Working With Data

Manning the helm during the data deluge.

Python-style Logging in R

Posted on August 24, 2016 by Jonathan Callahan

This entry is part 20 of 21 in the series Using R

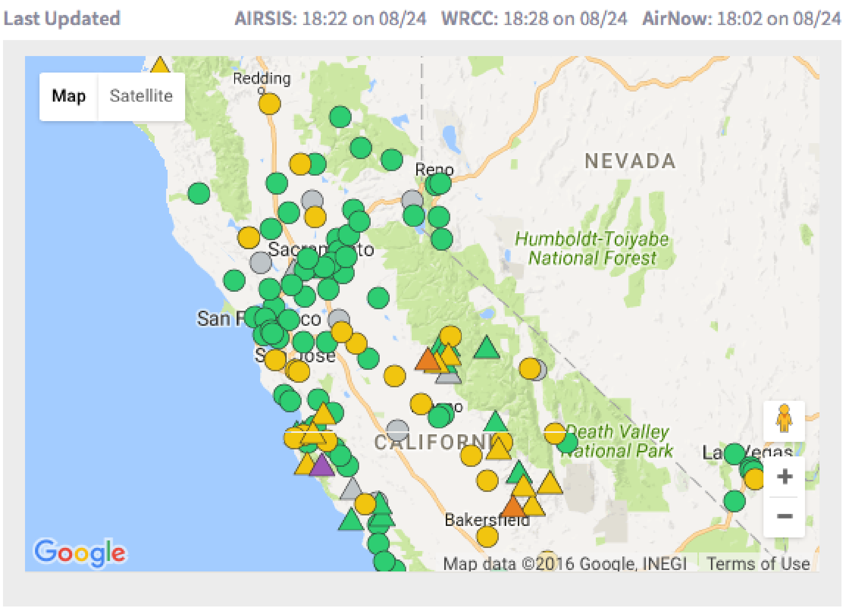
We are increasingly using R in "operational" settings that require robust error handling and logging. In this post we describe a quick-and-dirty way to get python style multi-level log files by wrapping the futile.logger package.

Making Messy Data Pretty

Our real world scenario involves R scripts that process raw smoke monitoring data that is updated hourly. The raw data comes from various different instruments, set up by different agencies and transmitted over at least two satellites before eventually arriving on our computers.

Data can be missing, delayed or corrupted for a variety of reasons before it gets to us. And then our R scripts perform QC based on various columns available in the raw (aka "engineering level") data.

Ultimately, the cleaned up data gets displayed in a user interface here.



No one needs to know all of the hard work that goes into cleaning and QC'ing the data that goes into this map. Our goal, after all, is to make it easier for stakeholders to make decisions without having to worry about the fussy details of raw data.

Logging When Things Go Wrong

However, neither the incoming data nor the R scripts we have written to process it are perfect. Things occasionally go wrong in a variety of ways and, when decision makers depend on the data being available, phone calls and text messages are next.

Bhuddist teacher Pema Chödrön's book *When Things Fall Apart* has several advice categories that actually apply in this situation. We will focus on one:

- Using painful emotions to cultivate wisdom, compassion, and courage
- Communicating so as to encourage others to open up rather than shut down
- Practices for reversing habitual patterns
- **Methods for working with chaotic situations**
- Ways for creating effective social action

One of the methods for working with chaotic situations in operational software is to have lots and Lots and LOTS of logging.

Python has the brilliant "logging" module that allows us to quickly set up separate output files for log statements at different levels: `DEBUG, INFO, WARNING, ERROR, WARNING, CRITICAL`. Inside our operational code we only need to issue single calls like:

```
logger.info('Ingesting data file: %s', data_file)
```

and this information will automatically be written out to both the DEBUG and the INFO log files if we have set them up.

In our situation it makes sense to write out three log files for quick perusal by authorized personnel:

- ~_DEBUG.log — for programmers needing to debug what went wrong
- ~_INFO.log — for managers trying to understand the overall situation
- ~_ERROR.log — (hopefully zero length!) for programmers and managers

These files get written to a uniform directory every time a data processing script runs with the '~' replaced by the name of the processing script.

Wrapping futile.logger

Luckily, most of the functionality we need is provided by futile.logger package (CRAN or github). Unfortunately, this package does not currently support multiple "appenders" per "logger" so we found ourselves initializing multiple named loggers to mimic the natural behavior of Python's logging module:

```
# Only FATAL messages go to the console
dummy <- flog.threshold(FATAL)

# Set up new log files
if ( file.exists(DEBUG_LOG) ) dummy <- file.remove(DEBUG_LOG)
dummy <- flog.logger("debug", DEBUG, appender.file(DEBUG_LOG))

if ( file.exists(INFO_LOG) ) dummy <- file.remove(INFO_LOG)
dummy <- flog.logger("info", INFO, appender.file(INFO_LOG))

if ( file.exists(ERROR_LOG) ) dummy <- file.remove(ERROR_LOG)
dummy <- flog.logger("error", ERROR, appender.file(ERROR_LOG))

# Silence other warning messages
options(warn=-1) # -1=ignore, 0=save/print, 1=print, 2=error
```

Then, inside the code, we created logging and error handling blocks like this:

```
  if ( class(result)[1] == "try-error" ) {
    err_msg <- geterrmessage()
    flog.error('Error creating SpatialPoitnsDataFrame: %s', err_msg, name='debug')
    flog.error('Error creating SpatialPointsDataFrame: %s', err_msg, name='info')
    flog.error('Error creating SpatialPointsDataFrame: %s', err_msg, name='error')
    stop(err_msg)
  }
  ...
  flog.info('Using rgdal::writeOGR to create geojson file %s', filename, name='debug')
  flog.info('Using rgdal::writeOGR to create geojson file %s', filename, name='info')
```

Yuck!

After writing just a few lines like this we tried a different approach and came up with a simple py_logging.R script that can be used to wrap futile.logger calls and give us the python style behavior we want. Setup is a cinch:

```
source('py_logging.R') # python style logging

# Set up logging
logger.setup(debugLog, infoLog, errorLog)

# Silence other warning messages
options(warn=-1) # -1=ignore, 0=save/print, 1=print, 2=error
```

And logging is much more natural:

```
  if ( class(result)[1] == "try-error" ) {
    err_msg <- geterrmessage()
    logger.error('Error creating SpatialPointsDataFrame: %s', err_msg)
    stop(err_msg)
  }
  ...
  logger.info('Using rgdal::writeOGR to create goejson file %s', filename)
```

Output from our logger looks great!

```
INFO [2016-08-24 10:11:54] Running process_data.R version 0.1.5

INFO [2016-08-24 10:11:54] Working on /home/data/location/datafile_1.RData
INFO [2016-08-24 10:11:55] Found 1232 data columns
INFO [2016-08-24 10:12:03] Working on /home/data/location/datafile_2.RData
INFO [2016-08-24 10:12:03] Found 103 data columns
INFO [2016-08-24 10:12:03] Working on /home/data/location/datafile_3.RData
INFO [2016-08-24 10:12:03] Found 44 data columns
```

At the end of the day, we can only hope we have succeeded at one or another other of:

- Using painful emotions to cultivate wisdom, compassion, and courage
- Communicating so as to encourage others to open up rather than shut down

Best wishes for excellent logging!

Series Navigation

When k-means Clustering Fails Improved Python-style Logging in R

This entry was posted in R, Toolbox and tagged logging, R. Bookmark the permalink.

page: 2

Introducing the PWFSLSmoke Package

Posted on March 12, 2017 by Jonathan Callahan

Mazama Science has just released the PWFSLSmoke package. Source code is available on GitHub. Here is the package description:

Utilities for working with air quality monitoring data with a focus on small particulates (PM2.5) generated by wildfire smoke. Functions are provided for downloading available data from the United States Environmental Protection Agency (US EPA) and it's AirNow air quality site. Additional sources of PM2.5 data made accessible by the package include: AIRSIS (password protected), the Western Regional Climate Center (WRCC) and the open source site OpenAQ.

In this post we discuss the reasons for creating this package and provide examples of its use.

For several years now, Mazama Science has been working with the AirFire team at The USFS Pacific Wildland Fire Sciences Lab. The **PWFSLSmoke** R package is being developed for PWFSL to help modelers and scientists more easily work with PM2.5 data from monitoring locations across North America. The package makes it easier to obtain data, perform analyses and generate reports. It includes functionality to:

- download and easily work with regulatory PM2.5 data from AirNow
- download and quality control raw monitoring data from other sources
- convert between UTC and local timezones
- apply various algorithms to the data (nowcast, rolling means, aggregation, *etc.*)
- provide interactive timeseries and maps through RStudio's Viewer pane
- create a variety of publication ready maps and timeseries plots

Data Model

The **PWFSLSmoke** package is designed to provide a compact, full-featured suite of utilities for working with PM 2.5 data used to monitor wildfire smoke. A uniform data model provides consistent data access across monitoring data available from different agencies. The core data model in this package is defined by the *ws_monitor* object used to store data associated with groups of individual monitors.

To work efficiently with the package it is important to understand the structure of this data object and which functions operate on it. Package functions that begin with `monitor_`, expect objects of class *ws_monitor* as their first argument. (*Note that 'ws_' stands for 'wildfire smoke'.*)

Monitoring data will typically be obtained from an agency charged with archiving data acquired at monitoring sites. For wildifre smoke, the primary pollutant is PM 2.5 and the sites archiving this data include AirNow, WRCC and AIRSIS.

The data model for monitoring data consists of an **R** `list` with two dataframes: `data` and `meta`.

The `data` dataframe contains all hourly measurements organized with rows (the 'unlimited' dimension) as unique timesteps and columns as unique monitors. The very first column is always named `datetime` and contains the `POSIXct` datetime in Coordinated Universal Time (UTC).

The `meta` dataframe contains all metadata associated with monitoring sites and is organized with rows as unique sites and columns as site attributes. The following columns are guaranteed to exist in the `meta`dataframe:

- `monitorID` – unique ID for each monitoring site (*i.e.* each instrument deployment)
- `longitude` – decimal degrees East
- `latitude` – decimal degrees North
- `elevation` – meters above sea level
- `timezone` – Olson timezone
- `countryCode` – ISO 3166-1 alpha-2 code
- `stateCode` – ISO 3166-2 alpha-2 code

(The MazamaSpatialUtils package is used to assign timezones and state and country codes.)

Additional columns may be available in the `meta` dataframe and these will depend on the source of the data.

It is important to note that the `monitorID` acts as a unique key that connects data with metadata. The `monitorID` is used for column names in the `data` dataframe and for row names in the `meta` dataframe. So the following will always be true:

```
rownames(ws_monitor$meta) == ws_monitor$meta$monitorID
colnames(ws_monitor$data) == c('datetime',ws_monitor$meta$monitorID)
```

File Size Reduction

Most providers of monitoring data store the data as it comes in, retaining station metadata with every single datum. However, individual monitors rarely move. Thus, replication of metadata vastly inflates the size of data in this "as-it-comes-in" data management strategy. The *ws_monitor* data model reflects the reality of monitors that rarely move.

Our BigData presentation describes how we used this data model to convert unwieldy EPA annual datasets into bite sized .RData files without losing any information. In one example we reduced a 665 Mb ASCII CSV file into a 3.3 Mb .RData file — 1/200th the size of the original.

Reduced file sizes make working with data from multiple monitors much easier and the remainder of this post demonstrates the package capabilities by looking at smoke generated from the 2015 wildfire season in the Pacific Northwest

Pacific Northwest 2015 Wildfires

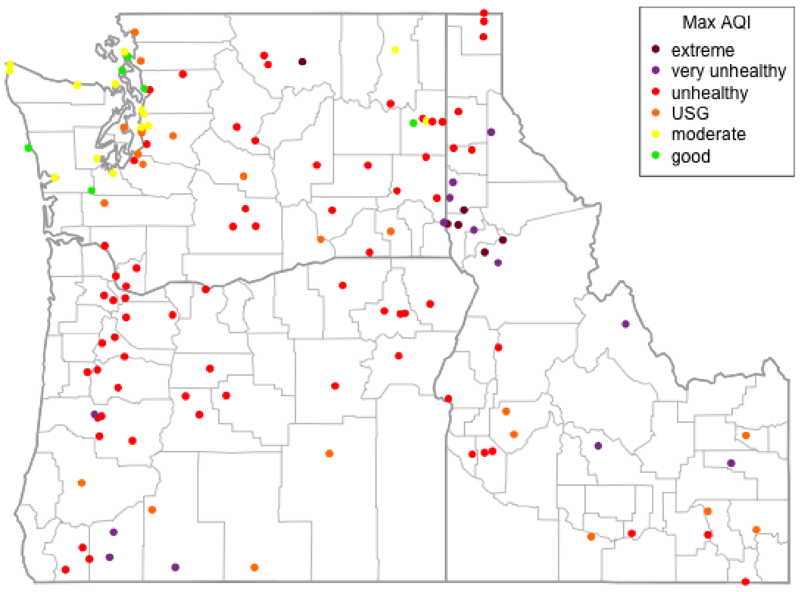
In the summer of 2015 Washington state had several catastrophic wildfires that led to heavy smoke in eastern Washington and northern Idaho for quite a few days. We will show how the mapping and timeseries plotting functions in the **PWFSLSmoke** package can help us visualize the spatial and temporal extent of wildfire smoke during these events.

To begin, let's have a broader look at AirNow ambient monitoring data for the Pacific Northwest – Washington, Oregon and Idaho – from June 1 through October 31, 2015. First, we create a 24-hr rolling mean for each monitor:

```
library(PWFSLSmoke)
PacNW <- Northwest_Megafires
# To work with AirNow data directly, uncomment the next two lines
#N_M <- airnow_load(startdate=20150531, enddate=20151101)
#PacNW <- monitor_subset(airnow, stateCodes=c("WA", "OR", "ID"))
PacNW_24 <- monitor_rollingMean(PacNW, width=24)
```

Now we can create a map where each monitor is color coded by the maximum value of this 24-hr rolling mean. By default, AQI colors and labels are used but arguments to `monitorMap()` and `addAQILegend()` allow users to specify their own.

```
monitorMap(PacNW_24, slice=max)
addAQILegend(title="Max AQI", cex=0.7)
```



The map shows that many areas of the Pacific NW had days with unhealthy air but a cluster of sites in Idaho were particularly bad.

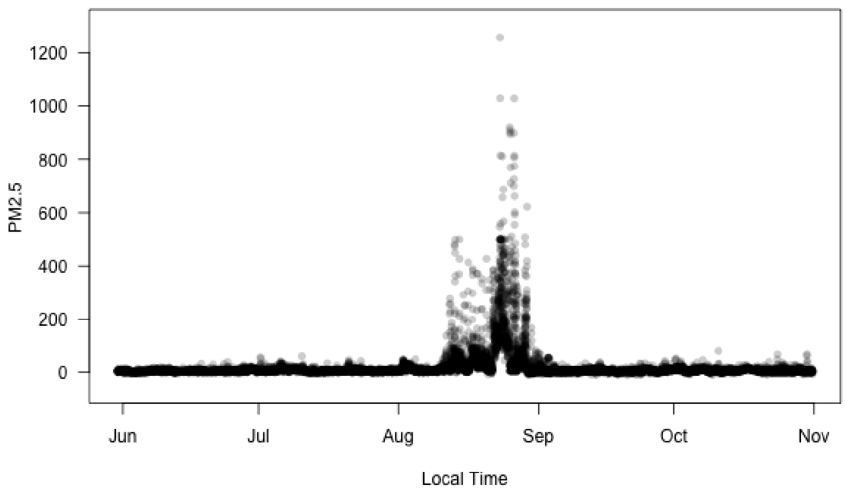
(***Note** that this is not the regulatory midnight-to-midnight AQI but a continuous 24-hr rolling mean.*)

We can us an interactive "leaflet" map to zoom in and get more information:

```
# Commented out for the blog post
#monitorLeaflet(PacNW_24, slice=max)
```

The `monitorMap()` and `monitorLeaflet()` plots show us pretty much the same information, except that the leaflet plot allows you to get monitor-specific metadata by clicking on a monitor. In this manner, we can assemble a list of monitorIDs in and around the Nez Perce Reservation in northern Idaho and generate a multi-monitor timeseries plot showing the terrible smoke in late August.

```
NezPerceIDs <- c("160571012","160690012","160690013","160690014","160490003","160491012")
NezPerce <- monitor_subset(PacNW, monitorIDs=NezPerceIDs)
monitorPlot_timeseries(NezPerce, style='gnats')
```

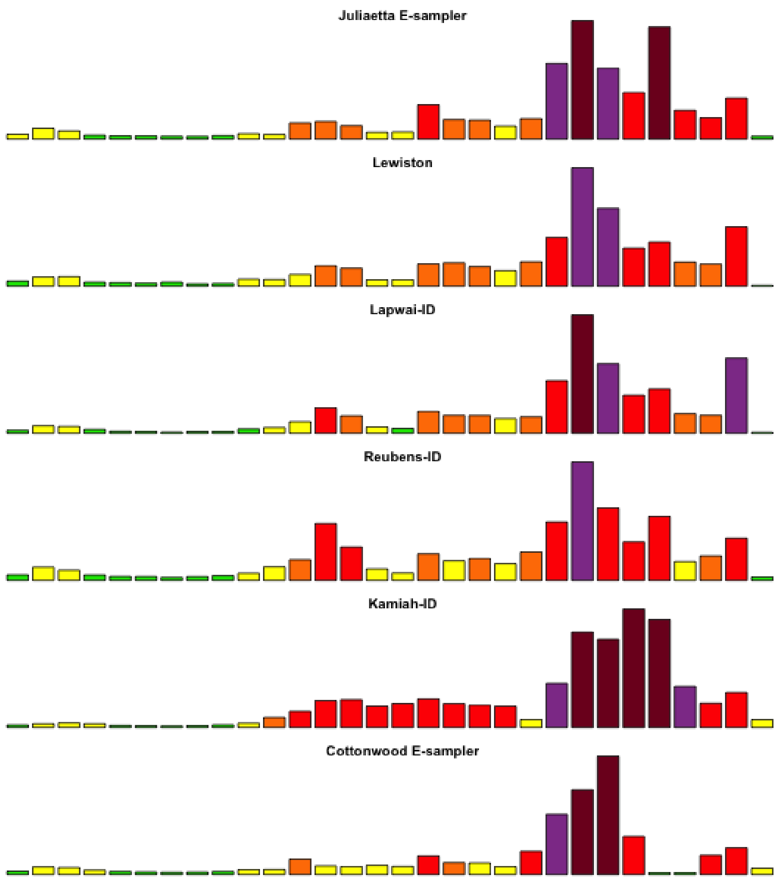


At this point it is clear that August is the month of interest so we'll subset all of our existing `ws_monitor` objects to cover the month of August with full days according to West Coast time.

```
PacNW <- monitor_subset(PacNW, tlim=c(20150801,20150831), timezone="America/Los_Angeles")
PacNW_24 <- monitor_subset(PacNW_24, tlim=c(20150801,20150831), timezone="America/Los_Angeles")
NezPerce <- monitor_subset(NezPerce, tlim=c(20150801,20150831), timezone="America/Los_Angeles")
```

We can use the `monitorPlot_dailyBarplot()` function to look at official, midnight-to-midnight AQI levels for each monitor during the month of August:

```
layout(matrix(seq(6)))
par(mar=c(1,1,1,1))
for (monitorID in NezPerceIDs) {
  siteName <- NezPerce$meta[monitorID,'siteName']
  monitorPlot_dailyBarplot(NezPerce, monitorID=monitorID, main=siteName, axes=FALSE)
}
par(mar=c(5,4,4,2)+.1)
layout(1)
```



We could also take a more automated approach and directly calculate the location with the worst acute smoke (worst hourly value) during this time period:

```
data <- PacNW$data[,-1] # omit 'datetime' column
maxPM25 <- apply(data, 2, max, na.rm=TRUE)
worstAcute <- names(sort(maxPM25, decreasing=TRUE))[1:6]
intersect(worstAcute, NezPerceIDs)
## [1] "160571012" "160490003" "160491012"
PacNW$meta[worstAcute[1],c('siteName','countyName','stateCode')]
##                      siteName countyName stateCode
## 160571012 Juliaetta E-sampler      LATAH        ID
```

We see that three of the sites we identified "by hand" also had the worst acute smoke of any sites in the Pacific NW. The site with the highest measured value of PM 2.5 for the month of august was Julietta in Latah county, Idaho.

Let's do a similar analysis for chronic smoke. In this case we will calculate the number of days at or above the AQI "unhealthy" level:

```
PacNW_dailyAvg <- monitor_dailyStatistic(PacNW, FUN=mean, minHours=20)
## Warning in monitor_dailyStatistic(PacNW, FUN = mean, minHours = 20): Found
## 2 timezones. Only the first will be used
data <- PacNW_dailyAvg$data[,-1]
unhealthyDays <- apply(data, 2, function(x){ sum(x >= AQI$breaks_24[4], na.rm=TRUE) })
worstChronic <- names(sort(unhealthyDays, decreasing=TRUE))[1:6]
intersect(worstChronic, NezPerceIDs)
## [1] "160490003" "160571012" "160690014"
PacNW$meta[worstChronic[1],c('siteName','countyName','stateCode')]
##            siteName countyName stateCode
## 160490003 Kamiah-ID      IDAHO        ID
```

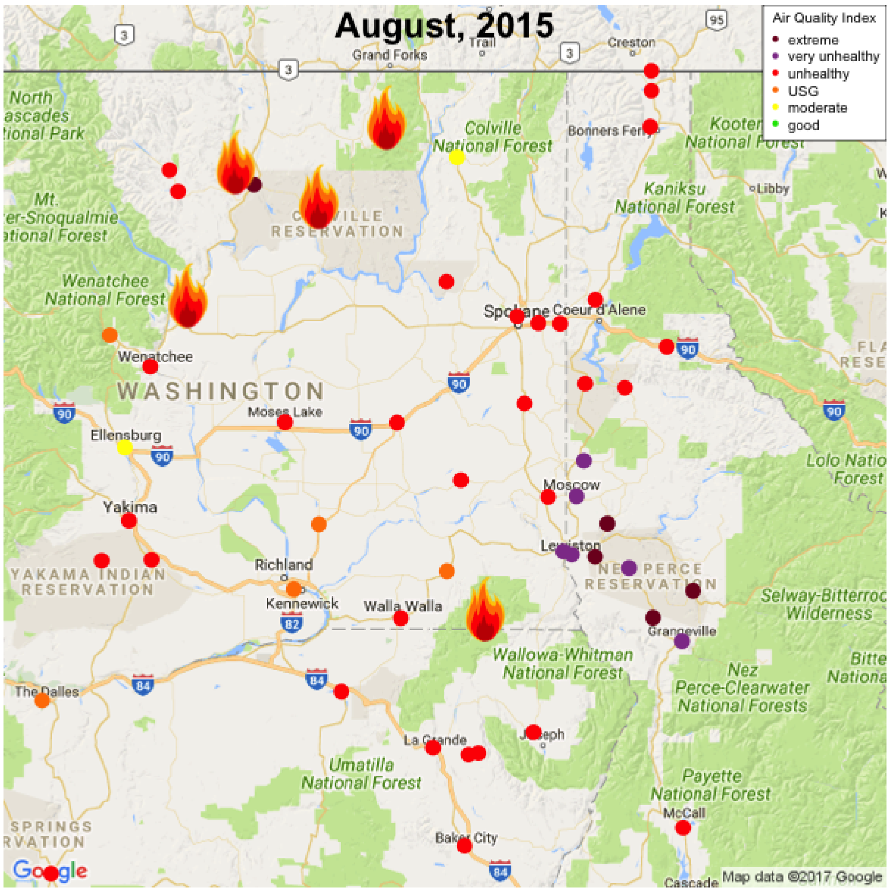
The area around the Nez Perce Reservation again had three of the worst sites for chronic smoke with the worst being Kamiah in Idaho county, Idaho.

Latah and Idaho counties were unfortunately downwind of several extraordinarily large Washington state wildfires in 2015 with the following ignition dates:

- Aug 11 – Kettle Complex
- Aug 13 – Grizzly Bear Complex
- Aug 13 – North Star
- Aug 14 – Chelan Complex
- Aug 15 – Okanogan Complex

A little googling and we can obtain a set of coordinates for these fires to use in a new "Google" map of the August daily average maxima with fire icons at fire locations:

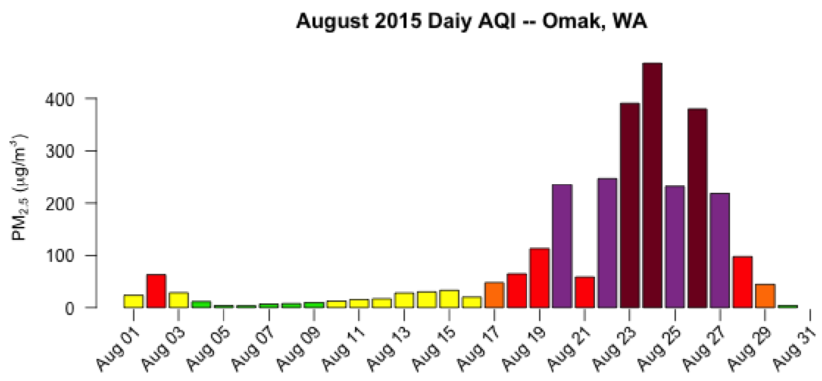
```
fireLons <- c(-118.461,-117.679,-120.039,-119.002,-119.662)
fireLats <- c(48.756,46.11,47.814,48.338,48.519)
gmap <- monitorGoogleMap(PacNW_dailyAvg, zoom=7, centerLon=-118, centerLat=47, slice=max)
addIcon('redFlame', fireLons, fireLats, map=gmap, expansion=0.2)
addAQILegend(cex=0.8)
title("August, 2015", line=-1.5, cex.main=2)
```

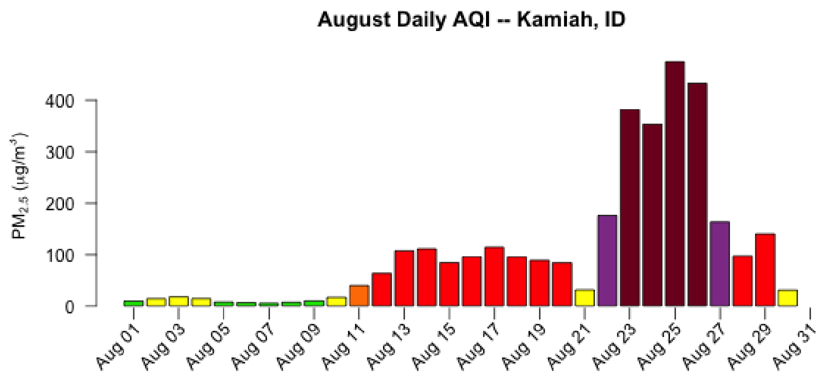


From this map we can see that the monitor in Omak on the Colleville reservation also registered a daily AQI level of "extreme" as it was in-between the two largest fires. The overall spatial pattern, however, shows that the worst impact from these large fires resulted from smoke drifting SE with the prevailing winds and bunching up in the valleys and plains just upwind of the Bitterroot mountains of northern Idaho.

We will now take a closer look at two tribal monitors: Omak for the Colleville tribe and Kamiah for the Nez Perce.

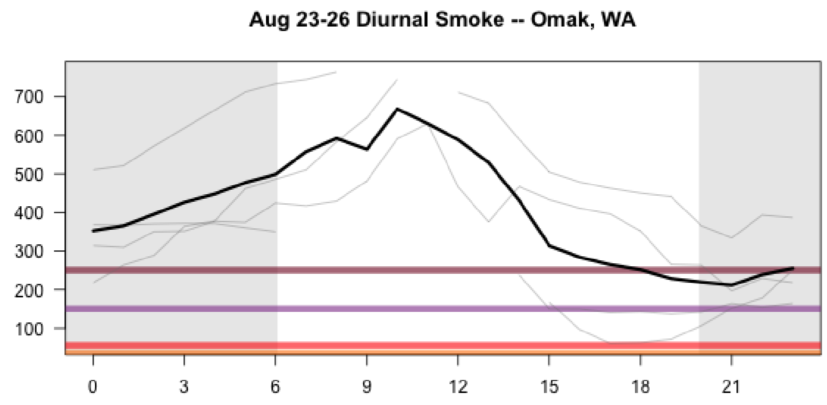
```r
Omak <- monitor_subset(PacNW, monitorIDs="530470013")
Kamiah <- monitor_subset(PacNW, monitorIDs="160490003")
layout(matrix(seq(2)))
monitorPlot_dailyBarplot(Omak, main="August 2015 Daiy AQI -- Omak, WA",
                         labels_x_nudge=0.8, labels_y_nudge=250)
monitorPlot_dailyBarplot(Kamiah, main="August Daily AQI -- Kamiah, ID",
                         labels_x_nudge=0.8, labels_y_nudge=250)
layout(1)
```

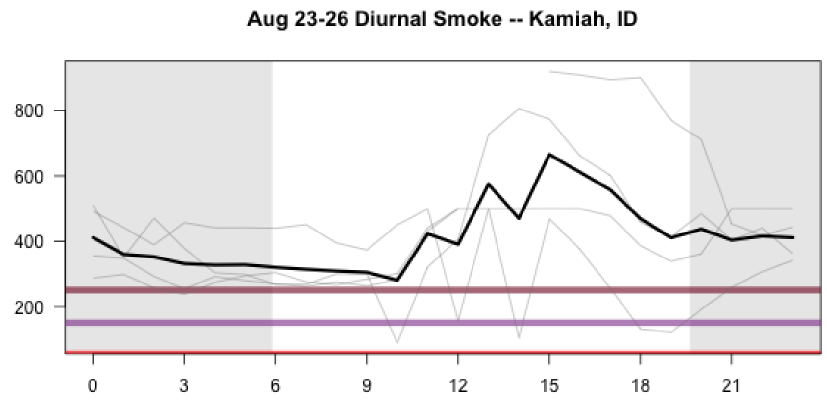




We can also examine the diurnal cycle during the very worst days:

```r
layout(matrix(seq(2)))
par(mar=c(3,4,4,2))
monitorPlot_timeOfDaySpaghetti(Omak, title="Aug 23-26 Diurnal Smoke -- Omak, WA",
                               xlab='', ylab='',
                               tlim=c(20150823,20150826))
monitorPlot_timeOfDaySpaghetti(Kamiah, title="Aug 23-26 Diurnal Smoke -- Kamiah, ID",
                               xlab='', ylab='',
                               tlim=c(20150823,20150826))
par(mar=c(5,4,4,2)+.1)
layout(1)
```





While both sites had horrible air all day, in Omak it was somewhat less horrible in the evenings. This is in contrast to Kamiah where the least horrible conditions were encountered in the mornings.

This ends the spatio-temporal exploration of smoke from Pacific NW mega-fire in 2015. We hope this inspires you to harness the mapping plotting functionality available in the **PWFSLSmoke** package.

Best Wishes for Healthy Air!

This entry was posted in R and tagged AirFire, AQI, metadata, PM2.5, PWFSL, R, R package, smoke. Bookmark the permalink.