Guide 2: Cleaning and Pre-Processing Weather Station Data

Marc Los Huertos

September 9, 2025 (ver. 1.01)

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.ncei.noaa.gov/products/land-based-station/global-historical-climatology-network-daily.

1.2 Background

1.2.1 What is GHCN-Daily (GNCN-d)?

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. Evaluate for Outliers (QACQ.fun)
6. Visualize data using ggplot2 (PlotStation.fun)
7. Other stuff?? We will see what we need to do as we go along. The data evolve each year.

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.

Add the following code to your Rmd file and run the script to load the functions into your environment:

```
source("/../../Regional_Climate_Trends/Guide2functions.R")
```

NOTE: Modify the path to the location of the file in your R file structure!

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/05_Regional_Climate_Trends/Data/FA25/"
ReadStations2.fun(datafolder)
```

Remember that datafolder is a path of hierarchical folders and should end with ".../Data/" for your path. Why is mine different? Because I often have to help students and want to save the data in a unique path for each spring course.

List the objects in the R environment The ls() function will list the objects in the R environment. Use this in the console. Notice nothing goes inside the parethesis.

```
ls()
```

```
##  [1] "coverage.fun"          "datafolder"            "df_exists"
##  [4] "df_head"               "df_names"              "fixDates.fun"
##  [7] "fixValues.fun"         "MonthlyAnomalies.fun" "MonthlyNormals.fun"
## [10] "MonthlyValues.fun"     "QAQC.fun"              "QAQC2.fun"
## [13] "ReadStations2.fun"     "SaveCleanUp.fun"       "sortStations.fun"
## [16] "USC00260507"           "USC00261485"           "USC00262708"
## [19] "USC00264698"           "USC00264935"           "USC00264950"
## [22] "USC00265168"           "USC00267908"           "USC00268160"
## [25] "USC00268761"           "USC00268988"           "USC00269229"
## [28] "USW00023154"           "USW00024121"           "USW00024128"
```

What do you see? Your list of objects should include several things like mine: some weather stations, You should see objects named after the stations that have been read into R. In addition, you should see several functions (.fun) and other miscilenous objects. If so, you have been making progress. If you don't see something parallel to my list, let me or mentors know so we can trouble shoot, or look at section 4.

Check the structure of the dataframes By using `str()`, we can ensure the datasets look right!

The syntax requires an object, in our case, the object name for a weather stations's data. For example:

```
## 'data.frame': 488209 obs. of  8 variables:
##  $ ID      : chr  "USW00024128" "USW00024128" "USW00024128" "USW00024128" ...
##  $ DATE    : int  18770701 18770702 18770703 18770704 18770705 18770706 18770707 187
##  $ ELEMENT : chr  "TMAX" "TMAX" "TMAX" "TMAX" ...
##  $ VALUE   : int  233 272 261 278 289 278 328 350 361 333 ...
##  $ M.FLAG  : chr  "" "" "" "" ...
##  $ Q.FLAG  : chr  "" "" "" "" ...
##  $ S.FLAG  : chr  "0" "0" "0" "0" ...
##  $ OBS.TIME: int  NA NA NA NA NA NA NA NA NA NA ...
```

What to look for? Is the object a data.frame? Does 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 – Skip This In theory, the stations with the greatest number of observations are the the ones we will want to evaluate. However, we may also want to evaluate stations to cover a geographic range.

THIS IS A BRAND NEW IDEA FROM WEDNESDAY'S LAB, AND WILL WORK ON THIS FOR NEXT SEMESTER. 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() Not working yet.
```

2.5 Clean Data

Next we'll "clean" the dataset by fixing the date format and preparing it for the analysis stages. I suggest you select two stations with the highest number of observations to work with. We will know if this is a problem soon enough and if it is, we can use one of the 13 remaining stations, if your state/terriborty has that many to choose from.

Function to Fix Dates This function converts date values to a format that R can understand as a date, so we can, among other things, create montly means.

Function: fixDates.fun()

Example of how to use the function (Note: the new dataframe has a name change with an "a" following the station. This is so we can keep the original data and the processed data separate.):

```
USW00024128a <- fixDates.fun(USW00024128)
# USC00040693a <- fixDates.fun(USC00040693)
```

Evaluation Temporal 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(USW00024128a)
#coverage.fun(USC00040693a)
```

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!

See Section 3.4.3 for more information on how to evaluate the coverage of the data.

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 has a name change with a "b" following the station. This is so we can keep the original data and the processed data separate.):

```
USW00024128b <- fixValues.fun(USW00024128a)
# USC00040693b <- fixValues.fun(USC00040693a)
```

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 A sudden data shift in GHCNd (Global Historical Climatology Network - Daily) could indicate several potential issues or changes in the dataset. Some possible explanations include:

1. **Instrument Changes or Malfunctions** A shift might result from a change in measurement instruments, sensor recalibration, or instrument failure, leading to discontinuities in recorded values.
2. **Station Relocation** If a weather station is moved, even slightly, it can cause abrupt changes due to differences in elevation, local microclimates, or exposure.
3. **Changes in Observation Practices** Adjustments in how data is recorded, such as shifts in time-of-day observation or methodology (e.g., moving from manual readings to automated sensors), can introduce noticeable changes.
4. **Data Quality Issues or Corrections** The dataset may have been updated to correct errors, remove outliers, or introduce new quality control procedures, affecting trends.
5. **Climatic or Extreme Weather Events** A real, significant climate event, such as a heatwave, cold snap, or precipitation anomaly, could explain the sudden shift.
6. **Urbanization or Land Use Changes** Increased urbanization around a station (e.g., heat island effects) or changes in land use (e.g., deforestation) can cause long-term shifts in climate data.
7. **Metadata Errors or Missing Data Fill-ins** If there are discrepancies in metadata or if missing data is interpolated with estimates, sudden shifts can appear.

If you notice a sudden shift, it's worth cross-referencing station metadata, quality control flags, and surrounding station records to determine whether the change is due to an actual climatic event or a data-related issue.

QA/QC Flags in GHCNd In the GHCNd dataset, Quality Assurance (QA) and Quality Control (QC) flags are used to indicate potential issues or modifications in the data. These flags help users identify values that may be erroneous, adjusted, or estimated.

- **Types of QA/QC Flags:** Each reported data value in GHCNd can have three associated flags:
  - **Measurement Flag** Indicates if the data value was derived, adjusted, or estimated.
  - **Quality Flag** Identifies if a value failed a quality control check.
  - **Source Flag** Specifies the source of the data.

- **Measurement Flags (Optional):** Indicate if the data value was derived, estimated, or adjusted. Some common flags include:

Flag	Meaning
B	Adjusted to remove systematic bias (e.g., urban heat island effects)
D	Data value was from a delayed source
E	Estimated value
I	Derived from an intermediate source
M	Missing data
Q	Data value adjusted from original observation
S	Value was computed from multiple sources
T	Trace precipitation (small, nonzero amount)

Table 1: Measurement Flags in GHCNd

- **Quality Control (QC) Flags:** Indicate if a value has failed a quality check. If no QC flag is present, the value has passed all checks. Common QC flags include:

Flag	Meaning
D	Failed duplicate check
G	Failed gap check
I	Failed internal consistency check
K	Failed streak/frequent-value check
L	Failed check on length of multiday period
M	Failed climatological outlier check
N	Failed plausible value check
O	Failed observational consistency check
S	Failed spatial consistency check
T	Failed temporal consistency check
W	Temperature too warm for snow

Table 2: Quality Control Flags in GHCNd

- **Source Flags:** Indicate the origin of the data, such as manual observations, automated weather stations, or third-party sources. Common flags include:

- **Interpreting GHCNd QA/QC Flags:**
  - If a value **has no QC flag**, it has passed all quality checks.
  - A value with a **measurement flag "E"** means it was estimated and should be used with caution.
  - A QC flag like **"M" (outlier check failed)** suggests the value may be erroneous.
  - The **source flag** helps track where the data originated.

Flag	Meaning
0-9	Climate Reference Network (CRN) source code
C	Data from the Cooperative Observer Network (COOP)
G	Data from the Global Summary of the Day (GSOD)
H	Data from hydrological stations
R	Data from air force stations
S	Data from satellite-derived observations
Z	Data from summary statistics

Table 3: Source Flags in GHCNd

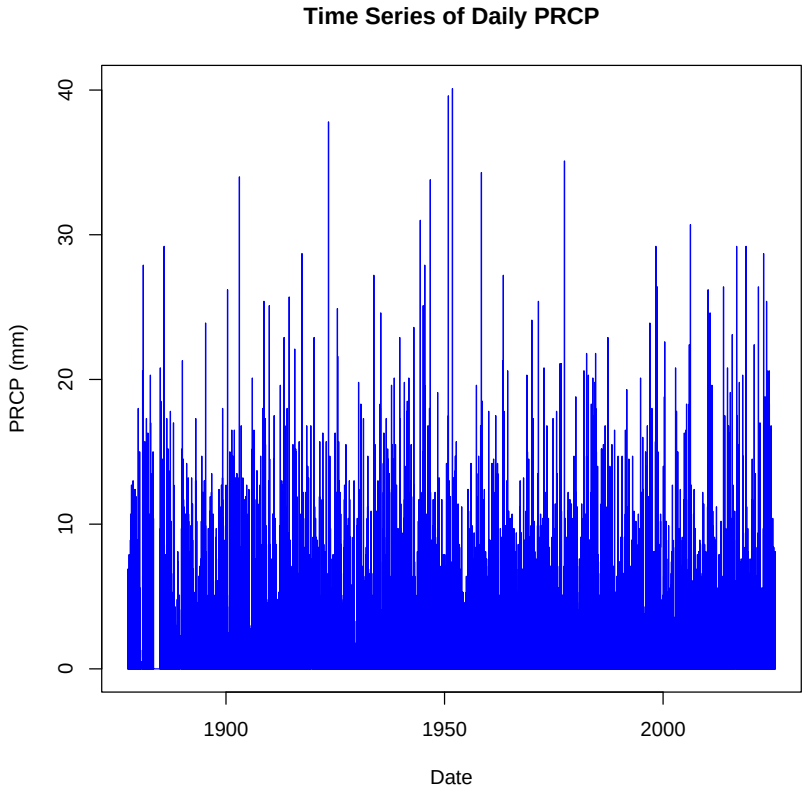
If you notice unexpected flags, consider cross-referencing metadata, station history, and nearby station records to determine whether the flagged values indicate a real climatic event or a data issue.

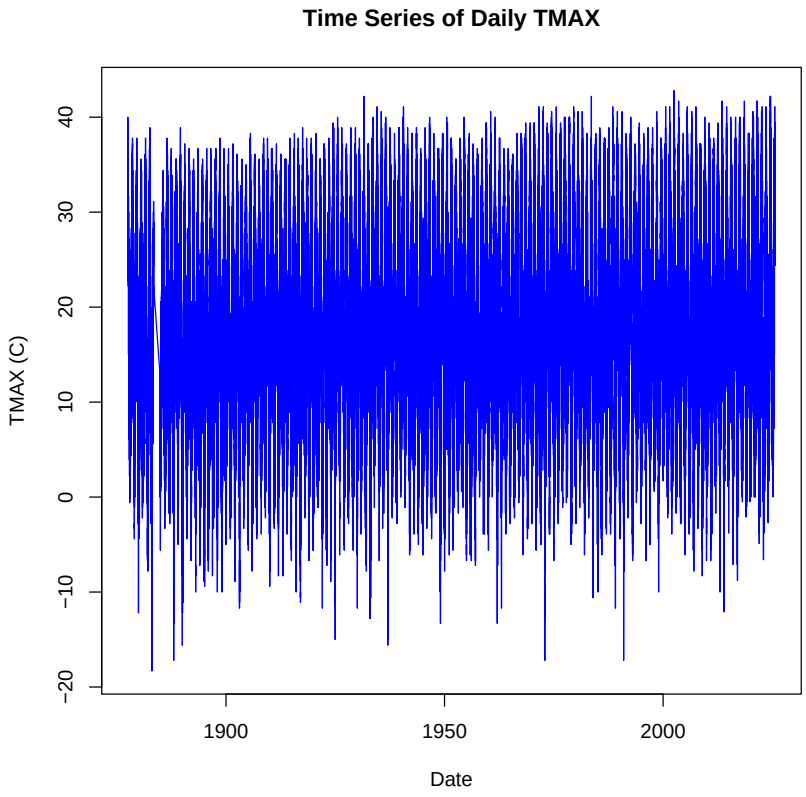
Function: QAQC.fun()

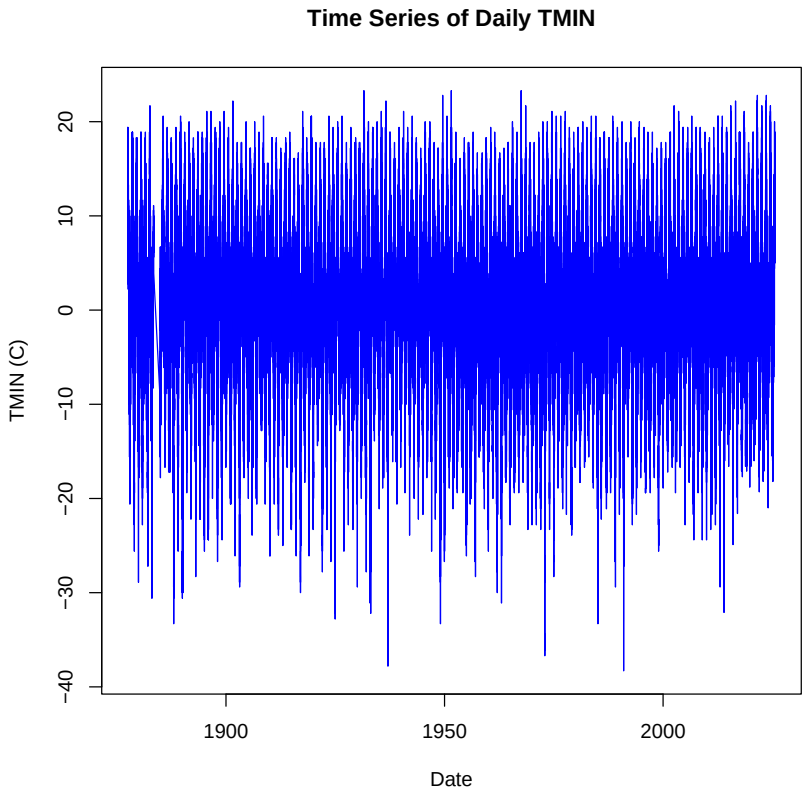
At this point, the function produced graphics and a summary of flags to examine the potential for QA/QC problems – in part becuase the issues seem rare. Maybe this will be the year where someone finds a problem that we need to address and I'll revise the function accordingly.

Example of how to use the function:

```
QAQC.fun(USW00024128b)
```


Time Series of Daily PRCP
PRCP (mm)
0
10
20
30
40
1900
1950
2000
Date


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

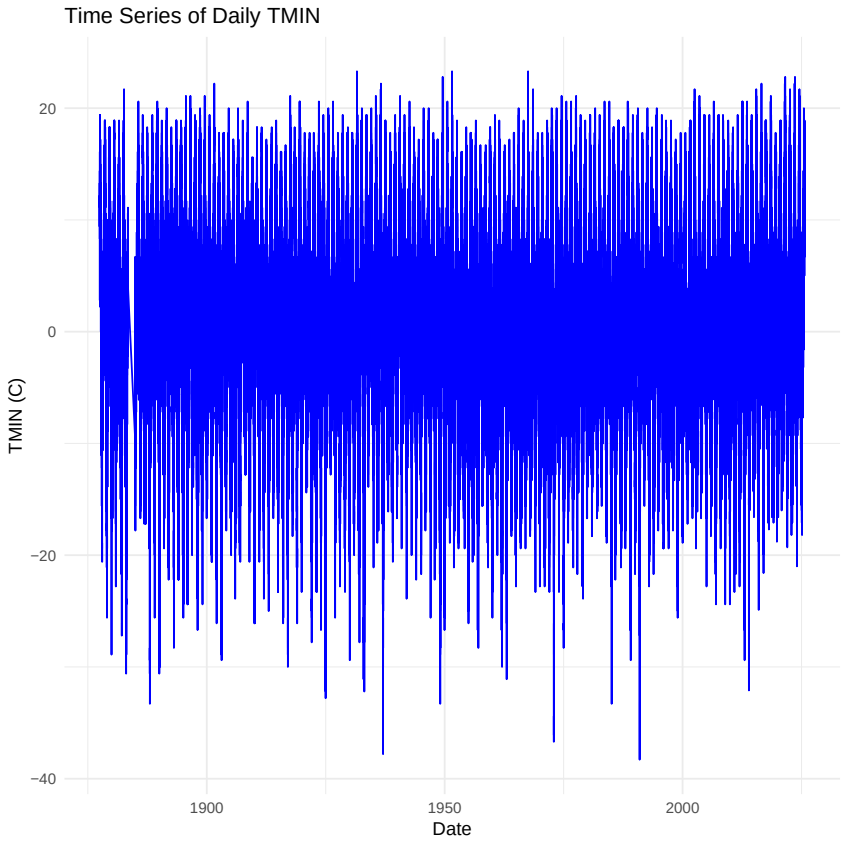



```
##  [1] ID       DATE     ELEMENT  VALUE    M.FLAG   Q.FLAG   S.FLAG   OBS.TIME
##  [9] Ymd      MONTH    YEAR
## <0 rows> (or 0-length row.names)

# QAQC.fun(USC00040693b)
```

No flags are reported here. If you have any hints of data problems (wacky values, odd patterns), let's sort them out together!

```
QAQC2.fun(USW00024128b)
```




```
# QAQC.fun(USC00040693b)
```

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:

```
USW00024128.monthly <- MonthlyValues.fun(USW00024128b)
USW00024128.normals <- MonthlyNormals.fun(USW00024128b)
#USC00040693.monthly <- MonthlyValues.fun(USC00040693b)
#USC00040693.normals <- MonthlyNormals.fun(USC00040693b)
```

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

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

```
USW00024128.anomalies <-
  MonthlyAnomalies.fun(USW00024128.monthly,USW00024128.normals)
#USC00040693.anomalies <-
#  MonthlyAnomalies.fun(USC00040693.monthly, USC00040693.normals)
```

2.6 Checking on the Results

You can double check that the dataframes you have been making are actually present by using the ls() function the console again.

```
ls()
```

```
##  [1] "coverage.fun"          "datafolder"            "df_exists"
##  [4] "df_head"               "df_names"              "fixDates.fun"
##  [7] "fixValues.fun"         "MonthlyAnomalies.fun"  "MonthlyNormals.fun"
## [10] "MonthlyValues.fun"     "QAQC.fun"              "QAQC2.fun"
## [13] "ReadStations2.fun"     "SaveCleanUp.fun"       "sortStations.fun"
## [16] "USC00260507"           "USC00261485"           "USC00262708"
## [19] "USC00264698"           "USC00264935"           "USC00264950"
## [22] "USC00265168"           "USC00267908"           "USC00268160"
## [25] "USC00268761"           "USC00268988"           "USC00269229"
## [28] "USW00023154"           "USW00024121"           "USW00024128"
## [31] "USW00024128.anomalies" "USW00024128.monthly"   "USW00024128.normals"
## [34] "USW00024128a"          "USW00024128b"
```

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:

- `USW00024128.anomalies[[1]]` for TMAX
- `USW00024128.anomalies[[2]]` for TMIN
- `USW00024128.anomalies[[3]]` for PRCP

2.7 Plotting the Results with ggplot2

2.7.1 Why is marc learning and teaching ggplot2?

I have been using R for a long time, and I have used base R graphics for most of that time. However, I have found that ggplot2 offers several advantages that make it worth learning and teaching:

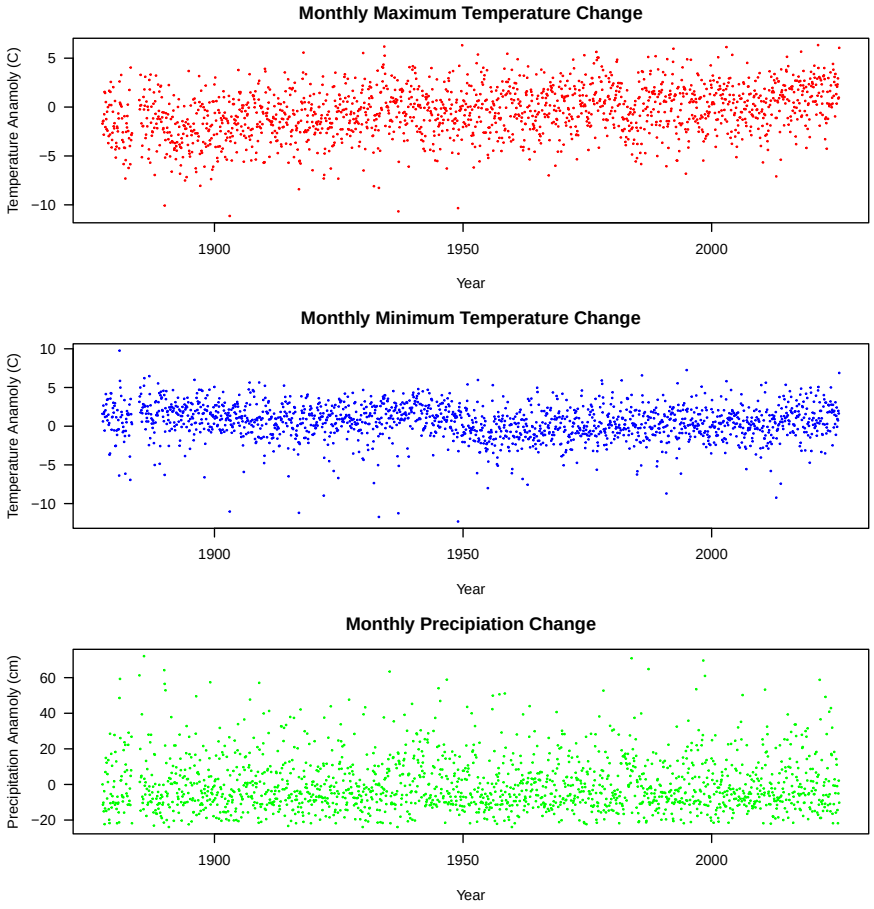


Figure 1: Here's a quick test to see if my data are coming out as expected and as a preview.

Consistency and Clarity ggplot2 is built upon the "Grammar of Graphics," a systematic framework for describing and building plots. This provides a logical and consistent way to map data variables to visual aesthetics (like color, shape, size) and geometric objects (like points, lines, bars). This systematic approach makes it easier to understand how plots are constructed and to create complex visualizations.

Layered Approach Plots in ggplot2 are built layer by layer. You start with a base plot and then add layers for geometric objects (e.g., geom_point(), geom_line()), statistical transformations (e.g., stat_smooth()), scales, coordinate systems, and themes. This modularity offers immense flexibility and control over every aspect of the plot.

Customization and Aesthetics ggplot2 offers extensive customization options for plot elements, including colors, fonts, labels, legends, and themes. This allows for the creation of publication-quality graphics with a high degree of aesthetic control.

Faceting The ability to easily create "facets" (sub-plots based on different subsets of the data) is a powerful feature for exploring relationships within data and comparing groups.

Consistency and Reusability The consistent syntax and layered structure make ggplot2 code highly reusable. Once you understand the grammar, you can apply the same principles to create a wide variety of plots, and modifications to the underlying data or desired plot type require minimal changes to the code.

Integration with Tidyverse ggplot2 is a core package within the Tidyverse, a collection of R packages designed for data science. This integration allows for seamless data manipulation and visualization workflows.

Community and Resources Due to its popularity, ggplot2 benefits from a large and active community, providing extensive documentation, online resources, and support for users.

Your are done with this guide, now take a break, a walk, and enjoy some screen free downtime. Next, go to Guide 3!

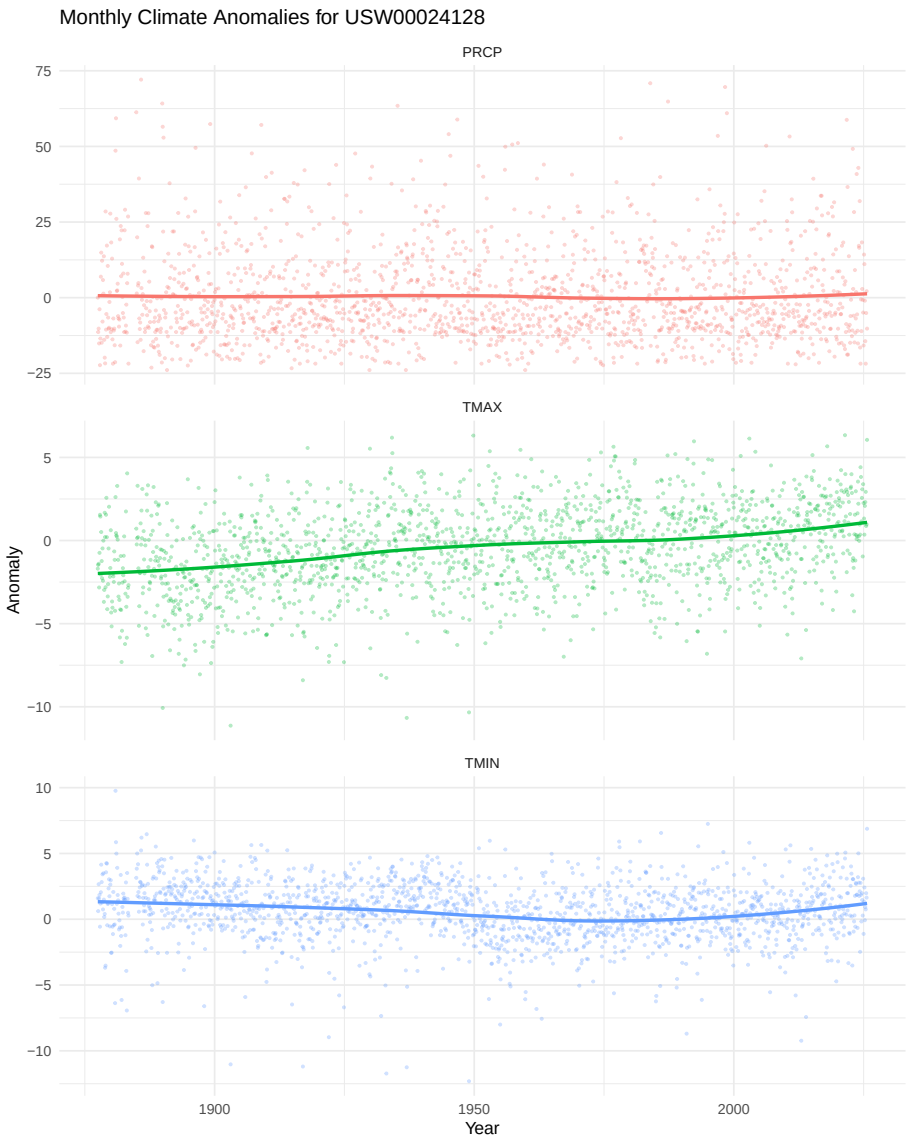


Figure 2: Here's a quick test to see if my data are coming out as expected and as a preview.

2.8 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, USW00024128.anomalies, "USW00024128")
```

For now, I am just saving the anomalies data into an RData file that I can use in other Rmd Guides by loading. I doubt you'll need to do that if you use only one Rmd file to knit all the functions together.

```
SaveCleanUp.fun(datafolder)
```

3 Describing Marc's Custom Functions

3.1 ReadStations2.fun

```
## function (datafolder)
## {
##     my.stations = read.csv(paste0(datafolder, "my.inventory.csv"))
##     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: 0x184d638>
```

3.1.1 Trouble Shooting

The function fails to read some csv files into the R environment for a variety of reasons that I haven't been able to solve.

To work around the issues, we'll have to read the csv filed directly. Here's how an example of how we might do it:

```
USW00024128 <- read.csv("data/USW00024128.csv")
```

3.2 fixdates.fun

```
## function (station)
## {
##     station$Ymd = as.Date(as.character(station$DATE), format = "%Y%m%d")
##     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 of Daily PRCP",
##         xlab = "Date", ylab = "PRCP (mm)")
##     plot(VALUE ~ Ymd, data = subset(station, subset = ELEMENT ==
##         "TMAX"), type = "l", col = "blue", main = "Time Series of Daily TMAX",
##         xlab = "Date", ylab = "TMAX (C)")
##     plot(VALUE ~ Ymd, data = subset(station, subset = ELEMENT ==
##         "TMIN"), type = "l", col = "blue", main = "Time Series of Daily TMIN",
##         xlab = "Date", ylab = "TMIN (C)")
##     station = subset(station, Q.FLAG != "")
##     station = subset(station, M.FLAG != "")
##     station = subset(station, S.FLAG != "")
##     return(station)
## }
## <bytecode: 0x14503728>
```

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$Ymd),
##         to = max(station$Ymd), by = "day"))
##     station.full = merge(Dates.all, station, all = TRUE)
##     station.coverage = sum(!is.na(station.full$VALUE[station.full$ELEMENT ==
##         element]))/length(station.full$VALUE[station.full$ELEMENT ==
##         element]) * 100
##     return(round(station.coverage, 2))
## }
## <bytecode: 0x20be9e8>
```

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.

If you get an error with this function, be sure you are using the correct file – in our case, it should end with an "a". If you are using the wrong file, you'll get an error.

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!

3.5 Failure to Read csv files

I noticed that the read.csv function is not working for some files. I'm not sure why, but I suspect it's because the files are too large. I'll work on this later. For now, let me know if you have any issues with this.

3.5.1 Plot Anomaly

Graphic has lots of issues and I am converting to ggplot (2025-26)!! – ut here's a start.

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

plot(TMAX.a ~ YEAR, data = subset(, 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 == 1))
abline(coef(temp.lm), col = "red")
```

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

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

3.6 MonthlyValues.fun

Here's the function for Monthly Normals:

```
## function (x)
## {
##     x.normals = subset(x, Ymd >= "1961-01-01" & Ymd <= "1990-12-31")
##     x.TMAX.normals.monthly = aggregate(VALUE ~ MONTH, data = subset(x.normals,
##         ELEMENT == "TMAX"), mean)
##     names(x.TMAX.normals.monthly) <- c("MONTH", "NORMALS")
##     x.TMIN.normals.monthly = aggregate(VALUE ~ MONTH, data = subset(x.normals,
##         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 = subset(x.PRCP.normals.month
##         mean)
##     names(x.PRCP.normals.monthly) <- c("MONTH", "NORMALS")
##     return(list(x.TMAX.normals.monthly, x.TMIN.normals.monthly,
##         x.PRCP.normals.monthly))
## }
## <bytecode: 0x144e988>
```

If you try to submit the wrong station file, e.g. 'USC00042294', you'll get an error. You need to submit the file that ends with an "b".

Here's the function for Monthly Values:

```
## function (x)
## {
##     x.TMAX.monthly = aggregate(VALUE ~ MONTH + YEAR, data = subset(x,
##         ELEMENT == "TMAX"), mean)
##     names(x.TMAX.monthly) <- c("MONTH", "YEAR", "TMAX")
##     x.TMIN.monthly = aggregate(VALUE ~ MONTH + YEAR, data = subset(x,
##         ELEMENT == "TMIN"), mean)
##     names(x.TMIN.monthly) <- c("MONTH", "YEAR", "TMIN")
##     x.PRCP.monthly = aggregate(VALUE ~ MONTH + YEAR, data = subset(x,
##         ELEMENT == "PRCP"), sum)
##     names(x.PRCP.monthly) <- c("MONTH", "YEAR", "PRCP")
##     return(list(x.TMAX.monthly, x.TMIN.monthly, x.PRCP.monthly))
## }
## <bytecode: 0x13ecd6a8>
```

If you try to submit the wrong station file, e.g. 'USC00042294', you'll get an error. You need to submit the file that ends with an "b".

3.7 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: 0x135af6f8>
```

If this function fails, it's likely because the products from the MonthlyNormals.fun or MonthlyValues.fun functions did not work or the products of these functions are not in the R environment and correctly specified in the function call.

4 Trouble Shooting and Work Arounds

4.1 Checking Results Step by Step

I generally check every step of the way to make sure the function is working. You can look the "Global Environment" in RStudio to see if the obects are there.

In addition, I have written some custom functions to evaluate the dataframes we have (hopefully) created!

- Does these object exist?

```
df_exists("USW00024128b")

## [1] TRUE
```

- What are the names within the dataframe?

```
df_names(USW00024128b)

##  [1] "ID"       "DATE"     "ELEMENT"  "VALUE"    "M.FLAG"   "Q.FLAG"
##  [7] "S.FLAG"   "OBS.TIME" "Ymd"      "MONTH"    "YEAR"
```

- What are the first few rows of the dataframe?

```
df_head(USW00024128.anomalies)

## $TMAX
##   MONTH YEAR      TMAX NORMALS     TMAX.a        Ymd
## 1     1 2004  3.029032    5.55 -2.5209677 2004-01-01
## 2     1 2001  4.248387    5.55 -1.3016129 2001-01-01
## 3     1 2003 11.683871    5.55  6.1338710 2003-01-01
## 4     1 1893  1.719355    5.55 -3.8306452 1893-01-01
## 5     1 1892  1.341935    5.55 -4.2080645 1892-01-01
## 6     1 2002  5.203226    5.55 -0.3467742 2002-01-01
##
## $TMIN
##   MONTH YEAR       TMIN   NORMALS      TMIN.a        Ymd
## 1     1 2004  -6.845161 -8.425591  1.58043011 2004-01-01
## 2     1 2001  -9.670968 -8.425591 -1.24537634 2001-01-01
## 3     1 2003  -2.612903 -8.425591  5.81268817 2003-01-01
## 4     1 1893 -10.948387 -8.425591 -2.52279570 1893-01-01
## 5     1 1892  -8.441935 -8.425591 -0.01634409 1892-01-01
## 6     1 2002  -7.038710 -8.425591  1.38688172 2002-01-01
##
## $PRCP
##   MONTH YEAR PRCP NORMALS PRCP.a        Ymd
## 1     1 2004 13.2    18.8   -5.6 2004-01-01
## 2     1 2001 14.7    18.8   -4.1 2001-01-01
## 3     1 2003 32.6    18.8   13.8 2003-01-01
## 4     1 1893 46.8    18.8   28.0 1893-01-01
## 5     1 1892 20.0    18.8    1.2 1892-01-01
## 6     1 2002 21.4    18.8    2.6 2002-01-01
```

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 Save and 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.

```
SaveCleanUp.fun(datafolder)
```

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