For those of you who don’t know, VLOOKUP is a function in Excel that takes two ranges, matches two of the columns, and gives you a result from a third column based on the match. There are some nuances, like that the matching column in the “from” range has to be the first one, and the difference between approximate and exact matches, but we’ll stay out of the Excel weeds on this post. Suffice it to say that dplyr merges are the ticket for 90% of what VLOOKUP gets used for and it’s way faster.

This post is largely insurance to make sure that I never do that Google search again. First, We’ll load the tidyverse (which includes dplyr) and grab some sample data. I’ll also load tibbletime to prepare the data more easily.

library(tidyverse)

library(tibbletime)

For some example data I hopped on to [kaggle](https://www.kaggle.com/) and grabbed a recent [dataset about some behives](https://www.kaggle.com/se18m502/bee-hive-metrics) that looked interesting. For the purpose of this exercise I’m going to calculate the average daily temperature and bee departures so that we can easily merge the two together.

temp <- read\_csv("data/temperature\_schwartau.csv")

## Parsed with column specification:

## cols(

## timestamp = col\_datetime(format = ""),

## temperature = col\_double()

## )

temp

## # A tibble: 253,430 x 2

## timestamp temperature

##

## 1 2017-01-01 14:10:00 NA

## 2 2017-01-01 14:15:00 12.3

## 3 2017-01-01 14:20:00 12.3

## 4 2017-01-01 14:25:00 12.3

## 5 2017-01-01 14:30:00 12.4

## 6 2017-01-01 14:35:00 12.4

## 7 2017-01-01 14:40:00 12.5

## 8 2017-01-01 14:45:00 12.5

## 9 2017-01-01 14:50:00 12.5

## 10 2017-01-01 14:55:00 12.5

## # ... with 253,420 more rows

flow <- read\_csv("data/flow\_schwartau.csv")

## Parsed with column specification:

## cols(

## timestamp = col\_datetime(format = ""),

## flow = col\_double()

## )

flow

## # A tibble: 2,513,836 x 2

## timestamp flow

##

## 1 2017-01-01 14:15:00 0

## 2 2017-01-01 14:16:00 0

## 3 2017-01-01 14:17:00 0

## 4 2017-01-01 14:18:00 0

## 5 2017-01-01 14:19:00 0

## 6 2017-01-01 14:20:00 0

## 7 2017-01-01 14:21:00 0

## 8 2017-01-01 14:22:00 0

## 9 2017-01-01 14:23:00 0

## 10 2017-01-01 14:24:00 0

## # ... with 2,513,826 more rows

To get the data ready for this demo I’m going to take the hourly average flow out of the hive (positive values) and the hourly average temperature.

flow <- flow %>%

drop\_na() %>%

mutate(exit\_flow = case\_when( # get only departures

flow >= 0 ~ flow

)) %>%

drop\_na() %>% # drop all arriving rows

arrange(timestamp) %>% # tble\_time needs the timestamp to be ordered

tbl\_time(index = timestamp) %>% # convert to tble\_time for ease of grouping by hour

collapse\_by(period = "hourly", clean = TRUE) %>% # "collapse" by hour

group\_by(timestamp) %>% # group by collapsed hourly column

summarise(mean\_exit\_flow = mean(exit\_flow)) # get the hourly average

flow

## # A time tibble: 20,953 x 2

## # Index: timestamp

## timestamp mean\_exit\_flow

##

## 1 2017-01-01 15:00:00 0

## 2 2017-01-01 16:00:00 0

## 3 2017-01-01 17:00:00 0

## 4 2017-01-01 18:00:00 0

## 5 2017-01-01 19:00:00 0.00847

## 6 2017-01-01 20:00:00 0

## 7 2017-01-01 21:00:00 0

## 8 2017-01-01 22:00:00 0

## 9 2017-01-01 23:00:00 0

## 10 2017-01-02 00:00:00 0

## # ... with 20,943 more rows

temp <- temp %>%

drop\_na() %>%

arrange(timestamp) %>%

tbl\_time(index = timestamp) %>%

collapse\_by(period = "hourly", clean = TRUE) %>%

group\_by(timestamp) %>%

summarise(mean\_temp = mean(temperature))

temp

## # A time tibble: 20,953 x 2

## # Index: timestamp

## timestamp mean\_temp

##

## 1 2017-01-01 15:00:00 12.4

## 2 2017-01-01 16:00:00 13.4

## 3 2017-01-01 17:00:00 16.7

## 4 2017-01-01 18:00:00 16.4

## 5 2017-01-01 19:00:00 13.7

## 6 2017-01-01 20:00:00 14.0

## 7 2017-01-01 21:00:00 12.4

## 8 2017-01-01 22:00:00 17.3

## 9 2017-01-01 23:00:00 17.8

## 10 2017-01-02 00:00:00 20.0

## # ... with 20,943 more rows

You can see that the first column of both tables match exactly, with one row for each hour and the right column of each table do not match. This is one scenario that I might use VLOOKUP for in Excel, especially if the data isn’t ordered, or if I don’t know if both datasets are complete.

As a side note, the flow dataset has 2.5 million rows and the temperature dataset has 250 thousand rows. Summarizing these by hour into tables of about 21 thousand rows and then combining them would be a trying task for Excel, but for R it’s a breeze.

To combine these two in R, I will use a join function from dplyr. VLOOKUP works like left\_join() or right\_join() because it looks for matches from one table in another table. If there aren’t any matches, it returns with an error. We also have the option to do a full\_join() which returns all rows from both tables and with NA wherever there isn’t a match or inner\_join() which only returns rows where there are matches from both tables.

Since I have created a nice clean dataset with matching timestamp columns, I will get the same result no matter which join function I use. dplyr will try to guess which column I want to join by and return its guess as a message unless I specify which column I want it to use with by =.

joined\_table <- flow %>%

left\_join(temp, by = "timestamp")

joined\_table

## # A time tibble: 20,953 x 3

## # Index: timestamp

## timestamp mean\_exit\_flow mean\_temp

##

## 1 2017-01-01 15:00:00 0 12.4

## 2 2017-01-01 16:00:00 0 13.4

## 3 2017-01-01 17:00:00 0 16.7

## 4 2017-01-01 18:00:00 0 16.4

## 5 2017-01-01 19:00:00 0.00847 13.7

## 6 2017-01-01 20:00:00 0 14.0

## 7 2017-01-01 21:00:00 0 12.4

## 8 2017-01-01 22:00:00 0 17.3

## 9 2017-01-01 23:00:00 0 17.8

## 10 2017-01-02 00:00:00 0 20.0

## # ... with 20,943 more rows

Pretty easy! Preparing this dataset took more work than joining it, but there are a lot of cases where the data will lend itself to a join right out of the box.

Since we’ve made it this far, let’s take a look at the relationship between the hourly average number of bees exiting the hive and the hourly average temperature.

joined\_table %>%

ggplot(aes(x = mean\_exit\_flow, y = mean\_temp)) +

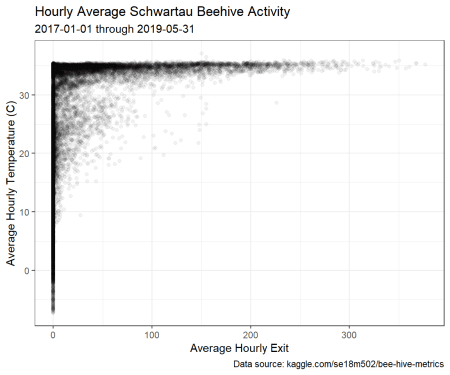
geom\_point(alpha = .05) +

labs(title = "Hourly Average Schwartau Beehive Activity", subtitle = "2017-01-01 through 2019-05-31", caption = "Data source: [kaggle.com/se18m502/bee-hive-metrics](http://kaggle.com/se18m502/bee-hive-metrics)") +

xlab("Average Hourly Exit") +

ylab("Average Hourly Temperature (C)") +

theme\_bw()



It looks like it needs to be about 10 C (50 F) before these bees want to head outdoors, and above 30 C (86 F) there is a huge spike in activity.

There is a pretty thick line of inactivity at all temperatures, though, which makes me wonder if the beehive has been active the whole time it was being observed. Let’s make a plot of activity over time.

joined\_table %>%

ggplot(aes(x = timestamp, y = mean\_exit\_flow, color = mean\_temp)) +

geom\_point() +

labs(title = "Hourly Average Schwartau Beehive Activity",

subtitle = "2017-01-01 through 2019-05-31",

caption = "Data source: [kaggle.com/se18m502/bee-hive-metrics](http://kaggle.com/se18m502/bee-hive-metrics)",

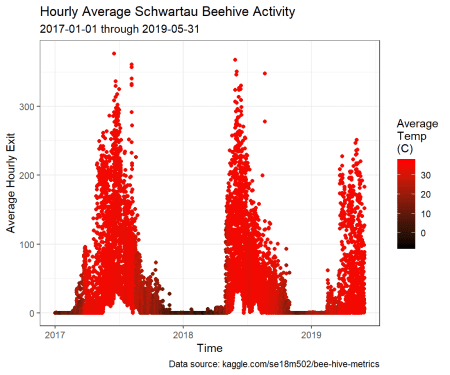
color = "Average\nTemp\n(C)") +

xlab("Time") +

ylab("Average Hourly Exit") +

scale\_colour\_gradient(low = "black", high = "red") +

theme\_bw()



As I expected, the beehive activity is function of both temperature and time of year. Also, there are some times of year where there are bees exiting the hive twenty-four hours per day.

There you have it! While there’s nothing wrong with using Excel, especially when it’s the medium that your colleagues are familiar with, R is much faster, and more reproducible and flexible. I can easily read through the code that I just wrote and follow what’s happening, but when I get an Excel workbook with similar calculations, it takes quite a bit of time to figure out what’s going on.