Session 3: Importing and cleaning data

Natalie Nelson, PhD, Biological & Agricultural Engineering, NCSU

02/02/2018

1 Data

The data analyzed in this exercise were collected in 2011 and 2012 as part of a study lead by Dr. Anders Huseth (Entomology and Plant Pathology, NCSU) and Dr. Russell Groves (Entomology, UW-Madison). With this study, Drs. Huseth and Groves aimed to answer the question: does potato pest management compromise water quality?
Potatoes are threatened by a variety of pests (Figure 1), and neonicotinoids are among the most commonly-applied group of insecticides used in potato pest management. Neonicotonoid insecticides can be applied through several methods, including foliar spray, seed treatment, in-furrow spray, or as an impregnated gel (Figure 2). These insecticides are largely sprayed during planting and early in the growing season (Figure 3).

In this study, researchers evaluated how neonicotinoid (1) application strategy and (2) active ingredient influenced groundwater quality. Potato crops received each of the four neonicotinoid applications shown in Figure 2 with one of two active ingredients, imidacloprid or thiamethoxam. Groundwater samples were collected throughout the growing season to determine if application strategy and/or active ingredient produced different total insecticide leachate loads, as well as different leaching trends (e.g., higher concentrations early or late in the growing season).

---

2 Questions

In today's problem session, we will analyze the neonicotinoid dataset to investigate the following questions:

- Which application strategy + active ingredient produces the *worst* groundwater quality? Define your metric for *worst*. Give some thought as to what's happening with the untreated control when coming up with your justification. Provide a data visualization(s) and relevant summary statistics to back up your argument.
- For the application strategy + active ingredient that creates the worst groundwater impacts, were the corresponding leachate concentrations similar between the two study years (2011 and 2012)? Provide a data visualization(s) and relevant summary statistics to support your argument.
- For the active ingredient associated with the worst groundwater quality, how long after planting did neonicotinoid groundwater concentrations peak for each of the application strategies? Support your argument with a data visualization(s).

For each plot you present, provide its practical interpretation. What is the data visualization saying? What are the real-world implications of the patterns you've uncovered in the data?

Additionally, what other types of data would you need to formulate "best neocotinoid practices" for potato farmers?

3 Analysis

Setting up your R script

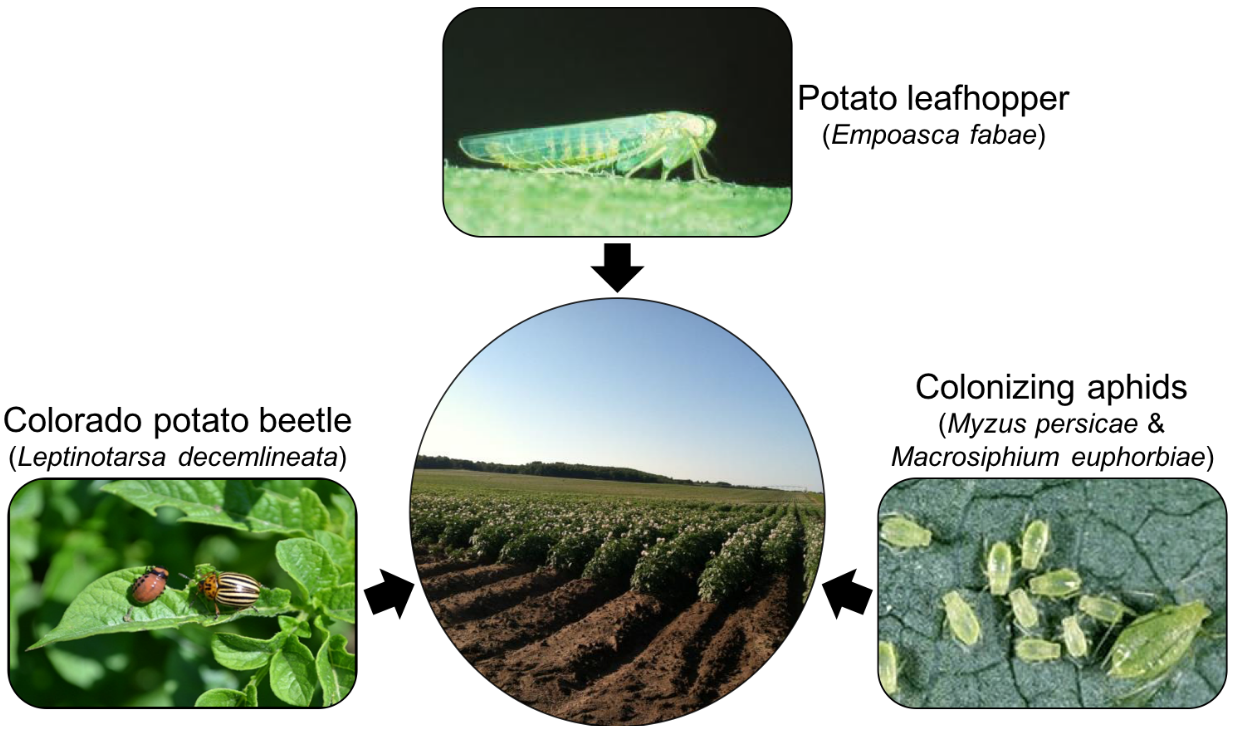


Figure 1: Examples of potato pests. Credit: Anders Huseth.

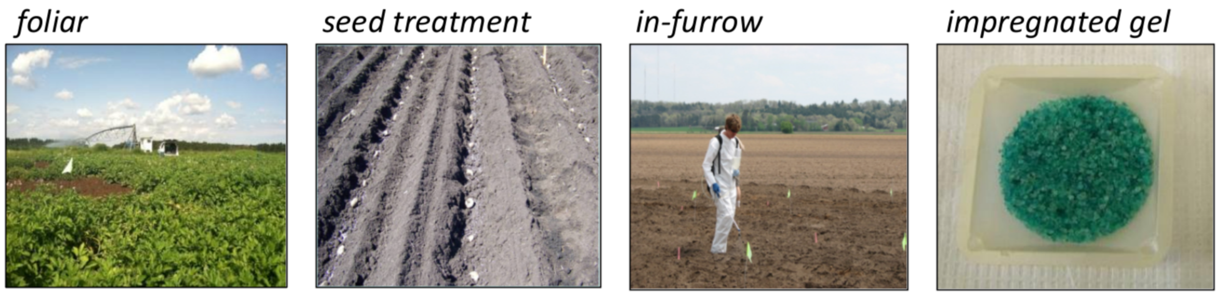


Figure 2: Neonicotinoid insecticide application methods. Credit: Anders Huseth.

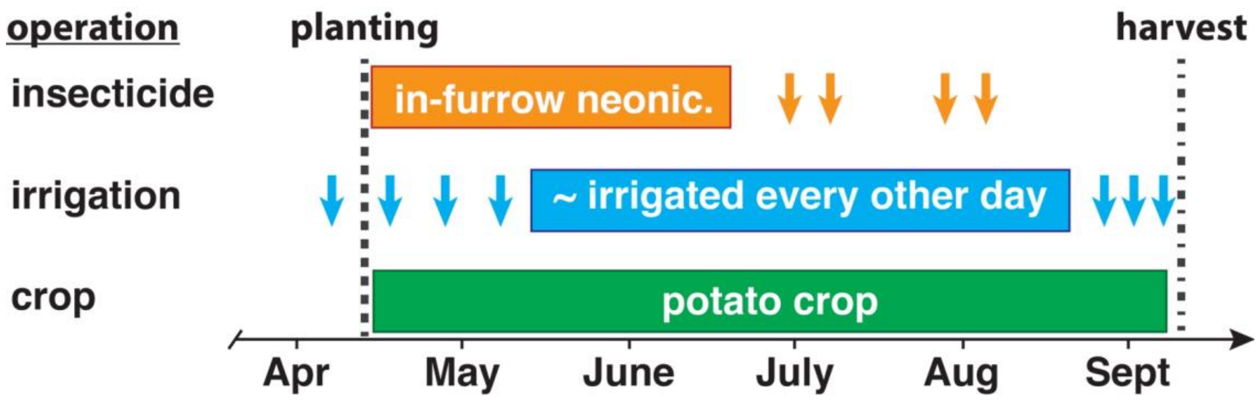


Figure 3: Example neonicotinoid application schedule. Credit: Anders Huseth.

New packages that need to be installed for today's session include `readxl`, `tidyr`, and `readr`.Remember, this can be done with the `install.packages("name-of-package")` function. In addition to these newly-installed packages, you will also need to load `ggplot2` and `dplyr` to your workspace.

```r
# Clear workspace
rm(list=ls(all=TRUE))

# Load packages
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(readxl)
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.4.3
```

```r
library(readr)

# Set working directory
#setwd("your-path-name") # Note: this line should not be commented out in your code!
```

Read in your data

The data are saved in a .xlsx file: neonicotinoids_messy.xlsx. To load Excel data, we use the function `read_excel()` and specify our filename as the argument.

```r
d <- read_excel("neonicotinoids_messy.xlsx")
```

```
## Warning in strptime(x, format, tz = tz): unknown timezone 'default/America/
## New_York'
```

Check the packaging and the top and bottom of your data

As always, our next step is to inspect our data.

```r
str(d)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    480 obs. of  8 variables:
##  $ year                   : num  2011 2011 2011 2011 2011 ...
##  $ sample_month           : chr  "june a" "june a" "june b" "june b" ...
##  $ day_of_year            : num  165 165 179 179 195 195 227 227 258 258 ...
##  $ trt_code               : chr  "UTC" "UTC" "UTC" "UTC" ...
##  $ insecticide_app_method: chr  "untreated control" "untreated control" "untreated control" "untreate
##  $ days_after_planting   : num  47 47 61 61 77 77 109 109 140 140 ...
##  $ imidacloprid           : chr  "0.88500000000000001" "NA" "0.96099999999999997" "NA" ...
##  $ thiamethoxam           : chr  "NA" "0.30299999999999999" "NA" "0.35599999999999998" ...
```

From `str()`, we can see that we have numeric and character variables. However, the last two columns (*imidacloprid* and *thiamethoxam*) are character variables, but should be numeric. This will cause problems later when we want to calculate summary statistics and produce data visualizations since R will think that these two variables include words, not numbers (and stats can't be calculated from numbers!). We can change the format of these variables using the `as.numeric()` function. There's a suite of functions like this that can be used to change the format of a variable (e.g., `as.factor()`, `as.integer()`, etc.).

```
d$imidacloprid <- as.numeric(d$imidacloprid)
```

```
## Warning: NAs introduced by coercion
```

```
d$thiamethoxam <- as.numeric(d$thiamethoxam)
```

```
## Warning: NAs introduced by coercion
```

```
str(d)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    480 obs. of  8 variables:
##  $ year                    : num  2011 2011 2011 2011 2011 ...
##  $ sample_month            : chr  "june a" "june a" "june b" "june b" ...
##  $ day_of_year             : num  165 165 179 179 195 195 227 227 258 258 ...
##  $ trt_code                : chr  "UTC" "UTC" "UTC" "UTC" ...
##  $ insecticide_app_method: chr  "untreated control" "untreated control" "untreated control" "untreat
##  $ days_after_planting   : num  47 47 61 61 77 77 109 109 140 140 ...
##  $ imidacloprid          : num  0.885 NA 0.961 NA 1.04 NA 0.908 NA 0.726 NA ...
##  $ thiamethoxam          : num  NA 0.303 NA 0.356 NA 0.436 NA 0.468 NA 0.457 ...
```

Looks much better now!
Also, note that d has been loaded as a "tibble" instead of a traditional dataframe (in the top right corner of the `str()` output, we can see the class is 'tbl_df', 'tbl' and 'data.frame' - which is how we know it's a tibble). A tibble is just a version of a dataframe with a few additional features. Although we could carry out this analysis with `d` as a tibble, we'll go ahead and convert it to a dataframe for the sake of sticking to what we're familiar with. This conversion is done with `as.data.frame()` (see your Data Import cheat sheet).

```
d <- as.data.frame(d)
head(d)
```

```
##   year sample_month day_of_year trt_code insecticide_app_method
## 1 2011       june a         165      UTC      untreated control
## 2 2011       june a         165      UTC      untreated control
## 3 2011       june b         179      UTC      untreated control
## 4 2011       june b         179      UTC      untreated control
## 5 2011         july         195      UTC      untreated control
## 6 2011         july         195      UTC      untreated control
##   days_after_planting imidacloprid thiamethoxam
## 1                  47        0.885           NA
## 2                  47           NA        0.303
## 3                  61        0.961           NA
## 4                  61           NA        0.356
## 5                  77        1.040           NA
## 6                  77           NA        0.436
```

```
tail(d)
```

```
##     year sample_month day_of_year trt_code insecticide_app_method
## 475 2012    september         259   foliar           foliar spray
## 476 2012    september         259   foliar           foliar spray
## 477 2012      october         289   foliar           foliar spray
## 478 2012      october         289   foliar           foliar spray
## 479 2012     november         320   foliar           foliar spray
## 480 2012     november         320   foliar           foliar spray
##     days_after_planting imidacloprid thiamethoxam
## 475                 142        0.334           NA
## 476                 142           NA         1.75
## 477                 172        0.333           NA
## 478                 172           NA         2.06
## 479                 203        0.000           NA
## 480                 203           NA         1.48
```

```r
str(d)
```

```
## 'data.frame':    480 obs. of  8 variables:
##  $ year                  : num  2011 2011 2011 2011 2011 ...
##  $ sample_month          : chr  "june a" "june a" "june b" "june b" ...
##  $ day_of_year           : num  165 165 179 179 195 195 227 227 258 258 ...
##  $ trt_code              : chr  "UTC" "UTC" "UTC" "UTC" ...
##  $ insecticide_app_method: chr  "untreated control" "untreated control" "untreated control" "untreat
##  $ days_after_planting   : num  47 47 61 61 77 77 109 109 140 140 ...
##  $ imidacloprid          : num  0.885 NA 0.961 NA 1.04 NA 0.908 NA 0.726 NA ...
##  $ thiamethoxam          : num  NA 0.303 NA 0.356 NA 0.436 NA 0.468 NA 0.457 ...
```

We can see that we have a dataframe with 480 rows and 8 columns. Let's look a bit more closely at our variables, specifically insecticide_app_method and trt_code.

```r
unique(d$trt_code)
```

```
## [1] "UTC"    "poly"   "if"     "st"     "foliar"
```

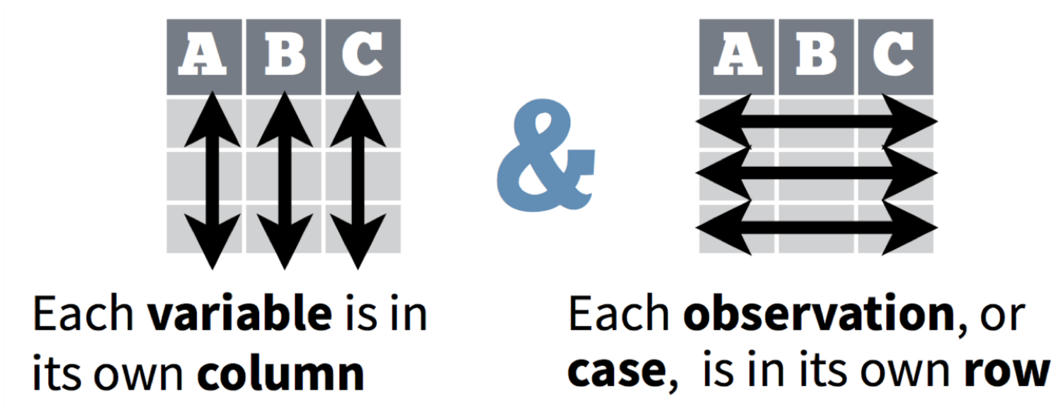
```r
unique(d$insecticide_app_method)
```

```
## [1] "untreated control"          "impregnated polyacrylamide"
## [3] "infurrow spray"             "seed treatment"
## [5] "foliar spray"
```

We see that *insecticide_app_method* includes each of the four application methods discussed in section 1 of this assignment, but we also have "untreated control" as an application. It appears that *trt_code* includes abbreviations of the application methods described in full in *insecticide_app_method*. *trt_code* could come in handy when we're making data visualizations, since it's often difficult to accommodate a long name like "impregnated polyacrylamide" as an axis label.

Tidying the data

In looking at our small data snippet with `head()`, it appears that the data need a bit of tidying. From our Data Import cheat sheet, we know that data are tidy when:


A B C
&
A B C
Each variable is in its own column
Each observation, or case, is in its own row

Do any of our columns not follow this format?
The answer is yes - our active ingredients are in separate columns, but should really be in the same column given the tidy data framework. If this is confusing, think back to our data visualization lab. In order to use functions in ggplot, we grouped data by the unique values in a *column*. If we want to compare the two active ingredients (we do!), when the active ingredients should be listed in the same column.
We can easily merge these columns using the `gather()` function. To do this, we provide the `gather()` function with the dataframe, names of the columns we'd like to gather, a "key" name to describe the new column in which the existing column names will be assigned as variable names, and a "values" name to describe the new column in which the values of the existing columns will be assigned. See your Data Import cheat sheet for more details.

```
d <- gather(d, imidacloprid, thiamethoxam, key = "active_ingredient", value = "concentration")
head(d,10)
```

```
##    year sample_month day_of_year trt_code insecticide_app_method
## 1  2011       june a         165      UTC      untreated control
## 2  2011       june a         165      UTC      untreated control
## 3  2011       june b         179      UTC      untreated control
## 4  2011       june b         179      UTC      untreated control
## 5  2011         july         195      UTC      untreated control
## 6  2011         july         195      UTC      untreated control
## 7  2011       august         227      UTC      untreated control
## 8  2011       august         227      UTC      untreated control
## 9  2011    september         258      UTC      untreated control
## 10 2011    september         258      UTC      untreated control
##    days_after_planting active_ingredient concentration
## 1                   47      imidacloprid         0.885
## 2                   47      imidacloprid            NA
## 3                   61      imidacloprid         0.961
## 4                   61      imidacloprid            NA
## 5                   77      imidacloprid         1.040
## 6                   77      imidacloprid            NA
## 7                  109      imidacloprid         0.908
## 8                  109      imidacloprid            NA
## 9                  140      imidacloprid         0.726
## 10                 140      imidacloprid            NA
```

Why do you think we used an underscore in the "key" name?
The active ingredient columns were successfully gathered into two new columns - one for the active ingredient name, and the other for the concentration value. But, we can see that we now have "NA" for several of our rows in the *concentration* column. These NAs were pulled from the previous dataframe. Although it's perfectly fine to have NAs in your dataframes, it can sometimes help to remove the NAs since they can

interfere with calculations (e.g., running `mean()`). To remove the NAs, we can use the `drop_na()` function. See your Data Import cheat sheet for more details.

```r
d <- drop_na(d, concentration)
head(d,10)
```

```
##    year sample_month day_of_year trt_code insecticide_app_method
## 1  2011       june a         165      UTC      untreated control
## 3  2011       june b         179      UTC      untreated control
## 5  2011         july         195      UTC      untreated control
## 7  2011       august         227      UTC      untreated control
## 9  2011    september         258      UTC      untreated control
## 11 2011      october         288      UTC      untreated control
## 13 2011       june a         165      UTC      untreated control
## 15 2011       june b         179      UTC      untreated control
## 17 2011         july         195      UTC      untreated control
## 19 2011       august         227      UTC      untreated control
##    days_after_planting active_ingredient concentration
## 1                   47      imidacloprid         0.885
## 3                   61      imidacloprid         0.961
## 5                   77      imidacloprid         1.040
## 7                  109      imidacloprid         0.908
## 9                  140      imidacloprid         0.726
## 11                 170      imidacloprid         0.628
## 13                  47      imidacloprid         0.000
## 15                  61      imidacloprid         0.503
## 17                  77      imidacloprid         0.383
## 19                 109      imidacloprid         0.259
```

Starting to look pretty tidy! An additional bit of "messiness" in the data appears in the *sample_month* column. Let's look a bit more closely.

```r
unique(d$sample_month)
```

```
## [1] "june a"    "june b"    "july"      "august"    "september" "october"  
## [7] "june"      "november"
```

Here, we see that there's a "june a" and "june b". Although the researchers surely had a reason for specifying "a" and "b", looking at this level of detail is outside of the scope of this assignment. If we leave the "a" and "b", the functions we call to produce summary stats and visualizations will think that "june a" is distinct from "june b", but we just want R to consider "june". We can fix this be renaming "june a" and "june b" to "june". We can do this with `recode()`. `recode()` is listed on your Data Transformation cheat sheet, but not with much detail. For more information on this function, call `help(recode)`.

```r
d$sample_month <- recode(d$sample_month, `june a` = "june", `june b` = "june")
head(d)
```

```
##    year sample_month day_of_year trt_code insecticide_app_method
## 1  2011         june         165      UTC      untreated control
## 3  2011         june         179      UTC      untreated control
## 5  2011         july         195      UTC      untreated control
## 7  2011       august         227      UTC      untreated control
## 9  2011    september         258      UTC      untreated control
## 11 2011      october         288      UTC      untreated control
##    days_after_planting active_ingredient concentration
## 1                   47      imidacloprid         0.885
## 3                   61      imidacloprid         0.961
## 5                    77      imidacloprid        1.040
## 7                   109      imidacloprid        0.908
## 9                   140      imidacloprid        0.726
## 11                  170      imidacloprid        0.628
```

Write your file

Let's save our cleaned data file so that we can use it again in the future. To save the file, we'll use the `write_csv()` function. This function saves dataframes as comma separated value (csv) files, which can be opened in Excel or a text editor. To run this function, we provide it with our dataframe object as well as a file name. The file will be saved to your working directory.

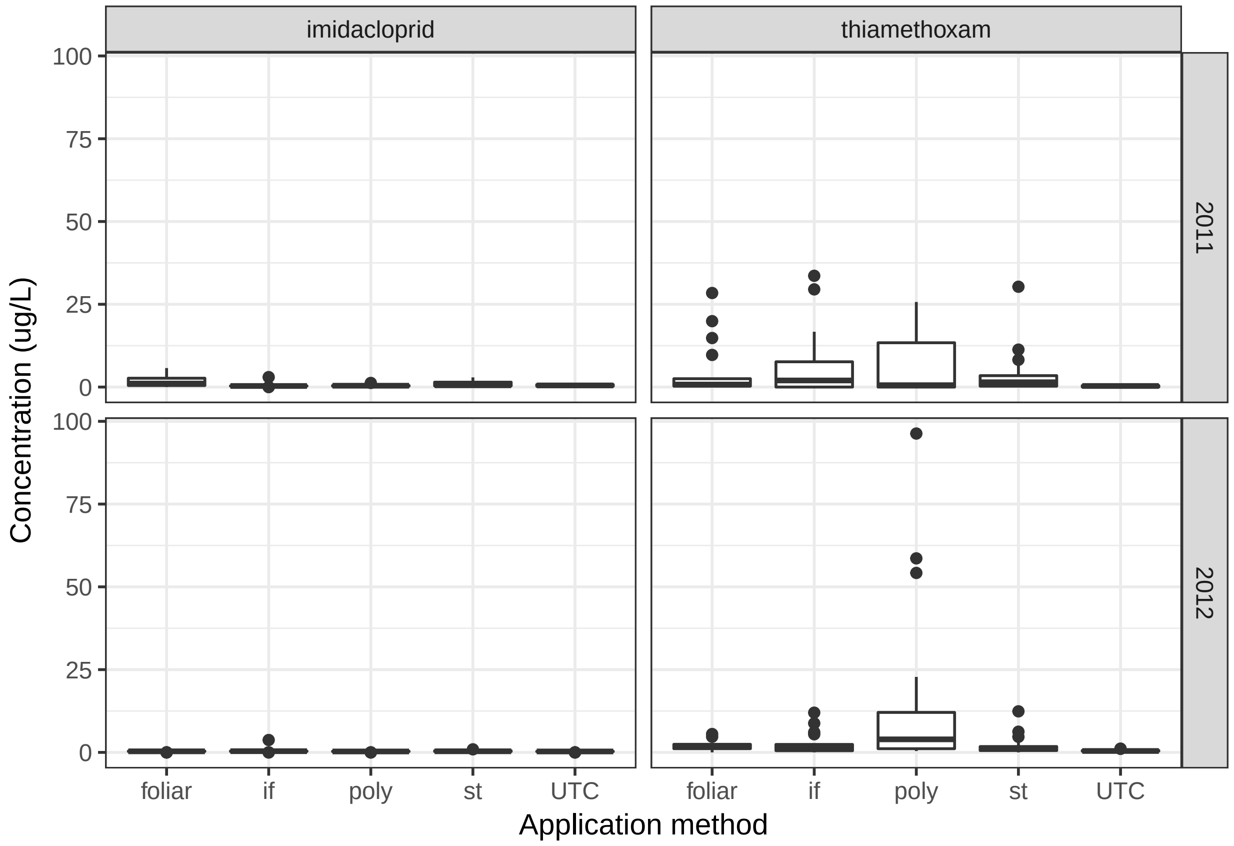
```r
write_csv(d,"neonicotinoids_clean.csv")
```

Note that we included the file extension (i.e., ".csv") when specifying our file name. Go ahead and check that your data are saved and can be opened in Excel.

Back to data viz: facet_grid()

You'll recall from our previous problem session that we used `facet_wrap()` to create multi-panel plots. We had divided the panels based on 1 variable (e.g., Year), but we can also create multi-panel plots based on 2 variables through the use of `facet_grid()`. To create a 2-dimensional multi-panel plot, you provide a formula argument to `facet_grid()` in the form of `rows~columns`. Additional details can be seen on the back of your Data Visualization cheat sheet, and an example is provided below.

```r
ggplot(d,aes(x=trt_code,y=concentration))+
  geom_boxplot()+
  facet_grid(year~active_ingredient)+
  xlab("Application method")+
  ylab("Concentration (ug/L)")+
  theme_bw()
```


imidacloprid
thiamethoxam
2011
2012
Concentration (ug/L)
100
75
50
25
0
foliar
if
poly
st
UTC
Application method