Using R in BST 600

RStudio is the program that is used to interface with R code. There are several ways you can work with R code in the RStudio program. For this class, we will use R Notebooks.

R Notebooks allow you to seamlessly integrate R code, R output, and written text in one document. In research (and in coursework!) we often want to create a report that contains both plain English text (interpretations, descriptions, headers, etc.) and output from data analyses in one document. Rather than having to copy/paste analyses from RStudio into a text editor such as Microsoft Word, R Notebooks allow you to create a report containing R code, R output, and written text in one document and save it as a visually-appealing .html file. HTML files are a great choice for reports because they can be opened on any type of computer using any internet browser without having to worry about differing appearance or functionality across different types of computers. HTML files also allow for interactivity such as the ability to click a button within the report to show or hide the R code.

BST 600 is a data analysis course, not a programming course. As such, all code that you need in order to run analyses in BST 600 is given to you in this document. Any modifications you need to make to the code to suit your needs are also described in this document. **You are not expected to struggle with coding in this class – if you’re having trouble with R Notebooks or you’re receiving error messages that you don’t know how to solve, post on the Canvas Discussion board or reach out to Dr. Slade via email**. This course is intended to give you experience selecting, performing, and interpreting statistical analyses; it is not intended to make you a statistical programmer. If you wish to learn more about programming in R, check out BST 535 (Introduction to R Programming).

Introduction to Coding in R

When we want R to do something to a particular variable in a dataset, we have to run a command (a line of code). R recognizes many different commands called functions, each with its own function name. Suppose we have a variable called “variable” in a dataset called “dataset”, and we want to execute a function called “function” on this variable. The command would look like this:

function(dataset$variable)

The dollar sign is used to tell R that the variable called “variable” is in the dataset called “dataset”. For more complex functions, you can give R more information about how you want the command to be executed. These additional pieces of information are called arguments. Arguments are given in a command inside the function parentheses, separated by commas, as such:

function(dataset$variable, arg1=\_\_\_, arg2=\_\_\_, arg3=\_\_\_)

A function’s arguments have specific names which must be spelled correctly in order for the function to recognize the arguments. In the above example, the arguments are called “arg1”, “arg2”, and “arg3”. This is just a general example; in reality, functions and arguments have more descriptive names. The blanks are where you would put the information that you want to give to each of the arguments.

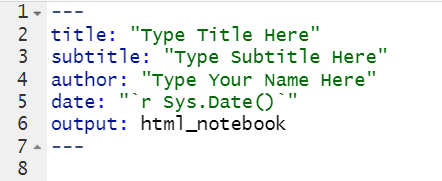
The examples above will execute the function and display any output that the function generates. If you want R to store the output as a named object rather than displaying it directly, you would do the following:

newName <- function(dataset$variable)

“newName” is a name of your choice. This would take the output that results from executing the function on the variable and save it as an object called “newName”. Then, any time you reference the object “newName” in your R code, it represents the output from running this function.

Introduction to R Notebooks

R Notebooks start with a header section at the top that looks like this:

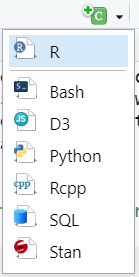


Any of the green text within the quotation marks can be changed. For the date, you can type a date within the quotation marks, or you can use ‘r Sys.Date()’ to automatically print the date that you generate the report. Don’t change any other text in this header area.

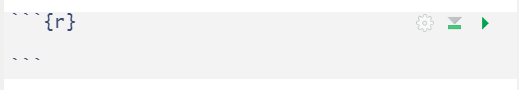
Beneath the header, you can start typing plain text that will appear in the report. R recognizes a wide range of HTML and Markdown style commands such as bold, italic, bullets, etc. The most commonly-encountered syntax is shown in the table below.

|  |  |
| --- | --- |
| Syntax | Description |
| <br> | Creates a blank line in the report |
| \*italics\* | Italicized text between \* symbols |
| \*\*bold\*\* | Bold text between \*\* symbols |
| # Section Header  ## Subsection Header  ### Subsubsection Header | Headers with various font sizes |
| \* item 1  \* item 2 | Bulleted list |
| 1. item 1  2. item 2 | Numbered list |
| <!-- This text won’t show up in the report. --> | Allows you to write messages in the R code script (Notebook) that won’t get printed in the report. This is useful if you need to leave notes to yourself while working on an analysis. |
| $H^0$ | Creates superscript, i.e., H0 |
| $H\_0$ | Creates subscript, i.e., H0 |
| $\sqrt{x}$  $\neq$  etc…. | Creates math symbols/equations between $ symbols. Note: \sqrt creates the square root symbol, and \neq creates the not equal to symbol. For other math symbols, see <https://www.caam.rice.edu/~heinken/latex/symbols.pdf> |
| $\mu$  $\beta$  $\sigma$ | Greek letters (link above also includes other Greek letters) |
| $\bar{x}$ | Creates |

To write some R code, click on the button that looks like a C in a green square with a plus sign on it (image below), then click on R in the dropdown menu. Alternatively, you can use the keyboard shortcut Ctrl+Alt+I on a PC or Cmd+Option+I on a Mac.



This will create a block in the R Notebook that looks like this:



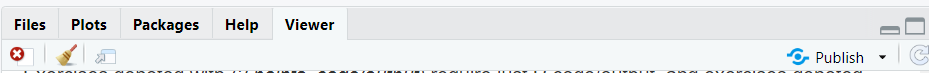
Type or copy/paste R code between the ```{r} line and the ``` line. To execute the R code, click on the green arrow on the right side of the block of R code. The output from the R code will appear beneath the code.

Near the top right corner of the block of R code is a gear symbol. If you click on the gear symbol, you’ll see a box that says “Output:” and has a dropdown menu. On this dropdown menu, you can select whether you would like the report to show just the output or show the code and the output. It should default to showing both the code and the output, which is recommended for this class.

To view the .html report, click on Preview (image below).



When you save your R code and/or when you preview the report, the .html file for the report will be automatically saved in the same folder as the R code. The report will also appear in the “Viewer” window in RStudio. You can click on the button circled below to view the report in an internet browser. What you see in an internet browser is how the report would appear to someone else who you sent the .html file to.



Preview your report frequently while working on a document. Personally, I preview the report after any new R code that I write/execute.

R Code for BST 600

* For each statistical analysis/command, template R code is written in the left box, and any relevant notes about the code are written in the right box.
* In the template R code, red text indicates words or phrases that you are required to edit in order to execute the code properly. Blue text indicates words or phrases that you can choose to edit (or not).
* Throughout the code, “dataset” refers to the name you specify for your dataset. You can call it anything you want when importing the data (no spaces), and then you must refer to it by this name throughout your code.
* Throughout this document, “variable” refers to the name of your variable of interest. You need to change it to the name of the variable you wish to analyze.
* R code is case sensitive, and punctuation matters!

Assignment 1

Import an Excel dataset

|  |  |
| --- | --- |
| library(readxl)  dataset <- read\_excel("filename.xlsx", col\_names=TRUE, na="") | Notes: This is two lines of R code; you need to use both of them. Change *filename* to the Excel document name. Be sure the dataset (Excel file) is in the same folder that your R code script is in. Change *dataset* to any name you wish to use for your dataset (no spaces allowed in the name). It is best to use a short but descriptive name, i.e., “lowbwt” is a good name for the low birthweight dataset. If you see an error message saying “there is no package called ‘readxl’”, you may need to revisit the “Instructions for setting up RStudio.pdf” document to be sure you’ve completed all steps. You only need to complete the steps in that document once. |

Display all variable names in dataset

|  |  |
| --- | --- |
| names(dataset) | Notes: |

Look at entire dataset

|  |  |
| --- | --- |
| View(dataset) | Notes: To look at the dataset, type this line in a chunk of R code, and execute just this chunk of code by pressing the green arrow on the upper right corner of the R chunk in your R Markdown document. This will open a new tab where you can see the dataset. You will want to erase this chunk of code before you knit the HTML report. |

Assignment 2

Mean

|  |  |
| --- | --- |
| mean(dataset$variable, na.rm=TRUE)  tapply(dataset$variable1, dataset$variable2, mean, na.rm=TRUE) | Notes: These are two separate lines of R code. The first one gives the mean of a continuous variable (change *variable* to the name of the variable). The second one gives the mean of a continuous variable within levels of a categorical variable (subgroups). For the second command, change *variable1* to the name of the continuous variable, and change *variable2* to the name of the categorical (grouping) variable. |

Median

|  |  |
| --- | --- |
| median(dataset$variable, na.rm=TRUE)  tapply(dataset$variable1, dataset$variable2, median, na.rm=TRUE) | Notes: These are two separate lines of R code. The first one gives the median of a continuous variable (change *variable* to the name of the variable). The second one gives the median of a continuous variable within levels of a categorical variable (subgroups). For the second command, change *variable1* to the name of the continuous variable, and change *variable2* to the name of the categorical (grouping) variable. |

Standard deviation

|  |  |
| --- | --- |
| sd(dataset$variable, na.rm=TRUE)  tapply(dataset$variable1, dataset$variable2, sd, na.rm=TRUE) | Notes: These are two separate lines of R code. The first one gives the standard deviation of a continuous variable (change *variable* to the name of the variable). The second one gives the standard deviation of a continuous variable within levels of a categorical variable (subgroups). For the second command, change *variable1* to the name of the continuous variable, and change *variable2* to the name of the categorical (grouping) variable. |

Variance

|  |  |
| --- | --- |
| var(dataset$variable, na.rm=TRUE)  tapply(dataset$variable1, dataset$variable2, var, na.rm=TRUE) | Notes: These are two separate lines of R code. The first one gives the variance of a continuous variable (change *variable* to the name of the variable). The second one gives the variance of a continuous variable within levels of a categorical variable (subgroups). For the second command, change *variable1* to the name of the continuous variable, and change *variable2* to the name of the categorical (grouping) variable. |

Interquartile range

|  |  |
| --- | --- |
| IQR(dataset$variable, na.rm=TRUE)  tapply(dataset$variable1, dataset$variable2, IQR, na.rm=TRUE) | Notes: These are two separate lines of R code. The first one gives the IQR of a continuous variable (change *variable* to the name of the variable). The second one gives the IQR of a continuous variable within levels of a categorical variable (subgroups). For the second command, change *variable1* to the name of the continuous variable, and change *variable2* to the name of the categorical (grouping) variable. |

Quantiles (minimum, first quartile, median, third quartile, maximum)

|  |  |
| --- | --- |
| quantile(dataset$variable, na.rm=TRUE)  tapply(dataset$variable1, dataset$variable2, quantile, na.rm=TRUE) | Notes: These are two separate lines of R code. The first one gives the quantiles of a continuous variable (change *variable* to the name of the variable). The second one gives the quantiles of a continuous variable within levels of a categorical variable (subgroups). For the second command, change *variable1* to the name of the continuous variable, and change *variable2* to the name of the categorical (grouping) variable. |

Histogram

|  |  |
| --- | --- |
| hist(dataset$variable, freq=TRUE, col="darkgray", xlab="Write your x axis label here", main="") | Notes: For a list of color names that you can use in place of *darkgray*, see this website: http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf |

Boxplot

|  |  |
| --- | --- |
| boxplot(dataset$variable, ylab="Write your y axis label here") | Notes: |

Side-by-side boxplots (boxplot of continuous variable within levels of categorical variable (subgroups))

|  |  |
| --- | --- |
| boxplot(variable1 ~ variable2, data=dataset, xlab="Write your x axis label here", ylab="Write your y axis label here") | Notes: *variable1* is the continuous variable, and *variable2* is the categorical variable. |

Scatterplot

|  |  |
| --- | --- |
| plot(yvariable~xvariable, data=dataset, xlab="Write your x axis label here", ylab="Write your y axis label here", main="", pch=19) | Notes: *xvariable* is the name of the variable that you want on the x-axis, and *yvariable* is the name of the variable that you want on the y-axis. |

Assignment 3

Confidence interval for a mean (continuous data, 1 group)

|  |  |
| --- | --- |
| t.test(dataset$variable, conf.level=.95)$conf.int | Notes: This creates a 95% confidence interval. Change *.95* to change the confidence level. The first line in the output gives the lower and upper bounds of the confidence interval. |

One-group t-test (continuous data, 1 group)

|  |  |
| --- | --- |
| t.test(dataset$variable, alternative="two.sided", mu=#, conf.level=.95) | Notes: The *#* character represents the null value; change this to the number in the null hypothesis. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. The confidence level is 1 minus the significance level, so *.95* should be changed if the significance level is something other than 0.05.  The p-value for the one-group t-test is at the end of the second line of the output. The output also includes a confidence interval for the population mean as well as the estimated sample mean. |

Assignment 4

Power and sample size calculations for one-group t-test (continuous data, 1 group)

|  |  |
| --- | --- |
| power.t.test(n=#, delta=#, sd=#, power=#, sig.level=.05, type="one.sample", alternative="two.sided") | Notes: Fill in numbers for the effect size (delta) and standard deviation (sd). To perform a sample size calculation, fill in a number for power (as a decimal, i.e., for 80% power you would type 0.8), and set n=NULL. To perform a power calculation for a given sample size, fill in a number for the sample size (n), and set power=NULL.  For a one-sided test, change *two.sided* to *one.sided*. The significance level can be changed to other values of alpha. |

Confidence interval for difference in means, paired data

|  |  |
| --- | --- |
| t.test(dataset$variable1, dataset$variable2, conf.level=.95, paired=TRUE)$conf.int | Notes: Use this code when your paired data is recorded as two separate variables (i.e., all observations in *variable1* are paired with an observation in *variable2*). The confidence level is 1 minus the significance level, so *.95* should be changed if the significance level is something other than 0.05.  The first line in the output gives the lower and upper bounds of the confidence interval. |

Paired t-test (continuous data, paired)

|  |  |
| --- | --- |
| t.test(dataset$variable1, dataset$variable2, alternative="two.sided", conf.level=.95, paired=TRUE) | Notes: Use this code when your paired data is recorded as two separate variables (i.e., all observations in *variable1* are paired with an observation in *variable2*). For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. The confidence level is 1 minus the significance level, so *.95* should be changed if the significance level is something other than 0.05.  The p-value for the paired t-test is at the end of the second line of the output. The output also includes a confidence interval for the mean difference in the population as well as the estimated mean difference in the sample. |

Power and sample size calculations for paired t-test (continuous data, paired)

|  |  |
| --- | --- |
| power.t.test(n=#, delta=#, sd=#, power=#, sig.level=.05, type="paired", alternative="two.sided") | Notes: Fill in numbers for the effect size (delta) and standard deviation (sd). The effect size is the expected mean difference. To perform a sample size calculation, fill in a number for power, and set n=NULL. To perform a power calculation for a given sample size, fill in a number for the sample size (n), and set power=NULL. Remember that the sample size is the number of pairs, and the standard deviation is the standard deviation of the differences. For a one-sided test, change *two.sided* to *one.sided*. The significance level can be changed to other values of alpha. |

Two-group t-test (continuous data, 2 independent groups)

|  |  |
| --- | --- |
| t.test(variable1 ~ variable2, data=dataset, alternative="two.sided", conf.level=.95, var.equal=FALSE) | Notes: *variable1* is the continuous variable, and *variable2* is the categorical (grouping) variable. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. The confidence level is 1 minus the significance level, so *.95* should be changed if the significance level is something other than 0.05. If *var.equal=FALSE*, the test will assume unequal variances. Change *FALSE* to *TRUE* to assume equal variances.  The p-value for the two-group t-test is at the end of the second line of the output. The output also includes a confidence interval for the difference in means in the population as well as the estimated difference in means in the sample. |

Power and sample size calculations for two-group t-test (continuous data, 2 independent groups)

|  |  |
| --- | --- |
| power.t.test(n=#, delta=#, sd=#, power=#, sig.level=.05, type="two.sample", alternative="two.sided") | Notes: Fill in numbers for the effect size (delta) and standard deviation (sd). The effect size is the expected difference in means in the two groups. To perform a sample size calculation, fill in a number for power, and set n=NULL. To perform a power calculation for a given sample size, fill in a number for the sample size (n), and set power=NULL. For a one-sided test, change *two.sided* to *one.sided*. The significance level can be changed to other values of alpha. |

Assignment 5

Test for equality of variances (two variances F-test)

|  |  |
| --- | --- |
| var.test(variable1 ~ variable2, data=dataset, alternative="two.sided", conf.level=.95) | Notes: *variable1* is the continuous variable, and *variable2* is the categorical (grouping) variable. This test is typically two-sided with a significance level of 0.05, but in the rare case that it’s not, *two.sided* and *.95* can be adjusted accordingly.  The p-value for the two-variances F-test is at the end of the second line of the output. |

ANOVA test

|  |  |
| --- | --- |
| AnovaModel.1 <- aov(variable1 ~ variable2, data=dataset)  summary(AnovaModel.1) | Notes: *variable1* is the continuous variable, and *variable2* is the categorical (grouping) variable. You can change *AnovaModel.1* to any name you wish to choose for the model. If you’re fitting more than one ANOVA model in the same R script, you should give them different names.  The first row of the table in the output will say the name of the categorical (grouping) variable. At the end of that row, the last column (labeled ‘Pr(>F)’) contains the p-value for the ANOVA test. |

Pairwise comparisons with Tukey’s adjustment

|  |  |
| --- | --- |
| TukeyHSD(AnovaModel.1) | Notes: This code is set up assuming that you have already fit an ANOVA model called *AnovaModel.1* that you wish to follow up with all pairwise comparisons. You must run the code to create the ANOVA model first before running this code (see above section).  The Tukey-adjusted p-values for each pairwise comparison are in the last column labeled ‘p adj’. |

Assignment 7

Correlation

|  |  |
| --- | --- |
| cor(dataset$variable1, dataset$variable2, use="complete") | Notes: *variable1* and *variable2* are both continuous variables. |

Confidence interval and hypothesis test for correlation

|  |  |
| --- | --- |
| cor.test(dataset$variable1, dataset$variable2, alternative="two.sided", method="pearson", conf.level=0.95) | Notes: *variable1* and *variable2* are both continuous variables. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. The confidence level is 1 minus the significance level, so *.95* should be changed if the significance level is something other than 0.05. The null value is automatically set to 0.  The p-value for the correlation test is at the end of the second line of the output. The output also includes a confidence interval for the correlation in the population as well as the estimated correlation in the sample. |

Simple linear regression model

|  |  |
| --- | --- |
| RegModel.1 <- lm(variable1~variable2, data=dataset)  summary(RegModel.1) | Notes: *variable1* is the outcome variable, and *variable2* is the predictor variable. You can change *RegModel.1* to any name you wish to choose for the model. If you’re fitting more than one regression model in a single R script, you should give them different names.  The most relevant part of the output is the table labeled ‘Coefficients’. In this table, ‘Estimate’ gives you the estimated coefficients (intercept and slope), and ‘Pr(>|t|)’ gives the p-value for the hypothesis test of the coefficient. The R2 value for the model can be found in the second-to-last line of the output. |

Predictions from a simple linear regression model

|  |  |
| --- | --- |
| as.numeric(predict(RegModel.1, data.frame(variable=value))) | Notes: This code is set up assuming that you have already fit a linear regression model called *RegModel.1* that you wish to make a prediction with. You must run the code to create the linear regression model first before running this code (see above section). If you call your regression model a different name, change *RegModel.1* to that name.  *variable* is the name of the predictor variable in the regression model. It must be spelled exactly how the variable is spelled in the dataset.  *value* is the value of the predictor variable that you would like to plug into the regression model. If it is a numerical value, this is a number. If it is a category name, the value must be written in quotation marks, i.e., “Male”. For categorical variables, the category must be spelled exactly as it is recorded in the dataset (case sensitive). |

Confidence intervals for linear regression coefficients

|  |  |
| --- | --- |
| confint(RegModel.1, level=0.95) | Notes:This code is set up assuming that you have already fit a linear regression model called *RegModel.1* that you wish to follow up with confidence intervals for the coefficients. You must run the code to create the linear regression model first before running this code (see above section). If you call your regression model a different name, change *RegModel.1* to that name. Change *.95* to a different number if you want a different confidence level for the intervals. |

Assignment 8

Add a simple linear regression line to a scatterplot

|  |  |
| --- | --- |
| abline(RegModel.1) | Notes: First create a scatterplot between the predictor and outcome variables as usual. Then, run this line to add the regression line. This code is set up assuming that you have already fit a regression model called *RegModel.1*. You must run the code to create the regression model first before running this code (see above section). If you call your regression model a different name, change *RegModel.1* to that name. |

Residual plot

|  |  |
| --- | --- |
| plot(RegModel.1, 1) | Notes: This code is set up assuming that you have already fit a regression model called *RegModel.1* that you wish to follow up with diagnostic plots. You must run the code to create the regression model first before running this code (see above section). If you call your regression model a different name, change *RegModel.1* to that name. |

Normal QQ plot

|  |  |
| --- | --- |
| plot(RegModel.1, 2) | Notes: This code is set up assuming that you have already fit a regression model called *RegModel.1* that you wish to follow up with diagnostic plots. You must run the code to create the regression model first before running this code (see above section). If you call your regression model a different name, change *RegModel.1* to that name. |

Create a new (transformed) variable

|  |  |
| --- | --- |
| dataset$newvariable <- log(dataset$variable1)  dataset$newvariable <- sqrt(dataset$variable1)  dataset$newvariable <- dataset$variable1^2 | Notes: This code shows three examples: a log transformation, a square root transformation, and a squared transformation. In each, *variable1* is the existing variable you wish to transform, and *newvariable* is the name of the new variable you are creating. You can choose whatever name you want for the new variable. |

Multiple linear regression model

|  |  |
| --- | --- |
| RegModel.1 <- lm(variable1 ~ variable2 + variable3, data=dataset)  summary(RegModel.1) | Notes: *variable1* is the outcome variable. *variable2* and *variable3* are predictor variables (you can add even more predictors with additional + signs). You can change *RegModel.1* to any name you wish to choose for the model. If you’re fitting more than one regression model in a single R script, you should give them different names.  The most relevant part of the output is the table labeled ‘Coefficients’. In this table, the first row represents the intercept, and each subsequent row represents the slope for the given variable. ‘Estimate’ gives you the estimated coefficient, and ‘Pr(>|t|)’ gives the p-value for the hypothesis test of the coefficient. The adjusted R2 value for the model can be found in the second-to-last line of the output. |

Assignment 9

Predictions from a multiple linear regression model (including those with indicator variables and/or interaction terms)

|  |  |
| --- | --- |
| as.numeric(predict(RegModel.1, data.frame(variable1=value1, variable2=value2, variable3=value3))) | Notes: This code is set up assuming that you have already fit a linear regression model called *RegModel.1* that you wish to follow up with confidence intervals for the coefficients. You must run the code to create the linear regression model first before running this code (see above section). If you call your regression model a different name, change *RegModel.1* to that name.  *variable1, variable2,* and *variable3* are the names of the predictor variables in the regression model. They must be spelled exactly how the variables are spelled in the dataset. You can adjust the code to have more or less than three variables, depending on how many predictor variables are in your model.  *value1, value2,* and *value3* are the values of the predictor variables that you would like to plug into the regression model. If it is a numerical value, this is a number. If it is a category name, the value must be written in quotation marks, i.e., “Male”. For categorical variables, the category must be spelled exactly as it is recorded in the dataset (case sensitive).  If your model includes a categorical predictor variable with more than 2 categories, R will automatically use multiple indicator variables to represent this information when fitting the model. However, to use this code to make a prediction, you only have to use the original categorical variable name (for the *variable* argument) and the category of interest (for the *value* argument). R will convert this to the necessary information to plug into the indicator variables in the model.  If the model includes an interaction between two variables, you should provide the value needed for each of the two variables. |

Change the reference category for a categorical variable

|  |  |
| --- | --- |
| dataset$variable <- factor(dataset$variable, levels=c("Category1", "Category2")) | Notes: *Category1* and *Category2* should be changed to the names of the categories in the categorical variable (must be spelled exactly as recorded in your dataset). Whichever one you list first will be the reference category. This code can be extended for cases with more than two categories. |

Create an indicator variable

|  |  |
| --- | --- |
| dataset$newvariable <- as.factor((dataset$variable1 == "Category1")\*1)  dataset$newvariable <- as.factor((dataset$variable2 > 100)\*1) | Notes: This code shows two examples. In each, *newvariable* is the name of the new variable you are creating. You can choose whatever name you want for the new variable.  In the first example, suppose that *variable1* is a categorical variable in your dataset. This code creates a new variable called *newvariable* that equals 1 for subjects who are in the category called *Category1* (change this to the category name that you want), and 0 for subjects who are not in this category.  In the second example, suppose that *variable2* is a continuous variable in your dataset. This code creates a new variable called *newvariable* that equals 1 for subjects who have a value greater than 100 (change this to whatever number you want), and 0 for subjects who have a value less than or equal to 100.  The variable names and category names must be spelled exactly as they are in the dataset. |

Linear regression model with an interaction term

|  |  |
| --- | --- |
| RegModel.1 <- lm(variable1 ~ variable2 + variable3 + variable2\*variable3, data=dataset)  summary(RegModel.1) | Notes: *variable1* is the outcome variable. *variable2, variable3,* and their interaction are predictors. This fits a model including a main effect of *variable2*, a main effect of *variable3*, and an interaction between *variable2* and *variable3*.  You can change *RegModel.1* to any name you wish to choose for the model. If you’re fitting more than one regression model, you should give them different names.  The most relevant part of the output is the table labeled ‘Coefficients’. In this table, the first row represents the intercept, and each subsequent row represents the slope for the given variable. ‘Estimate’ gives you the estimated coefficient, and ‘Pr(>|t|)’ gives the p-value for the hypothesis test of the coefficient. The interaction term is the row indicated by the two variables with a colon between them. The R2 value for the model can be found in the second-to-last line of the output. |

Create a centered variable

|  |  |
| --- | --- |
| dataset$newvariable <- dataset$variable1 - mean(dataset$variable1) | Notes: This creates a new variable called *newvariable* which is a centered version of *variable1*. You can choose whatever name you want for the new variable. |

Assignment 12

Bar graph

|  |  |
| --- | --- |
| barplot(table(dataset$variable), xlab="Write your x axis label here", ylab="Frequency") | Notes: |

Side-by-side bar graph

|  |  |
| --- | --- |
| tab <- table(dataset$variable1, dataset$variable2)  barplot(tab, beside=TRUE, xlab="Write your x axis label here", ylab="Frequency", legend.text=names(tab[,1]), args.legend=list(x="top", inset=c(1,-0.1))) | Notes: This is two lines of R code; you need to use both of them. This will summarize *variable2* stratified by *variable1*. Both variables should be categorical. Change *TRUE* to *FALSE* to stack the bars on top of each other. |

Pie chart

|  |  |
| --- | --- |
| pie(table(dataset$variable), radius=1) | Notes: |

Frequencies (counts) for 1 categorical variable

|  |  |
| --- | --- |
| table(dataset$variable) | Notes: |

Relative frequencies (percentages) for 1 categorical variable

|  |  |
| --- | --- |
| prop.table(table(dataset$variable)) | Notes: |

Table for 2 categorical variables (2x2 table), counts only

|  |  |
| --- | --- |
| xtabs(~ variable1 + variable2, data=dataset) | Notes: *variable1* is the row variable, and *variable2* is the column variable. |

Table of **overall** percentages for 2 categorical variables

|  |  |
| --- | --- |
| proportions(xtabs(~ variable1 + variable2, data=dataset)) | Notes: *variable1* is the row variable, and *variable2* is the column variable. This computes overall probabilities where the denominator is the entire sample. |

Table of **row** percentages for 2 categorical variables

|  |  |
| --- | --- |
| proportions(xtabs(~ variable1 + variable2, data=dataset), margin=1) | Notes: *variable1* is the row variable, and *variable2* is the column variable. This computes conditional probabilities where the denominator is the total in each row. |

Table of **column** percentages for 2 categorical variables

|  |  |
| --- | --- |
| proportions(xtabs(~ variable1 + variable2, data=dataset), margin=2) | Notes: *variable1* is the row variable, and *variable2* is the column variable. This computes conditional probabilities where the denominator is the total in each column. |

Change order of categories for a categorical variable

|  |  |
| --- | --- |
| dataset$variable <- factor(dataset$variable, levels=c('Category 1', 'Category 2')) | Notes: *Category 1* and *Category 2* should be changed to the names of the categories in the categorical variable (must be spelled exactly as recorded in your dataset). Whichever one you list first will be treated as the category of interest for proportions. |

Confidence interval for one proportion (exact binomial method)

|  |  |
| --- | --- |
| binom.test(xtabs(~variable, data=dataset), conf.level=.95)$conf.int | Notes: *variable* is the categorical variable of interest. This code calculates a confidence interval assuming that the proportion of interest is based off of the category that is listed first for this variable in R. It is strongly recommended that you run the following command:  table(dataset$variable)  BEFORE running this code to see which order the categories are in. If your category of interest is not the category listed first, you must reorder the factor levels for *variable* before running this command (see code above for reordering factor levels). The code here creates a 95% confidence interval. Change *.95* to change the confidence level. |

Confidence interval for one proportion (normal approximation method)

|  |  |
| --- | --- |
| prop.test(xtabs(~variable, data=dataset), conf.level=.95)$conf.int | Notes: *variable* is the categorical variable of interest. This code calculates a confidence interval assuming that the proportion of interest is based off of the category that is listed first for this variable in R. It is strongly recommended that you run the following command:  table(dataset$variable)  BEFORE running this code to see which order the categories are in. If your category of interest is not the category listed first, you must reorder the factor levels for *variable* before running this command (see code above for reordering factor levels). The code here creates a 95% confidence interval. Change *.95* to change the confidence level. |

Hypothesis test for one proportion (exact binomial method)

|  |  |
| --- | --- |
| binom.test(xtabs(~variable, data=dataset), alternative="two.sided", p=#, conf.level=.95) | Notes: *variable* is the categorical variable of interest. This code performs a hypothesis test assuming that the proportion of interest is based off of the category that is listed first for this variable in R. It is strongly recommended that you run the following command:  table(dataset$variable)  BEFORE running this code to see which order the categories are in. If your category of interest is not the category listed first, you must reorder the factor levels for *variable* before running this command (see code above for reordering factor levels).  The *#* character represents the null value; change this to the number in the null hypothesis. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. The confidence level is 1 minus the significance level, so *.95* should be changed if the significance level is something other than 0.05.  The p-value for the exact binomial test is at the end of the second line of the output. The output also includes a confidence interval for the population proportion as well as the estimated sample proportion. |

Hypothesis test for one proportion (normal approximation method)

|  |  |
| --- | --- |
| prop.test(xtabs(~variable, data=dataset), alternative="two.sided", p=#, conf.level=.95) | Notes: *variable* is the categorical variable of interest. This code performs a hypothesis test assuming that the proportion of interest is based off of the category that is listed first for this variable in R. It is strongly recommended that you run the following command:  table(dataset$variable)  BEFORE running this code to see which order the categories are in. If your category of interest is not the category listed first, you must reorder the factor levels for *variable* before running this command (see code above for reordering factor levels).  The *#* character represents the null value; change this to the number in the null hypothesis. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. The confidence level is 1 minus the significance level, so *.95* should be changed if the significance level is something other than 0.05.  The p-value for the normal approximation test is at the end of the second line of the output. The output also includes a confidence interval for the population proportion as well as the estimated sample proportion. |

Power and sample size calculations for one-group test of a proportion (binary data, 1 group)

|  |  |
| --- | --- |
| power.binom.test(p0=#, p\_true=#, n=#, power=#, sig.level=.05, alternative="two.sided")  Warning: This function is \*not\* included in the base configuration of R. A block of code will be provided in assignments that contains instructions for R to add this function to its repertoire. If you try to run this command and get an error message saying ‘could not find the function “power.binom.test”’, it’s because you need to run that block of code for initializing the function BEFORE running this command. | Notes: Fill in numbers for *p0* and *p1* which represent the hypothesized proportion (AKA the null value) and the expected true proportion, respectively. To perform a sample size calculation, fill in a number for power (as a decimal, i.e., for 80% power you would type 0.8), and set n=NULL. To perform a power calculation for a given sample size, fill in a number for the sample size (n), and set power=NULL.  For a one-sided test, change *two.sided* to *one.sided*. The significance level can be changed to other values of alpha. |

Assignment 13

Risk difference and confidence interval for risk difference

|  |  |
| --- | --- |
| risk\_diff(table(dataset$variable1, dataset$variable2))  Warning: This function is \*not\* included in the base configuration of R. A block of code will be provided in assignments that contains instructions for R to add this function to its repertoire. If you try to run this command and get an error message saying ‘could not find the function “risk\_diff”’, it’s because you need to run that block of code for initializing the function BEFORE running this command. | Note: *variable1* is the exposure variable and *variable2* is the outcome variable. You should run the command: table(dataset$variable1, dataset$variable2)  to look at how the 2x2 table is oriented BEFORE running this code. This code treats the first column listed as the category of interest for the outcome variable. It treats the first row listed as the first group and the second row listed as the second group for the exposure variable. You may need to reorder factor levels of *variable1* and *variable2* before running this code in order to match your research question of interest. This code outputs three numbers: the risk difference, the lower bound of the 95% CI for the risk difference, and the upper bound of the 95% CI for the risk difference. You can change .95 to a different number if you want a different confidence level for the confidence intervals (i.e., .90 would give 90% confidence intervals). |

Risk ratio and confidence interval for risk ratio

|  |  |
| --- | --- |
| risk\_ratio(table(dataset$variable1, dataset$variable2))  Warning: This function is \*not\* present in the base configuration of R. A block of code will be provided in assignments that contains instructions for R to add this function to its repertoire. If you try to run this command and get an error message saying ‘could not find the function “risk\_ratio”’, it’s because you need to run that block of code for initializing the function BEFORE running this command. | Note: *variable1* is the exposure variable and *variable2* is the outcome variable. You should run the command: table(dataset$variable1, dataset$variable2)  to look at how the 2x2 table is oriented BEFORE running this code. This code treats the first column listed as the category of interest for the outcome variable. It treats the first row listed as the first group and the second row listed as the second group for the exposure variable. You may need to reorder factor levels of *variable1* and *variable2* before running this code in order to match your research question of interest. This code outputs three numbers: the risk ratio, the lower bound of the 95% CI for the risk ratio, and the upper bound of the 95% CI for the risk ratio. You can change .95 to a different number if you want a different confidence level for the confidence intervals (i.e., .90 would give 90% confidence intervals). |

Odds ratio and confidence interval for odds ratio

|  |  |
| --- | --- |
| odds\_ratio(table(dataset$variable1, dataset$variable2))  Warning: This function is \*not\* present in the base configuration of R. A block of code will be provided in assignments that contains instructions for R to add this function to its repertoire. If you try to run this command and get an error message saying ‘could not find the function “odds\_ratio”’, it’s because you need to run that block of code for initializing the function BEFORE running this command. | Note: *variable1* is the exposure variable and *variable2* is the outcome variable. You should run the command: table(dataset$variable1, dataset$variable2)  to look at how the 2x2 table is oriented BEFORE running this code. This code treats the first column listed as the category of interest for the outcome variable. It treats the first row listed as the first group and the second row listed as the second group for the exposure variable. You may need to reorder factor levels of *variable1* and *variable2* before running this code in order to match your research question of interest. This code outputs three numbers: the odds ratio, the lower bound of the 95% CI for the odds ratio, and the upper bound of the 95% CI for the odds ratio. You can change .95 to a different number if you want a different confidence level for the confidence intervals (i.e., .90 would give 90% confidence intervals). |

Calculate expected cell counts for a chi-squared test

|  |  |
| --- | --- |
| with(dataset, chisq.test(variable1, variable2))$expected | Notes: *variable1* defines the two (or more) groups. *variable2* defines the outcome of interest; the category that represents a “success” (for your proportion of interest) is the one that comes first alphabetically, by default. If the other category is your category of interest, be sure to reorder factor levels before running this code. |

Hypothesis test for two or more proportions (chi-squared test)

|  |  |
| --- | --- |
| chisq.test(dataset$variable1, dataset$variable2) | Notes: *variable1* is the exposure (group) variable and *variable2* is the outcome variable. You should run the command: table(dataset$variable1, dataset$variable2)  to look at how the 2x2 table is oriented BEFORE running this code. This code treats the first column listed as the category of interest for the outcome variable. You may need to reorder factor levels of *variable2* before running this code in order to match your research question of interest.  The p-value for the chi-squared test is at the end of the output. |

Hypothesis test for two or more proportions (Fisher’s exact test)

|  |  |
| --- | --- |
| fisher.test(dataset$variable1, dataset$variable2) | Notes: *variable1* is the exposure (group) variable and *variable2* is the outcome variable. You should run the command: table(dataset$variable1, dataset$variable2)  to look at how the 2x2 table is oriented BEFORE running this code. This code treats the first column listed as the category of interest for the outcome variable. You may need to reorder factor levels of *variable2* before running this code in order to match your research question of interest.  The p-value for Fisher’s exact test is on the second line of the output. |

Power and sample size calculations for two-group test of a proportion (binary data, 2 independent groups)

|  |  |
| --- | --- |
| power.prop.test(n=#, p1=#, p2=#, power=#, sig.level=.05, alternative="two.sided") | Notes: Fill in numbers for the expected proportion in the first group (p1) and the expected proportion in the second group (p2). To perform a sample size calculation, fill in a number for power, and set n=NULL. To perform a power calculation for a given sample size, fill in a number for the sample size (n), and set power=NULL. For a one-sided test, change *two.sided* to *one.sided*. The significance level can be changed to other values of alpha. |

Assignment 14

Logistic regression model with untransformed coefficients

|  |  |
| --- | --- |
| dataset$outcome <- as.factor(dataset$outcome)  table(dataset$outcome)  GLM.1 <- glm(outcome ~ exposure, family=binomial, data=dataset)  summary(GLM.1)$coef | Notes: Change *outcome* to the name of the outcome variable, and change *exposure* to the name of the exposure variable. This is four lines of code. The second line prints a frequency table for the outcome variable. The logistic regression model will be modeling the log odds of the **second** category listed in this table for the outcome variable. For categorical *exposure* variables, the category being represented as the slope in the regression model will be indicated in the regression table output (the other category that is not listed is the reference category). You may need to reorder the categories for the *outcome* and/or *exposure* variables before fitting the logistic regression model in order to match your research question of interest.  To include more than one predictor variable, you can add additional variables separated by a plus sign after *exposure*. You can change *GLM.1* to any name you wish to choose for the model. If you’re fitting more than one regression model, you should give them different names.  The fourth line of code (*summary* command) gives the regression table where the first row represents the intercept, and each subsequent row represents the slope for the given variable. ‘Estimate’ gives you the estimated **untransformed** coefficient (β0, β1, etc.), and ‘Pr(>|z|)’ gives the p-value for the hypothesis test of the coefficient. |

Exponentiate coefficients from logistic regression model

|  |  |
| --- | --- |
| exp(coef(GLM.1)) | Notes:This code is set up assuming that you have already fit a logistic regression model called *GLM.1* that you wish to follow up with confidence intervals for the coefficients. You must run the code to create the logistic regression model first before running this code (see above section).  This code gives the **exponentiated** coefficients from the logistic regression model (exp(β0), exp(β1), etc.). |

Predictions from a logistic regression model

|  |  |
| --- | --- |
| as.numeric(predict(GLM.1, data.frame(variable=value))) | Notes: This code is set up assuming that you have already fit a logistic regression model called *GLM.1* that you wish to make predictions with. You must run the code to create the logistic regression model first before running this code (see above section). If you call your regression model a different name, change *GLM.1* to that name.  *variable* is the name of the predictor variable in the regression model. It must be spelled exactly how the variable is spelled in the dataset.  *value* is the value of the predictor variable that you would like to plug into the regression model. If it is a numerical value, this is a number. If it is a category name, the value must be written in quotation marks, i.e., “Male”. For categorical variables, the category must be spelled exactly as it is recorded in the dataset (case sensitive).  Predictions using this code are untransformed, meaning they are on the log odds scale. |

Assignment 15

Confidence intervals for logistic regression coefficients

|  |  |
| --- | --- |
| confint(GLM.1, level=0.95)  exp(confint(GLM.1, level=0.95)) | Notes:This code is set up assuming that you have already fit a logistic regression model called *GLM.1* that you wish to follow up with confidence intervals for the coefficients. You must run the code to create the logistic regression model first before running this code (see above section).  Change *.95* to a different number if you want a different confidence level for the intervals. The first line of code gives confidence intervals for the untransformed coefficients, and the second line of code gives confidence intervals for the exponentiated coefficients. |

AIC for a logistic regression model

|  |  |
| --- | --- |
| summary(GLM.1)  summary(GLM.1)$aic | Notes:This code is set up assuming that you have already fit a logistic regression model called *GLM.1* that you wish to calculate the AIC for. You must run the code to create the logistic regression model first before running this code (see above section).  You can run either one of these commands (do NOT need to run both) to find the AIC. The first line of code prints the entire model output, and the AIC can be found at the end of the output. The second line of code prints just the AIC for the model. |

Wilcoxon signed-rank test (one group)

|  |  |
| --- | --- |
| wilcox.test(dataset$variable, alternative="two.sided", mu=#) | Notes: The *#* character represents the null value; change this to the number in the null hypothesis. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. |

Wilcoxon signed-rank test (two paired groups)

|  |  |
| --- | --- |
| wilcox.test(dataset$variable1, dataset$variable2, alternative="two.sided", paired=TRUE) | Notes: *variable1* and *variable2* are the two paired (continuous) variables. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. |

Wilcoxon rank sum test (two independent groups)

|  |  |
| --- | --- |
| wilcox.test(variable1 ~ variable2, alternative="two.sided", data=dataset) | Notes: *variable1* is the continuous variable, and *variable2* is the binary variable that defines the two groups. For a one-sided test, change *two.sided* to *less* or *greater*, depending on the direction of the alternative hypothesis. |

Kruskal-Wallis test

|  |  |
| --- | --- |
| kruskal.test(variable1 ~ variable2, data=dataset) | Notes: *variable1* is the continuous variable, and *variable2* is the categorical variable that defines the groups. |