**WEEK 1:  
  
Discussion 1**

R- programming language for statisticians

R studio- way to use that programming language in an easy way

*RStudio Basics:*

Top left corner:

* Press the green plus sign and click on R Script to create a new untitled file

Bottom left corner:

* Console in the bottom left is used to write things line by line. Also when code is run from the a script, it’ll show up here as output.
* Terminal and Background Jobs tabs can be ignored

Top right corner:

* Environment tab- if you import any data into R or import variables like vector or matrix, all of this data/variables can be seen as values in this tab
* History, Connections and Tutorials tabs won’t be used

Bottom right corner:

* Files shows which directory you’re in or what folder you’re in on your computer
* Plots tab to display graphs if created
* Packages show what kind of libraries are installed
* Help tab if you don’t know how to use a certain function or you don’t know what something is doing
* Viewer and Presentation tabs won’t be used

Rename Lab File to anything: Labs consist of coding and non-coding portions- create a Word doc or Google docs for It. So take a screenshot of the code in the file and the output as well and then include in answer in Doc. Would not submit the R file itself.

Numeric values:

Ex: 2, 10, 1.2 (numbers or decimals) – can do anything with them, addition subtraction etc.

Character values:

Anything in quotation marks is a character: “1”, “ahygfg” (can also put numbers in “ ”, the’ll be treated as characters), “()!@#?{“ (can even put symbols in “ ”); usually to express qualitative data in stats ex: blue, green or Monday, Tuesday etc.

Comments:

Lines with a ### (pound) sign, R ignores them- these are comments or remarks for user to understand details in complex code and make the code more readable

Functions:

Functions in R are basically a word followed by parenthesis; kind of like f(x) in math- f is function and x inside () is the input.

* If we have an object in R, and want to what type (or class) of object it is then use: **class()**
* Example: if we run > class(1): output is [1] “numeric” or if we run > class(“ahdebjdb”): output is [1] “character”

Vectors/Arrays:

One of the biggest advantages of R is that compared to other programing languages, R is built around vectors- big building blocks of code in R. Makes it really easy to do data analysis.

Simply, just objects or values that have been grouped together into one array: use c()

C stands for concatenate- stands for appending them into one array.

Ex: c(1,2,3,4,5)

Output: [1] 1 2 4 5 *🡪 These 4 numbers have been grouped together into one array/vector*

Ex: c(“a”, “b”, “c”)

Output: [1] “a” “b” “c”

Vectors need to have the same data type always.

Ex: c(“a”, 2, 3 “d”)

Output: [1] “a” “2” “3” “d” *🡪 Output produced will have “ ” around all so now all characters; R automatically changes things to be the same data type in order to avoid errors. In this case, easy to convert numbers into characters, but it’s not easy to convert characters back into numbers.*

Assigning Variables:

Variables are basically used for storing something to be used later.

Ex: numbers

Output: Error: object ‘numbers’ not found 🡪 *numbers is undefined; if it’s just a word by itself without (), R will think this is an object trying to store some value, but since numbers hasn’t been assigned a value, R is trying to find a value that doesn’t exist and thus produces an error.*

Set the value of numbers using <- (not =):

Ex: numbers <- c(9,8,1,8)

Output: numbers <- c(9,8,1,8) & will pop up on the top right panel as num [1:4] 9 8 1 8 🡪 1 through 4 is just like the number of elements

Now if numbers are run by itself again, we get that vector back; thus basically we’re storing a value so that we can use it later and we don’t have to retype this vector over and over again.

Ex: numbers *🡪 now we can call numbers can retrieve the values we set*

Output: [1] 9 8 18

We can also overwrite stored values, but make sure not accidentally overwriting important information. Variables are not permanent, can change them if make a mistake:

Ex:

word <- “apple pie”

word

word <- “pie”

word

Output:

> word <- “apple pie”

> word

[1] “apple pie”

> word <- “pie”

> word

[1] “pie”

Vector Operations: (vector multiplication, division etc. is element by element)

counts <- c(4,1,2) 🡪 numeric vector

days <- c(“Monday”, “Tuesday”, “Friday”) 🡪 character vector

We can do math with numeric vectors:

**+** for addition

**-** for subtraction

**\*** for multiplication

**/** for division

**^** for exponent

Ex: counts^2

Output:

> counts^2

[1] 16 1 4

Ex: counts + 3

Output:

> counts + 3

[1] 7 4 5

🡪 Very useful since now can write code that works the same way with whether you have one number or a vector of like a thousand numbers and you don’t have to individually add to each element inside of the vector- useful to analyze large data sets.

Multiplying vectors is element-wise, NOT like linear algebra

Ex:

counts2 <- c(3,2,1) *🡪 creating a second vector counts2*

counts \* counts2

Output:

> counts2 <- c(3,2,1)

> counts \* counts2

[1] 12 2 1

Also can use include vectors with certain functions as inputs. We can combine vectors (if we have separate vectors that represent parts of a data set) into a matrix using: **cbind()** [c stands for column, have rbind() for rows]

Ex: new <- cbind(counts, days) *🡪 output of cbind(counts, days) assigned to variable new*

new

Output:

< new <- cbind(counts, days)

< new

counts days

[1,] “4” “Monday”

[2,] “1” “Tuesday”

[3,] “2” “Friday”

(also changed all counts to characters to maintain same data type)

🡪 Can further check what class this object is through

Ex: **class()**

Output: > class(new)

[1] “matrix” “array” (type of array is matrix)

🡪 Can finally check the dimensions of the matrix using function: **dim()**

Ex: dim(new)

Output: < dim(new) 🡪 *output is the number of rows and number of columns*

[1] 3 2

**Discussion 2:**

* One of the main reasons behind the usage of R is to do data analysis in statistics especially with large sets of data given.
* For this, need to learn how data is stored in files and how to take those data files and put them in R so can be worked with. Will be working with birth.csv for this. (csv means every value in the file is separated using a comma)

Downloading Data:

1. To work with data files such as csv files, we need to set the working directory (folder)
2. Click “Session > Set Working Directory” at the top of the screen
3. Or, use **setwd()**

setwd(“~/Desktop/STATS10”)

* Move the birth.csv. file in the file to be used (directory and folder is synonyms), and make sure the R script (saved R file working on) is in the same folder.
* Working directory is the folder that R is based in. If you think about each directory as like a folder in your computer, the working directory is kind of like what folder R thinks it’s in right now. It’s important since it’s the way files are accessed in R and need to specify not only the file name, but also the path to get to that file.
* Need to import the data set birth.csv into R and view it in R as well. Screenshot the code though.

1. After we’re in the same folder as the data, we can use **read.csv()** to load the data
2. Make sure to store the output to save it

NCbirths <- read.csv(“births.csv”) 🡪 *to* *read and import the data into R: read.csv(), and further storing it as a variable NCbirths*

* Can also use the import button instead of this

1. The data is stored as a data frame, a special type of table data.frame through which different operations and math can be done with the numerical values through R.

Ex: class(NCbirths)

Output: > class(NCbirths)

[1] “data.frame”

1. Check that the data is loaded properly through using function **head()**:

head(NCbirths) *🡪 this gives the first 6 rows*; *good way to check if data loaded properly*

* As long as read.csv translates into a table in the Console, then it’s good.
* It’s a little hard to debug in the console, that’s why use script file since it’s a good organizational tool to edit things.

Installing and Loading Packages:

* Packages or libraries are extra add-ons to R. Like an expansion pack, each package adds extra functions into R for certain uses.
* Different specialized fields in mathematics and statistics require the usage of different tools which can be accessed through certain packages that aren’t available in base/ general R.

1. Go to “Packages > Install” in the bottom right panel and type in name of package
2. Or use **install.packages()** :

install.packages(“maps”)

* Click box that says Install dependencies if not checked. Dependencies are like prerequisites since some packages are built upon others so need to have that installed.

1. Can verify that the package is installed using **find.package()** :

find.package(“maps”) 🡪 *output will produce a path of what folder on computer the maps package was installed to.*

1. Once the package is installed, we can load it using **library()** :

library(maps) 🡪 *need to open it to be able to use anything inside of it; don’t need “ ” for maps anymore*

1. After loading, we can use functions from the packages whenever we want

Ex: map(“world”) *🡪 will just give map of the world on plots tab*

1. If we need help with a function, use ?function

Ex: ?map  *🡪 will bring you to help screen and provide all details*

***9th October, Monday***

**Discussion 3:**

Lab is training to do Assignment. Don’t need to submit Lab, just to practice using R.

* ‘chr’ indicates character and thus categorical or qualitative data
* ‘int’ indicates integer and thus numerical or quantitative data

Basic Data Analysis:

Through this, we want to explore what’s inside of the data set, what does it look like, what are the descriptive statistics, etc.

1. We can extract a column from the data frame using **$**  🡪 *important since we have a mixture of columns with some quantitative and some categorical with differing units, so sometimes it’s better to do an analysis one column at a time.*
2. Extracted columns are treated as vectors 🡪 *format: fulldataset$variableextracted*

Ages <- NCbirths$Mage #Mage stands for Mother’s age

1. We can use [ ] to get a subset of a vector/data frame

Ex 1: ages[1]

Output: > ages[1]

[1] 25 *🡪 pulling the first entry from the vector ages*

Ex 2: ages[4:8]

Output: > ages[4:8]

[1] 37 33 29 30 20 *🡪 gives entry 4 up to entry 8 of vector ages*

*-* In most programming languages, the first entry/index of a data set is 0, but in R it is 1.

1. The negative sign will return everything except the subset you write

ages[-(1:1925)] 🡪 *excluding everything written or negating it. So instead of giving numbers like the first 1925 entries, it will give everything except those – 1926 up until the end*

* The concept of taking subsets from data and analyzing them will be important with sampling.

1. Based on the values in ages, the unit is probably in years. Can convert the values to months 🡪 *format: vector\*conversion unit*

ages\_in\_months <- ages \* 12

* R treats every number just as a number, so need to keep track of units. Thus, it’s a good practice to put units in the variable name or add a comment with the unit name.

1. Can check work using subsets

ages\_in\_months[1:10]

Summarizing data:

1. Use summary() to get a summary of the data

summary(data) 🡪 *gives a 5 number (min, Q1, median, Q3, max) summary, but also includes mean; can check range or inter-quartile range for a set of new data*

1. Calculate the mean and standard deviation

mean(ages)

sd(ages)

1. Find the maximum and minimum values

min(ages)

max(ages)

1. Another useful function is the tally() function from the mosaic package

install.packages(“mosaic”) *🡪 tally() not built into base R so install mosaic package*

library(mosaic)

1. The Habit variable counts whether or not each mother was a smoker

tally(NCbirths$Habit)

* Can add formatting to tally function. To get proportions:

*formats: \* percent, \* proportion* (proportion is same as % but /100, so as a decimal)

tally(NCbirths$Habit, format = “percent”) *🡪 gives same data of habit variable but now as a %*

Visualizing Quantitative Data:

weights <- NCbirths$weight *🡪 taking the weight column from the birth data set and storing it in the weights vector*

1. Create a dot plot of weight values

dotPlot(weights) *🡪 dotplot gives another type so pay attention to capitalization)*

1. Create Histograms of Data

histogram(weights, breaks = 30) 🡪 *breaks just determines the number of bins; each break is the line between the bins*

histogram(weights, breaks = 100)

1. Create Boxplots

boxplot(NCbirths$Meduc, NCbirths$Feduc) *🡪 add 1 or 2 variables; column 1 represents Mothers’ education levels and column 2 is Fathers’ levels*

Visualizing Categorical Data:

1. View the variables in the dataset

View(NCbirths) 🡪 *this command will bring us to the spreadsheet of the data set*

1. Checking if two categorical variables are related

* Need to pick two categorical variables ex. marital status and habit, that might have some statistical relationship or are correlated.
* 2 way table- each row represents one category of the habit table and each column represents one category of the marital variable, and then can see proportions for each combination basically.

table1 <- tally(~Habit | Marital, data = NCbirths, format = “proportion”)

table 1

~ anything that has this inside of a function means it’s giving it a formula to use

| in the formula this means ‘group by’

Thus, giving tally a formula and saying take the habit variable and group it by marital. Basically, for each category in marital, count up how many of each category of habit exist in that grouping.

1. Create a barplot of categorical variables

barplot(table1, legend.text = TRUE) *🡪 setting legend.text = TRUE will show which color is which – suggests whether the variables are related in some way, if there’s a causal relationship, or completely unrelated (how strong the relationship is or whether there’s a confounding variable).*

***11th October, Wednesday:***

**Discussion 4:**

Visualizing Data- Two Quantitative Variables:

plot(NCbirths$weight ~ NCbirths$Gained) 🡪 *~ formula given is weight plotted by gain [ kind of like y = f(x; weight is a function of the variable gain)]*

Plot Attributes:

* Basically modifiers used to make the plot look nicer:
* Add these inputs to your plot() function to change the output

col- color

cex – point size

pch – plotting character (point type)

xlab, ylab – axes labels

main – title (of graph)

Ex:

plot(NCbirths$weight ~ NCbirths$Gained, col = “blue”, cex = 2)

plot(NCbirths$weight ~ NCbirths$Gained, xlab = “Gained”)

plot(NCbirths$weight ~ NCbirths$Gained, ylab = “Weight”, main = “Weight vs Gained”)

***16th October, Monday:***

**Discussion 5:**

Assignment 1- can have pictures in the pdf for the written part of it being handwritten. Assignment 1 given in the latest announcement.

Screenshots are the easiest way to get input and output for R.

For Q11, use plot to make a scatter plot and can use attributes to change size or color.

To install mosaic, either:

*install.packages(“mosaic”)*

*library(mosaic)*

Could you go over how to convert weight in ounces to pounds in the coding?

*weights\_in\_pounds <- weights / 16*

To save file, use command or ctrl S or ‘Save’ for named/ ‘Save As’ for unnamed doc on R.

Google everything about R. Ex: how to fix \_\_\_ error in ‘R’ or how to do \_\_\_ in ‘R’.

Scatter plot is plot function- plot with 2 vectors trying to compare.

In Part 5:

1. 5 number summary
2. What we can interpret from data- whether or not outliers, show whether data is skewed or not (ex: minimum is super low and rest all are pretty close together and if it has a large left tail etc.), what direction it is skewed in

***18th October, Wednesday***

**Discussion 6:**

Lab 2 is about Data cleaning and Operations.

Logical Statement and Relational Operators:   
For R like in most programming languages, these logical statements boil down to ‘TRUE or FALSE’. (in other like C++ it’s Boolean values)

Operations can be done with logical values:

Ex: TRUE + TRUE

Output: > TRUE + TRUE   
[1] 2

Ex: TRUE + FALSE   
Output: > TRUE + FALSE   
[1] 1

Ex: (3\*TRUE) + (FALSE/4)

Output: > (3\*TRUE) + (FALSE/4)

[1] 3

🡪 The value for TRUE and FALSE are treated as binary values; TRUE = 1 and FALSE = 0. Take these 0s and 1s and convert it to a certain class as well.

Relational Operators:   
For checking the relationship between two objects. If the statement is true, then gives a true in output and same for false. The double == is the operator for checking equality. For not equal operator, != is used.

Ex 1: 4 > 3

Output: > 4 > 3

[1] TRUE

Ex 2: 1.4 > 3

Output: 1.4 > 3

[1] FALSE

Ex 3: c(1,2,3,4) >= 4

Output: > c(1,2,3,4) >= 4

[1] FALSE FALSE FALSE TRUE

Ex 4: 3 < c(3,2,1)

Output: > 3 < c(3,2,1)

[1] FALSE FALSE FALSE

Ex 5: 3 <= 3

Output: > 3 <= 3

[1] TRUE  
Ex 6: 3 ==2

Output: > 3 ==2

[1] FALSE

Ex 7: 3 != c(3,2,3)

Output: > 3 != c(3,2,3)

[1] FALSE TRUE FALSE 🡪 checks element by element

For equality:

a = 3 🡪 for creating a variable, but confusing to use since the best practice in R is to use <-

a <- 3 🡪 best way

Applications of Relational Operators:

🡪 Suppose we’re checking in the NCbirths chart ‘How many babies weigh over 100 oz’

For each baby in the data set, it’s either true or false- either over 100 oz or not.

NCbirths$weight > 100

sum(NCbirths$weight > 100)

* Will add all trues and the output will be [1] 1643

sum(NCbirths$weight < 100)

* Will add all falses and the output will be [1] 322

sum(NCbirths$weight <= 100)

* Output will be [1] 349

If you add all outputs will give the total number of values in the data set.

🡪 ‘How many babies were assigned Female at birth’.

sum(NCbirths$Gender == “Female”)

Output: [1] 957 (make sure to check for spellings properly)

🡪 ‘What percentage of babies weigh over 100 oz?’

mean(NCbirths$weight > 100)

[1] 0.8247992

* To check the proportion of babies that meet the criteria.

***23rd October, Monday***

**DISCUSSION 7:**

Subsetting

If you have a data frame in R or a matrix/vector and you only want to consider parts of that data frame or vector that satisfy a logical condition, can use this.

NCbirths$weight[1:10] *🡪 running elements 1 through 10*

Output: [1] 124 177 107 144 117 98 147 138 104 123

fem\_weights <- NCbirths$weight[NCbirths$Gender == “Female”]

length(fem\_weights)

Output: [1] 957

*🡪 Taking the weight vector, and selecting the elements where this statement inside the square bracket is true. So basically, looking at the weight column in the data set and only want the weight of the babies where their gender is set to female, and then further storing it in variable fem\_weights. Can confirm the number of elements through the length function which gives the number of elements in a vector.* Format: variable\_vector[logical condition]

Data Cleaning & Preparation:

6th November, Monday

Discussion

Lab 3: Linear Regression & Sampling

Importing Data

NCbirths <- read.csv(“births.csv)

soil <- read.table(“soil\_complete.txt”, header = TRUE)

ice <- read.csv(“sea\_ice.csv”)

Linear Regression

- Use the lm() function to create a linear regression model

- List is another way to store information in R

linear\_model <- lm(weight ~ Mage, data = NCbirths)

summary(linear\_model) 🡪 to summarize

- Use abline() to add a line to a plot

- Put your linear model inside abline() to plot a regression line

plot(weight ~ Mage, data = NCbirths,

xlab = “Mother’s Age (yeats)”,

ylab = “Weight (oz.)”,

main = “Regression of Babies’ Weights on Mothers’ Ages”)

abline(linear\_model, col = “red”, lw = 2)

plot(linear\_model$residuals ~ NCbirths$Mage, main = “Residuals Plot”)

LAB:

a)

lead\_zinc\_model <- lm(lead ~ zinc, data = soil)   
summary(lead\_zinc\_model)

b)

plot(lead ~ zinc, data = soil,

xlab = “Zinc Concentration (ppm)“,

ylab = “Lead Concentration (ppm)”,

main = “Lead vs Zinc Concentrations in Soil”)

abline(lead

c)

d)

summary(lead\_zinc\_model)

- Intercept estimate = 16.582928

- Zinc slope estimate = 0.291335

- The equation is lead = 16.582928 + 0.291335 \* zinc

e)

- Plug in new data point into the regression equation

16.582928 + 0.291335 \* 1000

- The prediction for lead concentration is 307.9179 ppm

f)

- lead\_A = 16.582928 + 0.291335 \* zinc\_A

- lead\_B = 16.582928 + 0.291335 \* zinc\_B

- lead\_A = 16.582928 + 0.291335 \*

g)

summary(lead\_zinc\_model)

- R^2 = 0.912

- This means that approximately 91.2% of the variance in lead concentrations can be explained by the zinc concentration of the soil.

h)

- Linearity: I believe that the linearity condition is met. Based on the lead vs zinc plot, the two variables have a clear linear relationship.

- Symmetry: Based on the residual plot, it seems like the residuals are scattered symmetrically across the x-axis. Thus, the symmetry condition has been reasonably met.

- Equal variance: Looking at the lead vs zinc plot, it looks like the points in the bottom left corner are more closely clustered than the points in the top right. Also, in the residual plot we can see that the residuals are close to zero on the left and more spread out on the right.

This indicates that the variance is not equal across all values of the explanatory variable.

(Exercise 2 is the same as Exercise 1)

- Sampling and Simulation

- Setting a seed

- pick any number (pseudo randomness in programming; giving procedure some input gives same output. If starting at a certain starting point then gives the )

set.seed(123)

die\_vals <- c(1,2,3,4,5,6) 🡪 giving numbers 1 through 6 and giving 2 numbers as output

- Sampling without replacement

Sample(die\_vals,1)

- With replacement

sample(die\_vals, 2, replace = TRUE)

6th November

**DISCUSSION:**

All about Lab 3- linear regression (ex 1& 2), probability& sampling

Importing Data: (from lab 3)

NCbirths <- read.csv(“births.csv”)

soil <- read.table(“soil\_complete.txt”, header = TRUE)

ice <- read.csv(“sea\_ice.csv”)

(for second one .txt instead of .csv for table; header = TRUE means that the first row of the data set contains the header- names of variables)

Linear Regression:

- By hand is super tedious to calculate slopes and intercept but in R it’s just one function – the lm function.

1. Use the lm() function to create a linear regression model. Need to give it a formula- *lm(response variable ~ explanatory variable, data = dataset name)*

linear\_model <- lm(weight ~ Mage, data = NCbirths) 🡪 *creates a list; basically another way to store data in R containing information about coefficients, residuals etc.)*

summary(linear\_model) 🡪 summary() function to look at output

2. Use abline() to add a line to a plot

- Put your linear model inside abline() to plot a regression line

plot(weight ~ Mage, data = NCbirths,

xlab = “Mother’s age (years)”,

ylab = “Weight (oz.)”,

main = “Regression of Babies’ Weights on Mothers’ Ages”)

abline(linear\_model, col = “red”, lw = 2) *🡪 If run after, then adds a the regression line to the plot*

3. Otherwise, use abline(a, b) to plot the line y = a + bx *(a and b stands for intercept and slope respectively; values give the respective line)*

plot(linear\_model$residuals ~ NCbirths$Mage, main = “Residuals Plot”)

abline(a = 0, b = 0, col = “red”, lw = 2)

*🡪 Since a and b are 0, the equation would just be 0 (y =0). In the residual plot, helps check whether the residuals are symmetric across the x-axis and whether the equal variance assumption is met (whether the residuals fan out or close in on a certain point)*

*🡪 In a list, $ gives a certain value stored under a certain name. Then plot that against ~ an explanatory variable.*

**Assignment 3:**   
Exercise 1:

*a) Use the soil data to check relationship between lead concentration and zinc concentration in the soil by making the linear model*. *(lead is response variable, zinc is explanatory variable)*

lead\_zinc\_model <- lm(lead ~ zinc, data = soil)

summary(lead\_zinc\_model)

*b) Plot the data of lead vs zinc, and then add regression line.*

plot(lead ~ zinc, data = soil,

xlab = “Zinc Concentration (ppm)”,

ylab = “Lead Concentration (ppm)”,

main = “Lead vs Zinc Concentration in Soil”)

abline(lead\_zinc\_model, col = “red”, lw = 2)

c) *Doing the residual plot with the line at 0, and then add horizontal line.*

plot(lead\_zinc\_model$residuals ~ soil$zinc,

xlab = “Zinc Concentration (ppm)”,

ylab = “Residuals”,

main = “Residuals vs Fitted Values”)

abline(0,0, col = “red”, lw = 2)

🡪 Using R for steps d) to h).

d*) Write the equation of regression line.*

summary(lead\_zinc\_model)

Looking at the output,

- Intercept estimate = 16.582928

- Zinc slope estimate = 0.291335

- The equation is lead = 16.582928 + 0.291335 \* zinc

*e) If we have a new data point of zinc concentration 1000 ppm, what would lead concentration at this point be?*

- Plug the new data point into the regression equation

16.582928 + 0.291335 \* 1000 (run this)

- the prediction for/expected lead concentration at this new data point is 307.9179 ppm. (ss output)

*f) There are data points at 2 locations A and B. Location A has a zinc level that is 100ppm higher than location B. Based on this, how much higher would we expect the lead concentration to be in location A compared to location B.*

- Starting with the equations that we got from the regression, and just putting it in terms of variables ( lead\_A, lead\_B, zinc\_A and zinc\_B). (we know that zinc A is 100 more than zinc B)

lead\_A = 16.582928 + 0.291335 \* zinc\_A

lead\_B = 16.582928 + 0.291335 \* zinc\_B

lead\_A = 16.582928 + 0.291335 \* (zinc\_B + 100)

lead\_B = 16.582928 + 0.291335 \* zinc\_B

lead\_A – lead\_B = 16.582928 + 0.291335 \* (zinc\_B + 100) – (16.582928 + 0.291335 \* zinc\_B)

lead\_A – lead\_B = 0.291335 \* (zinc\_B + 100) - 0.291335 \* zinc\_B

lead\_A – lead\_B = 0.291335 \* (100)

lead\_A – lead\_B = 29.1335

🡪 We expect the lead concentration at site A to be 29.1335 ppm higher than the concentration at site B.

*g) R-squared and its interpretation? (R-squared is the proportion of variance explained by the explanatory variable)*

summary(lead\_zinc\_model)

- R^2 = 0.912

- This means that approximately 91.2% of the variance in lead concentration can be explained by the zinc concentration of the soil.

h) *About linear regression assumptions- linearity, symmetry, and equal variance. (can interpret and use the graphs to answer all these)*

- Linearity: I believe that the linearity condition is met. Based on the lead vs zinc plot, the two variables have a clear linear relationship. (All the data points are in a line, there’s no curve or wave like pattern)

- Symmetry: Based on the residual plot, it seems like the residuals are scatters symmetrically across the x-axis. Thus, the symmetry condition has been reasonably met. (to check whether the positive and negative residuals line up with one another; there’s more positive than negative residuals since its range is around 0-100, but its overall reasonably symmetry)

- Equal Variance: Looking at the lead vs zinc plot, it looks like the points in the bottom left corner are more closely clustered than the points in the top right. Also, in the residual plot we can see that the residuals are close to zero on the left and more spread out on the right. This indicates that the variance is not equal across all values of the explanatory variable. (when the data is more spread out, it indicates a higher variance). Thus the equal variance assumption is not met (since the residuals fan out over the plot, indicating that the variance changes over the values of zinc)

Skipping Exercise 2- exactly same as Exercise 1

Sampling and Simulation

- R has a lot of really good random number generators. You can generate things from different probability distributions and there’s lots of functions that allow random sampling and to run statistical simulations.

*1. Setting a seed*

- When you randomly generate numbers on a computer, it’s not actually truly random – the numbers are generated by some procedure, but it’s a super complicated functions with lots of inner workings that make the output essentially random (which cannot be predicted by a human). This is called pseudo randomness in programming.

FINISH

8th November

Nothing in discussion

*15th November:*

Lab 4 & Assignment 4

Finding Probabilities (Given Values)

In R, every probability distribution has this format which is p followed by the name of the distribution ex: pbinom(). For normal distribution it is pnorm(), exponential distribution is pexp() etc. ‘P’ stands for probability, so basically p and then name of distribution to find a probability.

Further, give pnorm() a value and a mean and standard deviation, which then returns a tail probability; the probability that the random variable is less than or equal to the value that you give it. For finding the probability that the random variable is greater than or equal to the value that you give it, use the rule of complements in probability- if the pnorm() function gives the probability that it’s less than or equal to ex: 75, the complement of that is that it would be greater than or equal to. So 1 – pnorm() would give the answer.

*Exercise 2 in Lab :*

🡪 Adult male height follows a normal distribution with mean = 69 and sd = 2.8

a) What proportion of males are less than 65 inches tall? In other words, what is P(X < 65)?

Command:

pnorm(65, mean = 69, sd = 2.8) *🡪 pnorm will give the left tail probability*

Output:

> pnorm(65, mean = 69, sd = 2.8)

[1] 0.07656373 *🡪 thus, approx. 7.6% are less than 65 inches tall*

b) What proportion of males are more than 75 inches tall? In other words, what is P(X > 75)?

Command:

1 - pnorm(75, mean = 69, sd = 2.8)

Output:

> 1 - pnorm(75, mean = 69, sd = 2.8)

[1] 0.01606229  *🡪 thus, approx. 1.6% are more than 75 inches tall*

🡪 ANOTHER METHOD:   
Can use a setting in the pnorm function called lower.tail. By default lower.tail is set to TRUE and that’s why it’s giving us the lower tail probability, the probability that it’s less than or equal to. If lower.tail is set to FALSE, it will use the upper tail so that will give the probability that it’s greater than or equal to, instead of less than or equal to. This will give exact same values as previous method.

Command:   
pnorm(75, mean = 69, sd = 2.8, lower.tail = FALSE)

Output:

> pnorm(75, mean = 69, sd = 2.8, lower.tail = FALSE)

[1] 0.01606229

c) What proportion of males are between 66 and 72 inches tall? In other words, what is P(66 < X < 72)

- For this can’t give pnorm() different values, but think of overlapping area. So take the pnorm() with the higher value and subtract the pnorm() with the lower value.

Command:

pnorm(72, mean = 69, sd = 2.8) – pnorm(66, mean = 69, sd = 2.8)

Output:

> pnorm(72, mean = 69, sd = 2.8) – pnorm(66, mean = 69, sd = 2.8)

[1] 0.7160232 🡪 *thus, approx. 71.6% of males are between 66 and 72 inches tall*

20th November:

Finding Value (Given Probability)

This time, we’re given the mean and standard deviation, but instead of being given a measurement X and asked to find the probability, we’re given a probability and asked to find the corresponding X value.

The qnorm() function aims to do the opposite of pnorm(), ‘q’ stands for quantity (quantile actually in R) since we’re finding the required measurement.

*Exercise 3 in Lab:*

🡪 Adult male height follows a normal distribution with mean = 69 and sd = 2.8

a) How tall must a male be in order to be among the shortest 0.5% of males?

(Left/lower tail- want to find a value where everything to the left of that, the area under the curve, is 0.5% converted to decimal 0.005)

Command:

qnorm(0.005, mean = 69, sd = 2.8)

Output:

> qnorm(0.005, mean = 69, sd = 2.8)

[1] 61.78768 (inches) *🡪 approx. 5’1.8 or shorter*

b) How tall must a male be in order to be among the tallest 0.25% of males?

(Right/upper tail- need to switch it to the upper tail using the rule of complements for probability. Can think of 0.25% or 0.0025 as the probability that a male is taller than this value that we’re trying to find. The other way to say that is 1 – 0.0025 probability that the male is shorter than the value we’re trying to find. So instead of right tail, we’re looking at everything to the left of it.

Command:

qnorm(1 – 0.0025, mean = 69, sd = 2.8)

Output :

> qnorm(1 – 0.0025, mean = 69, sd = 2.8)

[1] 76.85969

🡪 Basically can think of qnorm() as always looking at the left tail, so if we get something where we’re looking at the right tail, then just need to do 1 minus the probability. Also 1 – goes inside the brackets in qnorm() unlike pnorm() where it goes outside the function, since the input here is the probability so we use the rule of complements here whereas for pnorm() the output is the probability so apply the rule there.

4th Dec:

Assignment 5:

Hypothesis Test in R,

Ex 1

a) (no direction of changes less or greater than)

IN R:   
#H0: p = 0.48

#Ha: p =/= 0.48

n <- 3625

p\_null <- 0.48

p\_hat <- 1830/n

SE <- sqrt(p\_null\*(1-p\_null)/n)

z\_star <- (p\_hat – p\_null)/ SE

p\_val <- 2\*(1 – pnorm(z\_star))

p\_val <= 0.05

#Based on the data, we have sufficient evidence to reject the null hypothesis since the population proportion is not equal to 0.48

b) Confidence Interval: p\_hat +- (z\*)\* SE (standard error)

(Can disregard the pnull. We can estimate SE using p-hat)

SE2 <- sqrt(p\_hat\*(1-p\_hat)/n)

z\_star <- 1.96

CI <- c(p\_hat – z\_star \* SE2, p\_hat + z\_star \*SE2)

IN project:   
For ex variables like story (or n\_baths), fireplaces need to be discrete variables since can’t use decimals to describe them.

Distance is continuous and are is distance x distance so continuous, thus sq.ft is continuous .

(it’s continuous if it contains decimals)

data <- read.csv(“~Downloads/data.csv”)

rowSums(is.na(data)) > 0

data$ID[rowSums(is.na(data)) > 0]

View(data)

data <- na.omit(data)

🡪 3. Price comparison does the presence of a fireplace make a difference

histogram(~ TotalPrice/1e6) | Fireplace, data = data, layout = c(1,2), breaks = 20)

with\_fire <- data$TotalPrice[data$Fireplace == TRUE]

without\_fire <- data$TotalPrice[data$Fireplace == FALSE]

mean(with\_fire)

mean(without\_fire)

plot(with\_fire)

points(without\_fire, add = TRUE, col = “red”)

6th December, Wednesday:

Final practice problems

🡪 Have a Z-table open in your notes during the final!!!

Project Disc section:

- Why did you plot these things?

- What was the goal in doing this analysis?

- Go over interpretation of output (results) from code

- R code should be in an appendix at the end. Use the code to answer questions in the report for graphs, tables etc which can be put in the actual report. But the code itself like inputs should be at the end after the report. This is so that focus can be more on explanations and visualizations.

- Main report should be 4-10 pages without the code at the end, but this includes graph, tables etc.

Variables:

ID, Zipcode – categorical variables

N\_baths, YearBuilt – discrete numerical

- can look at these and see what kind of relationships you expect there to be between these variables.

In the project should have 5 tasks:

- 1) Detecting and handling missing data

- 2) Summarizing variables is in Lab 1. This is using functions like summary, making histograms or bar charts

- 3) Comparing variables is looking at a scatterplot between two variables or using the table function to look at the proportion of 2 categorical variables. (as done in lab 2- la schools, housing prices)

- 4) Identifying continuous variables, looking at relationships between them and graphing them (similar to lab 3)

- 5) Linear Regression (in lab 4)

🡪 Make sure all graphs, charts, tables are labelled and give captions as well (don’t have to but that’s the standard for most academic projects)

🡪 When writing code, organize it by section with headers and sub-headers, and write comments as you code – for ex. if plotting x and y:

## Looking at the scatterplot for x and y

plot (x,y)

## Linear relationship