HRECOS data in R: a primer

This document is meant to serve as a guide for basic operations in R, using data from the Hudson River Environmental Conditions Observing System (HRECOS).

Contents

Using the dataRetrieval package

The United States Geological Survey (USGS) developed the dataRetrieval package to allow R users to get data from the National Water Information System (NWIS). You can use this package to get data from any USGS station in the country, but for this example we will focus on HRECOS sites. Note: These instructions are adapted from a post on the USGS website: https://waterdata.usgs.gov/blog/dataretrieval/

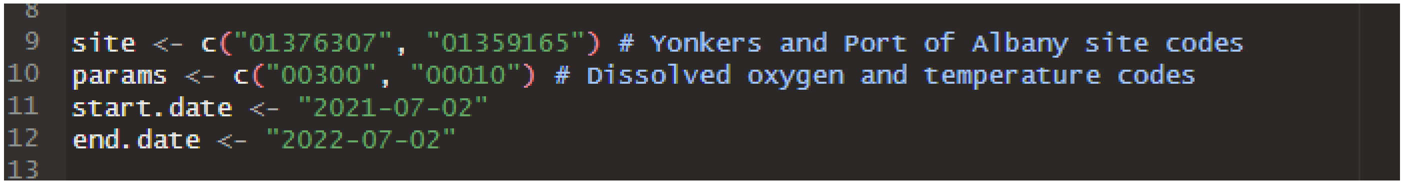
1. To install the package, run install.packages(“dataRetrieval”) to start
2. For this example, we will assume we already know what site and what data we want to download.
3. Let’s say we’re interested in the Yonkers and Port of Albany HRECOS stations dissolved oxygen and water temperature data.
4. First, we need the Site ID. Reference the table below or visit the HRECOS website to find the Site ID of your station(s).

Site Code	Site Name	Location	USGS Site ID
HRUTICA	Mohawk River at Ilion	Ilion Marina, Ilion, NY	01342732
HRLCK8	Mohawk River at Lock 8	Lock 8 Park, Schenectady, NY	01354330
HRREXBR	Mohawk River at Rexford	Rexford Bridge, Rexford, NY	01355475
HRALBP	Hudson River at Port of Albany	Albany, NY	01359165
HRSCHD	Hudson River near Schodack Landing	Schodack Island State Park, NY	0135980207
HRMARPHS	Hudson River near Poughkeepsie	Marist College, Poughkeepsie, NY	01372043
HRWSTPT	Hudson River at South Dock West Point	West Point, NY	01374019
HRPMNT	Hudson River at Piermont	Piermont Pier, NY	01376269
HRBZAK	Hudson River at Water Grant Street	Yonkers, NY	01376307
HRPR84	Hudson River at Pier 84	Pier 84, New York, NY	01376515
HRPR25	Hudson River at Pier 25	Pier 25, New York, NY	01376520
HRPVSC	Newark Bay at Oak Island Yards	Newark, NJ	404241074072202

5. Identify the parameter codes you need.
    a. https://help.waterdata.usgs.gov/codes-and-parameters/parameters
    b. Use param_codes <- readNWISpCode("all")
    c. Or reference the table below for the most common codes.

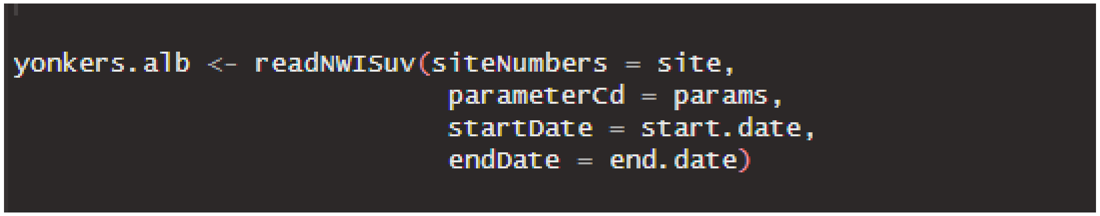
Parameter	**Code**
Hydrologic parameters	
Water temperature, Celsius	00010
Specific conductivity	00095
Dissolved oxygen, concentration	00300
Dissolved oxygen, percent saturation	00301
pH	00400
Turbidity	63680
Salinity	90860
Phycocyanin relative fluorescence (fPC) (RFU)	32321
Phycocyanin relative fluorescence (fPC) (ug/l)	32319
Chlorophyll relative fluorescence (fChl) (RFU)	32315
Chlorophyll relative fluorescence (fChl) (ug/l)	32316
Meteorologic parameters	
Air temperature, Celsius	00020
Wind direction	00036
Precipitation	00045
Relative humidity	00052
Wind gust speed	61727
Barometric pressure	75969
Wind speed	82127
PAR	99989

6. Set up variables.



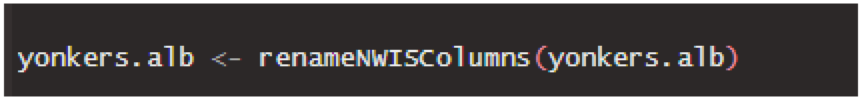
```r

site <- c("01376307", "01359165") # Yonkers and Port of Albany site codes
params <- c("00300", "00010") # Dissolved oxygen and temperature codes
start.date <- "2021-07-02"
end.date <- "2022-07-02"

```

7. Use the readNWISuv function as shown below:

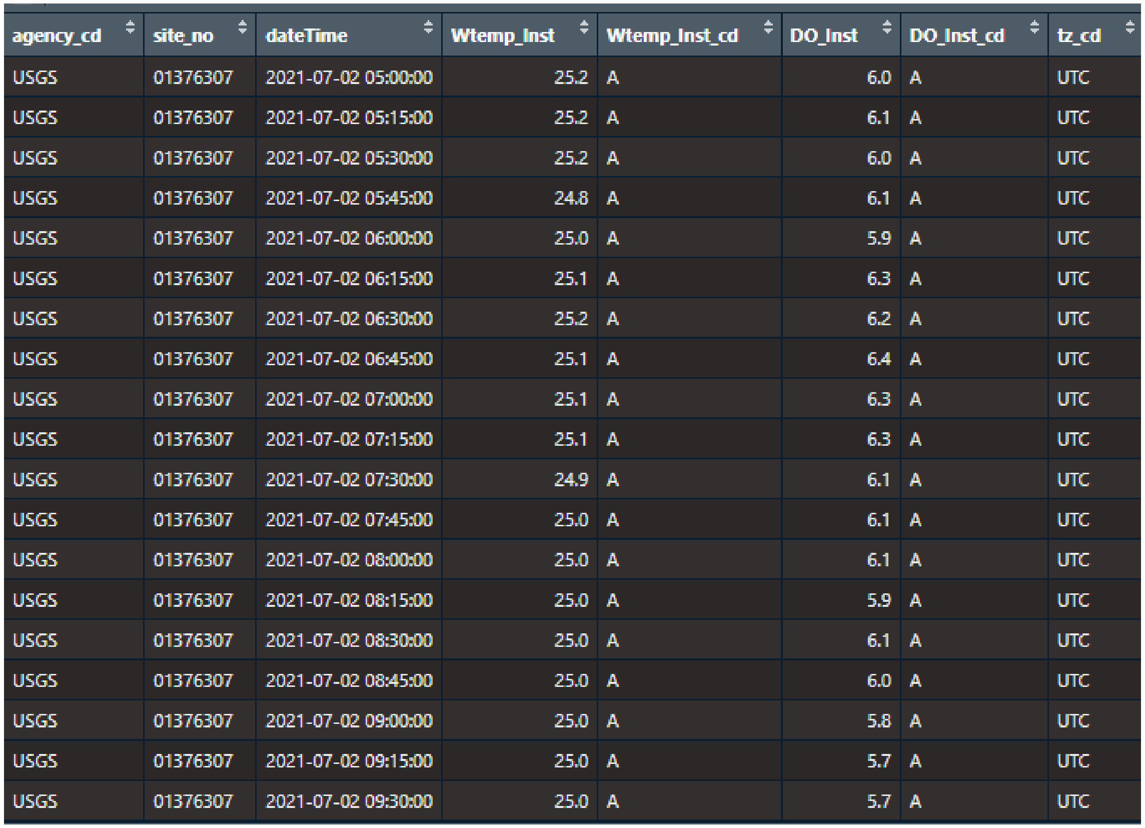


```r
yonkers.alb <- readNWISuv(siteNumbers = site,
                          parameterCd = params,
                          startDate = start.date,
                          endDate = end.date)
```

8. The default column names are not very descriptive. You can change them using renameNWIScolumns. Here’s a sample of what the data table looks like now:



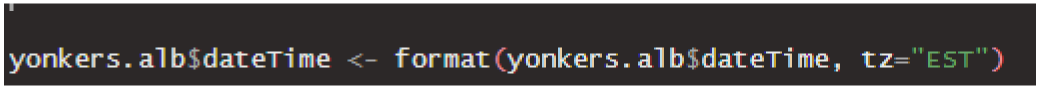
```r
yonkers.alb <- renameNWISColumns(yonkers.alb)
```



agency_cd	site_no	dateTime	Wtemp_Inst	Wtemp_Inst_cd	DO_Inst	DO_Inst_cd	tz_cd
USGS	01376307	2021-07-02 05:00:00	25.2	A	6.0	A	UTC
USGS	01376307	2021-07-02 05:15:00	25.2	A	6.1	A	UTC
USGS	01376307	2021-07-02 05:30:00	25.2	A	6.0	A	UTC
USGS	01376307	2021-07-02 05:45:00	24.8	A	6.1	A	UTC
USGS	01376307	2021-07-02 06:00:00	25.0	A	5.9	A	UTC
USGS	01376307	2021-07-02 06:15:00	25.1	A	6.3	A	UTC
USGS	01376307	2021-07-02 06:30:00	25.2	A	6.2	A	UTC
USGS	01376307	2021-07-02 06:45:00	25.1	A	6.4	A	UTC
USGS	01376307	2021-07-02 07:00:00	25.1	A	6.3	A	UTC
USGS	01376307	2021-07-02 07:15:00	25.1	A	6.3	A	UTC
USGS	01376307	2021-07-02 07:30:00	24.9	A	6.1	A	UTC
USGS	01376307	2021-07-02 07:45:00	25.0	A	6.1	A	UTC
USGS	01376307	2021-07-02 08:00:00	25.0	A	6.1	A	UTC
USGS	01376307	2021-07-02 08:15:00	25.0	A	5.9	A	UTC
USGS	01376307	2021-07-02 08:30:00	25.0	A	6.1	A	UTC
USGS	01376307	2021-07-02 08:45:00	25.0	A	6.0	A	UTC
USGS	01376307	2021-07-02 09:00:00	25.0	A	5.8	A	UTC
USGS	01376307	2021-07-02 09:15:00	25.0	A	5.7	A	UTC
USGS	01376307	2021-07-02 09:30:00	25.0	A	5.7	A	UTC

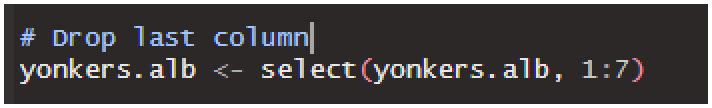
Changing the time zone

By default, data downloaded from NWIS are in Coordinated Universal Time (UTC). However, it may make more sense for your project to use Eastern Standard Time (EST). Here’s an easy fix using the data we retrieved in the section above. Since dateTime is an object of class POSIXct, we can use the base format() function to change it to EST.



```
yonkers.alb$dateTime <- format(yonkers.alb$dateTime, tz="EST")
```

Note: to avoid confusion, consider dropping the “tz” column of yonkers.alb since the data are no longer in UTC.

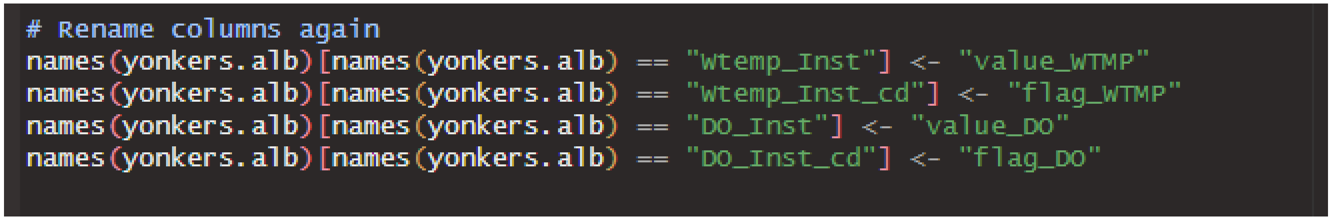


```
# Drop last column
yonkers.alb <- select(yonkers.alb, 1:7)
```

Using tidyr::pivot_longer() to clean up data

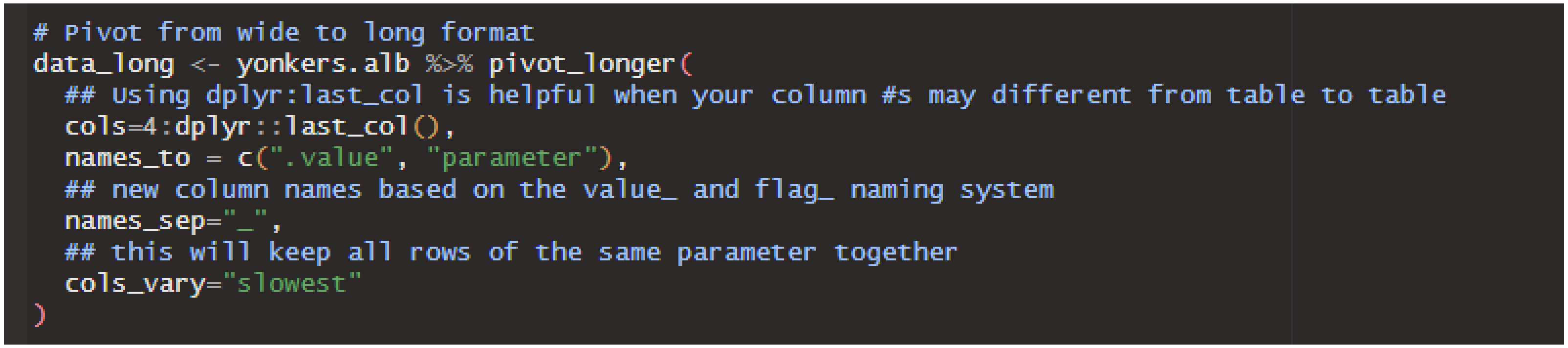
Data retrieved from NWIS are in “wide” format. For some operations, it may be useful for the data to be in a “long” format instead. Here’s one workflow you could use:

1. Rename columns (again)



```
# Rename columns again
names(yonkers.alb)[names(yonkers.alb) == "Wtemp_Inst"] <- "value_WTMP"
names(yonkers.alb)[names(yonkers.alb) == "Wtemp_Inst_cd"] <- "flag_WTMP"
names(yonkers.alb)[names(yonkers.alb) == "DO_Inst"] <- "value_DO"
names(yonkers.alb)[names(yonkers.alb) == "DO_Inst_cd"] <- "flag_DO"
```

2. It is recommended that you remove the “tz” column before proceeding. See “Changing the time zone” for more.
3. Use tidyr::pivot_longer() to transform the table

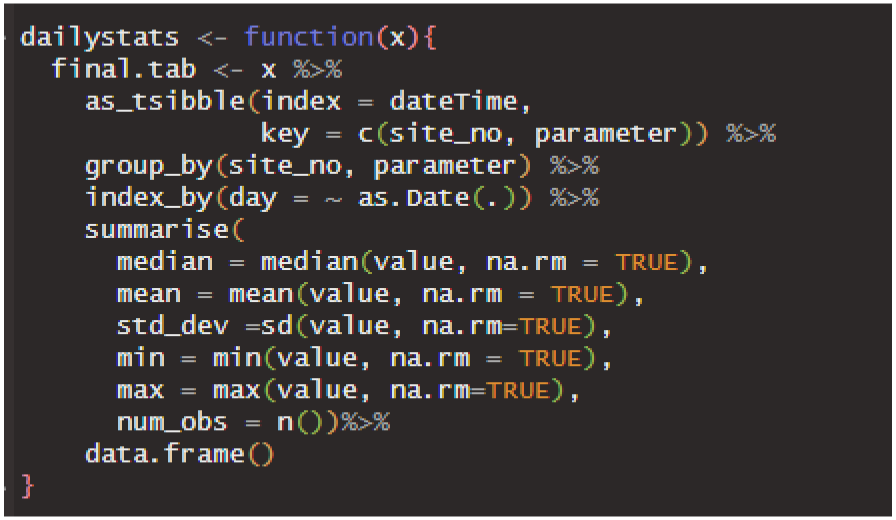


```
# Pivot from wide to long format
data_long <- yonkers.alb %>% pivot_longer(
  ## Using dplyr:last_col is helpful when your column #s may different from table to table
  cols=4:dplyr::last_col(),
  names_to = c(".value", "parameter"),
  ## new column names based on the value_ and flag_ naming system
  names_sep="_",
  ## this will keep all rows of the same parameter together
  cols_vary="slowest"
)
```

Calculating daily statistics

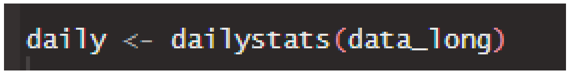
For some projects, you might want daily averages of observations instead of the measurements that are taken every 15 minutes. In this example, we will take the data_long table we made in the previous section and create our own function to calculate statistics. Note: you will need the tsibble and dplyr packages installed.

1. In our function, we will calculate the median, mean, standard deviation, minimum, and maximum daily values for each parameter. We also count the number of observations to assess if a full day of data were collected. In this case, 96 observations represents a 24-hour day.



```
dailystats <- function(x){
  final.tab <- x %>%
    as_tsibble(index = dateTime,
               key = c(site_no, parameter)) %>%
    group_by(site_no, parameter) %>%
    index_by(day = ~ as.Date(.)) %>%
    summarise(
      median = median(value, na.rm = TRUE),
      mean = mean(value, na.rm = TRUE),
      std_dev =sd(value, na.rm=TRUE),
      min = min(value, na.rm = TRUE),
      max = max(value, na.rm=TRUE),
      num_obs = n())%>%
    data.frame()
}
```

2. Use the function to calculate daily statistics on long data file:

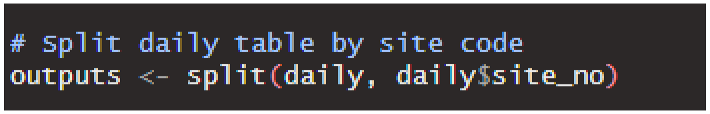


```
daily <- dailystats(data_long)
```

Exporting multiple files

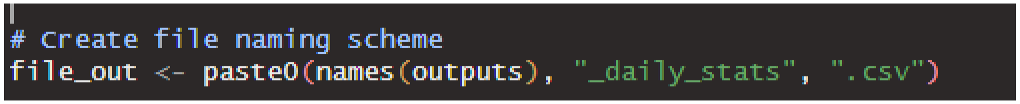
In our previous examples, we downloaded data from the Port of Albany and Yonkers HRECOS stations. For some projects, you might want to store this information in separate files on your computer.

1. Decide what variable you want to split your file by- in our example above, we might want to store the daily statistics data from the Port of Albany in a separate file from the Yonkers data.



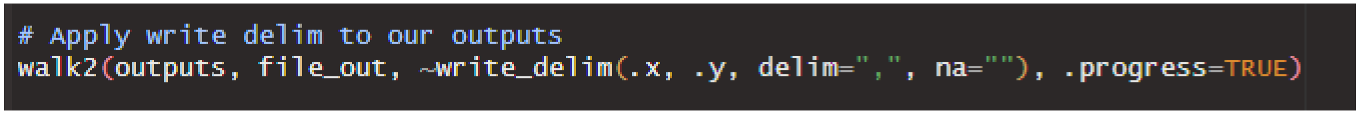
```
# Split daily table by site code
outputs <- split(daily, daily$site_no)
```

2. Decide how you want to name your files. In this example, we will take the site_no variable and append “daily_stats.csv” to it.



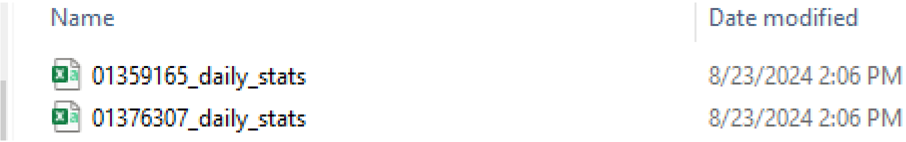
```
# Create file naming scheme
file_out <- paste0(names(outputs), "_daily_stats", ".csv")
```

3. Use purr::walk2() and readr::write_delim to write multiple files. Note: by default, these will save in your current working directory.



```
# Apply write delim to our outputs
walk2(outputs, file_out, ~write_delim(.x, .y, delim=",", na=""), .progress=TRUE)
```

4. Result:



Name	Date modified
01359165_daily_stats	8/23/2024 2:06 PM
01376307_daily_stats	8/23/2024 2:06 PM