**Preview Questions**

1. How do you create pictures of data that you can learn from?
2. Why should we work in plain text using RMarkdown?
3. Why should we use RStudio when working in R?
4. What are the basics of using RStudio?
5. What are the important things to know about R?
6. How do you get data into R?
7. How do you make a plot?
8. What is ggplot?
9. How does ggplot work?
10. What is tidy data?
11. What do mappings do?
12. Why do we build plots layer by layer?
13. What is the difference between mapping aesthetics and setting aesthetics?
14. How do you map aesthetics with geom?
15. How do you save your work?

**Summary Notes**

Chapter 2 – Get Started

* We use R and ggplot to create pictures of data.
* Use a text editor to take notes and write code in plain text.
* The data, code, and text are what’s “real”, not the formatted output.
* Finished output can be reproduced from code, data, and written text.
* Tweak and adjust the output programmatically through code.
* Three basic activities:
  + Write code
  + Review output
  + Take notes
* RMarkdown allows us to keep code and comments together in the same file.
* Finished output is created by knitting the file.
* R itself has virtually no user interface and does everything through a command line (i.e., console).
* You can also write your code in a text file as an R script (.r file) and send it to R.
* RStudio is an integrated development environment (IDE) that makes working with R much easier.
* Steps:
  + Create a project, File > New Project…
  + Create a new file, File > New File > RMarkdown or File > New File > R Notebook
  + Save the new file in the project folder immediately.
* You can use an R script instead of RMarkdown.
* An R script just contains R commands only and have the .r or .R file extension.
* To put notes in an R script, start the notes line with the hash character (i.e., #)
* In R, everything is an object.
  + Some objects are built into R, other objects are added via libraries, and still other objects are created by the user.
  + Create objects by assigning them to names using the assignment operator (<-).
  + {alt} + {-} is shortcut for the assignment operator.
* In R, every object has a name and you refer to objects by their name.
* In R, you perform actions using functions.
  + Functions are special kinds of objects that perform actions and produce output based on the input it receives; they have the form <Function Name>().
  + c() concatenates items into a vector composed of a series of comma-separated elements.
  + mean() calculates the arithmetic mean for a vector of numbers.
  + summary() produces a little table with named elements.
* The syntax for providing arguments to a function is <argument> = <value>.
* Libraries are families of useful functions.
* ggplot is a library of functions for plotting data.
* class() tells what an object is.
* Every object has a class.
* Certain actions may change an objects class.
* In a data frame the columns can be of different classes.
* str() shows you what’s inside an object.
* Be careful when using str() because some objects may cause it to output a very large amount of information to the console.
* Minimize errors by doing the following:
  + Make sure parentheses are closed.
  + Make sure expressions are complete.
  + Make sure the + character is at the end of the line, not the beginning.
* read\_csv() is used to read in comma separated data; the function is in the readr library which is a component of the tidyverse.
* haven library has functions to import various Stata, SAS, and SPSS formats directly.
* read\_csv() is the newer, more opinionated version of read.csv () function that is still widely used.
* read\_csv() does not classify variables as factors unless told to do so.

Chapter 3 – Make a plot

* Aesthetic mappings are logical connections between your data and plot elements.
* Geom is the overall type of plot.
* With long format table every variable is a column.
* With wide format tables, some variables are spread out across multiple columns.
* A tidy table does not present data in its most compact form.
* Tidy data is much more straightforward for specifying mappings.
* aes() specifies mappings in ggplot.
* Basic process:
  + Obtain tidy data.
  + Tell ggplot() what the data is.
  + Tell ggplot() what relationships we want to see.
  + Tell ggplot() how we want to see the relationships.
  + Layer on geoms as needed.
  + Use additional functions to adjust scales, labels, ticks marks, titles.
* The process of adding layers to a plot is additive.
* In the absence of other information, geoms look for instructions in the ggplot() function or the object created by it.
* Giving scale\_ functions a labels argument reformats the text print underneath the tick marks on the axes.
* A variable should have as many observations as there are rows in the data.
* color aesthetic affects the appearance of lines and points.
* fill aesthetic affects the appearance of filled areas of bars, polygons, and standard error ribbons.
* Aesthetics can be mapped specifically by geom.
* Set the default size of plots in .Rmd document by setting an option if the first code chunk.
  + knitr::opts\_chunks$set(fig.width=<#>, fig.height=<#>)
* To set size of specific code chunk plot without changing overall defaults, follow the code chunk label with a comma and provide the options.
  + ```{r example, fig.width=<#>, fig.height=<#>}
* Do NOT include apostrophes, backticks, spaces, slashes, and quotes in file names because they will make the code choke in the future.
* Use here() function to tell R what folder to save a plot.
* Vector formats (e.g., PDF, SVG) store as a set of instructions about lines, shapes, colors, and their relationships and enable resizing without distortion.
* Raster formats store images as a grid of pixels of a pre-defined size with information about the location, color, brightness, etc. of each pixel.
* Raster is more efficient for storage but they cannot be easily resized.
* Save work in several different formats

**Preview Questions**

1. How do you map U.S. state-level data?
2. What is an ur-chorepleth?
3. What are statebins?
4. How do you use small-multiple maps?
5. How do you determine if your data is really spatial?

**Summary Notes**

Chapter 7 – Draw Maps

* Choropleth maps show geographical regions colored according to some variable.
* Choropleth maps can sometimes be misleading or systematically misrepresent what you want to highlight because the data is not purely spatial.
* You don’t have to represent spatial data in a spatial manner.
* It’s important to know your data and variables well enough to check that you have merged them properly when doing joins and binds.
* Having geographical areas that only partially represent the variable we are mapping is a common choropleth map problem.
* Most choropleth maps of the U.S. show population density more than anything else for whatever variable.
  + They may present a geographical distribution to insinuate an explanation.
* Ur-choropleths don’t present variables as explanations of anything in isolation.
* Normalization by population has its limits.
  + When an event of interest is not very common the denominator starts to be expressed more than the measure.
  + There may be reporting constraints to protect privacy.
* Small differences in reporting combined with miscoding will produce spatially misleading results.
* Statebins are an alternative to state-level choropleths.
* Small multiple maps show changes over time for the same area of study.
* Differences in the geographical size of states makes spotting changes using small multiple maps more difficult.
* A lot of county, state, and national data is not properly spatial in the sense that the data is really about individuals or some other unit of interest rather than the geographical distribution of those units.
* drop\_na(<data>, <variable>) deletes rows that have observations missing on the specified variable.
* Don’t mistake the unit of observation for the variable of real interest.

**Preview Questions**

1. How do you use color to your advantage?
2. How do you layer color and text together?
3. How do you change the appearance of plots?

**Summary Notes**

Chapter 8 – Refine your plots

* The default settings in ggplot are good enough when doing exploratory data analysis.
* Refining plot means:
  + Perfecting the look.
  + Properly format for a specific use, such as submission to a journal.
  + Add detail not included in the default settings.
  + Completely change the look of the plot.
* Choose a color palette based on its ability to express the data.
  + Distinct color scheme for categorical data.
  + Graded color scheme for ordinal data.
* Use color to highlight some aspect of the data.
* Use theme\_set() to change to overall look of a plot all at once.
* The theme() function enables fine-grained control over the appearance of the plot.
* Using themes enables you to focus on the data itself rather than design elements.
* The problem with using two y-axes is that it makes it very easy to misrepresent the degree of association between the variables.
  + You can manipulate the scaling of the axes relative to one another.
* A lot of time series analysis is about making the serial nature of the data go away.
* The perceptual qualities of pie charts make comparison of categories difficult.
* Tracking ordered categories in a pie chart is difficult.
* Pie charts are good at emphasizing share or percentage of a total amount but there is a perceptual price to pay.