# Example code for Chapter 6, Part 2 to focus on:

library(tidyverse)

library(mdsr)

### Another example of reading .csv and Excel files from external sources:

**library(readr)**

**election2 <- read\_csv(file="https://people.stat.sc.edu/hitchcock/Minneapolis\_tidy.csv")**

**# Or could use the base R 'read.csv' function (not as fast with large files):**

**# election2 <- read.csv(file="https://people.stat.sc.edu/hitchcock/Minneapolis\_tidy.csv")**

**election2**

**election4 <- read\_csv(file="https://people.stat.sc.edu/hitchcock/Minneapolis\_tidy\_no\_headers.csv", col\_names = F)**

**# Or could use the base R 'read.csv' function (not as fast with large files):**

**# election4 <- read.csv(file="https://people.stat.sc.edu/hitchcock/Minneapolis\_tidy\_no\_headers.csv", header = F)**

**election4**

**# Then we should provide the names in a separate step:**

**names(election4) <- c("ward","precinct","registered","voters","absentee","total\_turnout")**

rm(election1, election2, election3, election4)

## For files in which the delimiter that separates data values is something other than a comma, can use

## read\_delim in the readr package:

# read\_delim(file="fullpathname.txt", delim = "|")

## There are lots of other options in the read\_csv and read\_delim functions...

**## Reading HTML tables from a website**

**library(rvest)**

**url <- "http://en.wikipedia.org/wiki/Mile\_run\_world\_record\_progression"**

**tables <- url %>%**

**read\_html() %>%**

**html\_nodes("table")**

## The resulting object, tables, is a list:

is.list(tables)

## There are 15 tables in the list 'tables':

length(tables)

**## plucking the 3rd of the 15 tables and saving it as 'amateur':**

**amateur <- tables %>%**

**purrr::pluck(3) %>%**

**html\_table()**

**print(amateur, n=Inf)**

**## Using parse\_number to extract numeric information from a character string and store it as a numeric column:**

**library(readr)**

**ordway\_birds <- ordway\_birds %>%**

**mutate(**

**Month = parse\_number(Month),**

**Year = parse\_number(Year),**

**Day = parse\_number(Day)**

**)**

**ordway\_birds %>%**

**select(Timestamp, Year, Month, Day) %>%**

**glimpse()**

**# Try to calculate the mean year for the data set now:**

**mean(ordway\_birds$Year, na.rm=T) # na.rm=T will remove the missing values before calculating the mean**

# Note TimeStamp (which has date-time information) was a character variable, so we can't do mathematical operations on it.

**## Use mdy\_hms to convert TimeStamp to a true date-time (dttm) object called 'When':**

**library(lubridate)**

**birds <- ordway\_birds %>%**

**mutate(When = mdy\_hms(Timestamp)) %>%**

**select(Timestamp, Year, Month, Day, When, DataEntryPerson)**

**birds %>%**

**glimpse()**

## Now we can plot 'When' on a meaningful numeric axis:

birds %>%

ggplot(aes(x = When, y = DataEntryPerson)) +

geom\_point(alpha = 0.1, position = "jitter")

**## the 'first', 'last' and 'interval' function can work on date-time values:**

**bird\_summary <- birds %>%**

**group\_by(DataEntryPerson) %>%**

**summarize(**

**start = first(When), # Picks out the earliest date-time value for a person**

**finish = last(When) # Picks out the latest date-time value for a person**

**) %>%**

**mutate(duration = interval(start, finish) / ddays(1)) # 'interval' computes the difference between date-time values**

**# Printing summary table:**

**bird\_summary %>%**

**na.omit()**

**## A date that does not include a time:**

**as.Date(now())**

**# Also:**

**today()**

**## Converting date-time information stored in a character object into a true date-time object:**

**library(lubridate)**

**example <- c("2021-04-29 06:00:00", "2021-12-31 12:00:00")**

**str(example)**

**converted <- ymd\_hms(example)**

**str(converted)**

**# See the difference:**

**now() - example**

**now() - converted**

**## math on date-time values:**

**converted**

**converted[2] - converted[1]**

## Example Chapter 7 code to focus on:

## loading packages:

library(tidyverse)

library(mdsr)

library(Lahman)

names(Teams)

**## Getting information about the columns in Teams:**

**## str(Teams)**

**glimpse(Teams)**

**## Vectorized operation (takes a vector as input, returns a vector as output):**

**exp(1:3)**

**## A summary function (takes a vector as input, returns a single number as output):**

**mean(1:3)**

## An iterative operation using a loop (not recommended):

averages <- NULL

for (i in 15:40) {

averages[i - 14] <- mean(Teams[, i], na.rm = TRUE)

}

names(averages) <- names(Teams)[15:40]

averages

## Simpler code using the colMeans function (recommended)

colMeans(Teams[,15:40], na.rm = TRUE)

# Note that using the numbers 15 and 40 in the code makes this code non-reproducible on other data tables

# or on a potentially altered version of this data table...

**## Same iterative operation using 'map\_dbl':**

**Teams %>%**

**select(15:40) %>%**

**map\_dbl(mean, na.rm = TRUE)**

**## This works:**

**Teams %>%**

**select(name) %>%**

**map(nchar)**

**## Using 'across' to specify WHICH variables to summarize:**

**Teams %>%**

**summarize(across(where(is.numeric), mean, na.rm = TRUE))**

**## A more updated syntax, avoids warning...**

**Teams %>%**

**summarize(across(where(is.numeric), \(x) mean(x, na.rm = TRUE)) )**

**## Another way to use 'across' to specify WHICH variables to summarize:**

**Teams %>%**

**summarize(across(c(yearID, R:SF, BPF), mean, na.rm = TRUE))**

## Summaries of the Angels franchise, separated by different versions of the team name:

angels <- Teams %>%

filter(franchID == "ANA") %>%

group\_by(teamID, name) %>%

summarize(began = first(yearID), ended = last(yearID)) %>%

arrange(began)

angels

**## Iterating manually to see how long each 'angels' team name is:**

**angels\_names <- angels %>%**

**pull(name)**

**angels\_names # a character vector containing the various Angels team names**

nchar(angels\_names[1])

nchar(angels\_names[2])

nchar(angels\_names[3])

nchar(angels\_names[4])

**## Using 'map\_int' to automate the iterated operations is better:**

**map\_int(angels\_names, nchar)**

**## Since 'nchar' is vectorized, using it directly is even better!**

**nchar(angels\_names)**

## writing our own function 'top5' to pick out the top 5 seasons based on Wins:

top5 <- function(data, team\_name) {

data %>%

filter(name == team\_name) %>%

select(teamID, yearID, W, L, name) %>%

arrange(desc(W)) %>%

head(n = 5)

}

**## -----------------------------------------------------------------------------**

**angels\_names %>%**

**map(top5, data = Teams)**

**## Each element of 'angels\_names' will in turn be the value of the 'team\_name' argument in the 'top5' function.**

**## 'map\_dfr' will return a data frame (which we can then summarize) rather than a list, which 'map' returns:**

**angels\_names %>%**

**map\_dfr(top5, data = Teams)**

**## Summary table separated by team name:**

**angels\_names %>%**

**map\_dfr(top5, data = Teams) %>%**

**group\_by(teamID, name) %>%**

**summarize(N = n(), mean\_wins = mean(W)) %>%**

**arrange(desc(mean\_wins))**

## Example Chapter 14 code to focus on:

## line plots of popularity of the male names "John", "Paul", "George", "Ringo"

library(tidyverse)

library(mdsr)

library(babynames)

Beatles <- babynames %>%

filter(name %in% c("John", "Paul", "George", "Ringo") & sex == "M") %>%

mutate(name = factor(name, levels = c("John", "George", "Paul", "Ringo")))

beatles\_plot <- ggplot(data = Beatles, aes(x = year, y = n)) +

geom\_line(aes(color = name), size = 2)

beatles\_plot

**## using 'plotly' package and 'ggplotly' function to make the beatles\_plot object interactive:**

**# install.packages("plotly")**

**library(plotly)**

**ggplotly(beatles\_plot)**

**beatles\_plot2 <- ggplot(data = Beatles, aes(x = year, y = n, color=name)) + geom\_point()**

**ggplotly(beatles\_plot2) # can try brushing/selecting with this plot ...**

## Creating interactive, searchable data table with the 'DT' package and 'datatable' function:

# install.packages("DT")

library(DT)

datatable(Beatles, options = list(pageLength = 10))

## Animation Plots:

## Before installing 'gganimate' initially, you may have to do:

# install.packages("gifski")

# install.packages("av")

# and then restart the R session ...

# install.packages('gganimate')

library(gganimate)

theme\_set(theme\_bw())

## Using 'gganimate' to create animated time series plots

**library(gganimate)**

**library(transformr)**

**beatles\_animation <- beatles\_plot +**

**transition\_states(**

**name,**

**transition\_length = 2,**

**state\_length = 1**

**) +**

**enter\_grow() +**

**exit\_shrink()**

**animate(beatles\_animation, height = 400, width = 800)**

## Maybe a better example of 'gganimate':

# Start with a static plot (we've seen a basic bar plot kind of like this before):

my\_plot <- ggplot(

data = Beatles,

aes(

x = name,

y = prop

)

) +

geom\_col() +

xlab("Name") +

ylab("Proportion with Name")

my\_plot

# This sums the proportions for each name over all the years in the data set (that's why the "proportions" are more than 1!)

**# The transition\_time variable specifies which variable you want to dynamic plot to change with**

**# (typically this would be a variable that measures time)**

**# The 'labs' function with 'frame\_time' allows the title to reflect**

**# the changing values of the transition\_time variable.**

**my\_plot + ylim(c(0,0.1)) + transition\_time(year) +**

**labs(title = "Year: {frame\_time}")**

**# The dynamic plot appears as a gif in a separate window.**

**# If you want to slow down the rate at which the frames change, then decrease the "frames per second" (fps):**

**a1 <- my\_plot + ylim(c(0,0.1)) + transition\_time(year) +**

**labs(title = "Year: {frame\_time}")**

**animate(a1, nframes = 138, fps = 5) # a lower fps produces a slower animation**

GenNeutral <- babynames %>%

filter(name %in% c("Riley", "Lauren", "Cameron", "Taylor")) %>%

mutate(name = factor(name, levels = c("Riley", "Lauren", "Cameron", "Taylor")))

my\_plot2 <- ggplot(

data = GenNeutral,

aes(

x = name,

y = prop

)

) +

geom\_col() +

xlab("Name") +

ylab("Proportion with Name")

my\_plot2 # This single plot is not really sensible, since again, it is summing annual proportions across many years.

**# Doing separate panels by sex with facet\_wrap:**

**a2 <- my\_plot2 + ylim(c(0,0.02)) + facet\_wrap(~sex) +**

**transition\_time(year) +**

**labs(title = "Year: {frame\_time}")**

**animate(a2, nframes = 138, fps = 5)**

# If you want the plot to stop at the end rather than wrap back around to the beginning, use loop=FALSE:

# animate(a2, nframes = 138, fps = 5, renderer = gifski\_renderer(loop=FALSE))

## magick::image\_write(path = here::here("gfx/beatles-gganimate.png"), format = "png")