NPS Data Wrangling and Visualization

Katie Willi

2025-06-10

Lesson Objectives

In the last lesson, we learned how to work with park visitation data and reduce redundancies in our workflows through functions and iteration. In this lesson we will learn how to manipulate data frames with the {tidyverse}, and plot elegant time series graphs with the {ggplot2}, {scales} and {plotly} packages.

Pulling in necessary packages and data sets

Remember to install the new packages!

```
library(tidyverse)
library(scales)
library(plotly)
```

Let's read in our parkwide visitation data from last week as well as our new unit visitation dataset:

```
# Read in the parkwide visitation data
parkwide_data <- read_csv("data/nps_parkwide_visitation.csv")

# Read in the unit-specific visitation data
unit_data <- read_csv("data/nps_unit_visitation.csv")
```

Let's pull park-wide visitor data from 1980-2024, and name the final object `parkwide`. (Code hack: we can use `1980:2024` to create a vector of years so we don't have to write each year out!)

```
years <- (1980:2024)

parkwide <- parkwide_data %>%
  filter(Year %in% years)
```

Let's also pull data for several individual park units:

```
parks <- c("ROMO", "ACAD", "LAKE", "YELL", "GRCA", "ZION", "OLYM", "GRSM")

units <- unit_data %>%
  filter(Year %in% years,
         UnitCode %in% parks)
```

Exploring our data

Look at the data frame structure of `parkwide` and `units`; they're exactly the same! So let's go ahead and bind those together:

```
visitation <- bind_rows(parkwide, units)
```

Now that we have a single data set containing all of the NPS recreational visitation data that we've pulled, let's start exploring it! But first, let's aggregate the monthly data into annual data using `group_by()` and `summarize()`:

```
annual_visitation <- visitation %>%
  group_by(UnitCode, Year) %>%
  # we only care about recreational visitors:
  summarize(RecVisitation = sum(RecreationVisitors))

annual_visitation
```

What does visitation data look like through time? First we can try to graph all of the park units together:

```
ggplot(data = annual_visitation)+
  geom_point(aes(x = Year, y = RecVisitation, color = UnitCode)) +
  geom_path(aes(x = Year, y = RecVisitation, color = UnitCode)) +
  scale_y_continuous(labels = scales::label_scientific()) +
  theme_bw(base_size = 10)
```

. . . yikes, not surprisingly, parkwide recreational visitation is much higher than our individual unit's visitation data, making our graph pretty useless. It might be nice to have each park unit in a graph of its own.

We can create individual graphs for each unit using `facet_wrap()`, and we can set the y-axes for each plot to `"free_y"`:

```
ggplot(data = annual_visitation) +
  geom_point(aes(x = Year, y = RecVisitation, color = UnitCode)) +
  geom_path(aes(x = Year, y = RecVisitation, color = UnitCode)) +
  scale_y_continuous(labels = scales::label_comma()) +
  facet_wrap(~UnitCode, scales = "free_y") +
  theme_bw(base_size = 10)
```

We can also make this plot interactive by feeding it into {plotly}'s `ggplotly()` function:

```
plotly::ggplotly(
  ggplot(data = annual_visitation) +
    geom_point(aes(x = Year, y = RecVisitation, color = UnitCode)) +
    geom_path(aes(x = Year, y = RecVisitation, color = UnitCode)) +
    scale_y_continuous(labels = scales::label_comma()) +
    facet_wrap(~UnitCode, scales = "free_y") +
    theme_bw(base_size = 10)
)
```

It is pretty clear that some park units get orders of magnitude more visitors than others. But just how much of the total park visitation do each of these parks account for from year to year? Here we walk through two methods to tackle this question, ***pivoting*** and ***joining***, to get park unit visitation side-by-side with park-wide data.

Pivoting

Currently, our annual visitation data is considered *long* because we have all of our NPS visitation data in one column, with multiple rows representing the same year. We can make this data *wide* by using the function `pivot_wider()`

```
wide_data <- annual_visitation %>%
  select(Year, UnitCode, RecVisitation) %>%
  pivot_wider(names_from = UnitCode, values_from = RecVisitation) %>%
  select(Year, Parkwide, everything())
```

... where `names_from` represents the column with the values you are hoping to spread into new columns, and `values_from` represents the data you want to fill these new columns with.

We can make the data set *long* again by using the function `pivot_longer()`:

```
long_data <- wide_data %>%
  pivot_longer(cols = -Year,
               names_to = "Park",
               values_to = "RecVisitation")
```

... where `cols` are the columns we want to gather into one column (or, the column(s) you DON'T want to gather), while `names_to` and `values_to` are the names and values for the new columns produced from the pivot.

Joining

Another approach is to join our datasets. Here's another way we can join unit data with parkwide data:

```
joined_data <- inner_join(x = units,
                          y = parkwide,
                          by = c("Year", "Month"))

glimpse(joined_data)
```

When joining, columns with the same name get ".x" and ".y" suffixes to distinguish them. To make the differences between the two datasets more clear, let's rename and select some columns ahead of joining them together:

```
joined_data_tidier <- inner_join(x = units %>% select(Year, Month, UnitCode, UnitRecVisitation = Recrea
                                 y = parkwide %>% select(Year, Month, ParwideRecVisitation = Recreation'
                                 by = c("Year", "Month"))
```