Downloading and processing NOAA weather station data in R

Michael Piccirilli & Elliot Cohen

January 8, 2015

Contents

Geo-referenced weather data pipeline

By Michael Piccirelli, Elliot Cohen and James McCreight ****

Motivation

Long observational records of high-resolution weather data are instrumental to a wide range of research and development applications. Such data are essential to energy demand forecasts (Segal et al. 1992; Sailor 2001; Crowley et al. 2003; Thatcher 2007, Cohen et al. 2014), building energy models (Deru et al. 2011), renewable energy feasibility and siting assessments (Lambert et al. 2005), crop insurance programs (Helmath et al. 2009), emergency preparedness and early warning systems (Rogers and Tsirkunov 2011), and much more.

Globally, national weather services and climate information centers such as the U.S. National Oceanic and Atmospheric Administration (NOAA), Britain's Met Office, and India's Institute for Tropical Meteorology (IITM), collect, curate, analyze and publish meteorological data from thousands of weather stations. NOAA's National Climatic Data Center (NCDC) is the world's largest archive of weather data.

NOAA scientists offer a wealth of meteorological/climatological information through the NCDC data portal. The data is available on the Online Climate Data Directory website. It is also available via FTP, which is more effecient for batch queries, and is the method we will use here.

The following article, and the accompanying `weatheR` library built for the statistical computing language R, is intended to facilitate geo-referenced and quality-control batch query of NOAA's National Climatic Data Center (NCDC)–the world's largest active archive of weather data. Our work builds on "Downloading and processing NOAA hourly weather station data" by Delameter et al. (YEAR). Our goal is to make meteorological data more accessible for a wide range of scientists, engineers and practicioners familiar with data analysis in R, but less familiar with the handling large meteorological datasets in particular. The weatheR libary introduces new functionality and automation previously unavailable:

- Identification and selection of nearest-neighbor weather stations for virtually any place on Earth. Locations of interest are geo-referenced via the Google Maps API.
- Batch query with a simple charachter string of location names (e.g. "New York City", "Tiennamen Square", "Abuja, Nigeria")
- Built-in quality controls for "best" data selection based on the geographic proximity and completeness of the data record.
- Quick visual inspection to make sure you're getting the right data.

The weatheR library

Open-Source and Available to All!

We created the `weatheR` package with all the necessary functions to download, transform, and plot NOAA weather station data. The package has 7 R package dependencies: `httr`, `ggmap`, `FNN`, `McSpatial`, `data.table`, `plyr`, `gridExtra`. The package is available on GitHub at https://github.com/mpiccirilli/weatheR. To start using the `weatheR` package immediately, all you need are the following three lines of code:

```
## install weatheR library (beta version from github)
require(devtools)
install_github("mpiccirilli/weatheR")
require(weatheR)
```

Dependencies

The following article goes step-by-step through the functions in the `weatheR` package.

Station List

The first step is to download the list of all ISD weather stations which can be found in the following location: ftp://ftp.ncdc.noaa.gov/pub/data/noaa/. The current list of weather stations can be found in a file called **`isd-history.txt`** or **`isd-history.csv`**.

```
allStations <- function() {
    isd <- "ftp://ftp.ncdc.noaa.gov/pub/data/noaa/isd-history.csv"
    response <- suppressWarnings(GET(isd))
    all <- read.csv(text = content(response, "text"), header = TRUE)
    colnames(all)[c(3, 9)] <- c("NAME", "ELEV")
    all <- all[!is.na(all$LAT) & !is.na(all$LON), ]
    return(all)
}
stations <- allStations()

head(stations)
```

```
##     USAF  WBAN                NAME CTRY STATE ICAO    LAT    LON   ELEV
## 3   7018 99999          WXPOD 7018                    0.000  0.000 7018.0
## 6   7026 99999          WXPOD 7026   AF               0.000  0.000 7026.0
## 17  7070 99999          WXPOD 7070   AF               0.000  0.000 7070.0
## 23  8268 99999           WXPOD8278   AF              32.950 65.567 1156.7
## 24  8307 99999          WXPOD 8318   AF               0.000  0.000 8318.0
## 45 10010 99999 JAN MAYEN(NOR-NAVY)   NO       ENJA 70.933 -8.667    9.0
##       BEGIN      END
## 3  20110309 20130730
## 6  20120713 20141120
## 17 20140923 20140923
## 23 20100519 20120323
## 24 20100421 20100421
## 45 19310101 20150105
```

Side note: this dataset is included in the package and can be called into memory by executing `data(station.list)`.

In the 5th line of the function above, we elimiated stations that do not contain either Latitude or Longitute coordines. We do this because we'll be using the Latitude and Longitude coordinates provided in this dataset to find the closest weather stations to our cities of interest. The amount of data that is lost from this step is minimal, approximately 1,300 of 29,000 weather stations.

Places of Interest

Now that we have the full list of weather stations, let's focus on what stations we want to hone in on. Let's create a vector of cities for which we'd like to find data. For demonstration, we'll use a few cities in the table of the World's Fastest Growing Cities mentioned above.

[comment: do we need to include city and country name to avoid mis-maatches? For example, there may be more than one "Victoria" in the world]

```
cities.of.interest <- c("Nairobi, Kenya", "Tema, Ghana", "Accra, Ghana",
    "Abidjan, Ivory Coast")
```

Nearest Neighbor Search

We now create a function that allows us the find the k-nearest weather stations to each of the cities mentioned above. This function finds a reference point for each city by using the Google Maps API from the package `ggmap` to find Latitude and Longitude coordinates of each city. It then passes these coordiates into a KNN function from the `FNN` package to find the k-nearest weather stations, which uses Euclidean distance. While the difference between Euclidean distance and distances calculated using the great circle formula are minimal, we did notice that some stations were indeed ordered incorrectly using Euclidean distance. Therefore we decided to calculate distance, coverted into Kilometers, based on the great circle formula.

The input of this function is a list of cities (or an individual city), the full list of ISD stations, and the k-nearest stations you wish to return. The format of the `city.list` should be "City, State" or "City, Country", whichever is applicable.

```
kNStations <- function(city.list, station.list, k) {
    coords <- suppressMessages(geocode(city.list))
    # Find k-nearest weather stations to the reference point
    kns <- get.knnx(as.matrix(station.list[, c(8, 7)]), as.matrix(coords),
        k)
    st <- station.list[kns$nn.index[, ], ]
    # Add additional fields to the dataset
    st$city <- rep(city.list, nrow(st)/nrow(coords))  # Reference City
    st$Ref_Lat <- rep(coords$lat, nrow(st)/nrow(coords))  # Reference Latitude
    st$Ref_Lon <- rep(coords$lon, nrow(st)/nrow(coords))  # Reference Longitude
    kilos.per.mile <- 1.60934
    st$kilo_distance <- geodistance(st$Ref_Lon, st$Ref_Lat, st$LON,
        st$LAT, dcoor = FALSE)$dist * kilos.per.mile  # Convert miles to kilos
    st <- st[with(st, order(city, kilo_distance)), ]
    st$rank <- rep(1:k, length(city.list))  # Rank is list from 1-k, closest to farthest
    st$BEGIN_Year <- as.numeric(substr(st$BEGIN, 1, 4))
    st$END_Year <- as.numeric(substr(st$END, 1, 4))
    # st <- st[st$BEGIN_Date <= beg & st$END_Date >= end, ] #
    # remove stations without complete data
```

```
    return(st)
}
```

Visual Inspection of Nearest Neighbors

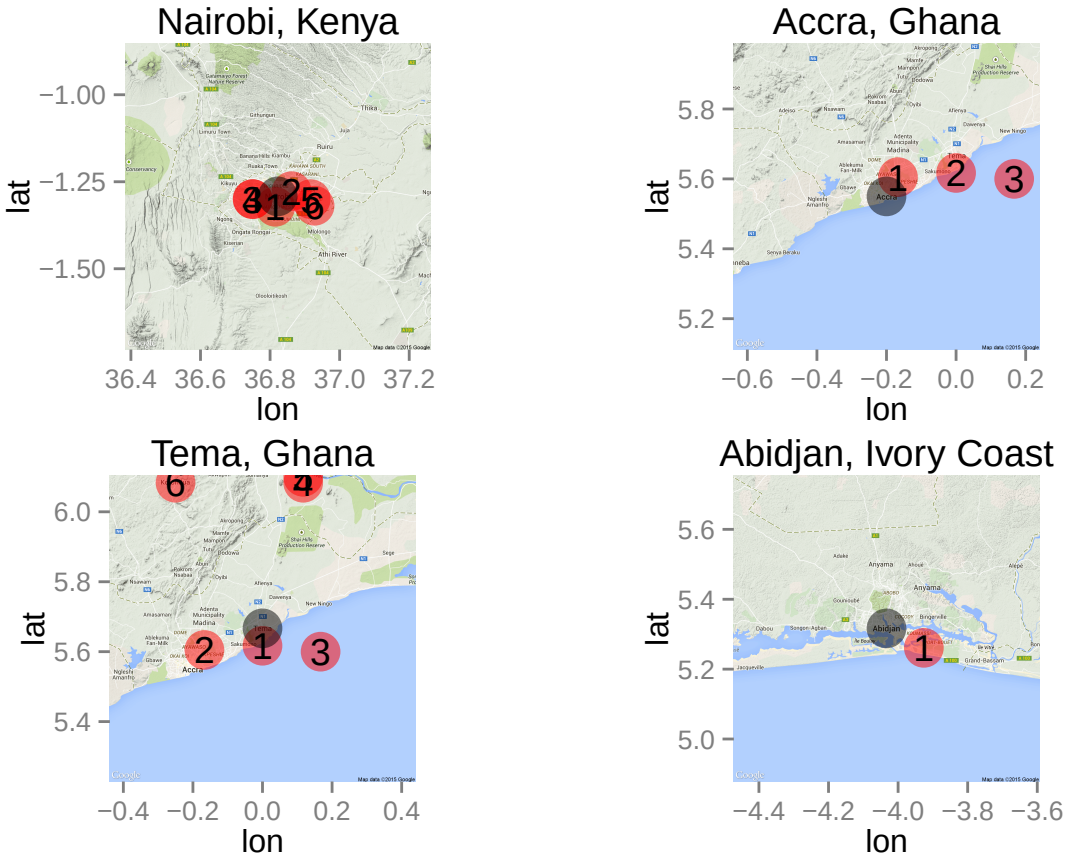
We now have all that is needed to begin downloading data, but before we do so we can visualize the stations of each city to understand how far away each station is from a city's central reference point. The code below first sets up the ability to print multiple plots in the case using a vector of cities, as we have above.

```
multiplot <- function(..., plotlist = NULL, file, cols = 1, layout = NULL) {
    # Make a list from the ... arguments and plotlist
    plots <- c(list(...), plotlist)
    numPlots = length(plots)
    # If layout is NULL, then use 'cols' to determine layout
    if (is.null(layout)) {
        # Make the panel ncol: Number of columns of plots nrow:
        # Number of rows needed, calculated from # of cols
        layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
            ncol = cols, nrow = ceiling(numPlots/cols))
    }

    if (numPlots == 1) {
        print(plots[[1]])
    } else {
        # Set up the page
        grid.newpage()
        pushViewport(viewport(layout = grid.layout(nrow(layout),
            ncol(layout))))
        # Make each plot, in the correct location
        for (i in 1:numPlots) {
            # Get the i,j matrix positions of the regions that contain
            # this subplot
            matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))
            print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                layout.pos.col = matchidx$col))
        }
    }
}
```

Next, we'll create the main function to plot the stations for each city. The inputs here are again the list of cities, full list of NOAA ISD weather stations, and the k-number of stations you wish to return. The output will be a map for each city, with one black dot and up to k red dots (numbered). The black dot is the reference point for that city and the red dots represent the weather stations, labeled 1 to k, ordered from closest to farthest away from the reference point.

```
plotStations <- function(city.list, station.list, k) {
    kns <- kNStations(city.list, station.list, k)
    nc <- length(city.list)
    plots <- list()
    for (i in 1:nc) {
        map <- suppressMessages(get_map(location = city.list[i],
            zoom = 10))
```

```
        p1 <- suppressMessages(ggmap(map) + geom_point(aes(x = LON,
            y = LAT), data = kns[kns$city == city.list[i], ],
            colour = "red", size = 7, alpha = 0.5) + geom_text(aes(x = LON,
            y = LAT, label = rank), data = kns[kns$city == city.list[i],
            ]) + geom_point(aes(x = lon, y = lat), data = geocode(city.list[i]),
            colour = "black", size = 7, alpha = 0.5)) + labs(title = city.list[i]) +
            theme(plot.margin = unit(c(0, 0, 0, 0), "mm"))
        plots[[i]] <- p1
    }
    if (nc == 1)
        plot(p1) else multiplot(plotlist = plots, cols = round(sqrt(nc)))
}


# Run the function:
plotStations(cities.of.interest[1:4], stations, 7)
```



plotStations-1.pdf

We can see that some cities have a lot of weather stations nearby, where as others only have a few. Now that we've confirmed that there should be at least one station close to each city, let's create a function to download the data.

Downloading Data via FTP

The data we'll be downloading are fixed width, and saved on the ftp in `.gz` format. The widths and column names of the station data is as follows.

```
col.width <- c(4, 6, 5, 4, 2, 2, 2, 2, 1, 6, 7, 5, 5, 5, 4, 3,
    1, 1, 4, 1, 5, 1, 1, 1, 6, 1, 1, 1, 5, 1, 5, 1, 5, 1)

col.names <- c("CHARS", "USAFID", "WBAN", "YR", "M", "D", "HR",
    "MIN", "DATE.FLAG", "LAT", "LONG", "TYPE.CODE", "ELEV", "CALL.LETTER",
    "QLTY", "WIND.DIR", "WIND.DIR.QLTY", "WIND.CODE", "WIND.SPD",
    "WIND.SPD.QLTY", "CEILING.HEIGHT", "CEILING.HEIGHT.QLTY",
    "CEILING.HEIGHT.DETERM", "CEILING.HEIGHT.CAVOK", "VIS.DISTANCE",
    "VIS.DISTANCE.QLTY", "VIS.CODE", "VIS.CODE.QLTY", "TEMP",
    "TEMP.QLTY", "DEW.POINT", "DEW.POINT.QLTY", "ATM.PRES", "ATM.PRES.QLTY")
```

Now we can begin building a function that will download data for a range of years for each of the k-nearest stations to each city that is found in the `kNStations` function. The function will have 3 inputs: the result of the `kNStations` function, a beginning year, and an ending year (both in 4-digit format).

```
dlStationData <- function(kns, beg, end) {
    base.url <- "ftp://ftp.ncdc.noaa.gov/pub/data/noaa/"
    nstations <- nrow(kns)
    yrs <- seq(beg, end, 1)
    nyrs <- end - beg + 1
    usaf <- as.numeric(kns$USAF)
    wban <- as.numeric(kns$WBAN)
    # The following dataframe will be printed to show which files
    # were successfully downloaded
    status <- data.frame()
    temp <- as.data.frame(matrix(NA, nstations, 5))
    names(temp) <- c("File", "Status", "City", "rank", "kilo_distance")
    temp$City <- kns$city
    temp$rank <- kns$rank
    temp$kilo_distance <- kns$kilo_distance
    # Setup for the list of data
    temp.list <- df.list <- list()
    city.names <- unlist(lapply(strsplit(kns$city, ", "), function(x) x[1]))  # City Name
    df.names <- paste(city.names, kns$USAF, sep = "_")  # City Name_USAF#

    # Download the desired stations into a list (does not save to
    # disk)
    for (i in 1:nyrs) {
        for (j in 1:nstations) {
            # Create file name
            temp[j, 1] <- paste(usaf[j], "-", wban[j], "-", yrs[i],
                ".gz", sep = "")
            tryCatch({
                # Create connect to the .gz file
                gz.url <- paste(base.url, yrs[i], "/", temp[j,
                  1], sep = "")
                con <- gzcon(url(gz.url))
                raw <- textConnection(readLines(con))
                # Read the .gz file directly into R without saving to disk
                temp.list[[j]] <- read.fwf(raw, col.width)
                close(con)
                # Some housekeeping:
                names(temp.list)[j] <- df.names[j]
```

```
                names(temp.list[[j]]) <- col.names
                temp.list[[j]]$LAT <- temp.list[[j]]$LAT/1000
                temp.list[[j]]$LONG <- temp.list[[j]]$LONG/1000
                temp.list[[j]]$WIND.SPD <- temp.list[[j]]$WIND.SPD/10
                temp.list[[j]]$TEMP <- temp.list[[j]]$TEMP/10
                temp.list[[j]]$DEW.POINT <- temp.list[[j]]$DEW.POINT/10
                temp.list[[j]]$ATM.PRES <- temp.list[[j]]$ATM.PRES/10
                temp.list[[j]]$city <- city.names[j]
                temp.list[[j]]$distance <- kns$kilo_distance[j]
                temp.list[[j]]$rank <- kns$rank[j]
                temp[j, 2] <- "Success"
            }, error = function(cond) {
                return(NA)
                next
            }, finally = {
                # if any of the files didn't download successfully, label as
                # such
                if (is.na(temp[j, 2]) == "TRUE")
                  temp[j, 2] <- "Failed"
            })
        }
        # Combine each year's status and list
        status <- rbind(status, temp)
        status <- status[order(status[, 3], status[, 4], status[,
            1]), ]
        df.list <- append(df.list, temp.list)
    }
    output.list <- list(status, df.list)
    names(output.list) <- c("dl_status", "station_data")
    return(output.list)
}
```

The output of the above function gives a list of two items. 1) The status of downloading data for each year of each station. It will show either "Failed" or "Success" 2) A list of dataframes for each year of station.

The next step is to reduce the number of dataframes we need to work with by combining the data of each year for the same station.

Data Compilation

```
combineWeatherDFs <- function(dfList) {
    combined.list <- list()
    keys <- unique(names(dfList$station_data))
    keys <- keys[keys != ""]
    nkeys <- length(keys)
    for (i in 1:nkeys) {
        track <- which(names(dfList$station_data) == keys[i])
        combined.list[[i]] <- as.data.frame(rbindlist(dfList$station_data[track]))
        names(combined.list)[i] <- keys[i]
    }
    output.list <- list(dfList$dl_status, combined.list)
    names(output.list) <- c("dl_status", "station_data")
```

```
    return(output.list)
}
```

We're now ready to put this all together into one function:

```
getStationsByCity <- function(city.list, station.list, k, begin,
    end) {
    kns <- kNStations(city.list, station.list, k)
    weatherDFs <- dlStationData(kns, begin, end)
    combined.list <- combineWeatherDFs(weatherDFs)
    return(combined.list)
}
```

Data Filters

This is essentially the guts of the package, however we need to perform two more steps for our future analysis. First, we only need one weather station for each city. Thus far, we have not filtered out any stations so the output of the function above could have up to k-stations for each city. We will establish minimum standards for each dataset, evaluate each station's data, then select station containing the best data. Second, we need observations at every hour of every year. No matter how good the weather station is, it is bound to have some missing observations. Therefore, we will use a method linear interpolation to estimate the missing observations.

These two steps will be add-ons to the function above, so let's start by creating a filtering mechanism. We have four steps in our filtering/selection process: 1) Remove stations with little to no data 2) Remove stations that exceed a maximum distance from each city's reference point 3) Remove stations that exceed a threshold of missing data, including NA values 4) Select closest station remaining for each city, as all remaining stations are deemed adequate

```
filterStationData <- function(comb.list, distance, hourly_interval,
    tolerance, begin, end) {

    dlStatus <- comb.list$dl_status
    comb.list <- comb.list$station_data

    city.names <- unlist(lapply(comb.list, function(x) unique(x$city)))

    # 1) remove stations with little to no data at all
    rm.junk <- names(comb.list[which(sapply(comb.list, function(x) dim(x)[1] <=
        10))])
    comb.list <- comb.list[which(sapply(comb.list, function(x) dim(x)[1] >
        10))]

    # 2) remove stations that exceed maximum distance
    rm.dist <- names(comb.list[which(sapply(comb.list, function(x) max(x["distance"])) >
        distance)])
    comb.list <- comb.list[which(sapply(comb.list, function(x) max(x["distance"])) <
        distance)]

    # keep track of which stations have been removed
    rm.tmp <- unique(c(rm.junk, rm.dist))

    lapply(comb.list, names)
```

```
# 3.a) remove stations that exceed threshold of missing data,
# start with counting the 999s:
cl <- c("TEMP", "DEW.POINT")  # Additional columns can be added
ix <- ix.tmp <- NULL
ix.ct <- as.data.frame(matrix(nrow = length(comb.list), ncol = length(cl)))
colnames(ix.ct) <- cl
rownames(ix.ct) <- names(comb.list)
for (L in 1:length(comb.list)) {
    for (i in 1:length(cl)) {
        ix.tmp <- which(comb.list[[L]][cl[i]] == 999.9 |
            comb.list[[L]][cl[i]] == 999 | comb.list[[L]][cl[i]] ==
            99.99)
        ix.ct[L, i] <- length(ix.tmp)
        ix <- union(ix, ix.tmp)
    }
    comb.list[[L]] <- comb.list[[L]][-ix, ]
}
ix.ct$temp_pct <- ix.ct[, cl[1]]/unlist(lapply(comb.list,
    nrow))
ix.ct$dew_pct <- ix.ct[, cl[2]]/unlist(lapply(comb.list,
    nrow))
# print(ix.ct)

# ms.obs <- (ix.ct$TEMP)+(ix.ct$DEW.POINT)

# 3.b) set a minimum number of observations and remove
# stations that do not meet requirement
yrs <- seq(begin, end, 1)
nyrs <- end - begin + 1
min.obs <- (24/hourly_interval) * 365 * nyrs * (1 - tolerance)
obs.ix <- which(sapply(comb.list, nrow) < min.obs)
rm.obs <- names(comb.list[which(sapply(comb.list, nrow) <
    min.obs)])
if (length(rm.obs) == 0)
    comb.list <- comb.list else comb.list <- comb.list[-obs.ix]

# update removed stations
rm.all <- unique(c(rm.tmp, rm.obs))

# 4) All current stations are assumed to be adequet, we
# therefore will take the closest to each reference point
kept.names <- substr(names(comb.list), 1, nchar(names(comb.list)) -
    7)
kept.ranks <- unname(unlist(lapply(comb.list, function(x) x["rank"][1,
    1])))
f.df <- data.frame(location = kept.names, ranks = kept.ranks)
kp.ix <- as.numeric(rownames(f.df[which(ave(f.df$ranks, f.df$location,
    FUN = function(x) x == min(x)) == 1), ]))
final.list <- comb.list[kp.ix]

# Show what was removed during the filtering process:
kept <- names(comb.list)
st.df <- data.frame(count(city.names))
```

```
    rm.df <- count(substr(rm.all, 1, nchar(rm.all) - 7))
    kept.df <- count(substr(kept, 1, nchar(kept) - 7))
    df.list <- list(st.df, rm.df, kept.df)
    mg.df <- Reduce(function(...) merge(..., by = "x", all = T),
        df.list)
    suppressWarnings(mg.df[is.na(mg.df)] <- 0)
    colnames(mg.df) <- c("city", "stations", "removed", "kept")
    filterStatus <- mg.df

    # Show the stations that will be in the final output:
    finalStations <- names(final.list)

    # Create a list for output
    finalOutput <- list(dlStatus, filterStatus, finalStations,
        final.list)
    names(finalOutput) <- c("dl_status", "removed_rows", "station_names_final",
        "station_data")
    return(finalOutput)
}
```

This function returns a list of four items. The download status, which is the same as prior functions, a dataframe showing the number and percentage of rows that were filled with NA values, the names of the stations selected in the last step, and the data of those stations. It should be noted that the fourth item is a list of dataframes.

At this point, you could add a line for `filterStationData` to the `getStationsByCity`, however we still need to interplate missing values. This intermediate step is included as a separate function in the package, however we will skip it here.

Data Interpolation

The final step is to create a function to interpolate all the missing values in each station's dataset.

```
interpolateData <- function(wx.list) {
    clean.list <- lapply(wx.list, function(x) {
        ddply(x, .(city, USAFID, distance, rank, YR, M, D, HR),
            summarise, LAT = mean(LAT), LONG = mean(LONG), ELEV = mean(ELEV),
            TEMP = mean(TEMP), DEW.POINT = mean(DEW.POINT))
    })

    # Create a column with the full posix date for each hour
    for (i in 1:length(clean.list)) {
        clean.list[[i]]$dates <- as.POSIXct(paste(paste(clean.list[[i]]$YR,
            "-", clean.list[[i]]$M, "-", clean.list[[i]]$D, " ",
            clean.list[[i]]$HR, sep = ""), ":", 0, ":", 0, sep = ""),
            "%Y-%m-%d %H:%M:%S", tz = "UTC")
    }

    # Create a list of dataframes of each hour
    hourly.list <- list()
    for (i in 1:length(clean.list)) {
        hourly.list[[i]] <- data.frame(hours = seq(from = as.POSIXct(paste(min(clean.list[[i]]$YR),
            "-1-1 0:00", sep = ""), tz = "UTC"), to = as.POSIXct(paste(max(clean.list[[i]]$YR),
```

```
            "-12-31 23:00", sep = ""), tz = "UTC"), by = "hour"))
    }

    wx.df <- data.frame()
    for (i in 1:length(clean.list)) {
        temp.df <- merge(hourly.list[[i]], clean.list[[i]], by.x = "hours",
            by.y = "dates", all.x = TRUE)
        temp.df$city <- unique(na.omit(temp.df$city))[1]
        temp.df$USAFID <- unique(na.omit(temp.df$USAFID))[1]
        temp.df$distance <- unique(na.omit(temp.df$distance))[1]
        temp.df$rank <- unique(na.omit(temp.df$rank))[1]
        temp.df$LAT <- unique(na.omit(temp.df$LAT))[1]
        temp.df$LONG <- unique(na.omit(temp.df$LONG))[1]
        temp.df$ELEV <- unique(na.omit(temp.df$ELEV))[1]
        temp.df$YR <- as.numeric(format(temp.df$hours, "%Y"))
        temp.df$M <- as.numeric(format(temp.df$hours, "%m"))
        temp.df$D <- as.numeric(format(temp.df$hours, "%d"))
        temp.df$HR <- as.numeric(format(temp.df$hours, "%H"))

        # Interpolation
        temp.int <- approx(x = temp.df$hours, y = temp.df$TEMP,
            xout = temp.df$hours)
        temp.df$TEMP <- temp.int$y
        dew.int <- approx(x = temp.df$hours, y = temp.df$DEW.POINT,
            xout = temp.df$hours)
        temp.df$DEW.POINT <- dew.int$y

        # Merge the dataframes together
        wx.df <- rbind(wx.df, temp.df)
    }
    return(wx.df)
}
```

The output of this function is a single, large, dataframe with hourly observations for the station with the 'best' data for each city.

Putting It All Together

Putting it all together, we end with the following function which will then feed data in the next step of our analysis.

```
getInterpolatedDataByCity <- function(city.list, station.list,
    k, begin, end, distance, hourly_interval, tolerance) {
    kns <- kNStations(city.list, station.list, k)
    weatherDFs <- dlStationData(kns, begin, end)
    combined.list <- combineWeatherDFs(weatherDFs)
    filteredData <- filterStationData(combined.list, distance,
        hourly_interval, tolerance, begin, end)
    interpolation <- interpolateData(filteredData$station_data)
    return(interpolation)
}
```

References

Segal, M., H. Shafir, M. Mandel, P. Alpert and Y. Balmor (1992). Climatic-related Evaluations of the Summer Peak-hours' Electric Load in Israel, *Journal of Applied Meteorology* 31 (1992): 1492-1498.

Thatcher, M. J. (2007). Modelling Changes to Electricity Demand Load Duration Curves as a Consequency of Predicted Climate Change for Australia. *Energy* 32 (2007): 1647-1659.

Hellmuth M.E., Osgood D.E., Hess U., Moorhead A. and Bhojwani H. (eds) 2009. *Index insurance and climate risk: Prospects for development and disaster management.* Climate and Society No. 2. International Research Institute for Climate and Society (IRI), Columbia University, New York, USA.

Rogers, David and Vladimir Tsirkunov (2011). *Implementing Hazard Early Warning Systems.* Global Facility for Disaster Reduction and Recovery (GFDRR) in support of strengthening Weather and Climate Information and Decision Support Systems (WCIDS).

Lambert T., Gilman P., Lilienthal P., Micropower system modeling with HOMER, Integration of Alternative Sources of Energy, Farret FA, Simões MG, John Wiley & Sons, December 2005, ISBN 0471712329

Lilienthal P.D., Lambert T.W., Gilman P., Computer modeling of renewable power systems, Cleveland CJ, editor-in-chief, Encyclopedia of Energy, Elsevier Inc., Volume 1, pp. 633-647, NREL Report No. CH-710-36771, 2004