Guide 2: Cleaning and Pre-Processing Weather Station Data

Marc Los Huertos

February 8, 2024 (ver. 0.88)

1 Introduction

1.1 Goals

The goal of this guide is to provide a step-by-step process for cleaning and pre-processing weather station data. The data is from the Global Historical Climatology Network (GHCN) Daily dataset. The data is available from the National Oceanic and Atmospheric Administration (NOAA) and is available from the National Centers for Environmental Information (NCEI) at https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatol

1.2 Background

1.2.1 What is GHCN-Daily (GNCNd)?

The new dataset is referred to as GHCN-Daily, version 3. The new dataset has a number of improvements over the older dataset, including a more comprehensive and robust quality assurance process. The new dataset also includes more station records and is updated more frequently than the older dataset.

The new dataset is also available in a more user-friendly format, including a set of pre-packaged data files that are available for download from the NCEI website. However, devoloping instructions to interact with their website is way to time consuming! So, we'll use the R file transfer protocol, via https, to download the data.

In 2011, there was a major update to the daily updates to the GHCN dataset: **??**IMMEDIATE – Changes to COOP Daily Form Access. It's a useful history.

1.3 Approach

We need to address several things that might get in the way of our analysis:

1. Read csv files into R (from station.csv files)
2. Fix date format (convert to POSIXct, adding month and year as variables)

3. Convert units (VALUE: PRCP = 0.1 mm, TMAX/TMIN = 0.1 C)
4. Evaluate missing data (Coverage.fun)
5. Evalutae for Outliers (QACQ.fun)
6. Other stuff??

2 Cleaning and Pre-Processing Functions

2.1 Before Starting the Process

Before you begin, make sure you have the stations to read into R. Got to the "Files" tab in RStudio and make sure you see the station csv files in your data folder. If not, please Slack Marc and mentors, so we can troubleshoot the issue!

2.2 R Code with Custom Functions

From the Canvas page, go to the Guide2functions.R file and download the file to your computer. Then upload the file to Rstudio directory you are using for the project.

Open the file in Rstudio and run the code, using the "source". button near the top of the editor window.

Run the Guide2functions.R code and the functions will be loaded into your environment automatically. Be sure the function has been updated to 2024-02-07!

2.3 Function Descriptions and Use

The following custom functions are used to read the weather stations data into R, clean and pre-process data so we can analyze the data with the next guide.

2.4 Reading csv and Checking dataframe objects

Read all csv files into R The data are now ready to be read into R environment, we will read them in using the following function:

Function: ReadStations2.fun()

Use this function read the stations in the R Environment.

Example of how to use the function:

```
datafolder = "/home/mwl04747/RTricks/04_Regional_Climate_Trends/Data/SP24/"
ReadStations2.fun(datafolder)
```

Check the dataframes in the environment You can do this by running the following code:

```
ls()
```

```
##  [1] "CleanUp.fun"          "coverage.fun"         "datafolder"
##  [4] "fixDates.fun"         "fixValues.fun"        "MonthlyAnomalies.fun"
##  [7] "MonthlyNormals.fun"   "MonthlyValues.fun"    "QAQC.fun"
## [10] "ReadStations.fun"     "ReadStations2.fun"    "sortStations.fun"
## [13] "USC00040693"          "USC00041614"          "USC00042294"
## [16] "USC00043157"          "USC00044412"          "USC00046074"
## [19] "USC00046136"          "USC00046719"          "USC00046826"
## [22] "USC00047821"          "USC00047902"          "USC00047916"
## [25] "USC00048351"          "USC00049866"          "USW00023271"
```

Check the structure of the dataframes Using str(), make sure the datasets look right!

For example:

```
## 'data.frame': 225076 obs. of  8 variables:
##  $ ID      : chr  "USC00043157" "USC00043157" "USC00043157" "USC00043157" ...
##  $ DATE    : int  18670601 18670602 18670603 18670604 18670605 18670606 18670607 186
##  $ ELEMENT : chr  "PRCP" "PRCP" "PRCP" "PRCP" ...
##  $ VALUE   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ M.FLAG  : chr  "" "" "" "" ...
##  $ Q.FLAG  : chr  "" "" "" "" ...
##  $ S.FLAG  : chr  "F" "F" "F" "F" ...
##  $ OBS.TIME: int  NA NA NA NA NA NA NA NA NA NA ...
```

What to look for? Is the object a data.frame? Doee it have the right number of observations? Are the expected variable names present? Are the data types correct? Are values in each variable reasonable?

Sorting Stations by Number of Observations In theory, the stations with the greatest numbrer of observations are the the ones we will want to evaluate. However, we may also want to evaluate stations to cover a geographic range.

For now, let use this function to sort the stations by the number of observations and use this to focus are attention.

THIS IS A BRAND NEW IDEA FROM WEDNESDAY'S LAB, SO i AM GOING BE DEVELOPING THIS NOW, BUT DONT' USE THIS NOW. Instead look at the number of observation in the R environment to select 3 stations with the highest number of observations.

Function: sortstations.fun()

Example of how to use the function:

```
# sortstations.fun() Nor working yet.
```

2.5 Clean Data

Next we'll "clean" the dataset by fixing the date format and preparing it for the analysis stages.

Function to Fix Dates Function: fixDates.fun()

Example of how to use the function (Note: the new dataframe name change):

```
USC00042294a <- fixDates.fun(USC00042294)
```

Evaluation Data Coverage We need to know how much data we have for each station. This is important for the next steps in the process.

Function: coverage.fun()

Example of how to use the function:

```
coverage.fun(USC00042294a)
```

In general, we want something like 95% coverage for the period of record. If you don't have that, please let us know and we'll help you get additional stations with better coverage!

Function to Fix Values (by order of magnitude) According the the NOAA website, the csv.gz files have five core elements include the following units, which we will convert:

PRCP = Precipitation (tenths of mm) → mm

SNOW = Snowfall (mm) → cm

SNWD = Snow depth (mm) → cm

TMAX = Maximum temperature (tenths of degrees C) → degrees C

TMIN = Minimum temperature (tenths of degrees C) → degrees C

Function: fixValues.fun()

Example of how to use the function (Note: the new dataframe name change):

```
USC00042294b <- fixValues.fun(USC00042294a)
```

Checking for Outliers NOAA website conducts a very rudimentary data quality check, but every year, we find stations with wacky numbers. Hopefully this function will find them. But if you do have some, let Marc and mentors know so we can figure out how to address them!

Here are some stuff we'll look for:

Extreme Values Plot values with time, is the scale crazy with just a few observations at the extreme?

Sudden Shift ???

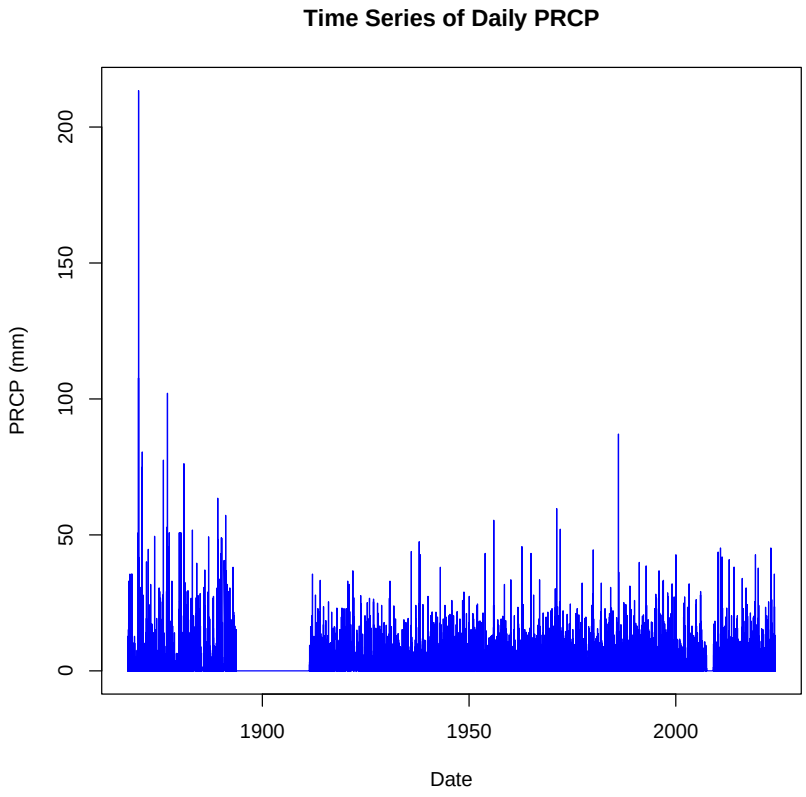
Note QA/QC Flags TBD

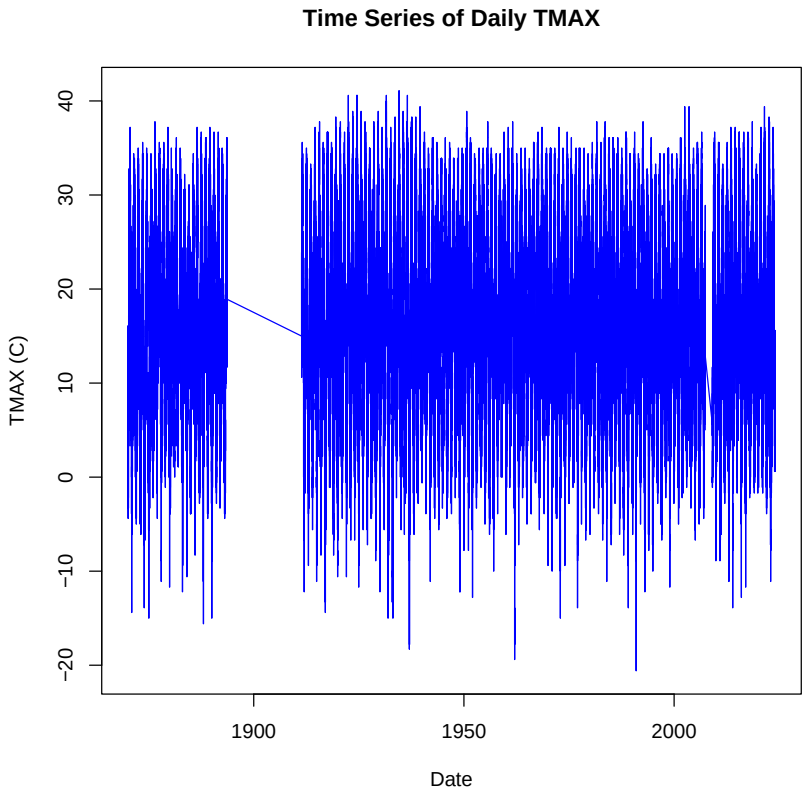
Yes, I am still working on this. QA/QC is a big deal and I want to make I can explain it AND make sure the code works.

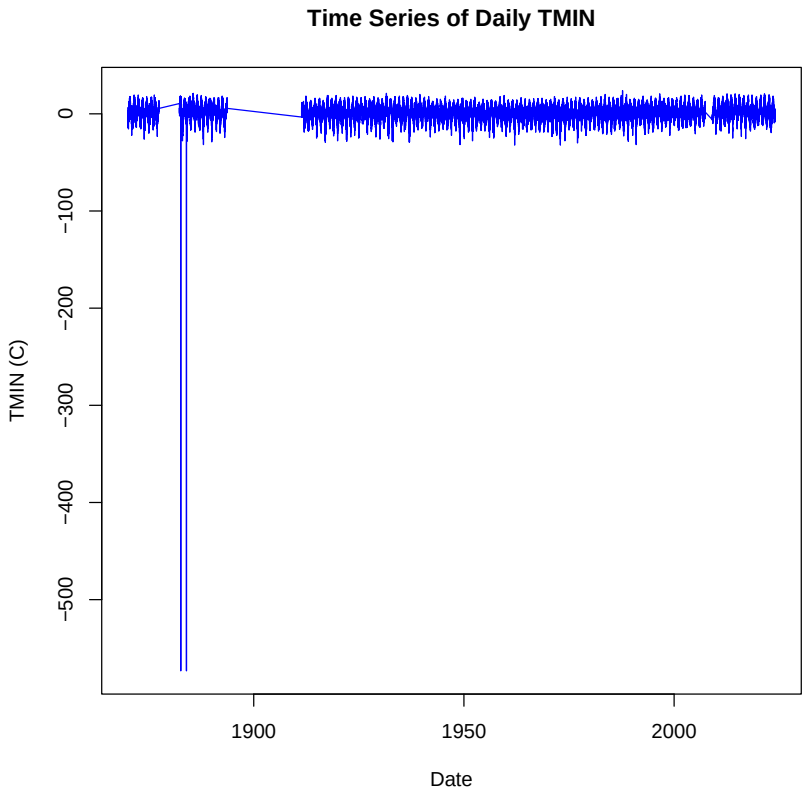
Function: QAQC.fun()

Example of how to use the function:

```
QAQC.fun(USC00042294b)
```


Time Series of Daily PRCP
PRCP (mm)
0
50
100
150
200
1900
1950
2000
Date


Time Series of Daily TMAX
TMAX (C)
40
30
20
10
0
−10
−20
1900
1950
2000
Date



If you have any hints of data problems, let's sort them out together!

Function to Create Monthly Values and Normals For TMAX and TMIN, we want monthly means, for rainfall, we'll want monthly totals.[1]

Function: MonthlyValues.fun()

Function: NormalValues.fun()

Example of how to use the functions:

```
USC00042294.monthly <- MonthlyValues.fun(USC00042294b)
USC00042294.normals <- MonthlyNormals.fun(USC00042294b)
```

[1] Brody/Evyln: We need code to exclue months with missing data, these might not be represenative of the month if missing, especically for PRCP!

Function to Create Anomalies Example of how to use the function:

```
USC00042294.anomalies <- MonthlyAnomalies.fun(USC00042294.monthly, USC00042294.normals)
```

2.6 Checking on the Results

Getting into the data is a bit tricky. The datasets is a list of dataframes. Each dataframe is a different variable, where 1 is TMAX, 2 is TMIN, and 3 is PRCP.

The get access to each you use the following code:

- `USC00042294.anomalies[[1]]` for TMAX
- `USC00042294.anomalies[[2]]` for TMIN
- `USC00042294.anomalies[[3]]` for PRCP

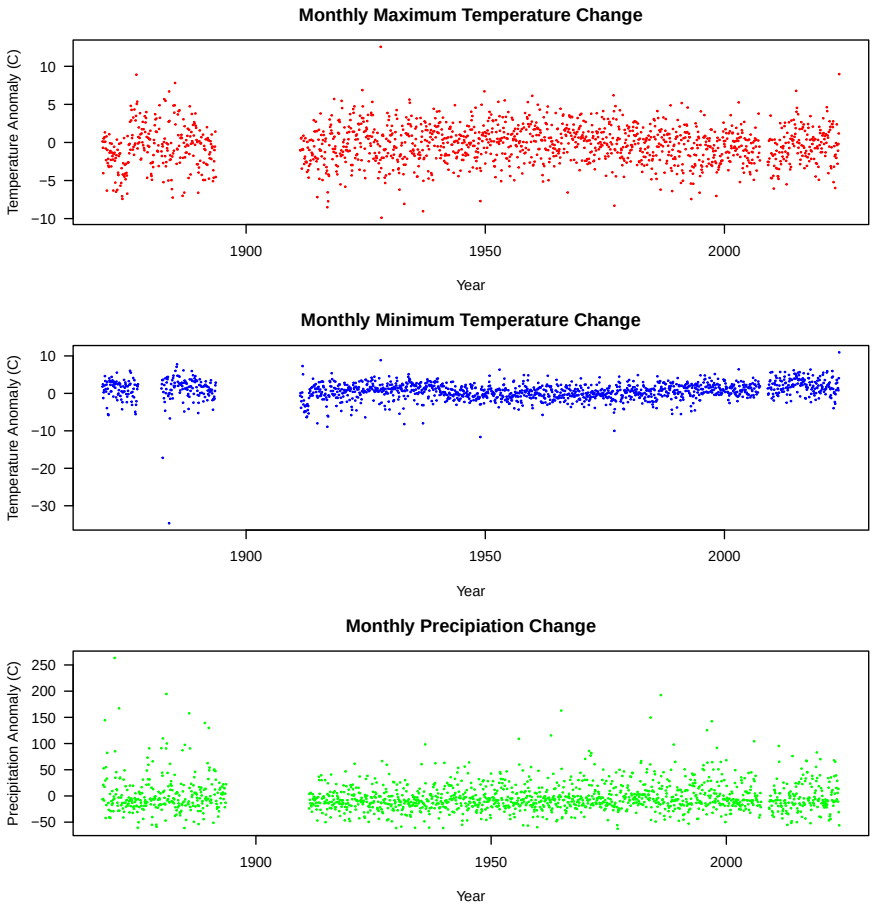
2.7 Clean Up R Environment

I tend to avoid having lots of objects in my R environment. I like to clean up after myself.

Also, I like to move the final products into csv files. Here's the function to do that. However, I noticed that it' deleting needed files. YIKES! It's also clunky, but not that important at this point, nevertheless, I'll work on it later.So, I suggest you don't run this code until I fix it.

```
CleanUp.fun(datafolder, USC00042294.anomalies, "USC00042294")
```

Your are done with this guide, now take a break, a walk, and enjoy some downtime without looking at a computer. Next, go to Guide 3!


Monthly Maximum Temperature Change
Temperature Anomaly (C)
10
5
0
−5
−10
1900
1950
2000
Year
Monthly Minimum Temperature Change
Temperature Anomaly (C)
10
0
−10
−20
−30
1900
1950
2000
Year
Monthly Precipiation Change
Precipitation Anomaly (C)
250
200
150
100
50
0
−50
1900
1950
2000
Year

3 Explaining the Marc's Custom Functions

3.1 ReadStations2.fun

```
## function (datafolder)
## {
##     my.stations = read.csv(paste0(datafolder, "my.inventory.cs
##     colnames <- c("ID", "DATE", "ELEMENT", "VALUE", "M-FLAG",
##         "Q-FLAG", "S-FLAG", "OBS-TIME")
##     for (i in 1:nrow(my.stations)) {
##         assign(my.stations$ID[i], read.csv(paste0(datafolder,
##             noquote(my.stations$ID[i]), ".csv"), header = TRUE
##             col.names = colnames), envir = parent.frame())
##     }
## }
## <bytecode: 0x2fcf400>
```

3.2 fixdates.fun

```
## function (station)
## {
##     station$Ymd = as.Date(as.character(station$DATE), format =
##     station$MONTH = as.numeric(format(station$Ymd, "%m"))
##     station$YEAR = as.numeric(format(station$Ymd, "%Y"))
##     return(station)
## }
```

3.3 ConvertUnits.fun

```
## function (station)
## {
##     station$VALUE = station$VALUE/10
##     return(station)
## }
```

3.4 QAQC.fun

```
## function (station)
## {
##     par(mfrow = c(1, 1))
##     plot(VALUE ~ Ymd, data = subset(station, subset = ELEMENT
##         "PRCP"), type = "l", col = "blue", main = "Time Series
##         xlab = "Date", ylab = "PRCP (mm)")
##     plot(VALUE ~ Ymd, data = subset(station, subset = ELEMENT
##         "TMAX"), type = "l", col = "blue", main = "Time Series
##         xlab = "Date", ylab = "TMAX (C)")
##     plot(VALUE ~ Ymd, data = subset(station, subset = ELEMENT
##         "TMIN"), type = "l", col = "blue", main = "Time Series
##         xlab = "Date", ylab = "TMIN (C)")
##     station = subset(station, Q.FLAG != "")
##     station = subset(station, M.FLAG != "")
##     station = subset(station, S.FLAG != "")
##     return(station)
## }
## <bytecode: 0x46d6668>
```

3.4.1 How to Evaluate QA/QC Problems

3.4.2 QA/QC Trouble Shooting

I will be getting all the guides working before working on this! But if there are errors with the custom function, this is where workarounds will be described! Please Slack me and mentors if you have any problems!

3.4.3 Coverage Problems (<95%)

```
## function (station, element = "TMAX")
## {
##     Dates.all = data.frame(Ymd = seq.Date(from = min(station$Y
##         to = max(station$Ymd), by = "day"))
##     station.full = merge(Dates.all, station, all = TRUE)
##     station.coverage = sum(!is.na(station.full$VALUE[station.f
##         element]))/length(station.full$VALUE[station.full$ELEM
##         element]) * 100
##     return(round(station.coverage, 2))
## }
## <bytecode: 0x228b598>
```

If you have too many stations that don't have enough data, we'll need to download additional stations. I can show you a way to decipher that ahead of time if you'd like to know. Otherwise, I suggest you double the number of stations selected in the code that generated my.inventory. I have increase the number of stations per state to 15, so perhaps that will give you enough to work with.

3.4.4 Missing Data

Similar to the problem above, missing data can be a problem. The real issue that I can see is that rainfall is so central because we use the data to calculate monthly totals, but if the month is missing data, then we have a severe bias. Same issue in temperature, but because we are looking at averages, it's less likely to be a major source of bias. But we should should check!

3.4.5 Plot Anomaly

Graphic has lots of issues. more next time! But here's a start.

```
options(scipen=5)
par(mar=c(4,6,2,5))

plot(ANOMALY ~ YEAR, data = subset(station1.TMAX, MONTH == 1),
     las=1, pch=19, col = "blue", cex=.5, #xlab = "Year",
     ylab = "Maximum Temp Anomaly (C)",
     main="January Maximum Temp Anomaly")
mtext("Maximum Temp Anomaly (C)", side = 2, line = 3)
temp.lm = lm(ANOMALY ~ YEAR, data = subset(station1.TMAX, MONTH =
abline(coef(temp.lm), col = "red")
```

We can see huge periods of time where no data was collecged. Yikes! I don't think I can use this station.

My custom functions are probably sensitive to missing values, need to work on that!

3.5 MonthlyValues.fun

Here's the function for Monthly Normals:

```
## function (x)
## {
##     x.normals = subset(x, Ymd >= "1961-01-01" & Ymd <= "1990-1
##     x.TMAX.normals.monthly = aggregate(VALUE ~ MONTH, data = s
##         ELEMENT == "TMAX"), mean)
##     names(x.TMAX.normals.monthly) <- c("MONTH", "NORMALS")
##     x.TMIN.normals.monthly = aggregate(VALUE ~ MONTH, data = s
##         ELEMENT == "TMIN"), mean)
##     names(x.TMIN.normals.monthly) <- c("MONTH", "NORMALS")
##     x.PRCP.normals.month.year = aggregate(VALUE ~ MONTH + YEAR
##         data = subset(x.normals, ELEMENT == "PRCP"), sum)
##     x.PRCP.normals.monthly = aggregate(VALUE ~ MONTH, data = s
##         mean)
##     names(x.PRCP.normals.monthly) <- c("MONTH", "NORMALS")
##     return(list(x.TMAX.normals.monthly, x.TMIN.normals.monthly
##         x.PRCP.normals.monthly))
## }
## <bytecode: 0x462b9d0>
```

Here's the function for Monthly Values:

```
## function (x)
## {
##     x.TMAX.monthly = aggregate(VALUE ~ MONTH + YEAR, data = su
##         ELEMENT == "TMAX"), mean)
##     names(x.TMAX.monthly) <- c("MONTH", "YEAR", "TMAX")
##     x.TMIN.monthly = aggregate(VALUE ~ MONTH + YEAR, data = su
##         ELEMENT == "TMIN"), mean)
```

```
##     names(x.TMIN.monthly) <- c("MONTH", "YEAR", "TMIN")
##     x.PRCP.monthly = aggregate(VALUE ~ MONTH + YEAR, data = su
##         ELEMENT == "PRCP"), sum)
##     names(x.PRCP.monthly) <- c("MONTH", "YEAR", "PRCP")
##     return(list(x.TMAX.monthly, x.TMIN.monthly, x.PRCP.monthly
## }
## <bytecode: 0x201bc10>
```

3.6 MonthlyAnomalies.fun

Here's the function:

```
## function (station.monthly, station.normals)
## {
##     for (i in seq_along(station.monthly)) {
##         TMAX <- merge(station.monthly[[1]], station.normals[[1
##             by = "MONTH")
##         TMAX$TMAX.a = TMAX$TMAX - TMAX$NORMALS
##         TMAX$Ymd = as.Date(paste(TMAX$YEAR, TMAX$MONTH, "01",
##             sep = "-"))
##         TMIN <- merge(station.monthly[[2]], station.normals[[2
##             by = "MONTH")
##         TMIN$TMIN.a = TMIN$TMIN - TMIN$NORMALS
##         TMIN$Ymd = as.Date(paste(TMIN$YEAR, TMIN$MONTH, "01",
##             sep = "-"))
##         PRCP <- merge(station.monthly[[3]], station.normals[[3
##             by = "MONTH")
##         PRCP$PRCP.a = PRCP$PRCP - PRCP$NORMALS
##         PRCP$Ymd = as.Date(paste(PRCP$YEAR, PRCP$MONTH, "01",
##             sep = "-"))
```

```
##         return(list(TMAX = TMAX, TMIN = TMIN, PRCP = PRCP))
##     }
## }
## <bytecode: 0x100dc408>
```

4 Trouble Shooting and Work Arounds

TBD!

5 Next Steps

5.1 Apply Function to All Stations

So far, I have only run function for 1 station, but I suspect you can figure out how to run it for each one!

This is all we need to do so far. Next week, we'll look at different way to visualize the data!

5.2 Clean Up R Environment

I'll save all the station data into csv files, then use them in the next guide to clean, process, and visualize data. I don't think the function is all that useful, so I can show you better ways of doing thin is class.

```
station1.clean=cleandataframe.fun(station1)
```