Chapter 14: Dynamic and Customized Data Graphics

- In Chapters 2 and 3, a large number of commonly used static graphical methods were discussed.
- In Chapter 14, we explore more advanced graphics.
- A *static* graph is a fixed, unchanging display of data information.
- Static graphs are well suited for publication in printed articles.
- *Dynamic* data graphics have to ability to change, either automatically or based on user control.
- Dynamic graphics are well suited for display on computers, tablets, smartphones, etc., where changes in the graphical output can be shown in real time.

A Few Uses of `htmlwidgets`

- With modern Web browsers, it is possible to have *interactive* data visualizations directly within web browsers.
- Dynamic data graphics on the Web can be facilitated with a JavaScript library called `D3`.
- The `htmlwidgets` package is a bridge between R and `D3`, so that R users can use `D3` without needing to learn JavaScript.
- All of these `htmlwidgets` plots that follow are automatically interactive.
- The `leaflet` package has become popular for making dynamic geospatial maps — it is covered in Chapter 17 of the textbook.

Dynamic Data Visualizations using `plotly`

- The `plotly` package allows R users to take advantage of the functionality of the `plotly.js` JavaScript library.
- You can convert any `ggplot2` object into a `plotly` object using the `ggplotly` function.
- From there, the user can immediately have interactive tools such as:
  1. *brushing* (highlighting/marking selected points)
  2. *mouse-over annotations* (where additional information is displayed when you hover over points with the mouse)
  3. *zooming* (showing close-up displays of selected portions of the graph)
- See the example from the textbook with the Beatles names plot.

Interactive DataTables, Dygraphs, and Streamgraphs

- The `DT` package allows the creation of interactive data tables.
- These tables are automatically searchable, sortable, and pageable.
- The `dygraphs` package produces interactive time series plots.
- With dygraphs, we can brush over and easily select particular time intervals and zoom in and out on the plot.
- The *streamgraph* package can be installed with the `install_github` function in the `remotes` package.
- The streamgraph uses area rather than magnitude to display values over time.
- See examples of these types of tables and graphs using the Beatles names plot.

Animation in Plots

- The `gganimate` package is convenient for producing animated plots that show as *GIF*s that change rapidly over time.
- When the information in a data set changes over time (or over the values of some other variables), it is possible to illustrate the changing picture with an *animation plot*.
- These appear like videos to our eye, since they are sequences of static plots that are frames that flip quickly as the values of the time variable change.

Customizing Animation Plots

- The `transition_time` argument will specify the name of the variable that is changing with the frames.
- If the plot is changing over levels of a discrete variable, the `transition_states` argument can specify the name of this discrete variable.
- You can customize the number of frames and the transition speed (frames per second) by creating a plot object and inputting that object into the `animate` function and adjusting the `nframes` and `fps` arguments.
- See the animation examples with the Beatles names and gender-neutral names plots.

Dashboards and Shiny Apps

- Section 14.3 discusses using the `flexdashboard` package to create visualizations as a *dashboard*, which mixes graphical and textual information.
- This is best done in conjunction with R Markdown and RStudio — we will not show it on the classroom computer.
- Section 14.4 presents the basics of Shiny apps.
- Shiny is a powerful framework that can create interactive web applications (apps) and dynamic dashboards.

Creating and Running Shiny Apps

- The basic components of a Shiny app are:
  1. a `ui.R` file with code that sets up and controls the user interface of the app;
  2. a `server.R` file with code that displays the results.
- These files can be stored in a folder on the local computer and the `source` command will send the contents of the files directly into R.
- Then calling `runApp` with the full directory path name will run the app.
- See a simple Shiny app with the Beatles names and a more complicated Shiny app with the New York City restaurant data.

Customization of ggplot2 Graphics

- In ggplot2, a *theme* is a comprehensive customization of the appearance of a plot.
- A theme contains over 100 attributes that define the appearance of axis labels, titles, background colors, grid lines, etc.
- The default theme in ggplot2 is `theme_grey`.
- We can pull out and examine certain aspects of a theme using `pluck`.
- There are *many* different built-in themes you can use in ggplot2, including `theme_bw`, `theme_minimal`, `theme_classic`, etc.
- See examples of changing the appearance of the Beatles names plot by using other themes.

Creating Your Own ggplot2 Theme

- With the `theme` function, you can create your own theme.
- You COULD write your own theme from scratch, specifying ALL the attributes
- But it's much easier to modify an existing theme using the `%+replace%` operator.
- The textbook example creates a new theme called `theme_mdsr`.
- The choices of colors in R can be found by typing: `colors()` or there are handy online cheat sheets showing all the colors in R and their names.
- There are *many* different built-in themes you can use in `ggplot2`, including `theme_bw`, `theme_minimal`, `theme_classic`, etc.
- The `ggthemes` packages has lots of specialized themes.
- See example of the Beatles names plot shown using `theme_mdsr`.

Figure 14.19

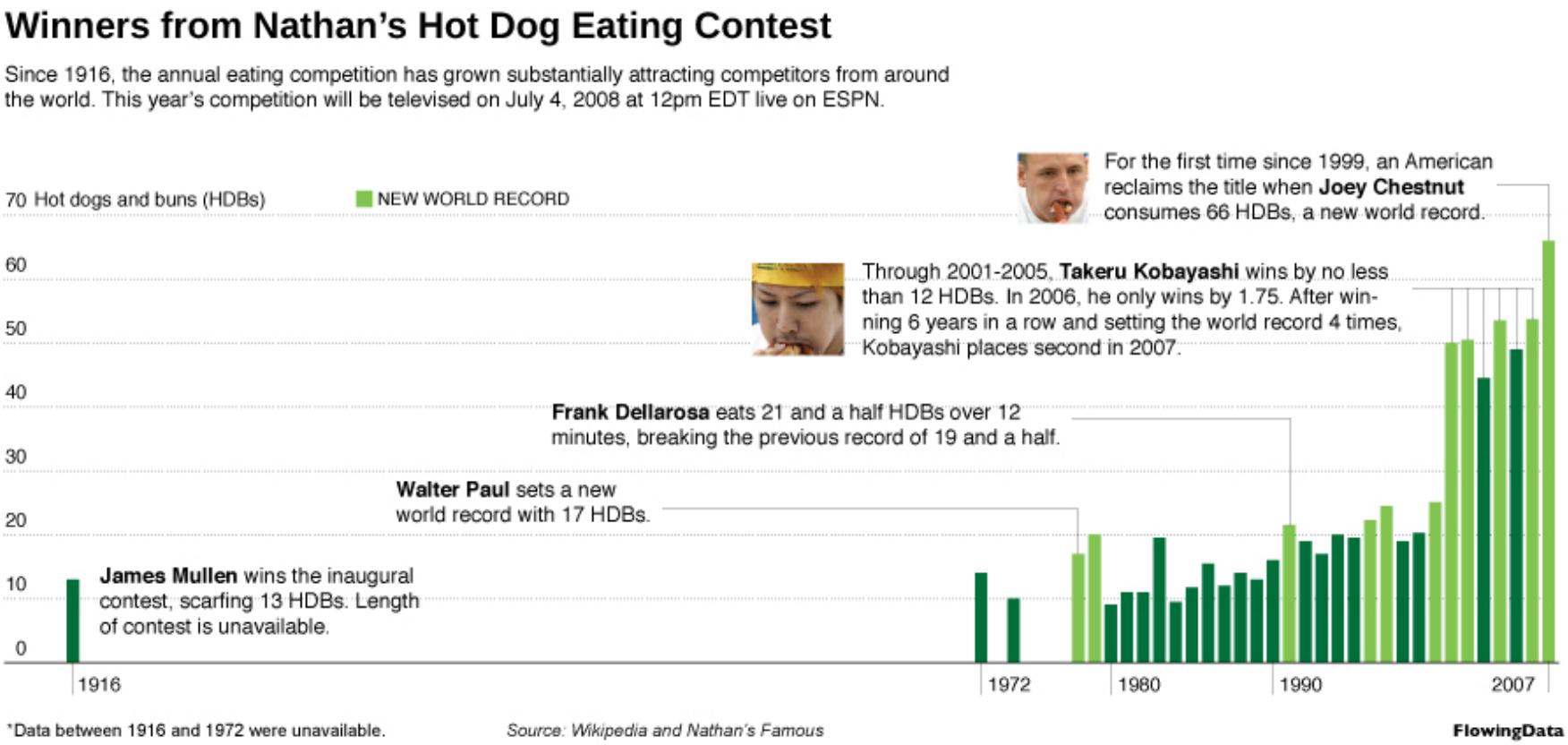

Figure: Nathan Yao's *New York Times* Hot Dog Eating graphic: Figure 14.19 from MDSR textbook

Extended Example: Hot Dog Eating Graph

- Figure 14.19 shows a very complicated graph with lots of text, color, lines, etc. created in *Adobe Illustrator*.
- In Section 14.6, an extended example shows how this graphic can be (almost) replicated in R using `ggplot2` and `tidyverse` tools.

A Quick Note about Fonts

- Internally, R only has a small number of fonts.
- These depend on the system R is running under, but three fonts (`sans`, `serif`, and `mono`, which are basically equivalent to Helvetica, Times, and Courier) are always available in R.
- The `extrafont` package allows access to all the fonts on your computer, not just the ones that R knows about.
- See Section 14.7 for a link to learn more about the `extrafont` package.

A Quick Note about Math Expressions

- If you need to use mathematical notation and expressions in your plot titles, axis labels, or text within plots, the `expression` function will allow you to type fairly complicated math expressions, Greek letters, etc.
- You can also mix plain text with mathematical expressions using the `paste` function.
- See the brief R examples on the course webpage.