The tidyverse ecosystem is steadily growing and adapting to the needs of its users. As part of this evolution, existing tools are being replaced by new and better methods. As useful as this flexibility is to the strength of the system, sometimes it can be hard to keep track of all the changes. **This blogpost will deal with two new developments: the 'curly-curly' operator for tidy evaluation and the new 'pivot' functions for data reshaping.**

We will need the following libraries; in particular make sure that you have installed at least `tidyr` version 1.0:

```
libs <- c('dplyr', 'tidyr',          # wrangling
          'readr', 'stringr',        # wrangling
          'knitr','kableExtra',      # table styling
          'ggplot2','gridExtra',     # plots
          'leaflet')                 # interactive maps
invisible(lapply(libs, library, character.only = TRUE))
```

For this post's dataset we'll be doing something different than sampling the usual in-built tables. We will work with the famous NYC Squirrel Census – cataloguing the squirrel population of New York's Central Park in October 2018. The squirrel census is a great example for a citizen science project that's both accessible fun and useful scientific contribution.

Talking about fun community projects: the specific dataset we're using has been prepared by the TidyTuesday project, a weekly social exercise to test our tidyverse skills and provide inspiration. TidyTuesday is run by the R for Data Science Online Learning Community. All datasets are available on github:

```
squirrels <- read_csv(str_c(
  "https://raw.githubusercontent.com/",
  "rfordatascience/tidytuesday/master/",
  "data/2019/2019-10-29/nyc_squirrels.csv"))
```

```
squirrels %>%
  select(lat, long, date, running, climbing, eating) %>%
  head(5) %>%
  kable() %>%
  kable_styling()
```

lat	long	date	running	climbing	eating
40.79408	-73.95613	10142018	FALSE	FALSE	FALSE
40.79485	-73.95704	10062018	TRUE	FALSE	FALSE
40.76672	-73.97683	10102018	FALSE	TRUE	FALSE
40.76970	-73.97572	10182018	FALSE	TRUE	FALSE
40.79753	-73.95931	10182018	FALSE	FALSE	FALSE

Among other features, the squirrels data includes the latitude and longitude of the sighting, the date, and flags that indicate whether this squirrel was spotted running, climbing, or eating. True to tidy form, each row corresponds to one squirrel. There are many more features in the full dataset, but we will focus on the ones above in the following examples.

First, we will cover tidy evaluation with the `{{ }}` operator aka 'curly-curly'.

Introduced as part of rlang version 0.4.0 back in June, curly-curly replaces the approach of quoting with `enquo` and unquoting with `!!` (aka 'bang-bang'). Instead of two operations it provides a compact shorthand for simple cases. Here is a brief example for counting groups of distinct feature values:

```
# new style
count_groups <- function(df, groupvar){
  df %>%
    group_by({{ groupvar }}) %>%
    count()
}

count_groups(squirrels, climbing) %>%
  kable() %>%
  kable_styling()
```

climbing	n
FALSE	2365
TRUE	658

In this example, `{{ groupvar }}` splices the value of `groupvar` into the `group_by` call, rather than its name. This is equivalent to the quote-unquote style of `!! enquo(groupvar)`:

```
# old style
count_groups_old <- function(df, groupvar){
  df %>%
    group_by(!! enquo(groupvar)) %>%
    count()
}

count_groups_old(squirrels, climbing) %>%
  kable() %>%
  kable_styling()
```

climbing	n
FALSE	2365
TRUE	658

The advantage of `{{ }}` lies in its relative cognitive ease: think of it as inserting the value of the variable into the expression. No intermediate quoting or unquoting needed.

Next, we will look at how to reshape data with the new functions `pivot_wider` and `pivot_longer`, which are replacing the previous `spread` and `gather` tools. Most importantly: `pivot_wider` is the inverse function to `pivot_longer`, and vice versa.

As the name suggests, `pivot_wider` makes a tibble wider by turning a single categorical column into multiple columns, one for each category.

Let's look at this aggregated data frame of sightings of climbing squirrels per day:

```
squirrels %>%
  count(date, climbing) %>%
  head(4) %>%
  kable() %>%
  kable_styling()
```

date	climbing	n
10062018	FALSE	253
10062018	TRUE	84
10072018	FALSE	283
10072018	TRUE	122

Pivoting to a wider format allows us to compare the numbers of climbing and non-climbing squirrels directly next to each other:

```
# new style with pivot_wider
squirrels %>%
  count(date, climbing) %>%
  pivot_wider(names_from = climbing, values_from = n, names_prefix =
"climbing_") %>%
  head(2) %>%
  kable() %>%
  kable_styling()
```

date	climbing_FALSE	climbing_TRUE
10062018	253	84
10072018	283	122

- The `names_from` argument indicates which column supplies the new column names. The values of this column are being picked from the feature defined via the `values_from` keyword. Here, we pick the names from the binary feature `climbing` and the values from the count column `n`.
- Among the additional keywords, `names_prefix` allows us to assign meaningful names to the new columns (which would otherwise simply be the rather generic 'FALSE' and 'TRUE' here). Another useful keyword is `values_fill`, which specifies a global replacement for any missing values.
- The new `pivot_wider` replaces the old `spread` function, which had comparable yet possibly more confusing parameters. For comparison, here is the same result with `spread`:

```
# old style with spread
squirrels %>%
  count(date, climbing) %>%
  spread(key = climbing, value = n) %>%
  head(2) %>%
  kable() %>%
  kable_styling()
```

date	FALSE	TRUE
10062018	253	84
10072018	283	122

The syntax is very similar, which should make it easy for those familiar with `spread` to switch to `pivot_wider`. Note, that `spread` had no `names_prefix` argument.

In most tutorials, `pivot_wider` is somewhat overshadowed by `pivot_longer`; but I use it frequently to quickly compute proportions for grouped columns. For instance, here are the top 3 days for spotting squirrels climbing:

```
squirrels %>%
  count(date, climbing) %>%
  pivot_wider(names_from = climbing, values_from = n, names_prefix =
"climbing_") %>%
  mutate(climbing_percentage = climbing_TRUE/(climbing_TRUE +
climbing_FALSE)*100) %>%
  arrange(desc(climbing_percentage)) %>%
  head(3) %>%
  kable() %>%
  kable_styling()
```

date	climbing_FALSE	climbing_TRUE	climbing_percentage
10072018	283	122	30.12346
10062018	253	84	24.92582

date	climbing_FALSE	climbing_TRUE	climbing_percentage
10082018	220	65	22.80702

Let's move on to `pivot_longer`. Being the inverse function to `pivot_wider`, this tool (often) reduces the number of columns by turning some of the columns into a single new (typically) categorical feature alongside an (often times) numerical feature. The different levels of the categorical column now describe the numerical column in a unique way (similarly to the wider column structure previously). This is all a bit abstract, so let's do a quick example:

Imagine we've extracted both the proportion of climbing squirrels and non-climbing squirrels for each day with the help of `pivot_wider`. (Of course, one number determines the other but let's use it as a simple illustration).

```
squirrels %>%
  count(date, climbing) %>%
  pivot_wider(names_from = climbing, values_from = n, names_prefix =
"climbing_") %>%
  mutate(prop_true = climbing_TRUE/(climbing_TRUE + climbing_FALSE)*100,
         prop_false = climbing_FALSE/(climbing_TRUE + climbing_FALSE)*100) %>%
  head(2) %>%
  kable() %>%
  kable_styling()
```

date	climbing_FALSE	climbing_TRUE	prop_true	prop_false
10062018	253	84	24.92582	75.07418
10072018	283	122	30.12346	69.87654

Now we will use `pivot_longer` to turn those two `prop_` columns into a categorical `climbing` and a numerical `percentage` feature. Here, we drop the `climbing_FALSE` and `climbing_TRUE` columns, so the pivoting doesn't change the number of columns but it changes the structure of the dataset:

```
squirrels %>%
  count(date, climbing) %>%
  pivot_wider(names_from = climbing, values_from = n, names_prefix =
"climbing_") %>%
  mutate(prop_true = climbing_TRUE/(climbing_TRUE + climbing_FALSE),
         prop_false = climbing_FALSE/(climbing_TRUE + climbing_FALSE)) %>%
  select(date, prop_true, prop_false) %>%
  pivot_longer(cols = c("prop_true", "prop_false"),
               names_to = "climbing", values_to = "percentage") %>%
  head(4)  %>%
  kable() %>%
  kable_styling()
```

date	climbing	percentage
10062018	prop_true	0.2492582
10062018	prop_false	0.7507418
10072018	prop_true	0.3012346
10072018	prop_false	0.6987654

- The `cols` argument specifies the columns that will be pivoted. Then, `names_to` gives the name of the new feature that will hold the (categorical) names of the original columns. And `values_to` is the new feature that hold their values (here the percentages).
- Note, that as with the older `gather` method, the new column names have to be passed as strings.
- The resulting data structure is often much better suited for plotting with `ggplot2`. The categorical

feature can directly become a colour, fill, or faceting variable.

Finally, let's combine curly-curly and pivoting into a comprehensive example. Here, we build a function that takes as arguments the name of a data frame and the name of a logical column, here a squirrel action, and then extracts the percentage of this action per coordinate bin. This is it:

```
extract_percentage <- function(df, col) {

  df %>%
    mutate(lat = round(lat*5, 2)/5,
         long = round(long*5, 2)/5) %>%
    count(lat, long, {{ col }}) %>%
    pivot_wider(names_from = {{ col }}, values_from = n,
                values_fill = list(n = 0)) %>%
    mutate(true = `TRUE`/(`TRUE` + `FALSE`)*100,
           false = `FALSE`/(`TRUE` + `FALSE`)*100) %>%
    select(lat, long, true, false) %>%
    pivot_longer(cols = c("true", "false"),
                 names_to = "action", values_to = "percentage") %>%
    filter(action == "true")
}

extract_percentage(squirrels, climbing) %>%
  head(3)  %>%
  kable() %>%
  kable_styling()
```

lat	long	action	percentage
40.764	-73.974	true	50.00000
40.766	-73.978	true	21.73913
40.766	-73.976	true	19.23077

- The coordinates are rounded the nearest 0.02 degrees to provide the sample size for summary statistics.
- We only keep the percentage of positive sightings for each action.

We're now using this function to extract the proportions of squirrels that were observed eating, climbing, or running. Then we visualise those proportions on an interactive map of Manhattan centred on Central Park. The map is constructed using the wonderful leaflet package. Such a map could be used to find locations in the park that might be more promising than others for spotting certain squirrel shenanigans.

```
loc <- extract_percentage(squirrels, eating)

pal <- colorNumeric(palette = "RdBu", domain = seq(0,100), reverse = TRUE)

leaflet(loc) %>%
  setView(lng = median(loc$long), lat = median(loc$lat), zoom = 13) %>%
  #addProviderTiles("Esri.NatGeoWorldMap") %>%
  addProviderTiles(providers$CartoDB.DarkMatter) %>%
  addCircleMarkers(~ long, ~ lat,
                   data = extract_percentage(squirrels, eating),
                   group = "Eating",
                   color = ~ pal(percentage),
                   radius = 6, fillOpacity = 0.7, stroke = FALSE) %>%
  addCircleMarkers(~ long, ~ lat,
                   data = extract_percentage(squirrels, climbing),
                   group = "Climbing",
                   color = ~ pal(percentage),
                   radius = 6, fillOpacity = 0.7, stroke = FALSE) %>%
  addCircleMarkers(~ long, ~ lat,
                   data = extract_percentage(squirrels, running),
                   group = "Running",
                   color = ~ pal(percentage),
                   radius = 6, fillOpacity = 0.7, stroke = FALSE) %>%
  addLayersControl(baseGroups = c("Eating", "Climbing", "Running"),
                   options = layersControlOptions(collapsed = FALSE)) %>%
  addLegend("bottomright", pal = pal, values = ~ percentage,
            title = "Percentage", labFormat = labelFormat(suffix = "%"),) %>%
  addScaleBar("bottomleft")
```

- This is an interactive map that is fully movable and zoomable, provided you are connected to the internet. *If you aren’t connected to the internet: how did you reach this website? Teach me your dark and mysterious skills.* Speaking of dark: I chose a dark background map because I like the way it contrasts with the colours of the circles. Looks especially sweet in dark mode. Click on the subgroups in the upper right corner to see the spatial distributions of their percentages.

- The code is pretty self explanatory, and I will save a more detailed exploration of leaflet for a future blog post. The different sub-datasets for each action are added as `CircleMarkers` using our `èxtract_percentages` function. For the colour-coding we define a diverging palette. Each dataset is a layer with a `group` attribute which is used to define the switch between them.

- Among the things we see in the data, there is a tendency for sightings of squirrels eating to be located in the inner part of the park vs the edges. A lot of climbing happens on the east side; and there’s more running in the north. *Feel free to supply fitting stereotypes or jokes about Manhattan here – I’m blissfully unaware of any.*

- For a more serious analysis there would the caveat that some of the spatial bins will have small numbers of squirrels overall, which leads to large uncertainties for those percentages. This is something that would need to be taken into account in a deeper analysis.