Downloading and processing EPA pollution data

Version *0.1-1*

Paul L. Delamater and Andrew O. Finley

1 Access to EPA data

The Environmental Protection Agency (EPA) delivers monitored hazardous air pollutant data using a java-based internet applet called the Air Quality System Data Mart (AQS Data Mart). The EPA requires that users request access to the data here prior to downloading. Once granted access, the AQS Data Mart can be found at: `http://www.epa.gov/ttn/airs/aqsdatamart/access/interface.htm`. Unfortunately, computers with updated version of the Linux operating system may encounter problems using the applet as it uses an outdated java security exemption for authenticating the username and password. We used a Macintosh computer to access the AQS Data Mart and submit data queries, then downloaded and processed data using Linux computers.

2 Obtaining EPA data

The AQS Data Mart user's guide is available at:
`http://www.epa.gov/ttn/airs/aqsdatamart/documentation/index.htm`.

For data used in this example, we performed a Values Query. To define the spatial boundaries of the monitors included, enter values for Min/Max Latitude and Longitude or select the appropriate state, county, city, etc. For access to Criteria Pollutants (those regulated by the EPA), select the entry under Substance Type. Then, select the name of the particular pollutant under Substance Name. To define the type of data and the temporal range, click the Time and Measures tab. Enter values for the Begin Date and End Date and select the Static Name. Click "Submit" to send the query to the system. If query returns a small dataset, the results can be downloaded (copied and pasted) from the AQS interface by selecting "Download". If the query returns a large amount of data, the results must be downloaded from a data serving site. If multiple queries are necessary, note the query parameters and the Transaction ID number. This number can be used to download data using a **wget** call in bash. The following is an example of a bash script that downloads multiple files. In the example, the Transaction IDs are from 200, 201, 202... to 210.

```
#!/bin/bash
for z in $(seq 200 1 210)
do
wget https://oasext.epa.gov/AQDM/AQDM_RR_"\$z".zip
done
```

3 Prepare data for import to R

Once the data have been downloaded and unzipped, the resulting .xml file must be slightly modified before it can be read into R. The follwing text must be completely removed from the header section of the .xml file for the R code included in this document to perform correctly.

Text 1 to remove:
<!--To produce a CSV file change the following xml-stylesheet instruction to a comment and uncomment the xml-stylesheet instruction after it.-->

Text 2 to remove:
<!--<?xml-stylesheet type="text/xsl" href="http://www.epa.gov/ttn/airs /aqsdatamart/documentation/geoCoded2CSV.xsl"?>-->

For one file, this can be accomplished by using a text editor such as gedit. For multiple files, the following bash script will remove the text for all files in a particular folder.

```
#!/bin/bash

sed -i 's|<!--To produce a CSV file change the following xml-\
stylesheet instruction to a comment and uncomment the xml-\
stylesheet instruction after it.-->||' *.xml

sed -i 's|<!--<?xml-stylesheet type="text/xsl" href="http://\
www.epa.gov/ttn/airs/aqsdatamart/documentation/geoCoded2CSV.xsl\
"?>-->||' *.xml
```

4 Importing and processing EPA data with R

We will make use of the **XML** library in R. Before starting, please be advised that using R to read in XML data utilizes a considerable amount of RAM and system resources. Large files (e.g., statewide daily data for an entire year) may need to be redownloaded and processed using smaller spatial and/or temporal boundaries if the memory requirements are too demanding for the system. Later, we will be using spatial functions in R to view data and create a quick pollution surface. These libraries include **rgdal**, **sp**, **spdep**, and **MBA**.

The files used in the following example are daily measurements (24 hour mean values) of the 8 criteria air pollutants for the state of Michigan during January of 2010. In the first section of code, we read the data in for one .xml file (Carbon Monoxide) and export it in a format that is easier to read and process. The code can be modified to process multiple files as a function or in a loop. It also contains generic calls to the .xml files in an effort to be transferable for use with data from other sources.

```
> x <- xmlRoot(xmlTreeParse("data/xml/AQDM_RR_206321.xml"))
> master <- xmlSApply(x, function(z) xmlSApply(z, xmlValue))
> master <- as.data.frame(t(master))
> row.names(master) <- NULL
```

The .xml data are stored as a series of "nodes" and "subnodes". Each top level node (TLN) corresponds to one attribute for each record in the data. However, the top level nodes may be made up of a number of subnodes which must be accessed individually to extract usable data. The first section of code is a generic retrieval of all the data. Any data in subnodes will be concatenated together in their respective TLN. The number of TLNs is the number of columns returned by the generic call to the .xml file.

```
> TLN <- length(master)
> print(TLN)
```

```
[1] 10
```

The EPA data files contain 10 TLNs, so the next step is to query each TLN for the presence of subnodes.

```
> Nodes <- rep(NA, TLN)
> for (i in 1:TLN) {
+     Nodes[i] <- length(x[[1]][[i]])
+ }
> print(Nodes)
```

```
 [1] 3 1 1 1 1 1 1 1 1 1
```

```
> print(master[1, ])
```

```
       GeographicalLocation AQSParameterDescription       Date
1 42.22862-83.2082181Meters         Carbon monoxide 2010-01-01
   Time TimeZone                          StatisticName MeasureValue
1 00:00  EASTERN Daily Means of Sample Measurements          .25
    MeasureUnitName ObservationCount DataSourceReferenceID
1 Parts per million               23       261630001421011
```

The values in Nodes are confirmed by viewing the first record in master. Under “GeographicalLocation”, the latitude, longitude, elevation, and label are all read in as one large entry (because of the earlier generic call to the .xml file). Next, we query the three subnodes under TLN 1 for the presence of more nested subnodes (note: this section of the code would need to be modified if there were more than one TLN with subnodes)

```
> whichNode <- which(Nodes > 1)
> print(whichNode)
```

```
[1] 1
```

```
> SN <- Nodes[whichNode]
> Subnode <- rep(NA, SN)
> for (i in 1:SN) {
+     Subnode[i] <- length(x[[1]][[whichNode]][[i]])
+ }
> print(Subnode)
```

```
[1] 1 1 2
```

The 3rd subnode of TLN 1 contains two more subnodes underneath it. Therefore, we can query that node to test for another set of nested subnodes.

```
> whichSub <- which(Subnode > 1)
> print(whichSub)
```

```
[1] 3
```

```
> SSN <- Subnode[whichSub]
> Sub <- rep(NA, SSN)
> for (i in 1:SSN) {
+     Sub[i] <- length(x[[1]][[whichNode]][[whichSub]][[i]])
+ }
> print(Sub)
```

```
[1] 1 1
```

After all the nested subnodes have been identified, the data they contain can be extracted.

```
> Coords <- xmlSApply(x, function(z) xmlSApply(z[[whichNode]],
+     xmlValue))
> Coords <- t(Coords)
> Coords <- as.data.frame(Coords)
```

This call to the .xml file returns 3 columns of data. Because the third subnode (elevation) is made up of two seperate subnodes (value, label), we remove it from this dataframe. Then, we extract the elevation data by itself.

```
> Coords <- Coords[, -(whichSub)]
> row.names(Coords) <- NULL
> Elev <- xmlSApply(x, function(z) xmlSApply(z[[whichNode]][[whichSub]],
+     xmlValue))
> Elev <- t(Elev)
> Elev <- as.data.frame(Elev)
> row.names(Elev) <- NULL
```

Now, the TLN 1 column can be removed from the original dataframe and the data from the nested subnodes can be appended. Also, unneccesary attributes can be removed from each of the observations, entries in the "StatisticName" column can be shortened, and the column names can be changed to a more R friendly format.

```
> master <- master[, -(whichNode)]
> master <- cbind(Coords, Elev, master)
> master$Time <- master$TimeZone <- master$DataSourceReferenceID <- NULL
> master$StatisticName <- "Daily Means"
> colnames(master) <- c("Latitude", "Longitude", "Elevation",
+     "ElevationUnits", "AQSParameter", "Date", "Statistic",
+     "StatisticValue", "StatisticUnit", "ObsCount")
```

```
> name <- paste("data/csv/", master$AQSParameter[1],
+     ".csv", sep = "")
> write.csv(master, file = name, row.names = F)
```

5 Merging pollutant data

Often, multiple pollutants are measured at the same monitoring location, therefore the following section of code can be used to import and merge data by location of collection. This code also tests for duplicate daily measures at each monitor location. These result when seperate, independent monitors measure the same pollutant at each location. In this special case, the mean of the monitors are calculated.

```
> names <- list.files("data/csv/", pattern = ".csv")
> files <- vector("list", length(names))
> for (i in 1:length(names)) {
+     files[[i]] <- read.csv(paste("data/csv/", names[i],
+         sep = ""))
+     dups <- files[[i]][duplicated(cbind(files[[i]]$Latitude,
+         files[[i]]$Longitude, files[[i]]$Date)),
+         ]
+     if (dim(dups)[1] > 0) {
+         files[[i]] <- aggregate(files[[i]][, c(3,
+             8, 10)], by = list(Latitude = files[[i]]$Latitude,
+             Longitude = files[[i]]$Longitude, Date = files[[i]]$Date),
+             mean, na.rm = TRUE)
+         files[[i]]$ElevationUnits <- dups$ElevationUnits[1]
+         files[[i]]$AQSParameter <- dups$AQSParameter[1]
+         files[[i]]$Statistic <- dups$Statistic[1]
+         files[[i]]$StatisticUnit <- dups$StatisticUnit[1]
+         files[[i]] <- files[[i]][c(1:2, 4, 7:8, 3,
+             9, 5, 10, 6)]
+     }
+     rm(dups)
+ }
```

Once the duplicate readings have been processed, the pollutant data can be merged into one large dataframe by location.

```
> key <- c("Latitude", "Longitude", "Date", "Elevation",
+     "ElevationUnits", "Statistic")
> colnames <- c("CO", "CO.unit", "CO.obs", "Pb", "Pb.unit",
+     "Pb.obs", "NO2", "NO2.unit", "NO2.obs", "O3",
+     "O3.unit", "O3.obs", "PM10", "PM10.unit", "PM10.obs",
+     "PM25", "PM25.unit", "PM25.obs", "SO2", "SO2.unit",
+     "SO2.obs", "TSP", "TSP.unit", "TSP.obs")
> m <- merge(files[[1]], files[[2]], by = key, all = TRUE)
> m$AQSParameter.x <- m$AQSParameter.y <- NULL
> names(m)[7:12] <- colnames[1:6]
> ct <- 13
> for (w in 3:length(files)) {
+     m <- merge(m, files[[w]], by = key, all = TRUE)
+     m$AQSParameter <- NULL
+     names(m)[ct:(ct + 2)] <- colnames[(ct - 6):(ct -
+         4)]
+     ct <- ct + 3
+ }
> write.csv(m, file = "data/out/merged.csv", row.names = F)
```

6 Create summary files

Summary files can be created with each monitor location and the number of observations recorded (per pollutant during the study period) and the mean of the daily observations.

```
> count <- function(x) sum(!is.na(x))
> mean.na <- function(x) mean(x, na.rm = T)
> obs <- aggregate(m[, c(7, 10, 13, 16, 19, 22, 25,
+     28)], by = list(Latitude = m$Latitude, Longitude = m$Longitude),
+     FUN = count)
```

```
> obs.m <- aggregate(m[, c(7, 10, 13, 16, 19, 22, 25,
+     28)], by = list(Latitude = m$Latitude, Longitude = m$Longitude),
+     FUN = mean.na)
```

7 Make pollution surface

Now, we can view the monitor locations on a map using R's spatial capabilities. First read in the .shp file of Michigan counties. According to the output, there are 252 polygons in the file. Many of these polygons are small islands due to lakes surrounding Michigan which slow down the ability to display in using R. To speed up the display, these can be removed.

```
> mi.state <- readOGR("data/gis/.", "michigan")

OGR data source with driver: ESRI Shapefile
Source: "data/gis/.", layer: "michigan"
with 252 features and 7 fields
Feature type: wkbPolygon with 2 dimensions

> mi <- mi.state[mi.state$AREA > 0.01, ]
> plot(mi, xlab = "Degrees", ylab = "Degrees", axes = T)
> points(obs$Longitude, obs$Latitude, pch = 16, col = "red")
```

We'll subset only Particulate Matter 2.5 (PM25) points from the observations and create a surface using the monthly average value at each monitor. Because so few monitors are located in the upper peninsula of Michigan, the surface will only be created over the lower peninsula (note: Creating surfaces using unprojected geographic data [decimal degrees] is not recommended in general practice.)

```
> pm25.m <- obs.m[!is.na(obs.m$PM25), c(2, 1, 8)]
> mba.bbox <- c(-88, -82, 41, 47)
> surf <- mba.surf(pm25.m, 100, 100, extend = TRUE,
+     sp = TRUE, b.box = mba.bbox)$xyz.est
> image(surf, axes = T, asp = 1.25)
> plot(mi, add = T)
> points(pm25.m$Longitude, pm25.m$Latitude, pch = 16)
```

Finally, clip the surface to the county boundaries.

```
> surf.c <- surf
> surf.c@data <- surf.c@data * (!is.na(overlay(surf,
+     mi)))
> surf.c$z[surf.c$z == 0] <- NA
> image(surf.c, axes = T, asp = 1.25)
> plot(mi, add = T)
> points(pm25.m$Longitude, pm25.m$Latitude, pch = 16)
```

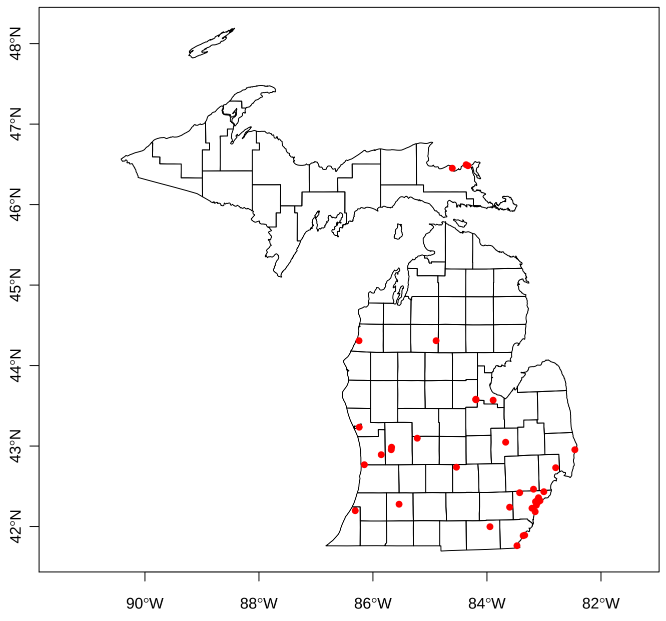

Figure 1: Monitor locations.

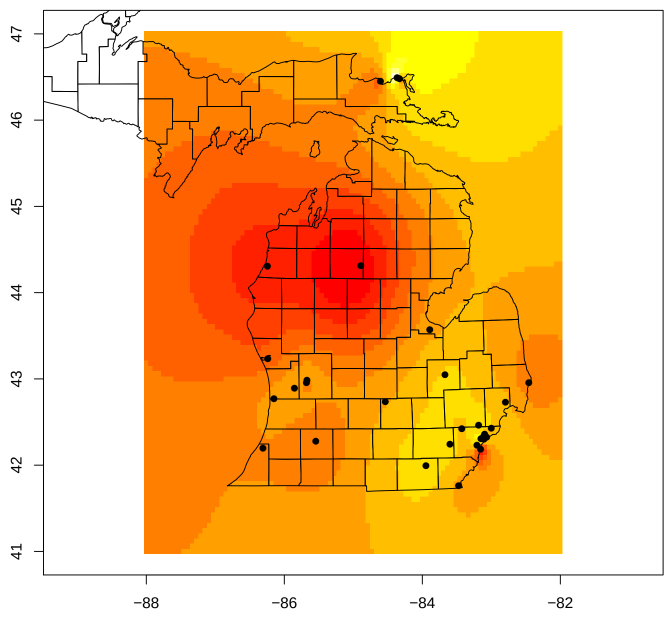

Figure 2: MBA PM 2.5 surface.

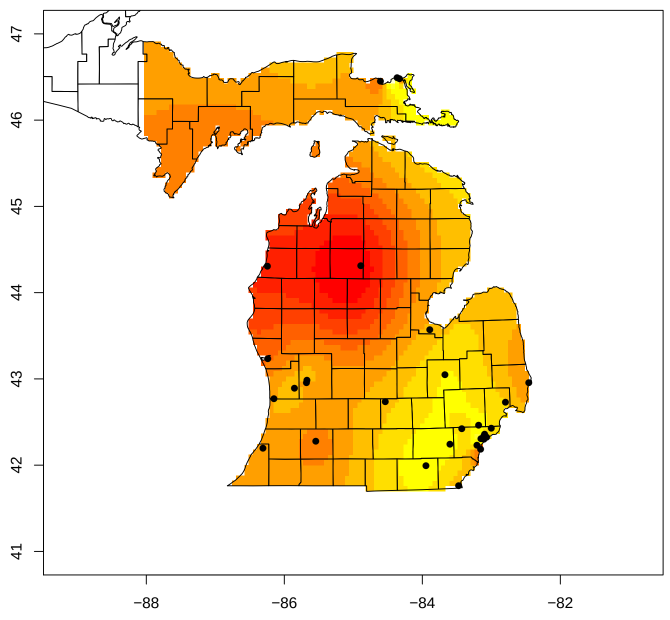

Figure 3: Clipped MBA PM 2.5 surface.