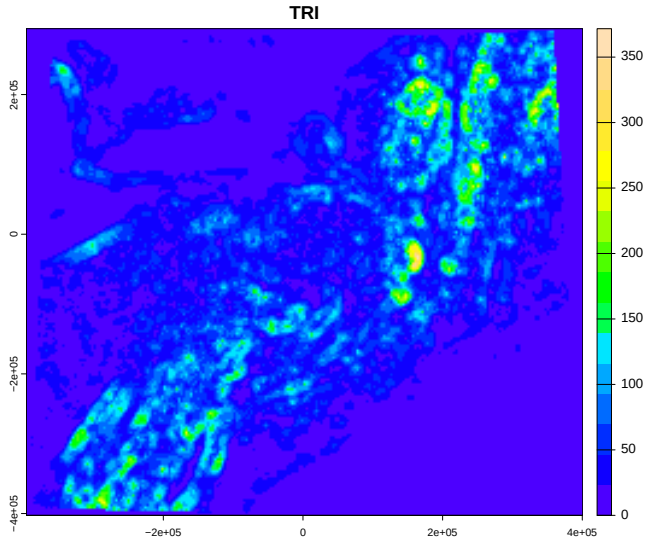


Figure 2: Terrain Ruggedness Index

```
## [1] TRUE
```

5.3 Population density

The reason for considering this as a covariable is that urban areas affect the local climate (typically making it warmer), and that very sparsely-populated rural areas (typically cooler, more precipitation as snow).

The Socioeconomic Data and Applications Center (SEDAC) at Columbia University (USA)[24] is a data centre in NASA's Earth Observing System Data and Information System (EOSDIS). One product available there is the Gridded Population of the World (GPW), v4[25] for several dates. We select the year 2000, since our climate data is 1971-2000. There are two grid resolutions: 2.5' (approx. 5 km) and 15' (approx. 30 km) resolutions; we try them both, because they give two scales of possible population effects. The units of measure in both cases are units: persons per km^2.

The GeoTIFF files `gpw_v4_population_density_rev11_2000_15_min.tif` and `gpw_v4_population_density_rev11_2000_2pt5_min.tif` were downloaded from SEDAC. These are gridded dataset and can be read and manipulated by the `terra` package.

Another interesting covariate, which we do not use here, is night light intensity, as a surrogate for population density and industrial activity. This is available from NASA[26].

[24] https://sedac.ciesin.columbia.edu
[25] https://sedac.ciesin.columbia.edu/data/set/gpw-v4-population-density-rev11
[26] https://www.earthdata.nasa.gov/learn/backgrounders/nighttime-lights

This predictor is highly-skewed, which will distort any method based on variances, such as regression trees. So, we transform these to logarithms, adding 1 to 0 values to have a valid base-10 logarithm.

Load the population density maps, transforming to base-10 logarithms.

```
pop2pt5.world <- log10(rast(
  "./gpw_v4_population_density_rev11_2000_2pt5_min.tif")+1)
pop15.world <- log10(rast(
  "./gpw_v4_population_density_rev11_2000_15_min.tif")+1)
print(pop2pt5.world)
print(pop15.world)
```

```
## class       : SpatRaster
## dimensions  : 4320, 8640, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : gpw_v4_population_density_rev11_2000_2pt5_min
## name        : gpw_v4_population_density_rev11_2000_2pt5_min
## min value   :                                     0.000000
## max value   :                                     4.720518
## class       : SpatRaster
## dimensions  : 720, 1440, 1  (nrow, ncol, nlyr)
## resolution  : 0.25, 0.25  (x, y)
## extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : gpw_v4_population_density_rev11_2000_15_min
## name        : gpw_v4_population_density_rev11_2000_15_min
## min value   :                                   0.000000
## max value   :                                   4.371986
```

These are in geographic coordinates, i.e., unprojected, and cover the whole globe.

Restrict the rasters to the bounding box of the study area, which we have from the DEM used in the exercise. For this the unprojected version of the bounding box must be used.

```
print(st_bbox(ne))

##   xmin   ymin   xmax   ymax
## -80.47  38.95 -71.98  44.93

pop15.ne <- terra::crop(pop15.world, st_bbox(ne))
pop2pt5.ne <- terra::crop(pop2pt5.world, st_bbox(ne))
names(pop15.ne) <- "pop15"
names(pop2pt5.ne) <- "pop2pt5"
print(pop15.ne)

## class       : SpatRaster
## dimensions  : 24, 34, 1  (nrow, ncol, nlyr)
## resolution  : 0.25, 0.25  (x, y)
## extent      : -80.5, -72, 39, 45  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : gpw_v4_population_density_rev11_2000_15_min
## name        :    pop15
## min value   : 0.000000
## max value   : 4.053805

print(pop2pt5.ne)

## class       : SpatRaster
## dimensions  : 143, 203, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
```

```
## extent      : -80.45833, -72, 38.95833, 44.91667  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : gpw_v4_population_density_rev11_2000_2pt5_min
## name        :  pop2pt5
## min value   : 0.000000
## max value   : 4.489077
```

Match the projection of the DEM and stations. This requires a **reprojection** into another grid. Here we want to keep the approximate resolution, since the value is a density per pixel.

```
pop15.ne.m <- project(pop15.ne, ne.crs$proj4string)
print(pop15.ne.m) # resolution approx. 23.3 x 23.3  km

## class       : SpatRaster
## dimensions  : 29, 32, 1  (nrow, ncol, nlyr)
## resolution  : 23272.99, 23272.99  (x, y)
## extent      : -389643.2, 355092.3, -387890.1, 287026.5  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=aea +lat_0=42.5 +lon_0=-76 +lat_1=39 +lat_2=44 +x_0=0 +y_0=0 +ellps=WGS84 +units
## source(s)   : memory
## name        :   pop15
## min value   : 0.00000
## max value   : 3.88183

pop2pt5.ne.m <- project(pop2pt5.ne, ne.crs$proj4string)
print(pop2pt5.ne.m)  # approx. 3.9 x 3.9 km

## class       : SpatRaster
## dimensions  : 173, 189, 1  (nrow, ncol, nlyr)
## resolution  : 3879.738, 3879.738  (x, y)
## extent      : -386276.8, 346993.7, -393578.4, 277616.3  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=aea +lat_0=42.5 +lon_0=-76 +lat_1=39 +lat_2=44 +x_0=0 +y_0=0 +ellps=WGS84 +units
## source(s)   : memory
## name        :  pop2pt5
## min value   : 0.000000
## max value   : 4.418226
```

Display the two maps on the same visual scale:

```
ymax <- max(values(pop15.ne), values(pop2pt5.ne), na.rm = TRUE)
plot(pop15.ne.m, main = "15' cells",
     col=rev(heat.colors(64)), range = c(0, ymax))
plot(pop2pt5.ne.m, main = "2.5' cells",
     col=rev(heat.colors(64)), range = c(0, ymax))
```

Notice how the maximum density is higher for the 2.5° product, because of the concentration of population in NYC.

6 Add covariables to the dataset

The overlay operation on `SpatRaster` requires the point object to also be from the `terra` package. For point objects these are `SpatVector` objects. Objects of class `sf` can be converted to `SpatVector` with the `vect` method of the `terra` package.

6.1 At the climate stations

```
ix.vals <- terra::extract(mrvbf, vect(ne.m))
eval(parse(text=paste("ne.m$mrvbf <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("ne.df$mrvbf <- ix.vals$",
                      names(ix.vals)[2], sep="")))
ix.vals <- terra::extract(tri3, vect(ne.m))
```

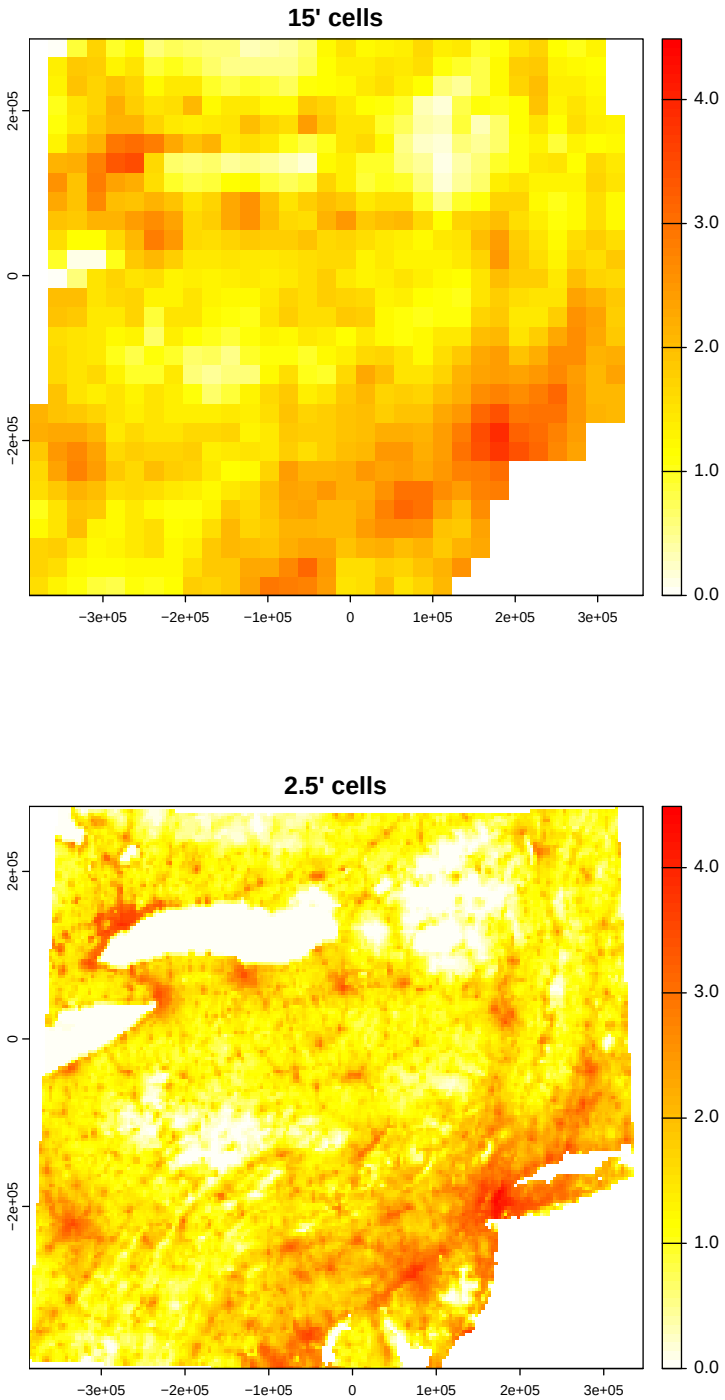


Figure 3: Log10(Population density) per square km

```
eval(parse(text=paste("ne.m$tri3 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("ne.df$tri3 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
ix.vals <- terra::extract(pop15.ne.m, vect(ne.m))
eval(parse(text=paste("ne.m$pop15 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("ne.df$pop15 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
ix.vals <- terra::extract(pop2pt5.ne.m, vect(ne.m))
eval(parse(text=paste("ne.m$pop2pt5 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("ne.df$pop2pt5 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
rm(ix.vals)
```

Check for cells with NA's, change these to 0 (no population).

```
length(ix <- which(is.na(ne.m$pop15)))

## [1] 5

if (length(ix) > 0) {
  ne.m[ix,"pop15"] <- 0; ne.df[ix,"pop15"] <- 0
}
length(ix <- which(is.na(ne.m$pop2pt5)))

## [1] 6

if (length(ix) > 0) {
  ne.m[ix,"pop2pt5"] <- 0; ne.df[ix,"pop2pt5"] <- 0
}
```

6.2 Over the prediction grid

Add the covariables to the grid. For the population densities, note these are *not* the populations of the cell, rather, the density of the area surrounding and including it.

```
ix.vals <- terra::extract(mrvbf, vect(dem.ne.m.sf))
eval(parse(text=paste("dem.ne.m.sf$mrvbf <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("dem.ne.m.df$mrvbf <- ix.vals$",
                      names(ix.vals)[2], sep="")))
ix.vals <- terra::extract(tri3, vect(dem.ne.m.sf))
eval(parse(text=paste("dem.ne.m.sf$tri3 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("dem.ne.m.df$tri3 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
ix.vals <- terra::extract(pop15.ne.m, vect(dem.ne.m.sf))
eval(parse(text=paste("dem.ne.m.sf$pop15 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("dem.ne.m.df$pop15 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
ix.vals <- terra::extract(pop2pt5.ne.m, vect(dem.ne.m.sf))
eval(parse(text=paste("dem.ne.m.sf$pop2pt5 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
eval(parse(text=paste("dem.ne.m.df$pop2pt5 <- ix.vals$",
                      names(ix.vals)[2], sep="")))
summary(dem.ne.m.sf)

##    ELEVATION_              geometry       dist.lakes
##  Min.   :   0.0   POINT        :40052   Min.   :     0
##  1st Qu.: 304.6   epsg:NA      :    0   1st Qu.: 84666
##  Median : 788.0   +proj=aea ...:    0   Median :222394
##  Mean   : 878.4                         Mean   :228099
##  3rd Qu.:1374.0                         3rd Qu.:346447
```

```
##  Max.   :4156.3                         Max.   :648952
##
##    dist.coast           mrvbf               tri3
##  Min.   :     2.3   Min.   :0.000000   Min.   :  0.00
##  1st Qu.:126409.8   1st Qu.:0.008421   1st Qu.: 10.66
##  Median :276842.2   Median :0.380940   Median : 29.11
##  Mean   :275295.6   Mean   :0.947745   Mean   : 39.98
##  3rd Qu.:406957.9   3rd Qu.:1.787373   3rd Qu.: 57.30
##  Max.   :708701.1   Max.   :3.991617   Max.   :371.70
##
##      pop15          pop2pt5
##  Min.   :0.000   Min.   :0.000
##  1st Qu.:1.241   1st Qu.:0.904
##  Median :1.571   Median :1.304
##  Mean   :1.629   Mean   :1.341
##  3rd Qu.:1.978   3rd Qu.:1.772
##  Max.   :3.882   Max.   :4.418
##  NA's   :5520    NA's   :6782

rm(ix.vals)
```

Check for cells with NA's, change these to 0 (no population).

```
length(ix <- which(is.na(dem.ne.m.df$pop15)))

## [1] 5520

if (length(ix) > 0) {
  dem.ne.m.sf[ix,"pop15"] <- 0; dem.ne.m.df[ix,"pop15"] <- 0
}
length(ix <- which(is.na(dem.ne.m.df$pop2pt5)))

## [1] 6782

if (length(ix) > 0) {
  dem.ne.m.sf[ix,"pop2pt5"] <- 0; dem.ne.m.df[ix,"pop2pt5"] <- 0
}
```

7 Saving the dataset

We can save these objects in R's internal `RData` format, using the `save` function. These can be read into an R workspace with the `load` function, with the same object names as they were saved with.

Task 27 : Save the spatial and dataframe versions of the original and projected point datasets and the prediction grid as R objects. Also save the CRS we developed for this region. •

```
save(ne, ne.m, ne.df, state.ne, state.ne.m,
     dem.ne.m.sf, dem.ne.m.df, dem.ne4.m, ne.crs,
     file="./StationsDEM_covariates.RData")
```

These are now ready for modelling and prediction.

References

[1] Lev M Bugayevskiy and John Parr Snyder. *Map projections: a reference manual*. Taylor and Francis, 1995.

[2] John C. Gallant and Trevor I. Dowling. A multiresolution index of valley bottom flatness for mapping depositional areas. *Water Resources Research*, 39(12):ESG4–1 – ESG4–13, Dec 2003. doi: 10.1029/2002WR001426.

[3] Shawn J. Riley, Stephen D. DeGloria, and Robert Eliot. A terrain ruggedness that quantifies topographic heterogeneity. *Intermountain Journal of Sciences*, 5(1–4):23–27, 1999.

[4] John P. Snyder. *Map projections: a working manual*. USGS Professional Paper 1395. US Government Printing Office, 1987. URL https://pubs.er.usgs.gov/publication/pp1395.

[5] John P. Snyder and Philip M. Voxland. *An album of map projections*. USGS Professional Paper 1453. U.S. Geological Survey, 1989. URL https://pubs.er.usgs.gov/publication/pp1453.

[6] J P Wilson and J Gallant. *Terrain analysis : principles and applications*. Wiley & Sons, 2000. ISBN 0-471-32188-5.