Assignment 4: Data Wrangling (Spring 2025)

Darpan Barua

OVERVIEW

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

Directions

1. Rename this file `<DarpanBarua>_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).

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

```r
#1a - Below loads the necessary packages.
library(tidyverse)
library(lubridate)
library(here)

#1b
here()
```

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

```r
#1c - Below reads csv in the Assignments folder and assigns appropriate variable to each file.
#It also makes sure to set string columns to be read as factors.

raw_data1 <- read_csv(here("Assignments","EPAair_O3_NC2018_raw.csv"),
                      col_types = cols(.default = col_factor()))
raw_data2 <- read_csv(here("Assignments","EPAair_O3_NC2019_raw.csv"),
                      col_types = cols(.default = col_factor()))
raw_data3 <- read_csv(here("Assignments","EPAair_PM25_NC2018_raw.csv"),
                      col_types = cols(.default = col_factor()))
raw_data4 <- read_csv(here("Assignments","EPAair_PM25_NC2019_raw.csv"),
                      col_types = cols(.default = col_factor()))

#2 - Below reveals the rows by column configuration.
dim(raw_data1)
```

```
## [1] 9737   20
```

```r
dim(raw_data2)
```

```
## [1] 10592    20
```

```r
dim(raw_data3)
```

```
## [1] 8983   20
```

```r
dim(raw_data4)
```

```
## [1] 8581   20
```

All four datasets should have the same number of columns but unique record counts (rows). Do your datasets follow this pattern? - Answer: Yes

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 - Below converts the dates (which may be stored as character/factor) to a
# proper date format; in this case mm-dd-yy.

raw_data1 <- raw_data1 %>%
    mutate(Date = mdy(Date))
raw_data2 <- raw_data2 %>%
    mutate(Date = mdy(Date))
raw_data3 <- raw_data3 %>%
    mutate(Date = mdy(Date))
raw_data4 <- raw_data4 %>%
    mutate(Date = mdy(Date))


# 4 - Check detailed comments below.

# Below defines the required columns to keep in the processed dataset
selected_columns <- c("Date", "DAILY_AQI_VALUE", "Site Name", "AQS_PARAMETER_DESC",
    "COUNTY", "SITE_LATITUDE", "SITE_LONGITUDE")


# Below selects only the required columns from raw_data(s) and store in
# processed_data(s)

processed_data1 <- raw_data1 %>%
    select(all_of(selected_columns))
processed_data2 <- raw_data2 %>%
    select(all_of(selected_columns))
processed_data3 <- raw_data3 %>%
    select(all_of(selected_columns))
processed_data4 <- raw_data4 %>%
    select(all_of(selected_columns))


# 5 - Below ensures all values in the 'AQS_PARAMETER_DESC' column are set to
# 'PM2.5' for the PM2.5 datasets (processed_data3 and processed_data4)

processed_data3 <- processed_data3 %>%
    mutate(AQS_PARAMETER_DESC = "PM2.5")

processed_data4 <- processed_data4 %>%
    mutate(AQS_PARAMETER_DESC = "PM2.5")



# 6 - Below saves the processed datasets in the 'Processed' folder.

write_csv(processed_data1, here("Data", "Processed", "EPAair_O3_NC2018_processed.csv"))
write_csv(processed_data2, here("Data", "Processed", "EPAair_O3_NC2019_processed.csv"))
write_csv(processed_data3, here("Data", "Processed", "EPAair_PM25_NC2018_processed.csv"))
write_csv(processed_data4, here("Data", "Processed", "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"

```r
#7 - Below combines the four processed datasets into a single dataset and checks the dimensions...

combined_data <- rbind(processed_data1, processed_data2, processed_data3, processed_data4)

dim(combined_data)
```

```
## [1] 37893     7
```

```r
#8

# Below loads the stringr package for text processing
library(stringr)

# Below defines the list of common sites that appear in all four datasets
common_sites <- c("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")

# Below starts wrangling the combined dataset using a pipe (%>%)
filtered_data <- combined_data %>%
  mutate(`Site Name` = str_trim(`Site Name`)) %>%  # This removes extra spaces in "Site Name"
  filter(`Site Name` %in% common_sites)%>% # This keeps only the common sites from the list
  # Below converts Date column to character format. Not doing this led to incorrect dimensions.
  mutate(Date = as.character(Date)) %>%
  # Below removes invalid/missing dates. Not doing this led to incorrect dimensions.
  filter(!Date %in% c("", "--", "NA", "Invalid")) %>%
  # Below converts Date to proper Date object.Not doing this led to incorrect dimensions.
  mutate(Date = parse_date_time(Date, orders = c("mdy", "ymd", "dmy")))%>%
  mutate(
    DAILY_AQI_VALUE = as.numeric(as.character(DAILY_AQI_VALUE)), # This ensures AQI values are numeric
    SITE_LATITUDE = as.numeric(as.character(SITE_LATITUDE)), # Convert latitude to numeric
    SITE_LONGITUDE = as.numeric(as.character(SITE_LONGITUDE))# Convert longitude to numeric
  ) %>%

  # Below groups by Date, Site Name, AQS Parameter, and County to aggregate
  #multiple measurements per day
  group_by(Date, `Site Name`, AQS_PARAMETER_DESC, COUNTY) %>%
  #Below finds the mean for AQI, latitude, and longitude..
  summarize(
    mean_AQI = mean(DAILY_AQI_VALUE),
    SITE_LATITUDE = mean(SITE_LATITUDE),
    SITE_LONGITUDE = mean(SITE_LONGITUDE),
  ) %>%

  # Below adds columns for "year" and "month" by extracting them from the Date column
  mutate(
    year = year(Date),
    month = month(Date)
  )

#Below checks dimensions.
dim(filtered_data)
```

```
## [1] 14752     9
```

```r
#9

library(dplyr)
library(tidyr)

# Below reshapes the dataset to spread AQI values into separate columns
spread_data <- filtered_data %>%
  # Below selects only relevant columns before reshaping

  select(Date, `Site Name`, COUNTY, AQS_PARAMETER_DESC, mean_AQI, SITE_LATITUDE, SITE_LONGITUDE, year,
         month) %>%
  pivot_wider(
    names_from = AQS_PARAMETER_DESC,  # This spreads values based on the AQS_PARAMETER_DESC column
    values_from = mean_AQI,  # This uses mean_AQI values to populate the new columns
    names_prefix = "AQI_"  # This adds a prefix to the new column names to indicate AQI data
  )

#10 - Below checks dimension.

dim(spread_data)
```

```
## [1] 8976    9
```

```
#11 - Below saves processed data with the requested name.

write_csv(spread_data, here("Data", "Processed", "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.

```
#12

# Below groups by Site Name, Month, and Year, then compute mean AQI values
summary_data <- spread_data %>%
  group_by(`Site Name`, month, year) %>%
  summarize(
    # Below calculates the mean AQI for Ozone while ignoring NAs
    mean_AQI_Ozone = mean(AQI_Ozone, na.rm = TRUE),
    # Below calculates the mean AQI for PM2.5 while ignoring NAs
    mean_AQI_PM25 = mean(AQI_PM2.5, na.rm = TRUE),
  ) %>%
  drop_na(mean_AQI_Ozone)  # This removes rows where Ozone AQI is missing

#13 - Below checks dimensions.
dim(summary_data)
```

```
## [1] 239   5
```

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: We used drop_na(mean_AQI_Ozone) because we only wanted to remove rows where Ozone AQI is missing, while keeping rows where PM2.5 AQI might still be available. Too much data would've been gone if we used na.omit().