File Management and Mapping Spatial Data

File Management

When we work with real-life data, it's important to know how to both read data from our computer and write data from R Studio as a file. File management is an essential part of any workflow, especially as the scope and scale of analyses increases. Sometimes each file can be quite large – often several dozen or hundred megabytes or more – so it's important to analyze data in a modular fashion (our computers have limited memory, disk space, processing power, and speed, after all). Fortunately, many tools exists within R and R Studio to make these tasks easier. In this R Module, you'll learn some skills and R Packages that are frequently used to import and export data from your R environment.

Let's go ahead and initialize a new R Project (`File > New Project...`) and call it *R Module 3*.

Importing and Exporting Data

Working in future R Modules, classes, and projects, you'll need to be able to load, save, and share data. Fortunately, there are tools to load this data for almost any data type, from .CSVs to Shapefiles, raster images, and many more. We'll go over a couple of the most useful tools and packages used in data management to work with files in R.

In this lab, you'll use a folder called `data` , which contains a file called `mtcars.csv` . When working within R, it's important to know how to access files and folders. When you create a project, the folder you set it up within is designated as your **working directory**, which is a way of easily accessing files for your project without writing the file's full path. For example, instead of accessing the file with `C:/Users/brownhr/Documents/R Module 3/data/example.file` , if you're in the working directory "R Module 3" (which can be checked with `getwd()`), you can simply type `data/example.file` .

N.B. within R, it is very important to distinguish between `/` and `\` ; `\` is called an "escape character" and is used in more advanced text entry such as Regex. If you want to enter a file path, use either `C:/Example/File/path` or `C:\\Example\\File\\path` ; adding two backslashes is necessary in this case.

`readr`

The `readr` package provides some excellent functionality reading and writing data from and to files; while many of its functions exist in base R, `readr` is markedly faster, and provides more functionality for importing different types of data in a "tidy" manner ("tidy" is a programming concept in R that we'll get to later). For more information about `readr` , visit https://readr.tidyverse.org (https://readr.tidyverse.org)

`readr` comes with functions that allow for easy manipulation of primarily table-based data, such as `.csv` and `.tsv` files, allowing us to work with data that we make in Excel or Google Sheets. Especially when working with large files, or large numbers of files, `readr` makes importing and exporting data easy, and is much faster at reading and writing than the base `R` functions.

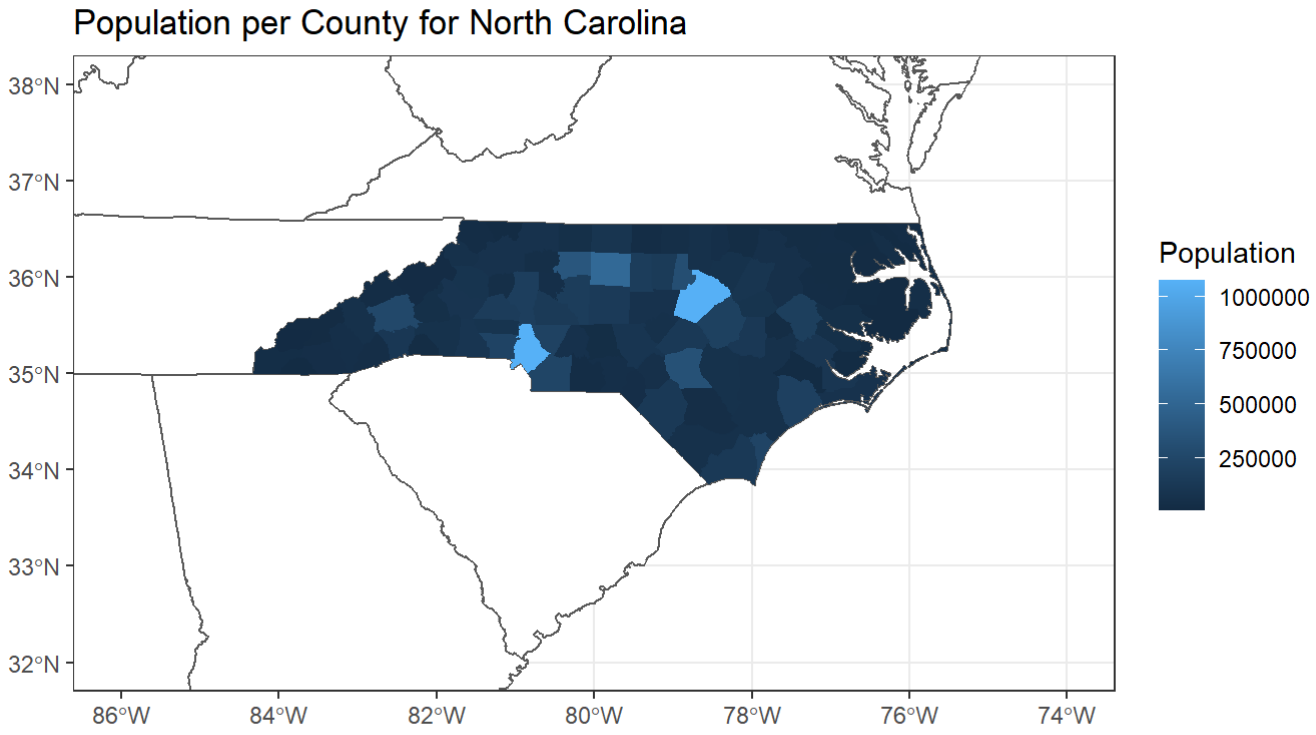
Mapping with `sf`

Simple Features for R, or `sf` , is an R package that allows for much easier geospatial analysis than older packages like `rgdal` . `sf` works with "simple features" – vector objects in GIS like points, lines, polygons, etc. – and stores their "geometry" – their list of coordinates – as a column in a data frame. This allows for geospatial data to be processed in exactly the same way as non-spatial data; you can run functions and analyses on the attribute table of a shapefile without needing to convert or export to any intermediate formats, for example.

`sf` can work with Shapefiles and other vector formats made by ArcGIS Pro, QGIS, etc., so if you have a shapefile, you can import it into R with `read_sf()` , perform your analysis, and export any sort of spatial result with `write_sf()` .

Even better, there's compatibility with `ggplot2` ! Suppose we have two Shapefiles that we wish to map: `NC_Counties` and `US_States` . Combining them is easy in `ggplot2` :

```r
# Load sf and the tidyverse, which contains packages like ggplot2 and dplyr

library(tidyverse)
library(sf)


# Read in the shapefiles with sf

US_States <- read_sf("data/US_States.shp")
NC_Counties <- read_sf("data/NC_Counties.shp")


# We can create a 'blank' ggplot object and define the data and mappings
# separately, which is useful if we have more than one input file or want to
# display more than one dataset.

ggplot() +
  geom_sf(data = US_States, fill = "white") +
  geom_sf(data = NC_Counties, aes(fill = Population), color = NA) +
  coord_sf(xlim = c(-86, -74), ylim = c(32, 38)) +
  theme_bw() +
  labs(title = "Population per County for North Carolina")
```



Looks good so far! You'll make a few maps of your own later in the Module, so feel free to read more about how `geom_sf()` and the other `ggplot2` and `sf` functions work.

Read more about `sf` by going to https://r-spatial.github.io/sf/ (https://r-spatial.github.io/sf/)

Create a new R Project, "R Module 3", to include your code, scripts, etc. Answer the following questions, and include them in your final document:

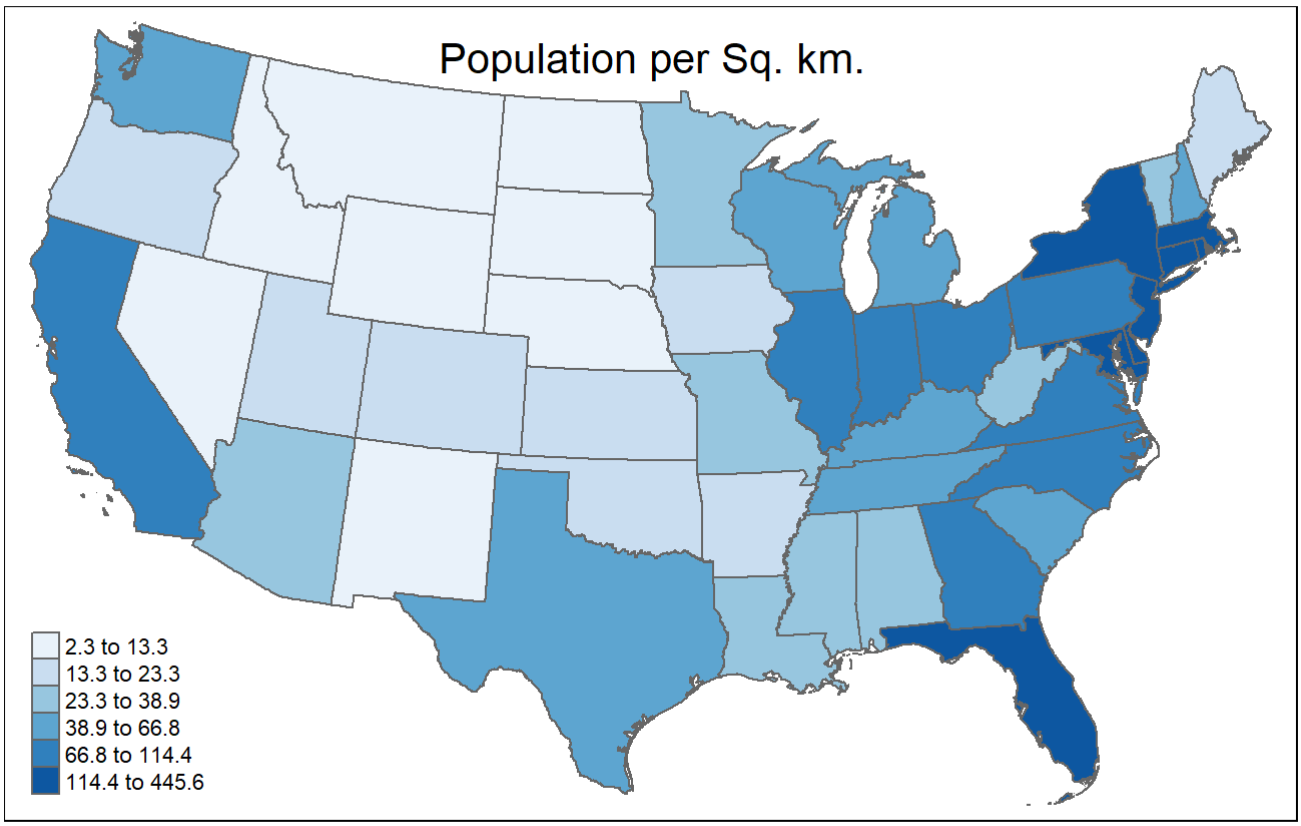
1. Within your `data/` folder, open the file `NC_Counties.prj` with a program like Notepad or Notepad++.
    - What kind of information does this file contain?
    - Why is it important to include these "auxiliary" files?
    - What would happen if you forgot to include the `.prj` file?
2. For the `US_States` layer, the `fill` argument stands on its own, while for `NC_Counties`, it's inside the `aes()` function. Why is this the case – what's the difference between these two layers?
3. What's the purpose of the `coord_sf()` function? Use `?coord_sf()` to view documentation and usage, and describe its arguments.

Choropleth Mapping

Choropleth maps are a type of thematic map that is used to display distinct geographic regions, such as states, counties, or countries. The colors used in choropleth mapping are **classified**; rather than a continuous range, the values are grouped into different classes, so that similar values share a color.

The `tmap` package is very useful for choropleth mapping, and offers a lot of functionality for customizing maps.

```r
library(tidyverse)
library(sf)
library(tmap)

# This function reprojects our shapefile into a different coordinate referenc
e
# system (CRS), and is useful when performing GIS-related tasks. In this cas
e,
# it's just to make the map look a little nicer.
data <- US_States %>%
  st_transform(crs = st_crs("ESRI:102004"))

tm_shape(data) +
  tm_polygons(
    # We're setting the color (col) mapping to the column "Population". Notic
e
    # that you have to quote the variable name here, which is different than
 in
    # ggplot2.
    col = "Population",

    # You can use RColorBrewer::brewer.pal.info to view many different kinds
 of
    # color palettes available in R.
    palette = "Blues",

    # This argument is used to normalize; i.e. convert from population to
    # population density. It's better to do this as a separate step in your
    # data, but can be done here for quick mapping purposes.
    convert2density = TRUE,

    # This sets the number of classes in our data.
    n = 6,

    # What method to use to create the classes. Note that some options might
    # overwrite your choice of n, so pay attention to how many classes your m
ap
    # actually has!
    style = "quantile",
  ) +
  tm_layout(
    title = "Population per Sq. km.", title.position = c("center", "top"),
    legend.title.color = NA
  )
```

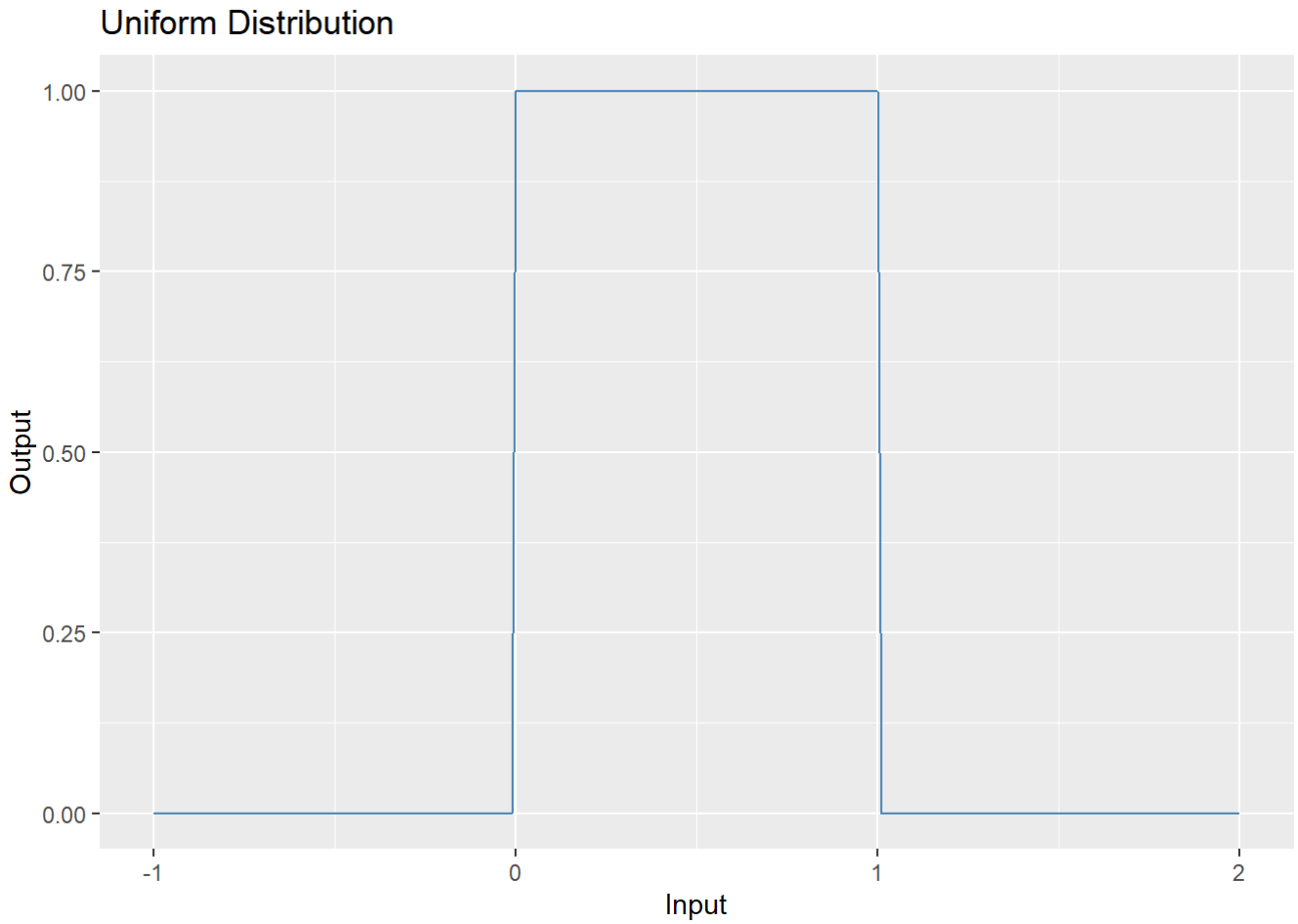


Probability and Distributions

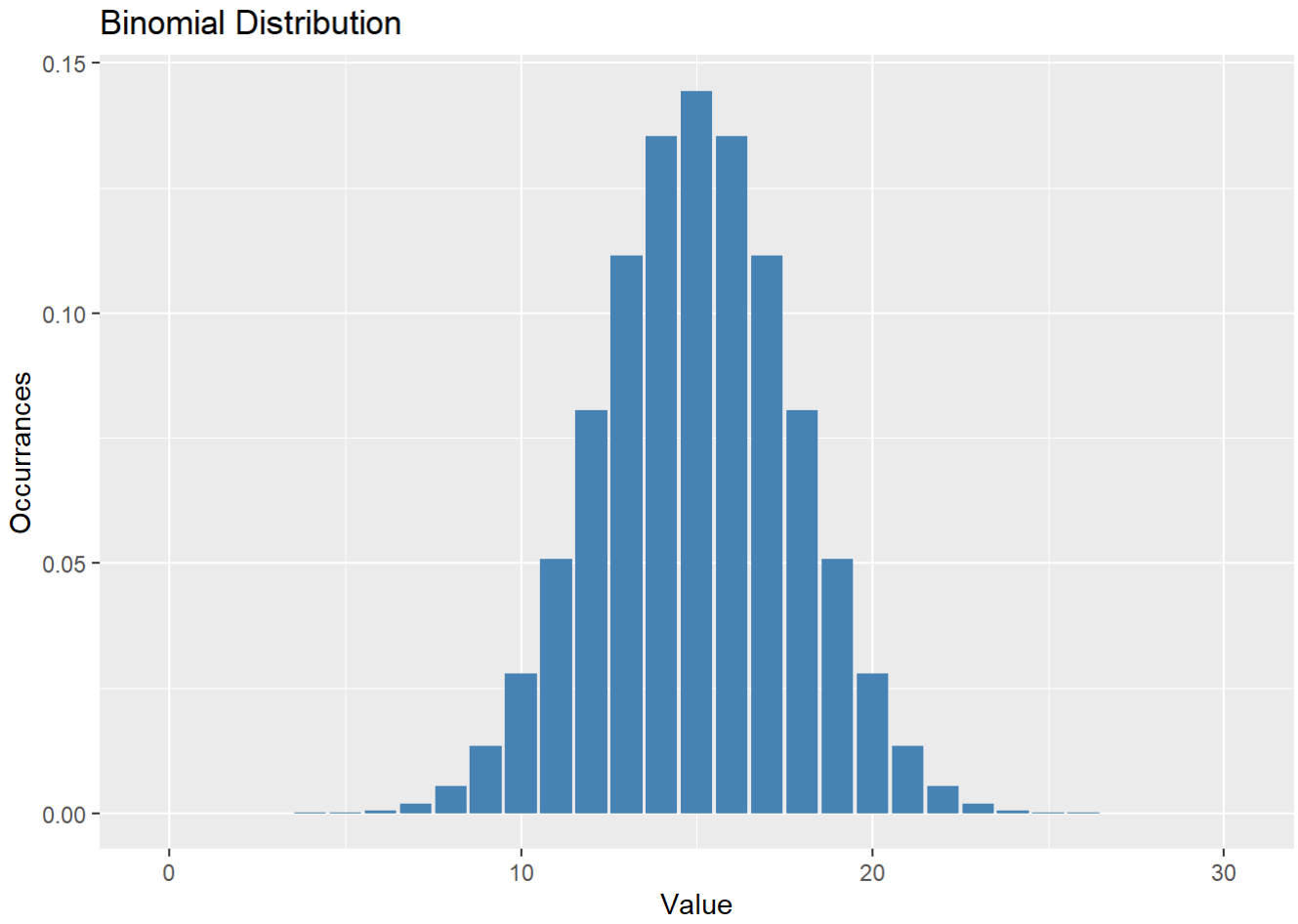
Discrete Distributions

Discrete probability distributions count occurrences that have countable or finite outcomes, such as the population of states within the U.S., days of rainfall, or average number of pets in households.

Uniform



Binomial



Poisson Distribution

A Poisson distribution refers to a kind of discrete distribution that shows the probability of a given number of events k occurring, given the parameter lambda, λ. λ can be thought of as the "average number of events"; for example, if the average number of goals in a World Cup match is 2.5, then $\lambda = 2.5$. Mathematically, the Poisson distribution is modeled by:

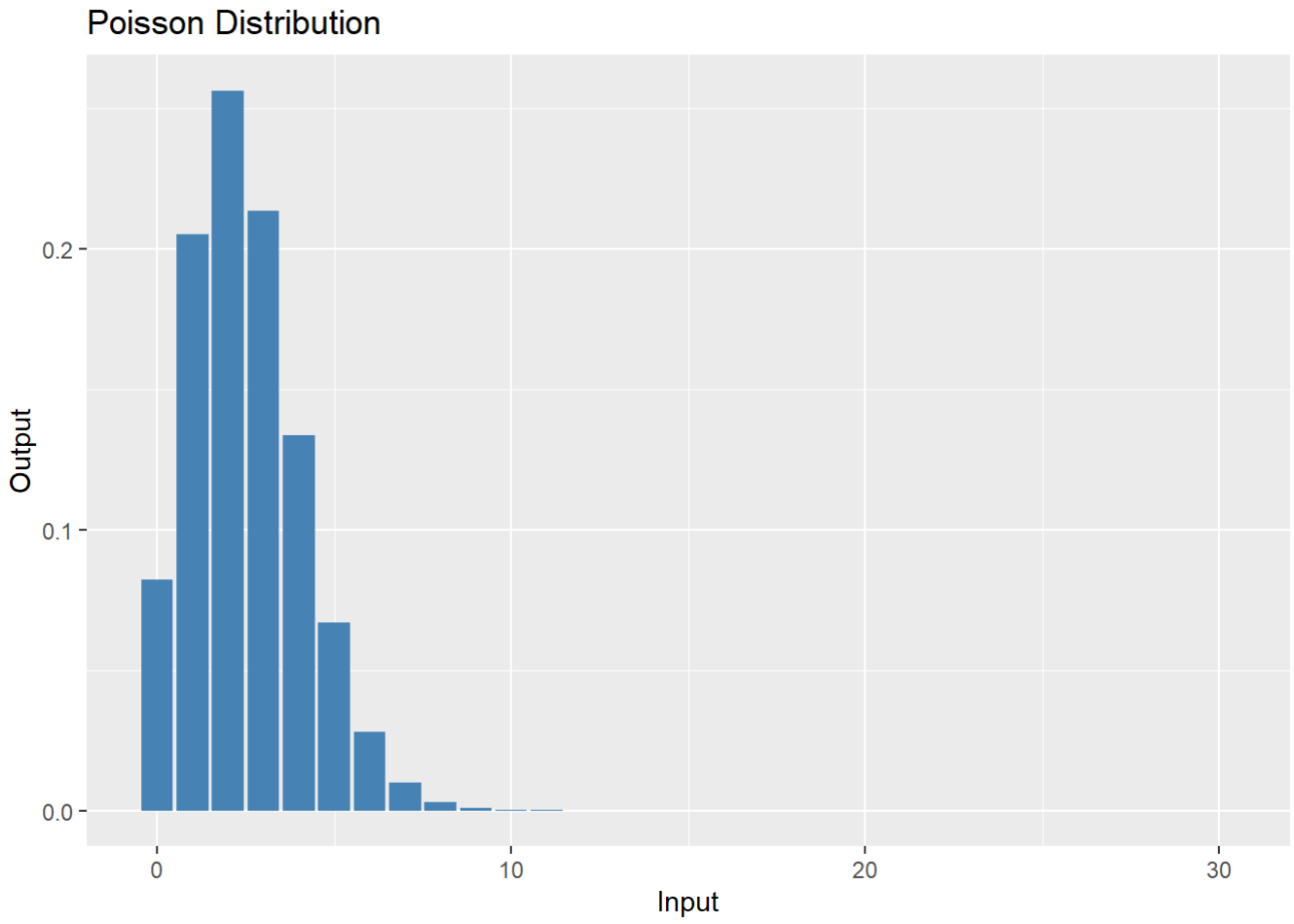
$$p(k;\lambda) = \frac{e^{-\lambda}\lambda^k}{k!},$$

where k is the given number of events that we want to calculate the probability for. For example, if we want to find the probability of 5 goals being scored in a World Cup game, we can calculate the following in R:

```
p <- dpois(x = 5, lambda = 2.5)

print(p)
```

```
## [1] 0.06680094
```

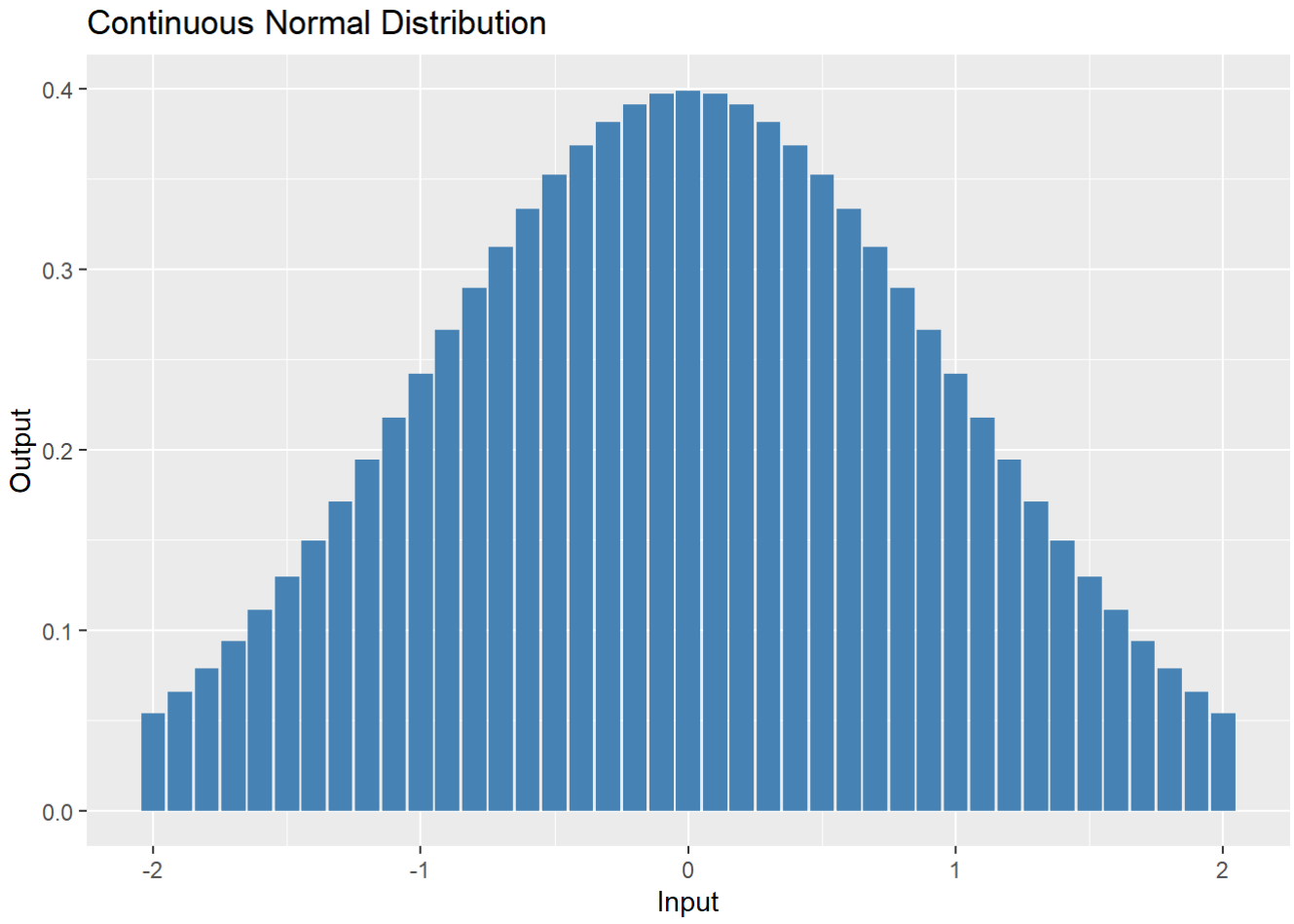
So, $\approx 6.68\%$ of the time, 5 goals will be scored. The functions `dpois()`, `ppois()`, `qpois()`, and `rpois()` are used to calculate the Poisson distribution.



Poisson Distribution with $\lambda = 2.5$

Continuous Distributions

Continuous distributions differ from discrete distributions in that outcomes can lie on a continuous range of values, for example, average daily temperature, atmospheric pressure, or height of buildings.



Assignment

In addition to questions 1-3 above, answer the following. Write up your responses, and submit your document to Google Classroom:

4. Using `ggplot2`, generate a histogram of the distribution of county population. Include axis labels, a title, and your name.
5. What type (continuous or discrete) does this distribution follow? Is the data normally distributed? If not, what kind of transformation can help "normalize" our data?
6. With a shapefile of your own (using ArcGIS, the `tigris` or `tidycensus` packages, or other), create a choropleth map using 5 classes, including a title and a legend. Create a map for both *quantile* and *equal-interval* classification, and briefly describe the difference in distribution between your two maps.
7. Repeat Question 6, using a different choice of classes.