STAT 29000 Project 11

Topics: tidyverse, data.table part 3

Motivation: `tidyverse` and `data.table` are both useful tools to work with data. Sometimes one package may be more suited to a task than another. Sometimes it is best to stick to base R. In this project, we will utilize multiple packages to solve problems.

Context: We now have some exposure to the `tidyverse` and its system of wrangling data. We understand an alternative package called `data.table`. We are able to use a variety of base R functions to manipulate data. Now, we are going to continue to use and hone these skills.

Scope: `data.table`, `tidyverse`, and the base R packages all provide utility to the R ecosystem. It is useful to deepen our understanding of what each of these contributes to our ability to analyze data.

You can read more about `data.table` in the official documentation:

https://cran.r-project.org/web/packages/data.table/data.table.pdf

You can read more about `tidyverse` in the official documentation

https://cran.r-project.org/web/packages/tidyverse/tidyverse.pdf

and on the `tidyverse` website:

https://www.tidyverse.org/

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`.

Question 1: diving into Disney data

Read some Disney data into R, using the following lines of R code. Note that the `read_excel` function is from the `readxl` library, so you will need to install and load this library first.

```
download.file("https://cdn.touringplans.com/datasets/touringplans_data_dictionary.xlsx",
"./dis.xlsx")
```

```
dat <- read_excel("./dis.xlsx")
```

```
colnames(dat) <- c("links", "content")
```

1a. Using the links in the newly created `tibble`, create new `tibbles` for the datasets that the links point to, and save those `tibbles` in their respective row in the original `tibble` named `dat`, under a column named `data`.

Hint: The links can be seen in the column `dat$links`.

Hint: Remember that we have several functions for reading in csv files (`fread`, `read_csv`, `read.csv`, which create (respectively), a `data.table`, a `tibble`, or a `data.frame`). One of these data reading functions is better suited for reading data from a link than others.

Hint: Remember to take a look at the data you read in, to make sure there isn't erroneous data. In this data, `NA` values are indicated as `-999`. All 3 of the functions we've used to read csv data have an argument to fix this as it is read in. Make sure to use these arguments and add this value as an `NA` value when reading the data.

1b. Add another column to `dat` that indicates `ride` for a dataset with data about a ride, and `other` for the metadata and entities files. Call this column `type`.

Hint: It is perfectly acceptable to do this naively (hard code).

1c. Add another column that identifies the `ride`. Use the name of the file that was read in, with the path and file extension removed. For example,

https://cdn.touringplans.com/datasets/toy_story_mania.csv

would be `toy_story_mania`. If the row doesn't contain a ride, leave the cell value as `NA`.

Hint: The `tidyverse` function called `case_when()` could be useful here.

1d. The data in its current form isn't particular condusive to accessing and exploring. Let's fix this. We can see that the data for each ride has 4 columns in common. In addition for each row, we have a date. We can supplement each rides dataset with metadata. For each row in our data, we will find a row in the metadata that has the same date, and append that row of metadata to the end of our data. If either our ride data or metadata doesn't have data for a certain date, let's throw it out. At the end, you should be left with 2910808 rows of data and 194 columns.

The first 5 columns should be: `date`, `datetime`, `SPOSTMIN`, `SACTMIN`, and `ride_name`, followed by every column from the metadata.

Hint: Consider using the function `unnest` from the `tidyr` package.

Hint: The method described for supplementing each ride's dataset with data from metadata is called a left join, or left outer join. To perform this, please look at the `data.table::merge()` function as well as the `tidyverse` functions with names of the form `.*_join`.

Question 2: I juuust can't wait . . . for the lines

2a. Add a column called `day_of_week`. Use the date to populate the new column with the name or abbreviation of the day of the week.

Hint: `lubridate` is a useful package from the `tidyverse` for dealing with dates.

2b. Let's say the most popular ride is the ride with the highest average posted wait time (`SPOSTMIN`). List the most popular rides for each day of the week by year. The results should be sorted by `day_of_week`, then by the year.

2c. Define "more precipitation" as having `WEATHER_WDWPRECIP >=.2`. List the five rides that decrease in popularity the most, as compared to the rest of the rides, when there is more precipitation. Is there any ride in particular that stands out as a ride that gets more popular?

Note: Be careful. Since our "popularity" metric is based on wait times, it is only natural that wait times will decrease with higher amounts of precipitation, because people leave the parks. Let's normalize days with less precipitation and days with more precipitation, so our wait times are mapped between 0-1 and can be compared. For example:

```
dat <- data.table(x=c(1,2,3), y=c(4,5,6))

dat[, c("xnorm", "ynorm"):=list( (x-min(x)) / (max(x)-min(x)),  (y-min(y)) / (max(y)-min(y))
)]
```

Hint: I found the `pivot_wider()` function from the `tidyr` to be useful.

Question 3: plotting out a disney trip

3a. Splash Mountain is an iconic Disney World ride. Use ggplot2's `facet_grid()` to create a grid of line graphs – one for each day of the week – where the average wait time is shown by hour on each graph. Add a theme to spruce up your graph.

Hint: The following functions are useful: `geom_line()`, `facet_grid()`.

Hint: If you want to make your labels look better, check out the example in the file called `stat29000project11examples.R`.

The resulting plot is depicted at

https://datamine.purdue.edu/seminars/fall2019/stat29000project11question3a.jpg

3b. Holidays are known to effect crowd levels at Disney World. Modify **(3a)** by separating the data for the holidays provided below, and adding another line to each graph showing the average wait times for holidays on the same chart.

```
newyear <- seq(mdy('01/01/2012'), mdy('01/01/2019'), by = '1 year')

christmaseve <- seq(mdy('12/24/2012'), mdy('12/24/2019'), by = '1 year')

christmas <- seq(mdy('12/25/2012'), mdy('12/25/2019'), by = '1 year')

halloween <- seq(mdy('10/31/2012'), mdy('10/31/2019'), by = '1 year')

independence <- seq(mdy('07/04/2012'), mdy('07/04/2019'), by = '1 year')

holidays <- c(newyear, christmaseve, christmas, halloween, independence)
```

The resulting plot is depicted at

https://datamine.purdue.edu/seminars/fall2019/stat29000project11question3b.jpg

Optional open problem. The chart in (3b) is peculiar. See if you can figure out what is going on with Sunday and Tuesday.

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/TrDmg7jV using the instructions found in the GitHub Classroom instructions folder on Blackboard.