Lab 3b - Workspaces, Dates, and Factors

Environmental Data Analytics | John Fay and Luana Lima

spring 2025

Objectives

1. More practice with working directories & here()
2. More practice with dates: the `lubridate()` package
3. Discuss `stringsAsFactors = TRUE` and what "factors" are

---

1. Working with Working directories:

Background

Recall that we use R in a project setting: We open our R sessions via the Project dropdown or click on an Rproj file to open a project directly. Normally when we do this, our project working directory is set to the project folder and we can locate all other files via relative paths, i.e., paths relative to where our working directory is set.

We typically issue the `getwd()` command to confirm that our working directory is as expected:

```
# The working directory should be set to the base folder for your forked
# course repository, i.e., the folder that houses the R Project file.

getwd()
```

```
## [1] "/home/guest/EDA_Spring2025"
```

We have options to change the working directory, e.g. via the `setwd()` statement or via RStudio's menus. Once changed, however, subsequent calls to relative paths may point to the incorrect locations leading to confusion.

An alternative: the `here` package

And thus we introduce the `here` package, designed to streamline this process. A more detailed description of the library and its rationale can be found here: - Overview: https://here.r-lib.org & http://jenrichmond.rbind.io/post/how-to-use-the-here-package/ - Github repo: https://github.com/jennybc/here_here

```
#install.packages('here')
library(here) #Note this package may conflict with plyr and lubridate
```

```
## here() starts at /home/guest/EDA_Spring2025
```

```
#The command `here()` points to the location containing the ".Rrpoj" file, i.e.
# the project directory...
here()
```

```
## [1] "/home/guest/EDA_Spring2025"
```

```
#`here()` is not affected by the setwd() command
setwd('./Data')
getwd()
```

```
## [1] "/home/guest/EDA_Spring2025/Data"
```

```
here()
```

```
## [1] "/home/guest/EDA_Spring2025"
```

```
#Return R's working directory to the project directory
setwd(here())

#Arguments in the `here()` command allow easy navigation within subfolders
filename <- here('Data','Raw','USGS_Site02085000_Flow_Raw.csv')
print(filename)
```

```
## [1] "/home/guest/EDA_Spring2025/Data/Raw/USGS_Site02085000_Flow_Raw.csv"
```

```
#See whether the file exists
file.exists(filename)
```

```
## [1] TRUE
```

```
#A tidied read.csv command
usgs_flow_data <- read.csv(
  file = here('Data','Raw','USGS_Site02085000_Flow_Raw.csv'),
  stringsAsFactors = T
)

#Create and print a list of all files in a folder using "dir" and "here"
filenames <- dir(here('Data','Raw'),pattern = ("*.csv"))
print(filenames)
```

```
##  [1] "ECOTOX_Neonicotinoids_Insects_raw.csv"
##  [2] "EPAair_O3_NC2018_raw.csv"
##  [3] "EPAair_O3_NC2019_raw.csv"
##  [4] "EPAair_PM25_NC2018_raw.csv"
##  [5] "EPAair_PM25_NC2019_raw.csv"
##  [6] "NEON_NIWO_Litter_massdata_2018-08_raw.csv"
##  [7] "NEON_NIWO_Litter_trapdata_raw.csv"
##  [8] "NTL-LTER_Lake_Carbon_Raw.csv"
```

```
##  [9] "NTL-LTER_Lake_ChemistryPhysics_Raw.csv"
## [10] "NTL-LTER_Lake_Nutrients_Raw.csv"
## [11] "NWIS_SiteFlowData_NE_RAW.csv"
## [12] "NWIS_SiteInfo_NE_RAW.csv"
## [13] "pr_1901_2016_BRA.csv"
## [14] "tas_1901_2016_BRA.csv"
## [15] "USGS_Site02085000_Flow_Raw.csv"
## [16] "Wind_Speed_PortArthurTX.csv"
```

Exercise: the here package in action

1. Use the `here` package with `read.csv()` to read the `NTL-LTER_Lake_ChemistryPhysics_Raw.csv` file into a dataframe.
2. Refer to the example code shown here (http://jenrichmond.rbind.io/post/where-is-here/) to read in all CSV files starting with "EPAair_03" into a single dataframe.

- Note: You'll have to import `tidyverse` for `%>%` to work. Also, you may need to look at additional arguments in the `dir()` function...

```
# 1. Read the 'NTL-LTER_Lake_ChemistryPhysics_Raw.csv' into a dataframe


# 2.
```

---

2. Working with dates

Date formatting in R: A Review

Remember formatting of dates in R:

%d day as number (0-31) %m month (00-12, can be e.g., 01 or 1) %y 2-digit year %Y 4-digit year %a abbreviated weekday %A unabbreviated weekday %b abbreviated month %B unabbreviated month

```
# Adjust date formatting for today
# Write code for three different date formats.
# An example is provided to get you started.
# (code must be uncommented)
today <- Sys.Date()
format(today, format = "%B")
```

```
## [1] "January"
```

```
#format(today, format = "")
#format(today, format = "")
#format(today, format = "")
```

The `lubridate()` package

Install and load the package lubridate into your R session. Lubridate offers fast and user friendly parsing of date-time data. Create a string for today's data and then convert it to R date object using lubridate.

More info on lubridate [here][https://cran.r-project.org/web/packages/lubridate/lubridate.pdf].

```
#install.packages("lubridate")
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
#Ex1
str_today <- "2023-feb-7"
#Since the format is year-month-day we will use function ymd()
date_obj_today <- ymd(str_today)
date_obj_today
```

```
## [1] "2023-02-07"
```

```
#Ex2
str_today2 <- "Feb 7, 2023"
#Since the format is month-day-year we will use function mdy()
date_obj_today <- mdy(str_today2)
date_obj_today
```

```
## [1] "2023-02-07"
```

```
#Ex_3 - on your own...
str_juneteenth <- "19 June 1865"
#date_juneteenth <- ()
#date_juneteenth
```

The century issue

What happens when you the year in your date column is represented by two digits? Is "55" 2055 or 1955?? Here we build code to deal with that issue.

```
#century issue
str_past <- "55-feb-3"
date_obj_past <- ymd(str_past)
date_obj_past
```

```
## [1] "2055-02-03"
```

```r
#Build a function to fix year that is more general than the one discussed in the lesson
fix.early.dates <- function(d, cutoff) {

       m <- year(d) %% 100
       # The %% operator does modular division i.e. integer-divides
       # year(d) by 100 and returns the remainder

       year(d) <- ifelse(m > cutoff, 1900+m, 2000+m)
       # This will update year(d), year() is the lubridate function that
       # returns the year for a date object
       return(d)
}

fixed_date_obj_past <- fix.early.dates(date_obj_past,cutoff=24)
#cutoff could be the current year to be more general or any other depending on data set


fixed_date_obj_past
```

```
## [1] "1955-02-03"
```

```r
#Fix for century issue
str_past <- "55-feb-3"

#Alternative 1
date_obj_past <- fast_strptime(str_past,"%y-%b-%d",cutoff_2000=24L)
date_obj_past
```

```
## [1] "1955-02-03 UTC"
```

```r
#Alternative 2
date_obj_past2 <- parse_date_time2(str_past,"ymd",cutoff_2000=24L)
date_obj_past2
```

```
## [1] "1955-02-03 UTC"
```

```r
#Functions ymd(), mdy(), ydm() do not take argument cutoff_2000
```

In some cases when dates are provided as integers, you may need to provide an origin for your dates. For example, excel date could be given as number of days since an origin date. Origin date can be different. When R looks at dates as integers, its origin is January 1, 1970. Check if that is true on your machine.

```r
#Check if "1970-01-01" is your origin date.
lubridate::origin
```

```
## [1] "1970-01-01 UTC"
```

---

3. Factors

Set up for the exercise

```
#Read packages
library(tidyverse);library(lubridate); library(here)

#Read in data, don't convert strings to factors
sites <- read.csv(here('Data/Raw/NWIS_SiteFlowData_NE_RAW.csv'))
dim(sites)
```

```
## [1] 152   4
```

```
#Convert date column to date object
sites$date <- ymd_hms(sites$date)
```

Examine the lakename column as a character column

Note: `wday` is a `lubridate()` function...

```
#Compute the day of the week each sample was collected
sites$dow <- wday(sites$date)

## What class is the dow column?
class(sites$dow)
```

```
## [1] "numeric"
```

```
#List the unique values in the column
unique(sites$dow)
```

```
## [1] 4 5 3
```

```
#Summary - is it meaningful?
summary(sites$dow)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   4.000   4.000   4.013   4.000   5.000
```

Convert the days of the week to a factor

```
#Convert to a factor
sites$dow <- factor(sites$dow)

## What class is the dow column now?
class(sites$dow)
```

```
## [1] "factor"
```

```r
#List the unique values in the column
unique(sites$dow)
```

```
## [1] 4 5 3
## Levels: 3 4 5
```

```r
#Summary - is it meaningful?
summary(sites$dow)
```

```
##   3   4   5
##   1 148   3
```

```r
#Show the levels associated with our factor column
levels(sites$dow)
```

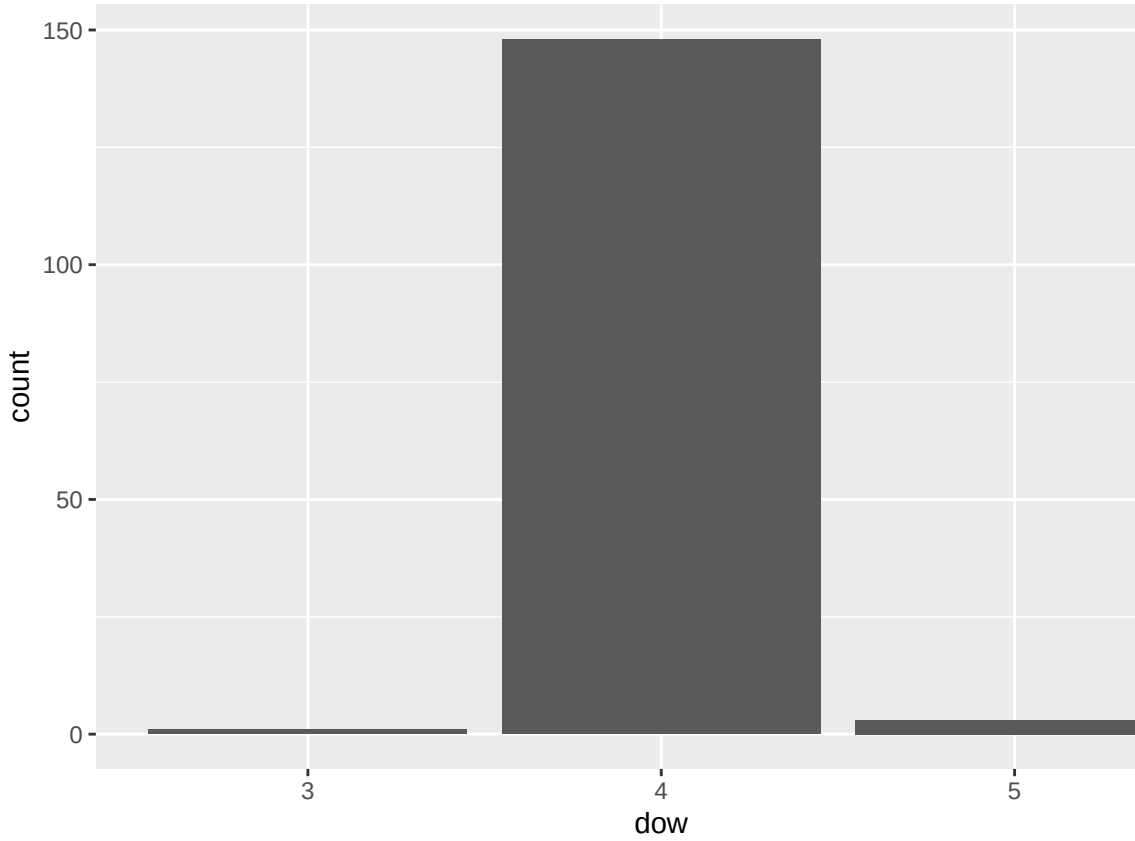
```
## [1] "3" "4" "5"
```

Factors & Levels

Factors defined A **factor** is a grouping variable: rows with the same value in the factor column are members of the same group. For example, all rows with a `dow` of 3 were collected on a Tuesday.

Levels defined **Levels** are the unique values that the factors can take. The `unique()` function reveals the levels as does the `levels()` function. By default, the levels includes only the values in the factor column. However, we can add additional levels and we can also re-order levels. Why would we want to do this? Well, one example would be if we wanted to plot the number of samples recorded each day of the week, and in that plot we wanted to include all days of the week, not just the one's found in our data...

```r
#Create a bar plot showing the count of samples by day of week
ggplot(sites,aes(x=dow)) +
  geom_bar()
```



Plotting with factors

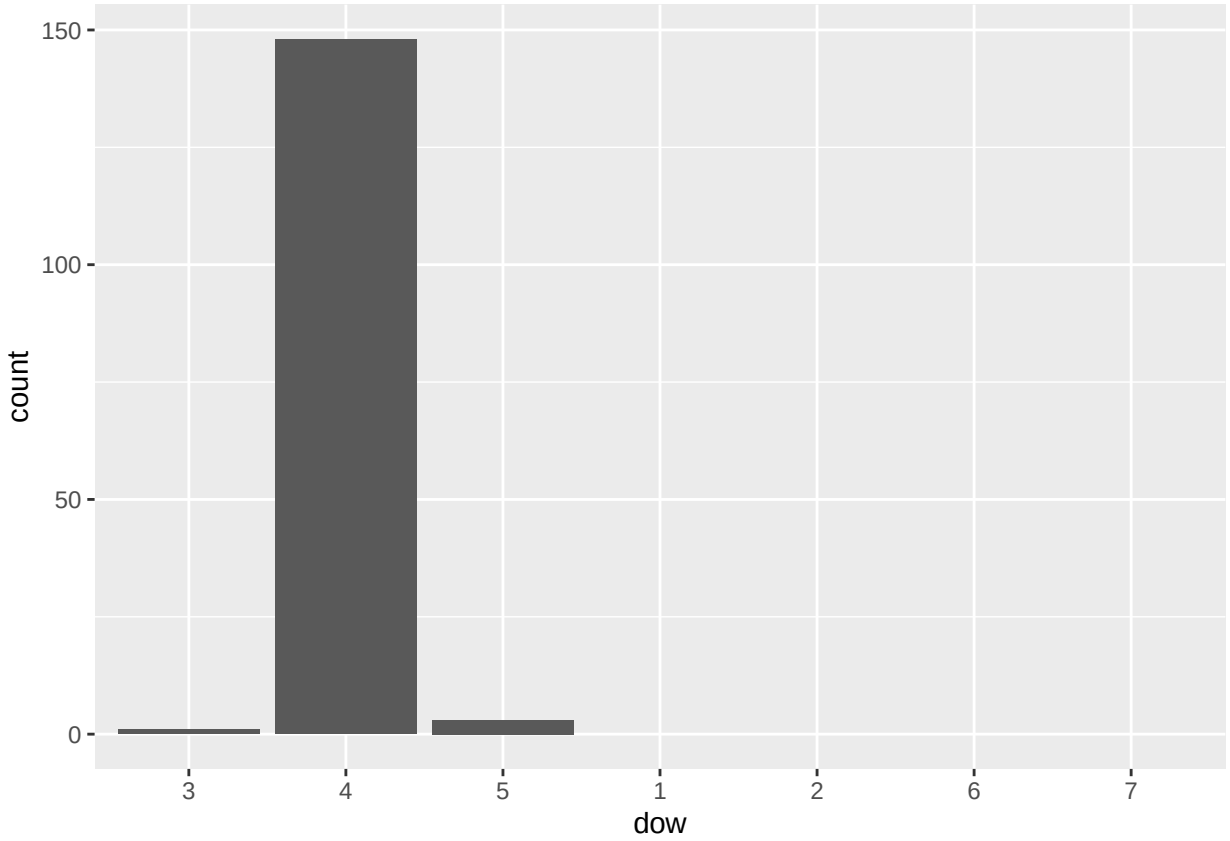
```
#Update the levels to include all seven days of the week
#  (note we have to start with existing levels)
levels(sites$dow) <- c(levels(sites$dow),"1","2","6","7")

#Confirm the change
levels(sites$dow)
```

Adding more (empty) levels to our dataset

```
## [1] "3" "4" "5" "1" "2" "6" "7"
```

```
#Plot again
ggplot(sites,aes(x=dow)) +
  geom_bar() +
  scale_x_discrete(drop=FALSE)
```



We now have all days of the week, but the order is incorrect. Also, we have day numbers, not day names. How to fix?

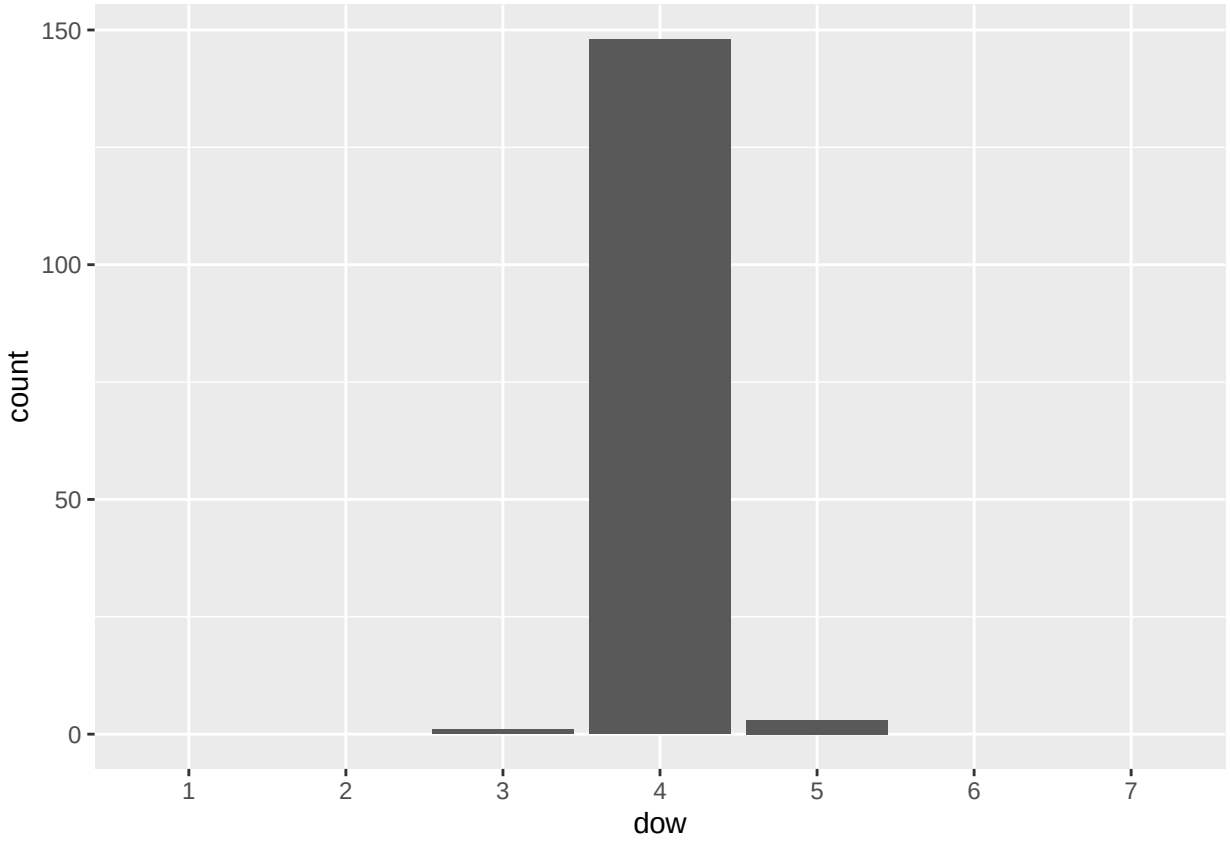
Reordering and renaming levels We can reorder levels simply by overwriting existing levels.

```
#Reorder by overwriting the column with the existing values
# assigned with a new order of levels
sites$dow <- factor(sites$dow, levels=c("1","2","3","4","5","6","7")) #Or... as.character(seq(1,7))

#See that the levels are updated
levels(sites$dow)
```

```
## [1] "1" "2" "3" "4" "5" "6" "7"
```

```
#View the plot
ggplot(sites,aes(x=dow)) +
  geom_bar() +
  scale_x_discrete(drop=FALSE)
```

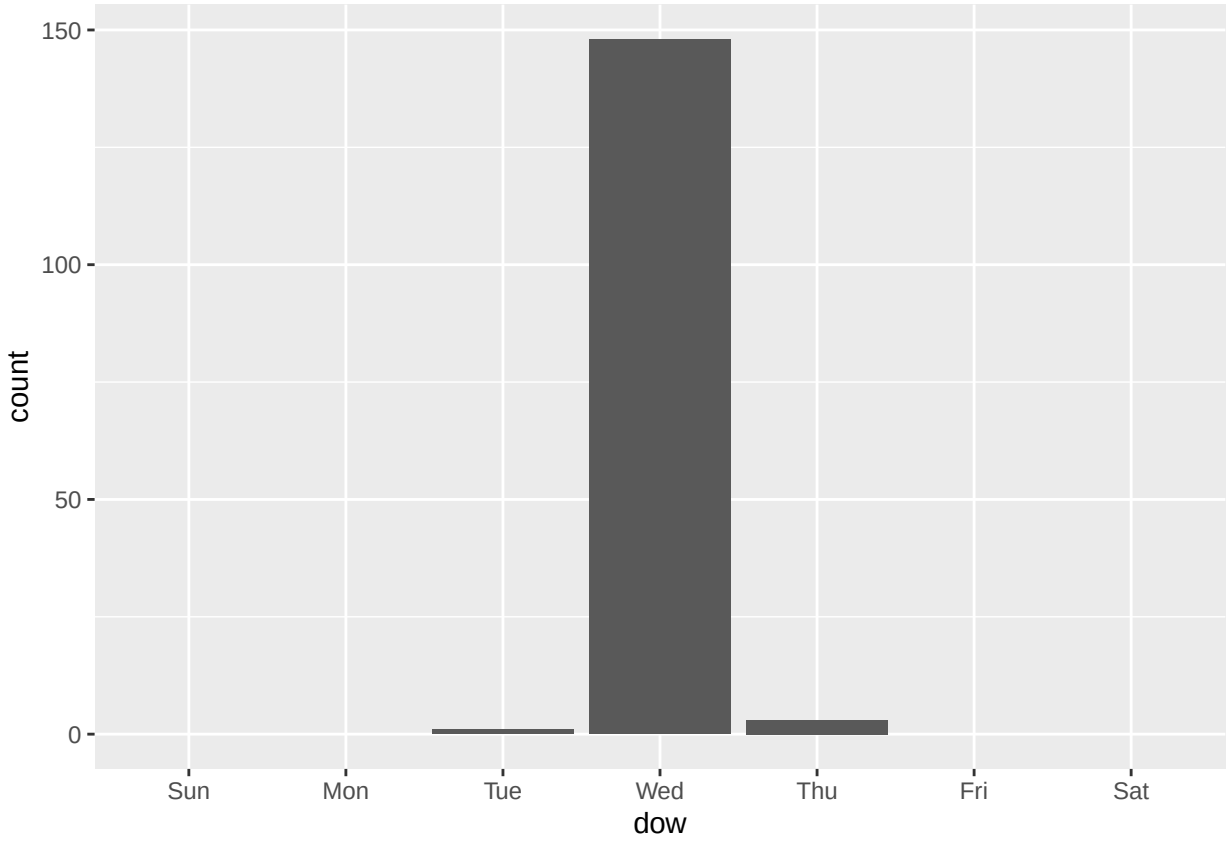



```
#Add **labels** to our factors
sites$dow <- factor(
  sites$dow,
  levels=c("1","2","3","4","5","6","7"),
  labels=c("Sun","Mon","Tue","Wed","Thu","Fri","Sat"))

#See that the levels are updated
levels(sites$dow)
```

```
## [1] "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat"
```

```
#View the plot
ggplot(sites,aes(x=dow)) +
  geom_bar() +
  scale_x_discrete(drop=FALSE)
```



Exercise: see if you can plot the number of samples recorded each month

Code to [re]Read in the USGS dataset is provided Hint: Follow the same steps above: - Create a column to hold the months, derived from the date-time column - Convert the months to a factor - Ensure the factor contains levels for all 12 months, in order - Ensure the factor contains labels for all 12 months Tip: - R has a built in variable `month.abb` that holds a list of all 12 months

```
#[RE]read in the USGS data
usgs_flow_data <- read.csv(
  file = here('Data','Raw','USGS_Site02085000_Flow_Raw.csv'),
  stringsAsFactors = T,
  colClasses = (c('site_no'='factor'))
)
#Convert date-time field to date
usgs_flow_data$datetime = mdy(usgs_flow_data$datetime)

#Create date column
usgs_flow_data$month <- month(usgs_flow_data$datetime)

#Convert to a factor, with labels and levels


#Plot
```