R Tutorial-09

T. Scofield

You may click here to access the .qmd file.

In this issue, we investigate applying operations involving one variable of a data frame in a manner that slices data based on the value of another variable. Specifically, we

- review some basic operations (calculating statistics, producing graphs, etc.)
- achieve similar results via techniques that slice the data

Univariate operations with mosaic and ggformula packages

frequency table

If you want a frequency table:

```
ssurv = read.csv("http://scofield.site/teaching/data/csv/ssurv.csv")     # some survey data
tally(~Species, data=iris)
```

```
Species
    setosa versicolor  virginica
        50         50         50
```

mean

If you want a mean/average:

```
mean(~cds, data=ssurv)
```

```
[1] NA
```

The reason this produced `NA` is that there were missing values in the `cds` column of the data frame. An additional instruction (see below) removes those entries and produces the desired result.

```
mean(~cds, data=ssurv, na.rm=TRUE)
```

```
[1] 50.53261
```

median

Computing the median:

```
median(~cds, data=ssurv, na.rm=TRUE)
```

```
[1] 35
```

variance

A (sample) variance

```
var(~cds, data=ssurv, na.rm=TRUE)
```

```
[1] NA
```

Five-number summary

A five-number summary, provides the minimum, the 25th percentile, the median, the 75th percentile, and the maximum:

```
qdata(~Petal.Length, data=iris)
```

```
  0%  25%  50%  75% 100%
1.00 1.60 4.35 5.10 6.90
```

Specific quantiles/percentiles in the data

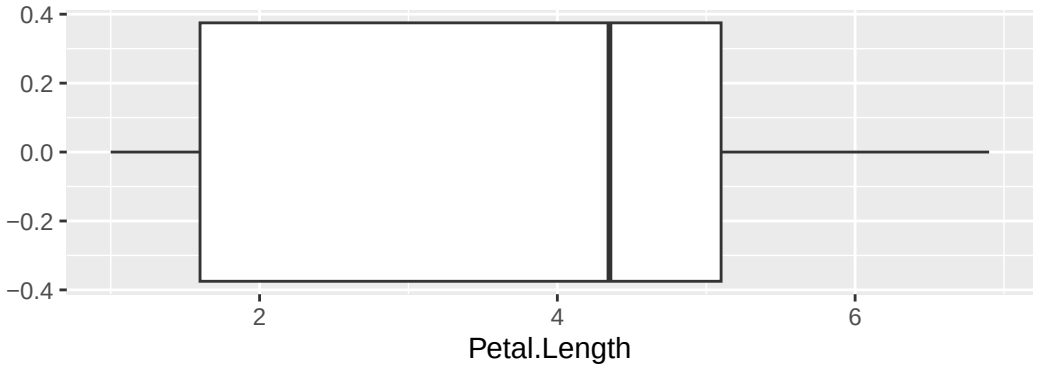
```
qdata(~Petal.Width, data=iris, p=seq(.1,.9,.1))
```

```
 10%  20%  30%  40%  50%  60%  70%  80%  90%
0.20 0.20 0.40 1.16 1.30 1.50 1.80 1.90 2.20
```

A boxplot

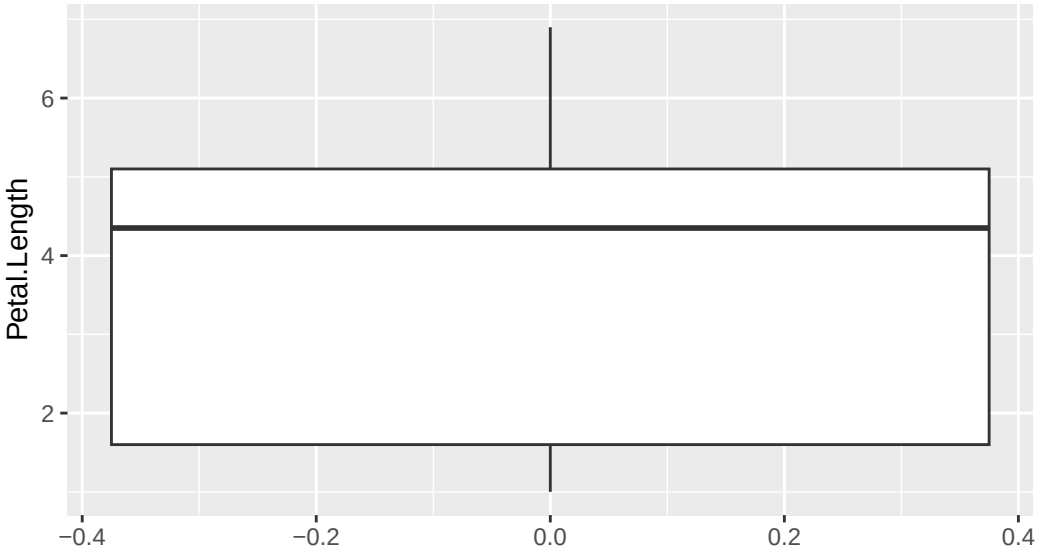
A boxplot, or box-and-whisker plot, is a visual representation of the 5-number summary:

```
gf_boxplot(~Petal.Length, data=iris)
```



or the same, but oriented vertically

```
gf_boxplot(Petal.Length ~ ., data=iris)
```

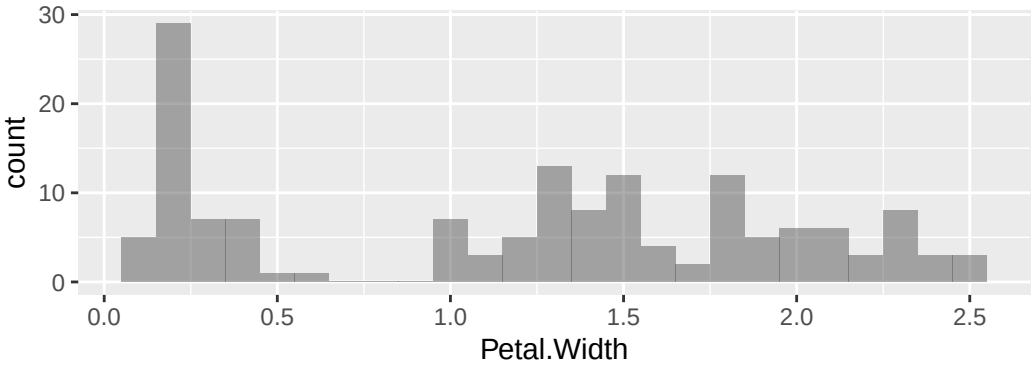


Note that, in each case, one axis contains value which are essentially meaningless.

A histogram

For some commands, you can pipe the data frame to the command, eliminating the need for `data=`. This seems to work with graph-producing commands, as I illustrate here with `gf_histogram()`:

```
gf_histogram(~Petal.Width, data=iris)
```



Applying values of another variable to get sliced results

frequency table for region by sex

The `ssurv` data frame contains information about the `region` where students live as well as their `sex`. We might want a two-way table for `region`, with frequencies contingent on this latter variable:

```
tally(~region | sex, data=ssurv)
```

```
          sex
region      F  M
            1  0
  Rural    25 25
  Suburban 92 98
  Urban    21 18
```

You might try out these modifications which also produce two-way tables:

```
tally(region ~ sex, data=ssurv)
tally(sex ~ region, data=ssurv)
```

mean computed individually for groups

There are 150 plants from 3 different species in the `iris` data frame. We might want mean `Petal.Length` calculated individually by species:

```
mean(~Petal.Length | Species, data=iris)
```

```
    setosa versicolor  virginica
     1.462      4.260      5.552
```

```
# mean(Petal.Length ~ Species, data=iris)  alternate syntax, does the same thing
# mean(Species ~ Petal.Length, data=iris)  Note: This one behaves quite differently!
```

quantiles grouped by values of another variable

Similarly, you may want the 0.3-quantile for each of the species:

```
qdata(Petal.Length ~ Species, data=iris, p=0.3)
```

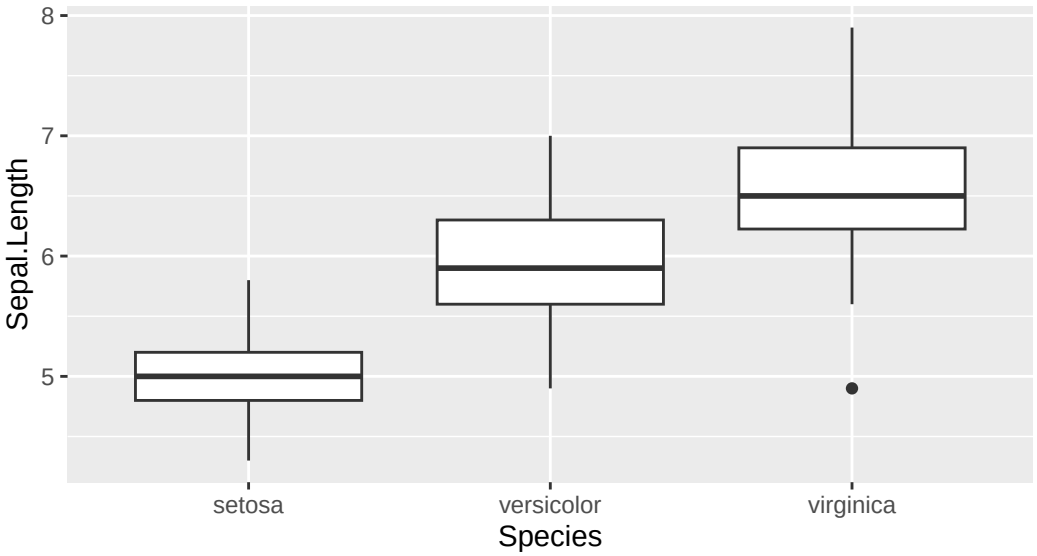
```
     Species 30%
1     setosa 1.4
2 versicolor 4.0
3  virginica 5.1
```

---

side-by-side boxplots

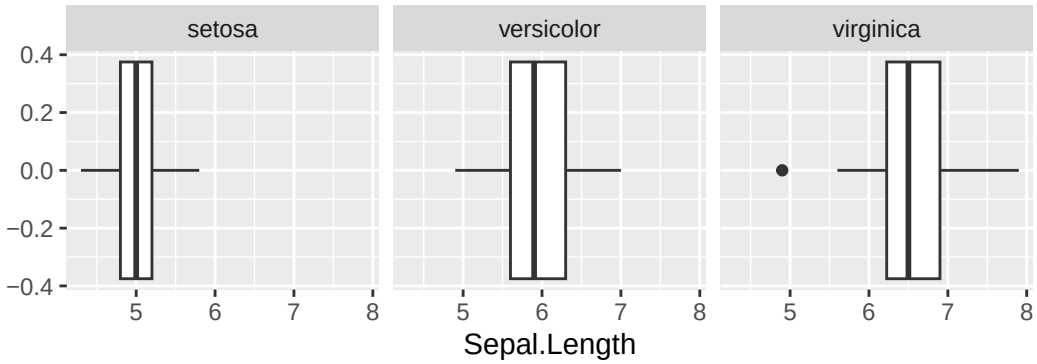
Graphics are, by nature, a little more interesting. Say you want a boxplot for each value of a grouping variable. The easiest side-by-side boxplots, in terms of syntax, might be these:

```
iris |> gf_boxplot(Sepal.Length ~ Species)
```



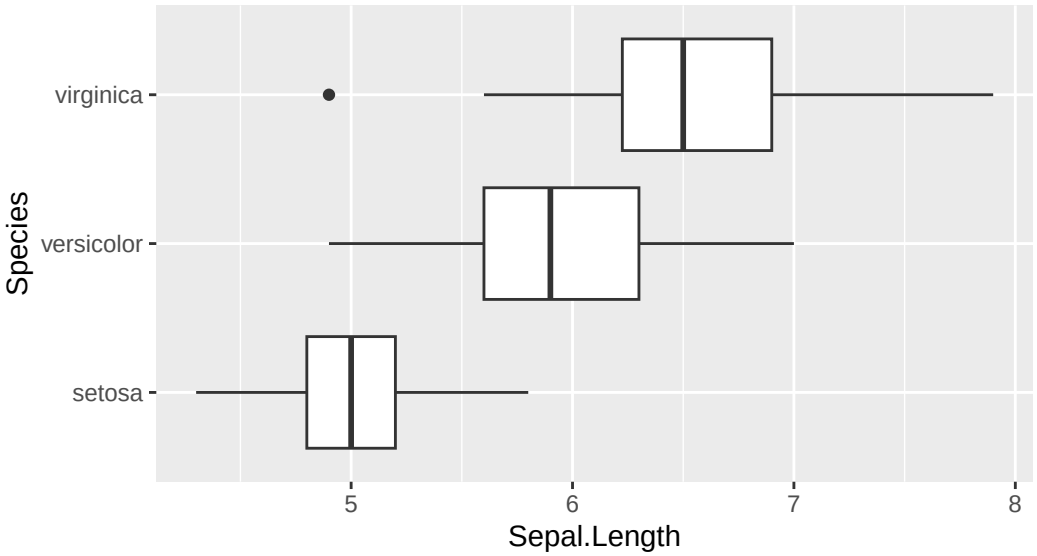
The presence of the grouping variable has made labels on the 2nd axis meaningful. Other variants work. This one is probably not as useful/appealing:

```
gf_boxplot(~Sepal.Length | Species, data=iris)
```

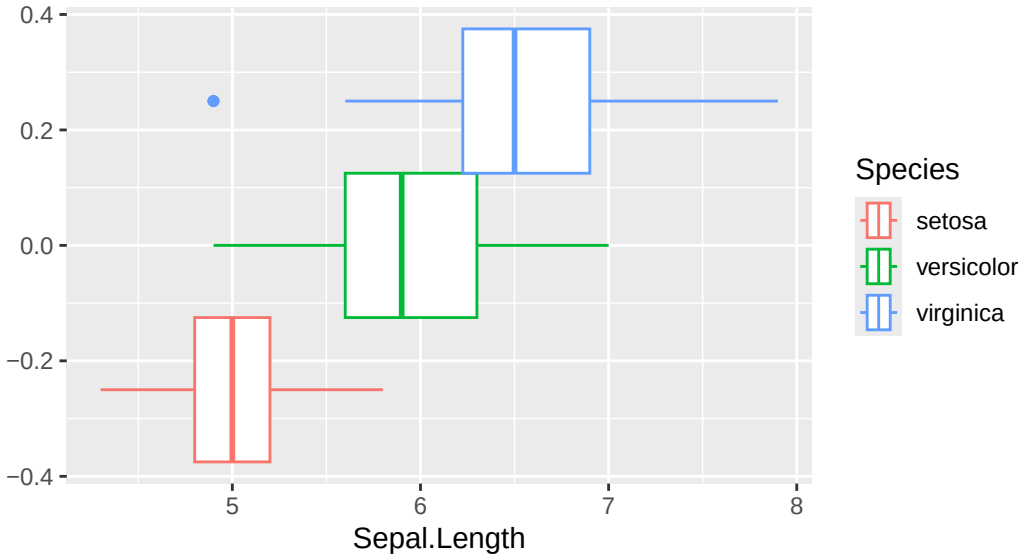


Two other variants which produce different takes on the slicing idea are these, one that reverses the role of the axes, and the other introducing color:

```
gf_boxplot(Species ~ Sepal.Length, data=iris)
```



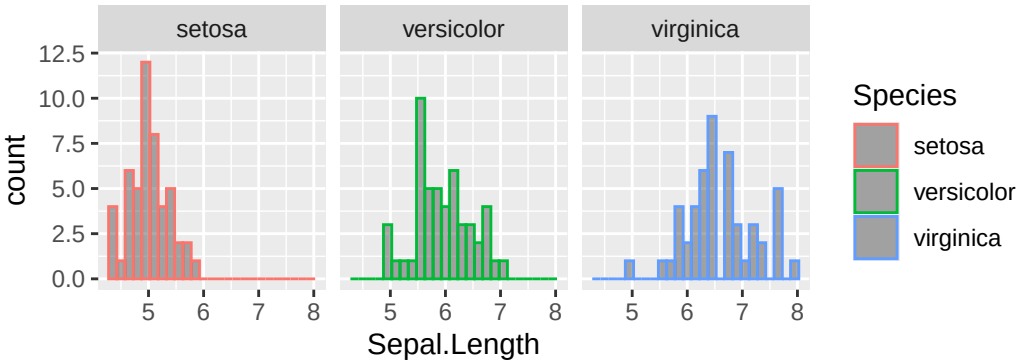

```
gf_boxplot(~Sepal.Length, color=~Species, data=iris)
```



histograms by slices

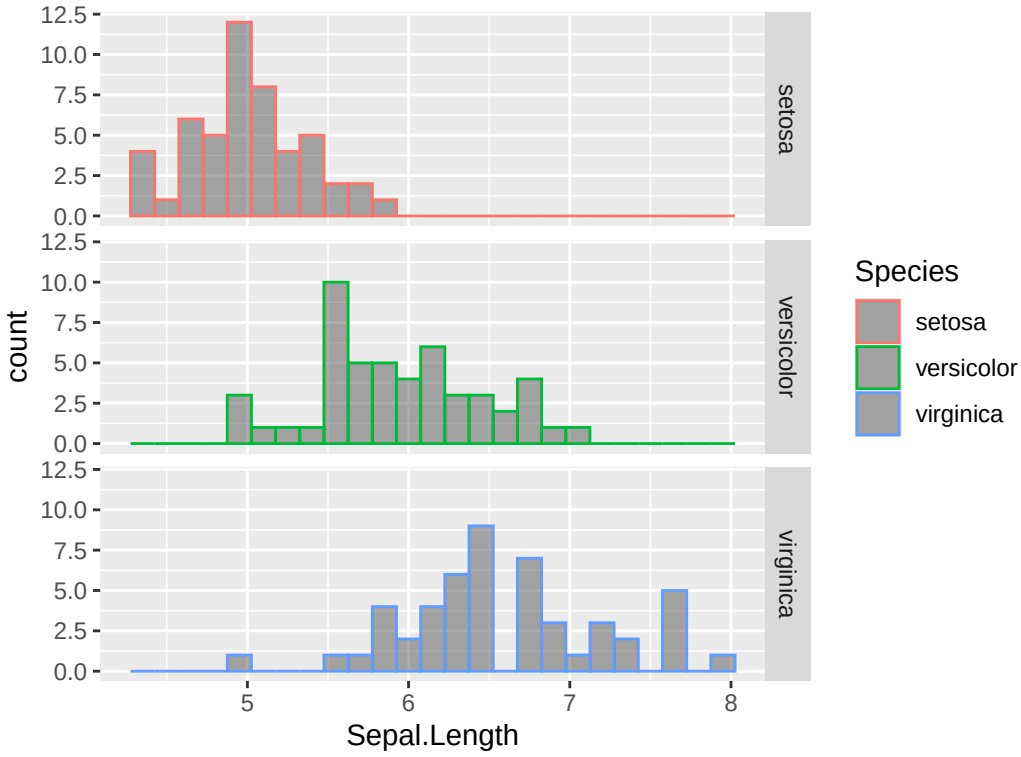
The view the same quantitative variable, `Sepal.Length`, compared across `Species` using histograms:

```
iris |> gf_histogram(~ Sepal.Length | Species, color=~Species)
```



One complaint might be that it would be easier to compare differences if the histograms were stacked. That's possible using this variant:

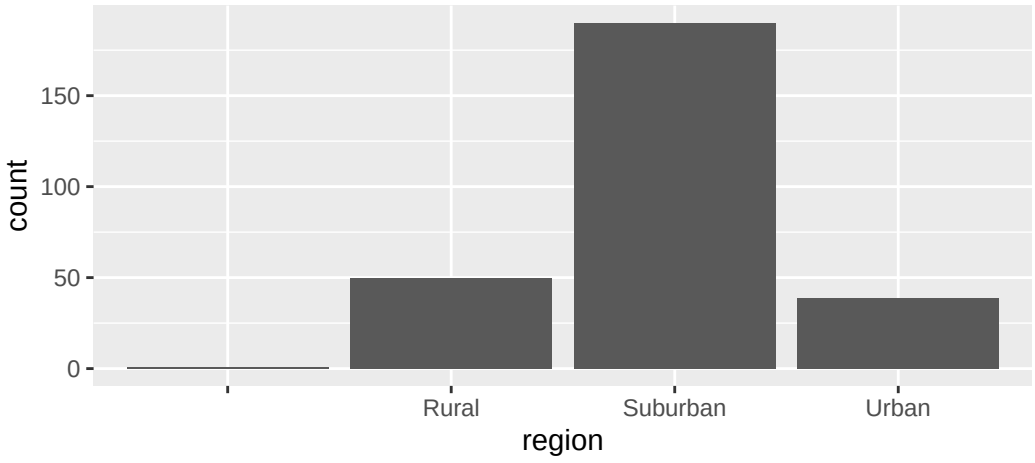
```
iris |> gf_histogram(~ Sepal.Length | Species ~ ., color=~Species)
```



bar charts without and with slicing

The `region` variable from the `ssurv` data (see above) can be visualized using a bar chart:

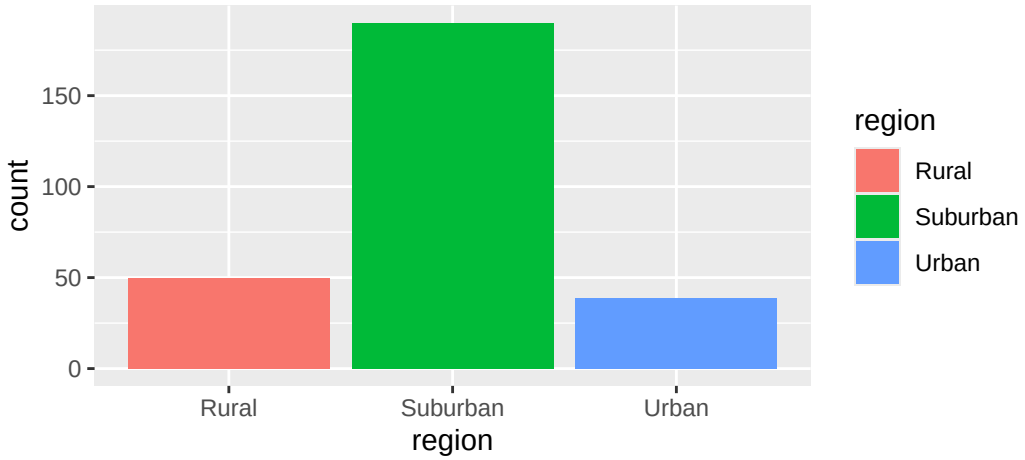
```
ssurv |> gf_bar(~region)
```




```
# gf_props(~region, data=ssurv)   # this version gives relative frequency
```

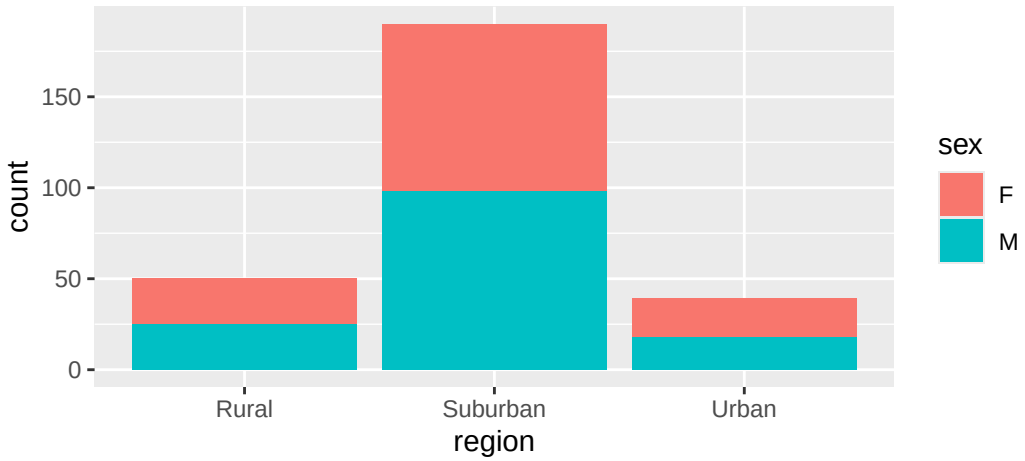
No slicing was done for this graph using values of a second variable. Before implementing slicing, let's filter out the one survey respondent who left `region` blank. In addition, I have specified to have the different regions colored in with different colors:

```
ssurv |> filter(region != "") |>
  gf_bar(~region, fill=~region)
```



We can change the `fill` variable to obtain slices by, say, `sex`:

```
ssurv |> filter(region != "") |>
  gf_bar(~region, fill=~sex)
```



You might try out this variant, not executed here, to see if you prefer the bars unstacked:

```
ssurv |> filter(region != "") |>
  gf_bar(~region, fill=~sex, position="dodge")
```

boxplots in Figure 8.2

The data frame, `brake`, from the `fosdata` package, has 20 variables measured on 80 cases. These variables include

`latency_p1`, `latency_p2`, `latency_p3`, all of which are quantitative

`age_group`, a binary categorical variable

The opening to Chapter 8 contains side-by-side boxplots of these three variables, broken down by `age_group`. To get this plot, the authors do some pre-processing—namely, they stack the values of `latency_p1, latency_p2,` and `latency_p3` into a single column (which means a data frame with 240 rows) called `time`, and creates a new column called `type` that says which of the three columns the value came from. The relevant command which achieves this stacking, `pivot_longer()`, is in the `tidyr` package:

```
nrow(fosdata::brake)
```

```
[1] 80
```

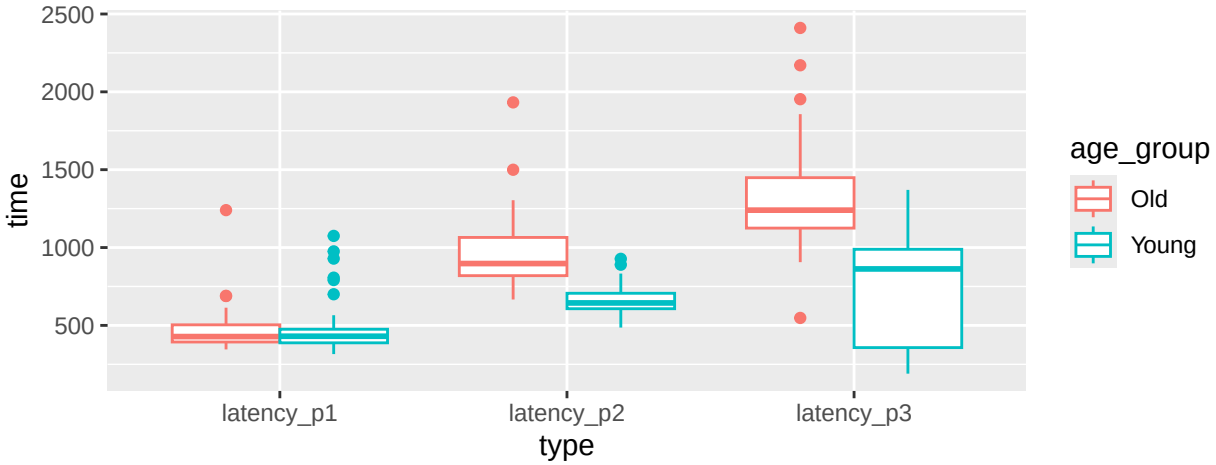
```
fosdata::brake |>
  tidyr::pivot_longer(
    cols=matches("^lat"),
    names_to = "type",
    values_to = "time"
  ) -> reorganizedBrake
nrow(reorganizedBrake)
```

```
[1] 240
```

This sort of pre-processing of data is facilitated by packages such as `tidyr` and `dplyr`. Chapter 6 is devoted in its entirety to discussing tricks using these packages, but would take us a bit away from the business of statistics. Nevertheless, it is worth a careful read, at some point *in the future*, by those who want to make a living mining data.

Figure 8.2 is the result of slicing up values in the newly-created `time` variable, but using values in two separate columns, `type` and `age_group`.

```
reorganizedBrake |> gf_boxplot(time ~ type, color=~age_group)
```



The resulting plot looks identical to Figure 8.2, and shows how the plotting functions we use (taken from `ggformula` package) offer `ggplot2`-style graphics.