Intro to Visualization with R - Review

* 1) Load libraries (ex. ggplot2)
  + >> **library**(library\_name)
* 2) Import csv
  + >> df = **read\_csv**(“name.csv”) #only if have library(readr) imported
  + >> df = **read.csv**(“name.csv”)
* 3) Inspect the first 6 rows of the dataset
  + >> **head**(df)
* 4) Visualize the data
  + *Example* – plotting data with the x-axis in descending order, adding a horizontal line & legend
  + p = ggplot(data = df, aes(x = x\_axis, y = y\_axis)) +  
     geom\_line() +   
     labs(title = “TITLE”, subtitle = “subtitle”, x = “x-axis label”, y = “y-axis label”,   
     fill = “legend title) +   
     scale\_x\_reverse(lim = c(x\_max, x\_min)) +  
     geom\_hline(aes(yintercept = hvalue, linetype = “name of line in legend”))  
    p
* Common graphs:
  + Scatterplot = geom\_point() layer in ggplot2
    - Visualize the relationship between 2 variables
  + Bar plot = geom\_bar() layer in ggplot2
    - Visualize values for variables that can be counted (i.e., discrete variables or categorical variables)

Histograms

**Histograms**

* Visualize the distribution of a **continuous** variable
  + Divide values of a variable into bins
    - Bin = Range of values that get counted together
      * *Ex*. Variable with values 1-100;   
         Want 5 bins 🡪 Each bin has range of 100/5 = 20;  
         1st bin = count frequency of values 1-20;   
         2nd bind = count frequency of values 21-40…
    - Default = Automatically calculate 30 equally sized bins
* >> **geom\_histogram()**
  + Arguments:
    - binwidth = The value range of each bin (*ex.* 10)
* **All together**:
  + >> ggplot(data, aes(x = x\_axis)) +   
     **geom\_hisogram**(binwidth = width\_of\_bins)

A graph of a graph showing the amount of oxygen

Description automatically generated

Heatmaps

**Heat Maps**

* Visualize frequencies along 2 variables
  + Looks like a scatterplot, but uses color-coded squares (rather than individual points) to indicate how many cases occurred at the intersection of x- and y- value ranges
  + Specify **bin widths** to control which ranges of values get counted together
    - Default = Automatically calculate 30 equally sized bins
* >> **geom\_bin2d()**
  + Arguments:
    - bindwidth = c(x\_bin\_width, y\_bin\_width)
      * Specify bin widths for each variable by passing a vector of widths
* **All together**:
  + >> heatmap\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis)) +   
     **geom\_bin2d**(binwidth = c(x\_bin\_width, y\_bin\_width))

A graph of blue squares

Description automatically generated

Box Plots

**Box Plots**

* Visualize the distribution of data by **quartiles**
  + A.k.a. “box-and-whisker plots”
  + Shows how much a variable varies across values of another variable
    - “Center” line = Median
    - Upper bound of box = 75th percentile
    - Lower bound of box = 25th percentile
    - Upper & Lower bounds of whisker = Extend up to 1.5x the distance between the 75th & 25th percentiles
      * Outliers shown as points beyond the whiskers
* >> **geom\_boxplot()**
* **All together**:
  + >> boxplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis)) +   
     **geom\_boxplot**()

A diagram of a temperature chart

Description automatically generated

Bar Plots – Stacked or Clustered

**Stacked Bar Plots**

* Visualize percentages within our data, or how different values add up, stacked on top of each other
* >> **geom\_bar() or geom\_col()**
  + Specifications: (for **geom\_bar()**)
    - fill = variable to be depicted as color-coded segments within our stacked bars
      * add to aes() mapping to ggplot
    - position = “stack”
      * stack different values of the **fill** variable on top of each other
      * \**Note* – assumed by default if we don’t specify any positioning
      * add to geom\_bar()
    - stat = “identity”
      * displays the values in our df as is (rather than displaying counts)
      * add to geom\_bar()
  + Specifications: (for **geom\_col()**)
    - stat = “identity” – assumed by default (don’t need to type out)
    - position = “fill”
      * Tells ggplot2 to represent each bar’s total as percentages out of 1 (rather than absolute counts)
      * Able to compare ratios between multiple variables
      * add to geom\_col()
* **All together**:
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis, fill = color\_variable)) +   
     **geom\_bar**(position = “stack”, stat = “identity”)
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis, fill = color\_variable)) +   
     **geom\_col**(position = “fill”)

A graph of a number of people

Description automatically generated with medium confidence

**Clustered Bar Plots**

* Visualize values of the variable side by side
* >> **geom\_bar() or geom\_col()**
  + Specifications: (for **geom\_bar()**)
    - fill = variable to be depicted as color-coded segments within our stacked bars
      * add to aes() mapping to ggplot
    - position = “dodge”
      * display each value of the **fill** variable next to each other
      * add to geom\_bar()
    - stat = “identity”
      * displays the values in our df as is (rather than displaying counts)
      * add to geom\_bar()
  + Specifications: (for **geom\_col()**)
    - stat = “identity” – assumed by default (don’t need to type out)
    - position = “dodge”
      * display each value of the **fill** variable next to each other
      * add to geom\_col()
* **All together**:
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis, fill = color\_variable)) +   
     **geom\_bar**(position = “dodge”, stat = “identity”)
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis, fill = color\_variable)) +   
     **geom\_col**(position = “dodge”)

A graph of a number of people

Description automatically generated

Statistical Summaries

**Statistical Summaries**

* \*Note – by default, **bar plots** (geom\_bar()) show the count of observations for each value
  + To show the mean instead:
    - stat = “summary”
      * tells ggplot2 to summarize values according to a provided function (calculate the mean values on the y--axis)
      * add to geom\_bar()
    - fun = “mean”
      * to summarize our y-axis variable by calculating mean values for each value in our x-axis variable (display the mean values on the y-axis)
      * add to geom\_bar()
* **All together**:
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis)) +   
     **geom\_bar**(stat = "summary", fun = "mean")

A graph of a number of hours asleep

Description automatically generated

Error Bars

**Error Bars**

* Show the standard error of a value
  + Tells us how much variation there is around the mean
* Steps: *Example* – Sleep data by diet
  + >> new\_df 🡨 df %>%  
     subset(status = “asleep”) %>%  
     na.omit() %>%  
     group\_by(diet) %>%  
     summarize(mean.hours = mean(hours), mean.se = std.error(hours)) %>%  
     mutate(se.min = mean.hours – mean.se, se.max = mean.hours + mean.se)
  + **1)** Calculate standard errors
    - Compute new variables in **summarize()**
      * **mean.hours** = mean(hours) *#mean of variable hours*
      * **mean.se** = std.error(hours) *#standard error of our means*
    - Add new variables to df using **mutate()**
      * **se.min** and **se.max** *#lower & upper bounds of error range*
  + **2)** Create bar plot showing means & standard errors
    - **geom\_errorbar()**
      * Specifications:
        + ymin and ymax variables of our error bar

inside aes()

* + - * + width = X of the bar’s width (ex. 0.2)
    - **geom\_bar()**
      * Specifications:
        + Add stat = “identity” to display **mean.hours** value as is
* **All together**:
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = mean.yvar)) +   
     geom\_bar(stat = “identity”) +   
     **geom\_errorbar**(aes(ymin = se.min, ymax = se.max), width = 0.2) +  
     labs(title = “title”)  
     A graph of a number of hours asleep

    Description automatically generated

Customizing Discrete Axes

**Discrete Axes**

* Discrete Variables (*Ex*. Categories on the x-axis of a bar plot)
* Customization:
  + Specify a particular order the values appear in
  + Rename the axes labels so they better describe each value
* **scale\_x\_discrete()** = Layer to customize discrete variables on the **x-axes**
* **scale\_y\_discrete()** = Layer to customize discrete variables on the **y-axes**
  + Arguments:
    - limits = c(“value1”, “value2”, “value3”)
      * pass a vector argument to specify that we only want to show bars for the values in the vector argument, in that order (rather than the default alphabetical ordering)
      * add to one of the scale\_?\_discrete() layers
    - labels = c(“value1” = “label1”, “value2” = “label2”, “value3” = “label3”)
      * pass a vector of value-to-label mappings to change the labels of the bars
      * add to one of the scale\_?\_discrete() layers
* *Example* – hours of sleep by diet bar plot
  + Pass c("omni", "carni", "herbi") vector to the **limits** argument in scale\_x\_discrete()
    - \**Note* – “insect” is omitted on the x-axis; “onmi” appears on the left (rather than on the right - default alphabetical ordering)
  + Pass c(“carni” = “Carnivore”, “herbi” = “Herbivore”, “omni” = “Omnivore”) to the **labels** argument in scale\_x\_discrete()
* **All together**:
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis)) +   
     geom\_bar(stat = “identity”) +   
     geom\_errorbar(aes(ymin = se.min, ymax = se.max), width = 0.2) +  
     labs(title = “title”) +   
     scale\_x\_discrete(limits = c(“value1”, “value2”, “value3”), labels = c(“value1”   
     = “label1”, “value2” = “label2”, “value3” = “label3”))

A graph of a number of hours asleep

Description automatically generated

Customizing Continuous Axes

**Continuous Axes**

* Continuous Variables
* Customization:
  + Specify a particular order the values appear in
  + Rename the axes labels so they better describe each value
* **scale\_x\_continuous()** = Layer to customize continuous variables on the **x-axes**
* **scale\_y\_continuous()** = Layer to customize continuous variables on the **y-axes**
  + Arguments:
    - breaks = c(n1, n2, n3)
      * pass a vector argument to specify which values we want to appear as tick marks on the axis
      * add to the scale\_?\_continuous() layer
    - labels = function
      * pass a custom function to take the automatic labels and apply the function to them
        + *Ex*. Add unit of measurement “hrs” to each # on the y-axis:  
           >> show\_as\_hours 🡨 function(x) {  
           output 🡨 past0(x, “ hrs”)  
           return(output)  
           }  
           >> scale\_y\_continuous(labels = show\_as\_hours)
        + *Ex*. Display y-asix as percentages % rather than ratios  
           >> scale\_y\_continuous(labels =   
           **scales::label\_percent()**)

**scales** package includes a variety of functions for transforming labels

* + - * add to the scale\_?\_continuous() layer
* **coord\_cartesian()** = Layer to specify the range of values shown on a given axis
  + Allows us to zoom in and zoom out of the plot region
  + Arguments:
    - ylim = c(ymin, ymax)
      * pass a vector argument to specify the min and max values of the y-axis
* **All together**:
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis)) +   
     geom\_bar(stat = “identity”) +   
     geom\_errorbar(aes(ymin = se.min, ymax = se.max), width = 0.2) +  
     labs(title = “title”) +   
     scale\_x\_discrete(limits = c(“value1”, “value2”, “value3”), labels = c(“value1”   
     = “label1”, “value2” = “label2”, “value3” = “label3”)) +   
     **coord\_cartesian**(ylim = c(8,12)) +  
     **scale\_y\_continuous**(labels = function, breaks = c(8,10,12))   
    A graph of a graph showing the amount of hours asleep

    Description automatically generated

Facets

**Facets**

* Allow us to visualize **multiple discrete variables** in 1 plot
  + Show each value of the facet variable in a different section
* **facet\_grid()** = Layer
  + Specify the variables it maps to
    - facet\_grid(rows = vars(row.var))
      * To show values of a facet variable as rows
    - facet\_grid(columns = vars(col.var))
      * To show values of a facet variable as columns
    - facet\_grid(rows = vars(row.var), columns = vars(col.var))
      * To show 2 facet variables in a grid
* **All together**:
  + >> barplot\_name 🡨   
     ggplot(data, aes(x = x\_axis, y = y\_axis)) +   
     geom\_bar(stat = “identity”) +   
     geom\_errorbar(aes(ymin = se.min, ymax = se.max), width = 0.2) +  
     labs(title = “title”) +   
     scale\_x\_discrete(limits = c(“value1”, “value2”, “value3”), labels = c(“value1”   
     = “label1”, “value2” = “label2”, “value3” = “label3”)) +   
     coord\_cartesian(ylim = c(8,12)) +  
     scale\_y\_continuous(labels = function, breaks = c(8,10,12)) +  
     **facet\_grid**(cols = vars(variable\_to\_map\_split\_into\_columns)

SUMMARY

*Example* – animal sleep data x diet x taxonomic order

Steps:

* 1) Process our df to include only what we want & calculate means/standard errors  
   A computer screen shot of white text

  Description automatically generated
* 2) Create our sleep x diet plot – map the **order** variable to a facet\_grid() layer split into columns  
   A computer screen shot of a program

  Description automatically generated

A graph of a number of hours asleep by diet

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

PROJECT – MUSEUMS & NATURE CENTERS

* Set the maximum width per line to no more than 8 characters using the function wrap\_format() from the scales package included with ggplot2
  + >> + scale\_x\_discrete(labels = scales::wrap\_format(8))
  + To apply a function example\_function() to axis labels on the x axis, we can add a layer like **scale\_x\_discrete(labels = example\_function())**
* **Filter museums\_df** to include a few states you might be interested in, using the State..Administrative.Location. column; for example, we can choose IL, CA, and NY
  + >> museums\_states <- museums\_df %>% filter(`State..Administrative.Location.` %in% c("IL", "CA", "NY"))
* Recreate our bar plot showing the distribution of museums vs non-museums and use facet\_grid() to display each state’s distribution in a separate panel
  + >> museum\_facet = ggplot(museums\_states, aes(x=Is.Museum)) + geom\_bar() + scale\_x\_discrete(labels = c("TRUE" = "Museum", "FALSE" = "Non-Museum")) + facet\_grid(cols = vars(Is.Museum))
* Add a **scale\_fill\_discrete()** layer to relabel the “TRUE” and “FALSE” labels in our legend
  + >> + scale\_fill\_discrete(labels = c("TRUE" = "Museum", "FALSE" = "Non-Museum"))
* Transform the plot to show values out of 100%, and transform the y-axis labels into percentage values
  + >> + geom\_bar(position = "fill") + scale\_y\_continuous(labels = scales::percent\_format())
* Create a new df that retains only unique values of Legal.Name in museums\_df; Filter this df to include only entities with Annual.Revenue greater than 0
  + >> museums\_revenue\_df <- museums\_df %>% distinct(Legal.Name, .keep\_all=TRUE) %>% filter(Annual.Revenue >= 0)
  + We can retain only unique rows of a data frame using the dplyr function **distinct()**, specifying which variable(s) we want to retain distinct values of.
    - Make sure to include the argument **.keep\_all = TRUE** so that all columns are retained, rather than just the one column we want to check uniqueness for.
* Change the x-axis labels to monetary values
  + >> + scale\_x\_continuous(labels = scales::dollar\_format()
  + **dollar\_format()** is a function from the scales library included in ggplot2 that adds dollar signs and commas to monetary data
* Change the y-axis labels to billions – so $1,000,000,000 reads $1B
  + >> + scale\_y\_continuous(labels = function(x) paste0("$", x/1e9, "B"))
* To round all labels on a continuous y-axis:
  + >> scale\_y\_continuous(labels = function(x) round(x))
* Add error bars – first calculate standard errors, then add error bars to plot
  + >> museums\_error\_df <- museums\_revenue\_df %>%   
     group\_by(Region.Code..AAM.) %>%   
     summarize(Mean.Revenue = mean(Annual.Revenue), Mean.SE =   
     std.error(Annual.Revenue)) %>%   
     mutate(SE.Min = Mean.Revenue - Mean.SE, SE.Max - Mean.Revenue +   
     Mean.SE)
  + >> revenue\_errorbar = ggplot(**museums\_error\_df**, aes(x = factor(Region.Code..AAM.), **y = Mean.Revenue**)) + geom\_bar(**stat = "identity"**) + **geom\_errorbar**(aes(ymin = SE.Min, ymax = SE.Max), width = 0.2 ) + scale\_x\_discrete(labels = c("1" = "New England", "2" = "Mid-Atlantic", "3" = "Southeastern", "4" = "Midwest", "5" = "Mountain Plains", "6" = "Western")) + scale\_y\_continuous(labels = function(x) paste0("$", x/1e6, "M")) + labs(title = "Mean Annual Revenue by Region", x = "Region", y = "Mean Annual Revenue ($)")
    - ggplot() 🡪 Change the df to the new one you just made
    - ggplot() 🡪 Change y-axis to Mean.Revenue (instead of column name Annual.Revenue)
    - geom\_bar() 🡪 Change the stat being used to stat = “identity”