Assignment 4: Data Wrangling (Fall 2024)

Hanbin Lyu

OVERVIEW

This exercise accompanies the lessons in Environmental Data Analytics on Data Wrangling

Directions

1. Rename this file `<FirstLast>_A04_DataWrangling.Rmd` (replacing `<FirstLast>` with your first and last name).
2. Change "Student Name" on line 3 (above) with your name.
3. Work through the steps, **creating code and output** that fulfill each instruction.
4. Be sure to **answer the questions** in this assignment document.
5. When you have completed the assignment, **Knit** the text and code into a single PDF file.
6. Ensure that code in code chunks does not extend off the page in the PDF.

Set up your session

1a. Load the `tidyverse`, `lubridate`, and `here` packages into your session.

1b. Check your working directory.

1c. Read in all four raw data files associated with the EPA Air dataset, being sure to set string columns to be read in a factors. See the README file for the EPA air datasets for more information (especially if you have not worked with air quality data previously).

```
#1a
#load packages
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(lubridate)
library(here)
```

```
## here() starts at /home/guest/EDE_Fall2024
```

```r
#1b
#check working directory
getwd()

#1c
#read data files and set string columns as factors
O3_2018 = read.csv("Data/Raw/EPAair_O3_NC2018_raw.csv", stringsAsFactors = TRUE)
O3_2019 = read.csv("Data/Raw/EPAair_O3_NC2019_raw.csv", stringsAsFactors = TRUE)
PM25_2018 = read.csv("Data/Raw/EPAair_PM25_NC2018_raw.csv", stringsAsFactors = TRUE)
PM25_2019 = read.csv("Data/Raw/EPAair_PM25_NC2019_raw.csv", stringsAsFactors = TRUE)

#check the data
view(O3_2018)
view(O3_2019)
view(PM25_2018)
view(PM25_2019)
```

2. Add the appropriate code to reveal the dimensions of the four datasets.

```r
#2
dim(O3_2018)
dim(O3_2019)
dim(PM25_2018)
dim(PM25_2019)
#reveal dimensions of data sets
```

> All four datasets should have the same number of columns but unique record counts (rows). Do your datasets follow this pattern? Answer: yes, all my data sets follow this pattern. They have same number of columns, 20, but different numbers of rows.

Wrangle individual datasets to create processed files.

3. Change the Date columns to be date objects.
4. Select the following columns: Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LONGITUDE
5. For the PM2.5 datasets, fill all cells in AQS_PARAMETER_DESC with "PM2.5" (all cells in this column should be identical).
6. Save all four processed datasets in the Processed folder. Use the same file names as the raw files but replace "raw" with "processed".

```r
#3
#change the Date columns to be date objects
mdy(O3_2018$Date)
mdy(O3_2019$Date)
mdy(PM25_2018$Date)
mdy(PM25_2019$Date)
```

```
#4
#select columns
select(O3_2018, Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LONGI
select(O3_2019, Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LONGI
select(PM25_2018, Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LON
select(PM25_2019, Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LON

#5
#fill all cells in AQS_PARAMETER_DESC with "PM2.5"
PM25_2018$AQS_PARAMETER_DESC = "PM2.5"
PM25_2019$AQS_PARAMETER_DESC = "PM2.5"

#combine codes in questions 3, 4, and 5 together
p318 = O3_2018 %>%
  select(Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LONGITUDE) %
  mutate(Date = mdy(Date))

p319 = O3_2019 %>%
  select(Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LONGITUDE) %
  mutate(Date = mdy(Date))

p2518 = PM25_2018 %>%
  select(Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LONGITUDE) %
  mutate(Date = mdy(Date)) %>%
  mutate(AQS_PARAMETER_DESC = "PM2.5")

p2519 = PM25_2019 %>%
  select(Date, DAILY_AQI_VALUE, Site.Name, AQS_PARAMETER_DESC, COUNTY, SITE_LATITUDE, SITE_LONGITUDE) %
  mutate(Date = mdy(Date)) %>%
  mutate(AQS_PARAMETER_DESC = "PM2.5")

#6
#save all four processed data sets in the Processed folder
write.csv(p318, row.names = FALSE, file = "Data/Raw/EPAair_O3_NC2018_processed.csv")
write.csv(p319, row.names = FALSE, file = "Data/Raw/EPAair_O3_NC2019_processed.csv")
write.csv(p2518, row.names = FALSE, file = "Data/Raw/EPAair_PM25_NC2018_processed.csv")
write.csv(p2519, row.names = FALSE, file = "Data/Raw/EPAair_PM25_NC2019_processed.csv")
```

Combine datasets

7. Combine the four datasets with `rbind`. Make sure your column names are identical prior to running this code.

8. Wrangle your new dataset with a pipe function (%>%) so that it fills the following conditions:

- Include only sites that the four data frames have in common:

"Linville Falls", "Durham Armory", "Leggett", "Hattie Avenue",
"Clemmons Middle", "Mendenhall School", "Frying Pan Mountain", "West Johnston Co.", "Garinger High School", "Castle Hayne", "Pitt Agri. Center", "Bryson City", "Millbrook School"

(the function `intersect` can figure out common factor levels - but it will include sites with missing site information, which you don't want…)

- Some sites have multiple measurements per day. Use the split-apply-combine strategy to generate daily means: group by date, site name, AQS parameter, and county. Take the mean of the AQI value, latitude, and longitude.
- Add columns for "Month" and "Year" by parsing your "Date" column (hint: `lubridate` package)
- Hint: the dimensions of this dataset should be 14,752 x 9.

9. Spread your datasets such that AQI values for ozone and PM2.5 are in separate columns. Each location on a specific date should now occupy only one row.
10. Call up the dimensions of your new tidy dataset.
11. Save your processed dataset with the following file name: "EPAair_O3_PM25_NC1819_Processed.csv"

```
#7
#combine 4 processed data sets
cpds = rbind(p318, p319,p2518,p2519)

#8
#wrangle the data set
wtds = cpds %>%
  filter(Site.Name == "Linville Falls" | Site.Name == "Durham Armory" | Site.Name == "Leggett" | Site.Na
  group_by(Date, Site.Name, AQS_PARAMETER_DESC, COUNTY) %>%
  summarize(DAILY_AQI_MEAN = mean(DAILY_AQI_VALUE),
    SITE_LATITUDE_MEAN = mean(SITE_LATITUDE),
    SITE_LONGITUDE_MEAN = mean(SITE_LONGITUDE)) %>%
  mutate(Month = month(Date), Year = year(Date))
```

```
## ‘summarise()‘ has grouped output by ’Date’, ’Site.Name’, ’AQS_PARAMETER_DESC’.
## You can override using the ‘.groups‘ argument.
```

```
dim(wtds)

#9
#spread the data set
stds = wtds %>%
  pivot_wider(names_from = AQS_PARAMETER_DESC, values_from = DAILY_AQI_MEAN)

#10
#check the dimension
dim(stds)

#11
#save the processed data sets
write.csv(stds, row.names = FALSE, file = "Data/Raw/EPAair_O3_PM25_NC1819_Processed.csv")
```

Generate summary tables

12. Use the split-apply-combine strategy to generate a summary data frame. Data should be grouped by site, month, and year. Generate the mean AQI values for ozone and PM2.5 for each group. Then, add a pipe to remove instances where mean **ozone** values are not available (use the function `drop_na` in your pipe). It's ok to have missing mean PM2.5 values in this result.

13. Call up the dimensions of the summary dataset.

```r
#12
#generate a summary data frame with 'drop_na'
gsdf = stds %>%
  group_by(Site.Name, Month, Year) %>%
  summarize(mean_Ozone = mean(Ozone), mean_PM2.5 = mean(PM2.5)) %>%
  drop_na(mean_Ozone)
```

```
## `summarise()` has grouped output by 'Site.Name', 'Month'. You can override
## using the `.groups` argument.
```

```r
#generate a summary data frame with 'na.omit'
gsdfo = stds %>%
  group_by(Site.Name, Month, Year) %>%
  summarize(mean_Ozone = mean(Ozone), mean_PM2.5 = mean(PM2.5)) %>%
  na.omit()
```

```
## `summarise()` has grouped output by 'Site.Name', 'Month'. You can override
## using the `.groups` argument.
```

```r
#13
#check the dimension
dim(gsdf)
dim(gsdfo)
```

14. Why did we use the function `drop_na` rather than `na.omit`? Hint: replace `drop_na` with `na.omit` in part 12 and observe what happens with the dimensions of the summary date frame.

    Answer: with the function 'drop_na', it has 182 rows and 5 columns; with 'na.omit' function, it has 101 rows and 5 columns. We use 'drop_na' because it removes missing values only from specific columns, mean_Ozone here, allowing us to keep rows with valid data in other columns. This helps us maintain important information in our dataset without removing too many rows uncessary.