R Video Notes

Datasets used in this document are either preloaded in R, or are custom and posted at the top bullet point under each video. It is recommended to load/install tidyverse as pipes (%>%) are frequently used, and many packages covered in this document are part of tidyverse.

Table of Contents

Visualizations

Simple Tables

(*Package*: Base R, *Dataset*: iris)

- **A frequency table shows the count for *one* variable (3:13)**
  - R Code: table(iris$Species)
- **A proportion table shows the proportion for *one* variable (7:15)**
  - R Code: prop.table(table(iris$Species)
- **Table with percentages instead (5:45)**
  - R Code: prop.table(table(iris$Species))*100
- **A contingency table shows the count for *two or more* variables (8:32)**
  - R Code: xtabs(~Petal.Width +Species, iris)

Presentation Summary Tables

(*Package:* gtsummary, *Dataset*: CO2)

- **Install/load gtsummary (0:10)**
  - R Code: library(gtsummary)
- **Basic Summary Table (0:44)**
  - R Code: CO2 %>% select(!c(Plant,conc)) %>% tbl_summary()
    - Removing 'Plant' and 'conc' variables
- **Summary split by categorical variable (1:42)**
  - R Code: CO2 %>% select(!c(Plant,conc)) %>% tbl_summary(by = Type)
    - 'by=Type' organizes the data by a categorical variable
- **Summary split by categorical variable with p-values (2:13)**
  - R Code: CO2 %>% select(!c(Plant,conc)) %>% tbl_summary(by = Type) %>% add_p()
    - 'add_p()' adds p-values to the table
- **Summary including overall, extra heading, and other statistics (3:02)**
  - R Code: CO2 %>% select(!c(Plant,conc)) %>% tbl_summary(by = Type, statistic = list(all_continuous() ~ "{mean} ({sd})", all_categorical() ~ "{n} / {N} ({p}%)"), digits = all_continuous() ~ 2) %>% add_p() %>% add_overall() %>% modify_spanning_header(c("stat_1", "stat_2") ~ "**Location**")
    - Code adds mean and standard deviation
- **Create crosstab with p-values (4:27)**
  - R Code: CO2 %>% tbl_cross(row = Type, col = Treatment, percent = "cell") %>% add_p()
    - Crosstab shows the relationship between two categorical variables

Barplot (1)

(*Package*: Base R, *Dataset:* custom)

- **Custom data (0:10)**
  - R code: values <- c(.4, .75, 0.2, 0.6, 0.5)
- **Make a barplot (0:30)**
  - R Code: barplot (values)
- **Add color to barplot (1:23)**
  - Text Color R Code: barplot(values, col = "#1b98e0")
  - Name of Color R Code: barplot(values, col = "darkgreen")
  - Link for more names of colors
- **Change bar orientation to horizontal (2:00)**
  - R Code: barplot(values, horiz = TRUE)
- **Add labels to barplot (2:27)**
  - 1st Step: Create a vector that contains that labels for the barplot

        - R Code: group <- LETTERS[1:5]
            - Creates labels from 'A' to 'E' (first 5 letters of alphabet)
    - 2nd Step: Assign the new vector to 'names.arg'
        - R Code: barplot(values, names.arg = group)
- **Creating a stacked bar plot(3:33)**
    - 1st Step: Create matrix for the data that includes two different values for each column/label (since you will have two bars of differing heights for one column)
        - R Code: data <- as.matrix(data.frame(A = c(0.2, 0.4), B = c(0.3,0.1), C = c(0.7, 0.1), D = c(0.1, 0.2), E = c(0.3, 0.3)))
    - 2nd Step: Create row names for the matrix
        - R Code: rownames(data) <- c ("Group 1", "Group 2")
    - 3rd Step: Plot stacked bar plot with matrix data with 2 different colors (4:49)
        - R Code: barplot(data, col = c("#1b98e0", "#353436"))
- **Add a legend to the bar plot (5:04)**
    - R Code: legend("topright", legend = c("Group 1", "Group 2"), fill = c("#1b98e0", "#353436"))
    - Important: Be sure to select/highlight the bar plot *and* the legend together before running. Running the bar plot before the legend will result in an error.
- **Create grouped barchart (columns side-by-side) by adding 'beside = TRUE' (6:25)**
    - R Code: barplot(data, col = c ("#1b98e0", "#353436") , beside = TRUE)
        - The groups should be some type of categorical data
- **Manually grouped barchart: Color subgroups by mean then create grouped barplot(24:28)**
    - Step 1: Create object containing mean price by subgroups
        - R Code: diamonds_m_cl_co <- aggregate(diamonds, price ~ clarity + color, mean)
    - Step 2: Plot grouped barplot with object containing subgroups
        - R Code: ggplot(diamonds_m_cl_co, aes(x=clarity, y = price, fill = color)) + geom_bar(stat = "identity", position = "dodge")
            - 'Position = "dodge" makes the barplot grouped. Without it, plot becomes a stacked barplot

Barplot (2)

(*Package*: ggplot2, *Dataset:* custom)

- **Custom data (0:10)**
    - R code: values <- c(.4, .75, 0.2, 0.6, 0.5)
- **Install and loading ggplot (7:48)**
    - Install R Code: install.packages("ggplot2")
        - Only need to install a package *one* time in R
    - Load R Code: library("ggplot2")

        - Need to load the package in *every* Rscript or .Rmd
- **Ggplot2 only takes dataframes as input, not matrices like baseR (8:09)**
    - Make dataframe R Code: data_ggp <- data.frame(group, values)
        - Dataframe is a slightly different way to store data than a matrix.
            - If unsure if data type is a data frame, run 'typeof(name_of_data)'
- **Create a bar plot in ggplot2(8:47)**
    - R Code: ggplot(data_ggp, aes(x = group, y = values)) + geom_bar(stat = "identity")
        - 'geom_bar' specifies that we are creating a bar chart based on our data

Ordering Bars of a Barplot

(*Package*: ggplot2, *Dataset*: custom)

- **Custom data (0:15)**
    - R Code: data <- data.frame(x = c("A", "B", "C", "D", "E"), y = c(0.5, 2, 1.2, -0.2, 0.7))
- **Manually ordering bars (1:55)**
    - Step 1: Create duplicate of data
        - R Code: data1 <- data
    - Step 2: Change factor levels of data
        - R Code: data1$x <- factor(data1$x, levels = c("B", "D", "E", "C", "A"))
            - This code changes the factor ordering to 'B', 'D', 'E', 'C', 'A'
            - By default, the factor ordering is 'A', 'B', 'C', 'D', 'E'
- **Order bars in increasing order (3:30)**
    - Step 1: Replicate data
        - R Code: data2 <- data
    - Step 2: Change factor levels to increasing order
        - R Code: data2$x <- factor(data2$x, levels = data2$x[order(data2$y)])
            - Replace x and y with variable names used in your plot
    - Step 3: Create plot with new data
        - R Code: ggplot(data2, aes(x, y)) + geom_bar(stat = "identity")
- **Order bars in decreasing order (4:30)**
    - Step 1: Replicate data
        - R Code: data3 <- data
    - Step 2: Change factor levels to decreasing order
        - R Code: data3$x <- factor(data3$x, levels = data3$x[order(data3$y, decreasing = TRUE)])
            - Replace 'TRUE' with 'FALSE' to organize in increasing order
    - Step 3: Create plot with new data
        - R Code: ggplot(data3, aes(x, y)) + geom_bar(stat = "identity")

Scatter plots, Changing X-Axis Range, and Facet Layers

(*Package*: ggplot2, *Dataset*: custom)

- **Custom data (2:22)**
  - R Code: data <- data.frame(x = 1:9, y = c(3, 1, 4, 3, 5, 2, 1, 2, 3), group = rep(LETTERS[1:3], each = 3))
- **Create base layer for a variety of plots (3:22)**
  - R Code: ggplot(data, aes(x = x, y = y))
    - This will show x and y axes, and tick marks
- **Create scatterplot (3:55)**
  - R Code: ggplot(data, aes(x = x, y = y)) + geom_point()
    - 'geom_point' specifies a scatter plot
- **Scatter plot: Change point size (4:45)**
  - R Code: ggplot(data, aes(x = x, y = y)) + geom_point(size =3)
- **Scatter plot: Specify the colors by group (5:10)**
  - R Code: ggplot(data, aes(x = x, y = y, col = group)) + geom_point(size =3)
    - Automatically adds a legend for the color groupings
    - '+ theme (legend.position = "none")' removes the legend
- **To save a plot, assign an object to ggplot2 code (6:08)**
  - R Code: ggp_simple <- ggplot(data, aes(x = x, y = y, col = group)) + geom_point(size =3)
- **Change the x-axis range (7:01)**
  - R Code: ggp_simple + scale_x_continuous(limits = c(-3, 15))
    - c(-3,15) changes the numeric range of x-axis
    - You can also add '+ scale_x_continuous(limits = c(-3, 15))' to the rest of the plot
- **Manually change colors of points by groups (7:52)**
  - R Code: ggp_simple + scale_color_manual(breaks = c("A", "B", "C"), values = c("#1b98e0", "#353436", "#e32f08"))
- **Add multiple scale layers by adding a '+' between layers (8:58)**
  - R Code: ggp_simple + scale_x_continuous(limits = c(-3, 15)) + scale_color_manual(breaks = c("A", "B", "C"), values = c("#1b98e0", "#353436", "#e32f08"))
- **Facet layers: Creates side-by-side subplots of a variable or dataset(9:45)**
  - R Code: ggp_simple + scale_x_continuous(limits = c(-3, 15)) + scale_color_manual(breaks = c("A", "B", "C"), values = c("#1b98e0", "#353436", "#e32f08")) + facet_wrap(group ~ .)
    - Facet layers are helpful when your original plot has too much data
    - '+ theme_bw()' changes graph background from gray to white

Density Plots

(*Package*: ggplot2, *Dataset*: diamonds)

- **Add regression line to facet wrapped subplots (18:04)**
  - R Code: ggplot(diamonds, aes(x = price, y = carat)) +geom_point () + facet_wrap(clarity ~ .) + geom_smooth(method = "lm", formula = y ~ x)
    - 'geom_smooth(method = "lm", formula = y ~ x)' adds a regression line
- **Density plot: Helpful plot for showing distribution of a number (19:44)**
  - R Code: ggplot(diamonds, aes(x = depth)) + geom_density()
    - For density plots only input one column/variable since density will be graphed on the y-axis
- **Density plot: Draw density plots by group (20:36)**
  - R Code: ggplot(diamonds, aes(x = depth, fill =cut)) + geom_density()
    - 'fill = cut' adds multiple density graphs to the same plot
    - 'fill' colors the area between the density line and the x-axis, as opposed to coloring just the line
    - Make color more transparent: 'geom_density(alpha = .3)'. Can be set to any number lower than 1.

Pie Charts

(*Package*: ggplot2, *Dataset*: diamonds)

- **Step 1: Assign colors to object (0:33)**
  - R Code: colors <- c("#FFFFFF","#F5FCC2","#E0ED87","#CCDE57", "#B3C732","#94A813","#718200")
- **Step 2: Shape dataset to have 3 columns: categorical variable, count, and percentage (0:37)**
  - R Code: data <- diamonds %>% group_by(color) %>% summarize(counts = n(), percentage = n()/nrow(diamonds))
- **Step 3: Create Pie Chart (0:56)**
  - R Code: pie <- ggplot(data = data, aes(x="", y = percentage, fill = color)) + geom_col(color = "black") + coord_polar("y", start = 0) + geom_text(aes(label = paste0(round(percentage*100), "%")), position = position_stack(vjust = 0.5)) + theme(panel.background = element_blank(), axis.line = element_blank(), axis.text= element_blank(), axis.ticks = element_blank(), axis.title=element_blank(), plot.title = element_text(hjust = 0.5, size = 18)) + ggtitle("Pie chart of Diamond Color") + scale_fill_manual(values = colors)
    - 'coord_polar("y", start = 0) ' this creates the pie chart. The rest of the code is mostly formatting.

Bubble Plots

(*Package*: ggplot2, *Datasets:* mtcars)

- **A bubble plot is a scatter plot except it has a 3rd numeric variable mapped to a size aesthetic (0:05)**
- **Subset mtcars dataset (0:25)**
  - R Code: data <- mtcars %>% mutate(cyl = factor(cyl), Model = rownames(mtcars))
- **Create basic bubble plot (0:30)**
  - R Code: plot1 <- data %>% ggplot(aes(x = wt, y = mpg, size = hp)) + geom_point(alpha = 0.5)
    - 'size = hp' is the 3rd variable that makes this a bubble plot
    - 'alpha .5' makes bubbles more transparent
- **Add color and custom bubble size to bubble plot (1:09)**
  - R Code: plot2 <- data %>% ggplot(aes(x = wt, y = mpg, size = hp, color = cyl, label = Model)) + geom_point(alpha = 0.5) + scale_size(range = c(.1, 15))
    - 'color = cyl' adds color
- **Convert ggplot bubble plot into plotly plot (1:40)**
  - Plotly maps are interactive. You can filter data by clicking on the legend and obtain point-specific data by hovering the cursor over a bubble point
  - Step 1: Load/install plotly
    - R Code: library(plotly)
  - Step 2: Convert ggplot to plotly plot
    - R Code: p <- ggplotly(plot2, width=500, height=500) %>% layout(xaxis = list(range = c(1, 6)), yaxis = list(range = c(8, 35)), legend = list(x = 0.825, y = .975))

Data Management

Using the Which and Order Command

(*Package*: Base R, *Dataset*: iris)

- **Find a row's values at a specified variable's *maximum* value (0:23)**
  - R Code: iris[which.max(iris$Sepal.Length),]
    - Returns the entire row where the variable 'Sepal' is the *highest* value
  - The brackets [ ] are called "indexing"
  - Important: Be sure to add a comma before the last bracket
- **Find a row's values at a specified variable's *minimum* value (1:22)**
  - R Code: iris[which.min(iris$Sepal.Length),]
    - Returns the entire row where the variable 'Sepal' is the *lowest* value
- **'order' selects any location of a variable (i.e. 10th or 23rd highest value) (1:40)**

    - R Code: iris[order(iris$Sepal.Length) [11],]
        - Returns the entire row where the variable 'Sepal' is the *11th highest*

Using File Paths

(Package: Base R)

- **Best way use to specify file/directory path is with function 'file.path'(0:38)**
    - R Code: my_directory <- file.path("C:", "Users", "Joach", "Desktop")
        - Specify components of a directory path and separate by commas
    - my_directory can then be called or used in code to refer to the directory path
- **Function 'file.path' can also be used to specify a specific file (1:44)**
    - R Code: My_file <- file.path("C:", "Users", "Joach", "Desktop", "my_file.csv")
        - If 'my_file.csv' wasn't added, this would specify a directory path

Handling NAs in R

(*Package*: Base R, *Dataset:* airquality)

- **Find missing values (NA) (1:05)**
    - R Code: is.na(airquality)
        - Returns matrix where TRUE is a missing value, FALSE is a numeric value
- **Count the amount of missing values in a data set (2:00)**
    - R Code: sum(is.na(airquality))
        - Returns the total amount of missing values
- **Remove all missing values from a dataset (3:10)**
    - R Code: na.omit(airquality)
    - Be careful! This deletes the entire row where the missing value is present, which could include useful data
- **Calculate the mean of a variable where there is missing values (4:15)**
    - R Code: mean(airquality$Ozone, na.rm = TRUE)

Data Analysis

Using lapply and sapply

(*Package*: Base R, *Dataset*: mtcars)

- **lapply runs a function on each element of a dataset and returns *a list* (0:38)**
    - Step 1: Load data (0:45)
        - R Code: data <- mtcars
    - Step 2: Create demo function (0:50)
        - R Code: mpg_category <- function(mpg){ if(mpg > 30){return("High")}
        - else if (mpg > 20){return("Medium")} else if(mpg <21){return("Low")}}
    - Step 3: Use lapply (1:10)
        - R Code: lapply(X = data$mpg, FUN = mpg_category)

        - This applies the function created in step 2 to every value of data$mpg and returns a list.
- **sapply runs a function on each element of a dataset and returns *a vector or matrix* (1:57)**
    - Step 1-2: Same as lapply
    - Step 3: Use sapply
        - R Code: sapply(X = data$mpg, FUN = mpg_category)
- If the data input *is a column*, sapply and lapply will apply the function to *each value* in the specified column.
- If the data input *is a dataset*, sapply and lapply will apply the function to *each column.*
- Using these functions is often quicker than using a for loop.