# Adding Change

# R CHEAT SHEETS LINKS

Master list of all cheat sheets you could ever want: <https://www.rstudio.com/resources/cheatsheets/>

The most useful to start with is probably <https://www.rstudio.com/wp-content/uploads/2016/10/r-cheat-sheet-3.pdf>

# GENERAL R FUNCTIONS AND FILE MANAGEMENT

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Brief description | Command | Details/Comments |
| Open, Close and Directories |  |  |  |
|  | Quit R | quit( ) |  |
|  | Install package | Install.packages(“package\_name”)  If you want to use the package, you have to give the following command:  Library(“package\_name”) | DON’T FORGET to put “ “ around package name, or it will give you error messages |
|  | Update all installed packages | update.packages() |  |
|  | Getting help | Help(function)  e.g. help(plot)  OR  ?function  e.g. ?plot  OR  Apropos(“function”)  Lists all functions with string “function” in the name  OR  Example(function)  Gets example of using the function |  |
|  | See current working directory | Getwd() |  |
|  | List contents of directory | Dir(“directory\_path”)  Dir() this will list contents of current directory  List.files() this will do the same thing as dir | Whenever you want to find a path or file, use TAB COMPLETION between the |
|  | Make a new directory | Dir.create(“directory\_name”)  Will create new directory in the current working directory |  |
|  | Remove a directory | Unlink(“directory\_name”, recursive = TRUE)  This will remove a directory and everything in it | DANGEROUS, this won’t ask you any questions, it’ll just delete it |
|  | Change working directory | Setwd(“directory\_path” ) | The path has to be in “quotes”. You can use “..” to indicate one directory above |
| Working with Files |  |  |  |
|  | Import a data file, like a .csv file | mydata <- read.table( “path\mydata.csv” , header = TRUE, sep = “,” , row.names = “id”)  OR  mydaya <- read.csv(“path\datafile.csv”)  You can read in other types of files if you indicate a different separator, for instance, for a tab separated file you would do:  mydata <- read.table( “path\mydata.tsv” , header = TRUE, sep = “\t” , row.names = “id”) | **DON’T FORGET** to use the header and sep arguments and to include the .csv or it won’t read the file in correctly |
|  | Import a .csv file, if you know it’s a .csv file | mydata <- read.csv( “path\mydata.csv” , header = TRUE, row.names = “id”) | Here, you don’t have to specify the separator, cause it assumes comma separated |
|  | Import a stata file | Library(foreign)  mydata <- read.dta( “mydata.dta” ) |  |
|  | Export to a csv file | write.table ( mydata, “path\mydata.csv”, sep = “ ,“ )  OR  To write to a tab separated file  write.table ( mydata, “path\mydata.csv”, sep = “ \t“ ) | **BE VERY CAREFUL!**  **DON’T FORGET** to use the header and sep arguments and to include the .csv or it won’t write the file correctly and might overwrite your old file and mess it up.  **BEST NEVER TO WRITE FROM R TO A FILE THAT ALREADY EXISTS AS A CSV FILE** |
|  | Export to a stata file | Library(foreign)  write.dta(mydata, “path/mydata.dta” ) |  |
|  | Make a new file | file.create(“filename”)  e.g.  file.create(“myfile.R”)  this will create a file that R will recognize as an R file |  |
|  | Check whether a file exists | File.exists(“filename”) | Useful for programming to make sure a file exists before the program uses it |
|  | Rename a file | File.rename (“old\_name”, “new\_name”) |  |
|  | Get info on a file | File.info(“filename”) |  |
|  | Run command current line | cmd-return |  |
|  | List the objects in the current workspace | Ls( )  OR  Objects(\_) | Ls lists all the objects in the directory, as opposed to list.files or dir, which only list files and other directories |
|  | Display recent commands | History( )  This will display last 25 commands  OR  History( max.show = inf)  This will display all previous commands |  |
|  | Run a specific script from a file | source(“myfilename”)  If no filename or path is specified, will look in current directory |  |
|  | Direct output to a specific location/file | Sink(“myfilename”, append=FALSE, split=FALSE)  Sink( ) This latter command returns output to terminal |  |
|  | Direct graphic output to a location/file | Jpg(“myfilename.jpg”)  This will output the graph to a jpg file  Dev.off( )  The tells R to redirect output back to terminal | Instead of jpg, can also say:  pdf (“myfile.pdf”)  bmp (“myfile.bmp”)  png (“myfile.png”)  postscript (“myfile.ps”) |
| Saving output/Loading output |  |  |  |
|  | Save Load dataset | save(object1\_tobesaved, object2\_tobesaved, …, file = ‘ filename’)  load( ‘filename’)  e,g.  save(staff.data, file=’staffdata.RData’  load(‘staffdata.RData’) | In this case, we are saving an object that happens to be a data frame and loading it. You can also save other objects such as lists or vectors, all in the same file, you just separate each object from the next using a comma |
|  | Attaching data frame to avoid having to preface each query with the data frame name | Attach(frame\_name)  e.g.  attach(staff.data)  staff.data[age > 30, ]  name height age  1 howie short 51  2 yann tall 40  e.g.  height  [1] short tall tall  Levels: short tall  detach(frame\_name) |  |
|  | Save all objects in current working directory | Save.image()  This saves all objects in the current working directory. These objects will automatically be loaded when you open R in this directory | Saves objects in a file called .RData |
|  | Saving your recent commands | savehistory ( file = “myfilename”)  The default is to save in a file called .Rhistory |  |
|  | Saving your recent commands | loadhistory ( file = “myfilename”)  The default is to save in a file called .Rhistory |  |
| Writing Functions |  |  |  |
|  | Format of a function | Function\_name <- function(arguments) {  Operations  return output  } | Arguments are usually values that are given to the function, which could be numbers, variables with numbers in them, and paramters |
|  | Saving output of a function | Output\_object <- function\_name(arguments)  Key is that you have to remember that the objects you make in a function, or modifications to objects that you make, only exist within the function unless you save the output of the function as an objects  e.g.  frame.with.new.column <- function\_that\_adds\_column(frame.without.new.column) |  |
|  | Saving a function in R studio | Write it in the top left window  Check source on save  Hit save |  |
|  | To write a function that will take a column name as an argument | Use double square brackets [[ ]] to refer to the column in the function, instead of the $  Have to pass the column name as argument in quotes when you call the function  e.g.  function\_name <- function(data.frame.name, “column\_name”) {  result\_variable <- function( data.frame.name [[column\_name]][element\_number]  }  e.g.  makemean <- function(frame.name, variable)  average <- mean(data.frame[[variable]])  }  if you call this function using the variable age, this function will take the mean of the column labeled age and store it in the object called average. Like this: makemean(frame.name, “age”)  OR  e.g.  getsecondvalue <- function(frame.name, variable)  second <- frame.name[[variable]][2]  }  if you call this function using the variable age, this function will get the second observation in the column labeled age and store it in the object called second. Like this: getsecondvalue(frame.name, “age”) |  |
|  | Control functions | If-else  if (cond) expression if (cond) expression1 else expresssion2  OR  for (var in seq) expresssion  OR  While (cond) expression  OR  Ifelse (test, yes, no) |  |
|  | Logical Operations | & (and, specifies conjunction, e.g. if number meets condition 1 and condition 2)  | (or, specifies dysjunction, e.g. if number meets condition 1 or condition 2)  == (for identifying values for classifying, this is like saying  “is equal to”, and evaluates whether both sides of the operator are the same)  = (for equations)  != (is not equal to)  && and || will only evaluate the first expression on the one side of the operand:  e.g.  TRUE & c(TRUE, FALSE, FALSE)  TRUE FALSE FALSE  TRUE && c(TRUE, FALSE, FALSE)  TRUE |  |
|  | My first code for a function. | # This takes a data frame as an argument (e.g. test.frame),  # adds a visit counter variable that is represented by outvar  # and counts visits for each participant and puts the visit numbers in outvar  # it takes a string for outvar, like "visitnumber" and takes a string name to  # tell it the name of the variable to use to identify subjects, like "PIDN"  counter <- function(x.frame, outvar, idvar) {  i <- 1  x.frame[[outvar]] <- 1  viscountr <- 2  while(i < length(x.frame[[idvar]])) {  i <- i + 1  if(x.frame[[idvar]][i] == x.frame[[idvar]][i-1]) {  x.frame[[outvar]][i] <- viscountr  } else x.frame[[outvar]][i] <- 1  if(x.frame[[idvar]][i] == x.frame[[idvar]][i-1]) {  viscountr <- viscountr+1  } else viscountr <-2  }  x.frame  }  To call this and save the new data frame with the new visit column, you would do:  New.frame <- counter(old.frame, “visitvar”, “idvariable”) | Has a lot of good functionality that I learned. Uses the double bracket approach to refer to columns in the data frame |

# WORKING WITH LISTS

|  |  |  |  |
| --- | --- | --- | --- |
| Working with lists |  |  |  |
|  | Creating a list | List\_name <- list(object1\_name = value, object2\_name = value, object3\_name = value, …)  e.g.  howie.lst <- list(stud.id = 3344, stud.name = “Howie”, stud.scores = c(99, 73, 95, 85)  howie.lst  $stud.id  [1] 3344  $stud.name  [1] “Howie”  $stud.scores  [1] 99 73 95 85 | A list contains several objects. The objects don’t have to have the same length or mode |
|  | Extract value of an element from a list | List\_name[[list\_item\_number]]  howie.lst[[3]]  $stud.scores  [1] 99 73 95 85  OR  List\_name$object\_name  e.g.  howie.lst$stud.scores  [1] 99 73 95 85 | If you just use single brackets, you will just create a new list based on the object you have queried (in this case, the third object). If you want the value of an element, you have to use the double brackets |
|  | Naming objects in a list | Names(List\_name) <- c(“name1”, “name2”, “name3”, etc.)  e.g.  names(howie.lst <- c(“id”, “name”, “scores”)  Howie.lst  $id  [1] 3344  $name  [1] “Howie”  $scores  [1] 99 73 95 85 |  |
|  | Extending a list (adding new objects) | List\_name$object\_name <- c (value1, value2, etc.)  e.g.  Howie.lst$parents <- (“dora”, “seymour”)  Howie.lst  $id  [1] 3344  $name  [1] “Howie”  $scores  [1] 99 73 95 85  $parents  [1] “dora” “Seymour” |  |
|  | Concatenating lists | New\_list <- c(list1, list2)  e.g.  list2 <- c(age = 51, sex = “man”)  new.lst <- c(Howie.lst, list2)  new.lst  $id  [1] 3344  $name  [1] “Howie”  $scores  [1] 99 73 95 85  $parents  [1] “dora” “Seymour”  $age  [1] 51  $sex  [1] “man” |  |
|  | Unflatten list | Unlist(listname) | All list elements will become elements in a single vector, and be forced to have the same mode (e.g. character, numeric…), and each element will have a name |

# GENERATING DATA

## Objects and Vectors

|  |  |  |  |
| --- | --- | --- | --- |
| Generating Objects and Vectors |  |  |  |
|  | Storing an object | Object\_name <- data to be stored  e.g. x <- 100  This will store the number 100 in the object named x  Typing the name of the object will show the contents  e.g.  x  [1] 100 | Object names can contain numbers, letters and periods  Object names are case sensitive  Each object is by definition a vector  Objects can have numbers, logical values (e.g. true false), characters, complex numbers  Character data must be surrounded by quotes |
|  | Remove object from memory | rm(object\_name) |  |
|  | Combine arguments into a vector | C(element1, element2, element3, …)  e.g. x <- c(1, 9, 3, 5, 9)  x  [1] 1 9 3 5 9  This puts all those numbers in the object x. Typing the name of the object will tell R to list the elements in the vector, starting with the first one, which is what the [1] means | Need commas between elements going into the object  All elements in the vector must have the same mode  Vector can include NA, for missing data |
|  | Cut a continuous variable into a factor variable | equal.count(dataset\_name$variable, number=#, overlap = #/#)  e.g.  agegroups <- equal.count(my.data$age, number =4)  This command will take the age variable and cut it into 4 equally sized groups and save the group designation as a nominal factor variable with 4 levels called agegroups |  |
|  | Paste together or concatenate strings | paste(“string1”, “string2”, “string3”, …, sep = “ “, collapse = NULL)  sep tells the function what character to use to separate the strings |  |
|  | place value in a particular element location in a vector | Object\_name[element\_number] <- value  e.g. x[7] <- “great”  This will put the word “great” in the 7th element in the vector called x |  |
|  | Making factors | factor(object\_name)  e.g. x <- c(“m”, “m”, “m”, “f”, “f”)  x <- factor(x)  x  [1] m m m f f  level: f m  you can do this more efficiently like this:  x <- factor(c(“m”, “m”, “m”, “f”, “f”)  if your vector only happens to have one value, but you want to preserve the ability to have multiple levels, you have to specify that  x <- factor(c(“m”, “m”, “m”), levels = c(“f”, “m”))  x  [1] m m m  levels: f m |  |
|  | Generating names for elements | Names(object\_name) <- (“element\_name1” “element\_name2, “element\_name3”, …)  e.g.  x <- c(15 13)  names(x) <- c(“eli”, “jonah”) | Can also use this to rename all the variables in a dataset, but you must enter names for every variable |
|  | Generating labels for values in a vector | Mydata$v1 <- factor(mydata$v1,levels = c(1, 2, 3), labels = c( “young”, “middle-age”, “old” )  OR if data are ordinal instead of factor variable  Mydata$v1 <- ordered(mydata$v1,levels = c(1, 2, 3), labels = c( “young”, “middle-age”, “old” ) |  |
|  | Vector operations | Operation(object\_name)  e.g.  v  [1] 3 5 10  If the object v has 3 elements, the command below will multiply each element by 2, and the new values will be placed in object x  x <- 2\*(v)  x  [1] 6 10 20  Doing operations where one vector is combined with another will result in the operation being done for the corresponding element location in the two vectors  e.g.  v  [1] 3 5 10  y  [1] 5 10 10  z <- v+y  z  [1] 8 15 20 | When doing operations with 2 vectors, if they are not the same length, R “recycles” elements in the shorter vector staring with the first element |
|  | Generate numbers from a normal distribution | rnorm(number\_values, mean = #, sd = #)  e.g.  rnorm(100, mean = 50, sd = 10)  This command will create a vector with 100 random numbers from a normal distribution with a mean of 50 and standard deviation of 10 |  |
|  | Recode data into a factor variable | Datafilename$newfactorvariablename[existingcontinuousvariablename criterion] <- “factorlevelname”  This example will make a new factor variable agecat that will make levels corresponding to three age ranges  attach(mydata) mydata$agecat[age > 75] <- "Elder" mydata$agecat[age > 45 & age <= 75] <- "Middle Aged" mydata$agecat[age <= 45] <- "Young" detach(mydata) |  |
|  | Centering a variable | scale(variable\_name, center = TRUE, scale = FALSE) |  |
|  | Generating z-scores | scale(variable\_name, center = TRUE, scale = TRUE) |  |
|  |  |  |  |
|  | Changing a string to a date | object\_name <- as.date(date\_variable, format)  e.g.  strDates <- c("01/05/1965", "08/16/1975") dates <- as.Date(strDates, "%m/%d/%Y")  ALSO  If you want to calculate time between dates  XXXXX | Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates  If string is indicated by capital letter, it indicates longer/unabbreviated version, for instance %Y is YYYY, %y is yy. %A is full weekday name (like Monday) and %a is abbreviated weekday name, like Mon. |
|  | Generate a sequence | Seq()  This command can be used in several ways:  Seq( start\_number, end\_number, length = number)  e.g. seq( 0, 10, length =3)  [1] 0 5 10  this generates a sequence of numbers between 1 and 10 such that the length of the resulting vector is 3  OR  Seq( start\_number, end\_number, by = number)  e.g. seq( 0, 10, by =2)  [1] 0 2 4 6 8 10  this generates a sequence of numbers between 1 and 10 counting by 2  OR  Seq(starting\_number, ending\_number, interval)  e.g. seq( 0, 2, 0.5)  [1] 0.0 0.5 1.0 1.5 2.0 |  |
|  | Generate a sequence the same length as an object | Seq\_along(object\_name) | This is really useful if you want to generate a counting variable the same length as another variable in your data |
|  | Generate a sequence | Use the colon  e.g.  x <- 1:5  x  [1] 1 2 3 4 5 | Note that you don’t need to do c(1:5), you can just do 1:5 |
|  | Generate a repetitive sequence | rep( number, repeats)  e.g. rep( 10, 5)  [1] 10 10 10 10 10  you can also do it with sequences  e.g. rep( 1:3, 3)  [1] 1 2 3 1 2 3 1 2 3  OR you can have each number in the sequence repeated  e.g. rep( 1:3, each = 3)  [1] 1 1 1 2 2 2 3 3 3 |  |
|  | Generate repetitive values within factors | gl( number of levels, repeats in each level)  e.g. gl( 3, 4)  [1] 1 1 1 1 2 2 2 2 3 3 3 3  levels: 1 2 3  another example with character data  e.g. gl( 2 , 3, labels = c(“m”, “f”))  [1] m m m f f f  levels: m f |  |
|  | Generate random numbers | rfunction( number\_of\_data, parameters)  e.g. rnorm(5)  [1] 1.2241399 0.8170304 -0.2244951 -0.3439073 1.2656202  this generates 5 random numbers from a normal distribution with a mean of 0 and sd of 1. You can change the parameters of the distribution you draw from  e.g. rnorm(5, mean = 10, sd = 3)  [1] 14.642944 7.436970 12.740121 9.238453 4.280766 |  |
|  | Grab a random sampling of data from a vector | Sample(“variable\_name”, number)  e.g. sample(my.data$height, 4)  this command will take 4 random values from the variable height  You specify the number of elements you want to grab  OR  e.g.  Sample(1:6, 4, replace = TRUE)  This will grab 4 random values from the sequence 1:6. The replace=TRUE argument says you can grab the same observation/number more than once  OR  e.g.  Sample(my.data$height)  This command will randomly rearrange all the values in the variable height. It will “sample” all the values because a number to sample was not specified  e.g.  Sample(c(0, 1), 100, prob = c(0.3, 0.7), replace = TRUE)  This command will grab 100 values from the vector containing 0 and 1, with a probability of grabbing 0 being 0.3 and the probability of grabbing 1 being 0.7. In this command, you had to say replace = TRUE because you don’t have 100 observations in the vector, so you have to replace the value each time you grab it |  |

## Matrices and Data Frames

|  |  |  |  |
| --- | --- | --- | --- |
| Generating Matrices and Data Frames |  |  |  |
|  | Separate vector into matrix | dim(object\_name) <- (nu\_of\_rows, nu\_of\_columns)  e.g.  x <- c(1, 2, 3, 5, 10, 14, 19, 22)  dim(x) <- c(2, 4)  [1] 1 3 10 19  [2] 2 5 14 22 |  |
|  | Create a matrix | Object\_name <- matrix (c(element1, element2, …), nu\_rows, nu\_columns)  e.g.  x <- matrix (c(1, 2, 3, 5, 10, 14, 19, 22), 2 ,4)  x  [1] 1 3 10 19  [2] 2 5 14 22  OR  Object\_name <- matrix (c(element1, element2, …), no\_rows, no\_columns, byrow = T)  e.g.  x <- matrix (c(1, 2, 3, 5, 10, 14, 19, 22), 2, 5, byrow = T))  x  [1] 1 2 3 5  [2] 10 14 19 22 | If you make a matrix with the standard command, R will fill the column with consecutive values and go to the next column. If you want to fill a whole, then move to the next, use the byrow = T parameter. |
|  | Generating names for columns and rows in a matrix | colnames(matrix\_name) <- c(“column\_name1” “column\_name2, “column\_name3”, …)  rownames(matrix\_name) <- (“row\_name1” “row\_name2, “row\_name3”, …)  e.g.  > x <- matrix (c(1, 2, 3, 5, 10, 14, 19, 22), 2, 5, byrow = T))  > colnames(x) <- c(“visit1”, “visit2”, “visit3”, “visit4”)  > rownames(x) <- c(“wait1”, “wait2”)  > x  visit1 visit2 visit3 visit4  wait1 1 2 3 5  wait2 10 14 19 22  You can also save the column names or row names in variables and use them  e.g.  > cnames <- c(“visit1”, “visit2”, “visit3”, “visit4”)  > colnames(x) <- cnames |  |
|  | Change name of a column in a dataframe | Colnames(data.frame)[col\_number] <- “new\_name” |  |
|  | Join two vectors/matrices by columns | cbind(object\_name, object\_name)  e.g.  x  [1] 1 2 3 5  [2] 10 14 19 22  y  [1] 3 3  [2] 4 5  cbind(x[ ,4], y[ ,1])  [ , 1] [ ,2]  [1, ] 5 3  [2, ] 22 4 |  |
|  | Join two vectors/matrices by rows | rbind(object\_name1[position], object\_name2[position])  e.g.  x  [1] 1 2 3 5  [2] 10 14 19 22  y  [1] 3 3 4 9  [2] 4 5 10 20  rbind(x[ 1, ], y[2, ])  [ , 1] [ ,2] [ ,3] [ ,4]  [1, ] 1 2 3 5  [2, ] 4 5 10 20 |  |
|  | Creating a data frame | Dateframe\_name <- data.frame(object1\_name = c(value1, value2, …), object2\_name = c(value1, value2, …), object3\_name = c(value1, value2,…)  e.g.  staff.data <- data.frame(name = c(“Howie”, “yann”, “gabe”), height = c(“short”, “tall”, “tall”), age = c(51 40 24))  name height age  1 howie short 51  2 yann tall 40  3 gabe tall 24  If already created the name, height and age variables, can just do this:  Staff.data <- c(name, height, age) | A data frame is the most common way of representing data for analysis. The rows are cases, with each case being described by a series of variables (in the columns). |
|  | Adding columns to a data frame | Frame\_name$new\_variable\_name <- c(value1, value2, …)  e.g.  staff.data$howmuchhair <- c(“not much”, “a lot”, “a lot”)  staff.data  name height age howmuchhair  1 howie short 51 not much  2 yann tall 40 a lot  3 gabe tall 24 a lot | New variable must have the same number of rows as rest of the data frame (can fill in with NA if have missing data). |
|  | Adding columns to a data frame other ways | data$size <- c("small", "large", "medium")  data[["size"]] <- c("small", "large", "medium") data[,"size"] <- c("small", "large", "medium")  data$size <- 0 |  |
|  | A procedure to move a column from one location to the beginning of a dataset | e.g.  col\_idx <- grep("string", names(dataset))  dataset <- dataset[, c(col\_idx, (1:ncol(dataset))[-col\_idx])]  names(dataset) | Here, you are using grep to find the location of the columns with names containing the “string” you have specified, you save those locations to an objects, then you rewrite the dataset using data from all rows of the dataset in the columns saved in col\_idx first, and then all other columns in the dataset excluding those at the positions saved in col\_idx. |
|  | Sorting data | Newdata <- my.data [order (mpg), ]  This will sort all the data in my.data by the variable mpg and save the sorted data to a new variable, “newdata”  Newdata <- my.data [order (mpg, cyl), ]  This will sort all the data in my.data by the variable mpg and then the variable cyl and save the sorted data to a new variable, “newdata”  Newdata <- my.data [order (mpg, - cyl), ]  This will sort all the data in my.data by the variable mpg in ascending order and then the variable cyl in descending order and save the sorted data to a new variable, “newdata” |  |

# EXAMINING DATA

## Examining Datasets

|  |  |  |  |
| --- | --- | --- | --- |
| Examining Datasets |  |  |  |
|  | Import dataset | Object\_name <- read.csv(“filepath/filename”) | You are importing the dataset into an object in R, for a typical dataset it would be a dataframe. If you do this from the GUI, it will read the dataset into an object where the name of the object will be the filename |
|  | Reading a dataset | View(object\_name) |  |
|  | Attach a dataset | Attach(object\_name) |  | Even though you’ve read in and viewed the file, R commands might not work on it until you attach it |
|  | See the type of an object | Class(object\_name)  This will return date.frame if it’s a dataframe, numeric if it’s numeric, etc… |  |
|  | See the dimensions of an object | Dim(object\_name) |  |
|  | Counting columns and rows in data frame | nrow(frame\_name)  ncol(frame\_name) |  |
|  | Read beginning and ends of datasets | Head(dataset\_name)  Tail(dataset\_name)  OR  Head(dataset\_name, #rows)  Tail(dataset\_name, #rows)  To specify the number of rows to be shown (instead of the default 6) | Head will write first six lines of file to screen, tail will write last six lines of file to screen |
|  | List names of variables in dataset | names(dataset\_name) |  |
|  | List variables in dataset with info about each variable (type, initial values…) | str(dataset\_name) |  |
|  | Summarize properties of variables in dataset | summary(dataset\_name)  For numeric variables, will give mean, median, max, min and 1st and 3rd quartiles  For character variables, will give counts for each level |  |

## Examining Objects, Vectors

|  |  |  |  |
| --- | --- | --- | --- |
| Examining Objects, Vectors |  |  |  |
|  | Find length of an objects | length(object\_name)  e.g.  x  [1] 1 9 3 5 9  length(x)  [1] 5 |  |
|  | Find type of data in objects | mode(object\_name)  e.g.  x  [1] 1 9 3 5 9  mode(x)  [1] “numeric” |  |
|  | Evaluate whether two arguments are identical | Identical(argument1, argument2)  e.g.  > identical(7, 7)  TRUE |  |
|  | Evaluate two arguments and return TRUE if they are not both TRUE or both FALSE | xor(argument1, argument2)  This function will return TRUE if one argument is FALSE and one argument is TRUE. If both are TRUE, or both are FALSE, the function will return FALSE  e.g.  > xor(7>7, 3==3)  TRUE |  |
|  | Find the value for a particular element in a vector | Object\_name[element\_number]  e.g. x[7]  This will show the value of the 7th element in the vector called x  e.g.  x[length(x)  This will find the value of the last element in a vector, regardless of how long it is |  |
|  | List levels in a factor variable | Levels(dataset\_name$variable\_name) |  |
|  | Look at factor level as its integer | as.integer(dataset\_name$variable\_name) |  |
|  | Look for a particular string in a dataset | Grep(“string”, object)  e.g.  myname <- grep(“Howie”, namelist)  This would find the position of the name howie in a column of names and assign it to the variable myname | The variable myname would not have the string “Howie” in it, just the row number or column number where it is in the larger dataset |
|  | Subsetting based on name of an element | Object\_name[“element\_name”]  e.g.  > x <- c(1 = “first”, 2 =”second”, 3 =”third”)  > x  first second third  1 2 3  > x[“second”]  second  2 |  |
|  | Probe elements in a vector to get the positions of those meeting certain criteria | which(object\_name criterion)  e.g.  x <- c(1, 3, 8, 10)  which( x >7)  3, 4 |  |
|  | See if any elements in a dataset meet specific criteria | any(object\_name criterion)  e.g.  x <- c(1, 3, 8, 10)  any( x > 7)  TRUE |  |
|  | See if all elements in a dataset meet specific criteria | all(object\_name criterion)  e.g.  x <- c(1, 3, 8, 10)  all( x > 7)  FALSE |  |
|  | Subsetting, or identifying elements that meet specific criteria | Object\_name[test]  e.g.  > x <- c(1, 2, 3, 5)  > x [ x>3 ]  [1] 5  this command gave us the element in object x that was greater than 3 |  |
|  | Subsetting, or identifying elements at certain positions in a vector | Object\_name[c(position, position, …)]  e.g.  x <- c(1, 2, 3, 5)  x [ c(1,4) ]  [1] 1 5  OR  Object\_name[c(sequence)]  e.g.  x <- c(1, 2, 3, 5)  x [1:3]  [1] 1 2 3  OR  Object\_name[-position\_to\_ignore]  e.g.  x <- c(1, 2, 3, 5)  x [-3]  [1] 1 2 5  OR  Object\_name[-c(position, position, …)]  e.g.  x <- c(1, 2, 3, 5)  x [-c(2, 3)]  [1] 1 5  this last command uses the negative sign to tell R to give the elements in the object, ignoring the specified positions |  |
|  | Get descriptors for a variable | Mean(dataset\_name$variable, na.rm = T)  Median(dataset\_name$variable, na.rm = T)  Min(dataset\_name$variable, na.rm = T)  Max(dataset\_name$variable, na.rm = T)  sd(dataset\_name$variable, na.rm = T)  Using na.rm = T means you should leave NA observations out of the equation |  |

## Examining Matrices/Data Frames

|  |  |  |  |
| --- | --- | --- | --- |
| Examining Matrices and Data Frames |  |  |  |
|  | Get value of element at specific position in a matrix | Object\_name[row\_position, column\_position]  e.g.  x <- matrix (c(1, 2, 3, 5, 10, 14, 19, 22), 2, 4, byrow = T))  x[2,3]  [1] 19  OR to get a whole row  x[2, ]  [1] 10 14 19 22  OR to get a whole column  x[ , 4]  [1] 5 22 | You can use all the same commands described above for extracting variables from vectors in matrices |
|  | Get values from a matrix and keep the result as a matrix instead of a vector | Object\_name[row\_position, column\_position, drop = F ]  e.g.  x <- matrix (c(1, 2, 3, 5, 10, 14, 19, 22), 2, 4, byrow = T))  x[2, , drop = F]  [ , 1] [ ,2] [ ,3] [ ,4]  [1, ] 10 14 19 22 | If you do this command and do not including the drop =F parameter, R would just make a vector, not a matrix because the result would have just one row of values |
|  | Subsetting a data frame | Subset(frame\_name, condition)  e.g  Subset(staff.data, age > 30 )  name height age  1 howie short 51  2 yann tall 40  Can combine multiple conditions for subset  Subset(frame\_name, variable1==condition | variable1==condition)  e.g.  subset(patient\_data, dx==”AD” | dx==”FTD”)  This will get all the column data for the rows where diagnosis is AD or where diagnosis if FTD |  |
|  | Extract value of an element in a data frame | Frame\_name[ row\_position, column\_position]  e.g.  staff.data[3, 2]  [1] tall  levels: short tall  OR  To list all the names of a particular variable in a data frame:  frame\_name$object\_name  e.g.  staff.data$height  [1] “short” “tall” “tall” |  |
|  | Querying data frame using conditions | Frame\_name[frame\_name$variable\_name condition]  e.g.  staff.data[staff.data$height == “tall”]  name height age  2 yann tall 40  3 gabe tall 24  Another example  e.g.  staff.data[staff.data$age > 30, ]  name height age  1 howie short 51  2 yann tall 40  Another example  e.g.  staff.data[staff.data$height == “tall”, “name”]  [1] “yann” “gabe”  Another example  e.g.  staff.data[staff.data$height == “tall”, c(“name”, “age”]  name age  2 yann 40  3 gabe 24 |  |
|  | Get descriptive statistics for each object in a list or data frame | lapply(dataset\_name, function, na.rm = TRUE)  e.g.  means <- lapply(my.data, mean, na.rm = TRUE)  This command will try to take the mean of every column in the dataset my.data, and save those means in a new list  Other functions include s**d, var, min, max, median, range, and quantile, or any other function**  **e.g.**  **class\_list <-** lapply(my.data, class)  This command will assess the class of each column in the my.data dataset and pass the class values to a list called class\_list |  |
|  | Iteratively get statistics for each object in a list and save the results in a vector | sapply(dataset\_name, function, na.rm = TRUE)  e.g.  means <- sapply(my.data, mean, na.rm = TRUE)  This command will try to take the mean of every column in the dataset my.data, and save those means in a vector. Because the output of mean is numbers, the vector will be numeric  **e.g.**  **class\_list <-** sapply(my.data, class)  This command will assess the class of each column in the my.data dataset and pass the class values to a vector called class\_list. Because the output of class is a string, the vector will be a character vector  sapply will also make a matrix if the data it brings back is a series of vectors  e.g.  maxmin <- sapply(my.data, range)  This will take the minimum and maximum of the numbers in each column of my.data and save it in a matrix that would be 2 rows by however many columns are in the dataset | If using sapply results in a bunch of elements of length 1, it will save them in a vector  If using sapply results in a bunch of vectors that all have multiple elements and have the same length, it will save them in a matrix.  If using sapply results in a bunch of vectors that all have multiple elements that are not the same length, it will save the results as a list, which is the same thing that lapply will do |
|  | Specify the type of output provided by apply | Vapply(dataset\_name, function, vector\_type)  e.g.  vapply(my.data, class, character(1)  This command will determine the class of each object in my.data and the output is specified as a character vector of length 1. | Using this function does the same thing as sapply, but it is ‘safer’ than sapply because it will give an error if the output is not in the form you expect. In contrast, sapply will output the result in whatever format it needs to, and you won’t know if it’s not what you expect |
|  | Get statistics for an object in a list split according to another factor vector | tapply(variable\_to\_be\_assessed, factor\_variable, function)  e.g.  tapply(my.data$age, mydata$diagnosis, mean)  this command with provide the means for the variable age at each level of diagnosis |  |
|  | Apply a function to some data according to levels of another variable | by(data, factor\_variable, function)  e.g.  by(my.data$age, my.data$diagnosis, mean)  this command with provide the means for the variable age at each level of diagnosis |  |
|  | Apply a function to some data specifying the data that goes into the function (or limiting the input) | with(data, expresssion)  e.g.  with(my.data$diagnosis == “FTD”, mean(my.data$age))  this command with provide the means for the variable age for those with a diagnosis of FTD |  |
|  | Aggregating data | Newobjects <- Aggregate(data.frame, by = list (factor\_var1, factor\_var2…, FUN = function, na.rm = TRUE)  e.g.  means <- aggregate (size.data, by = list(agegroup, sex), FUN = mean, na.rm = TRUE)  This command will aggregate all the numeric data in the dataset called size.data and produce means for each numeric variable grouped according to the agegroup and sex variable (4 means per numeric variable) |  |
|  | Find observations with missing data in a variable of interest | is.na(dataset\_name$variable)  or can find lines where the variable of interest is not missing  !is.na(dataset\_name$variable) |  |
|  | Omit observations with missing data | na.omit(dataset\_name$variable) |  |
|  | Identify rows with NA values in ANY of the variables in the row | Complete.cases(dataset\_name)  e.g.  my.data[complete.cases(my.data), ]  This command will extract the rows that have complete data in the my.data dataset  e.g.  my.data[ ! complete.cases(my.data), ]  This command will extract the rows that DO NOT have complete data in the my.data dataset | The complete cases command returns a vector with one line for each line in the data frame, and the value TRUE for those lines that are complete |
|  | Remove rows from a data frame | Dataset\_name <- dataset\_name [ -c(row#, row#...), ] |  |
|  | Correlations of variables in dataset | Cor(dataset [row, column], use = “complete.obs”)  e.g.  cor (my.data [ , 1:5], use = “complete.obs”)  OR  Symnum(Cor(dataset [row, column], use = “complete.obs”))  This second command will produce a symbolic version of the | The “complete.obs” tells R to disregard observations with NA. The command will produce a correlation matrix between all indicated variables |
|  | Make tables for factor data | table(Object\_name)  e.g.  table(x)  x  f m  3 2  you can also use table to cross-tabulate multiple factors for the same cases. For instance if you also have a second factor with age categories for the same people references in x, you can show sex and age in a table  e.g.  y <- factor(c(“old” “young” “young” “young” “old”))  table(x,y)    f m  old 1 1  young 1 2 |  |
|  | Getting frequencies from cross-tabulated data | First put the table in an object  e.g.  t <- table(x,y)  Then, run  prop.table(table\_object\_name, dimension\_number)  e.g.  prop.table (t, 1)  f m  old .5 .33  young .5 .66  this gives you the proportions of other levels within each level of the first dimension in the table, which is sex, or x, and says that half the females are old and half are young, and 2/3 of the males are young and 1/3 are old  OR  prop.table (t, 1)  f m  old .5 .5  young .66 .33  this gives you the proportions of other levels within each level of the second dimension in the table, which is age, or y, and says that half the old people are male and half female, and 2/3 of the young people are male and 1/3 are female  OR  e.g.  prop.table (t)  f m  old .2 .2  young .2 .4  this gives you the proportions with respect to all categories, and says that 20% of all the people are old females, 20% are young females, 20% are old males and 40% are young males  OR  You can run the prop.table command with the table command:  e.g.  prop.table(table(x,y), 1)  prop.table(table(x,y), 2) |  |
|  | Getting percentages from cross-tabulated data | prop.table(table\_object\_name, dimension\_number)\*100  prop.table (t)\*100  f m  old 20 20  young 20 40 |  |
|  | Getting marginal counts from cross-tabulated data | First put the table in an object  e.g.  t <- table(x,y)  then use the margin function  Margin.table(table\_object\_name, dimension\_number)  e.g.  margin.table (t, 1)  x  f m  2 3  margin.table (t,2)  y  old young  3 2  OR  You can run the margin.table command with the table command:  e.g.  margin.table (table(x,y), 1)  x  f m  2 3 |  |

# HYPOTHESIS TESTING

|  |  |  |  |
| --- | --- | --- | --- |
| Chi-square |  |  |  |
|  | Chi-square | Summary(table\_data)  If you run the summary command on data that is in a table, it will run a chi square  First you have to make an object with the table data in it  e.g.  demograph <- table(groupdata$DX, groupdata$gender)  summary(demograph)  OR  You can just run the summary command on the table command:  Summary(table(groupdata$DX, groupdata$gender)) |  |
| T-test |  |  |  |
|  | t-test, independent, two groups | t.test(x~y)  e.g.  t-test (mmse ~ diagnosis)  This would do t-test comparing subgroups of x according to binary factor y  OR  t-test(y1, y2)  This would do t-test comparing two numeric variables | Var.equal = TRUE option specifies equal variance and pooled estimate of variance  Alternative = “less” and alternative = “greater” options specify one-tailed t-test |
|  | t-test, paired | t.test(y1, y2, paired = TRUE)  e.g.  t-test(bpbefore, bpafter, paired = TRUE)  This would do t-test comparing two numeric variables representing repeated values from the same cases (e.g. before and after treatment) |  |
|  | t-test, one sample | t.test(y1, mu = #)  e.g.  t-test(glucose, mu=0)  This would do t-test comparing the observed glucose values to test whether the mean is significantly different from 0 |  |
| Regression |  |  |  |
|  | Run regression model | Object\_name <- lm (dep\_variable ~ indep\_variable1 + indep\_variable2 + indep\_varialble3 + …, data = dataset\_name)  e.g.  lm.weight <- lm(weight ~ height + age, data = my.data) | This type of linear model cannot handle missing data |
|  | View regression results | summary (regression\_name)  e.g.  summary(lm.weight) |  |
|  |  |  |  |
|  | Rerun regression with different predictors | Object\_name( <- update(regression\_name, . ~ new\_variable1, new\_variable2, …)  e.g.  lm2.weight <- update(lm.weight , . ~ . -age)  This command will run the prior linear model without age and save to a new objects |  |
|  | Run stepwise regression on linear model | Object\_name <- Step(model1) |  |
|  | Run anova on previously created linear model | Anova(regression\_name)  e.g.  anoval(lm.weight) |  |
|  | Compare two linear models using anova | Anova (model1, model2)  e.g.  anova(lm.weight, lm2.weight) |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# GRAPHICS

|  |  |  |  |
| --- | --- | --- | --- |
| Graphics |  |  |  |
|  | Graph labeling | Ylab = “label name”  Xlab = “label name”  e.g.  hist(clinical.data$age, ylab = “age frequencies”) |  |
|  | Histogram | Hist(dataset\_name$variable)  This will give counts for each interval  Hist(dataset\_name$variable, prob = T)  This will give probabilities for each interval instead of counts |  |
|  | Histogram using lattice package | Library(lattice)  Histogram( ~ variable | factor\_variable, data = dataset\_name)  OR  Histogram(condition ~ variable | condition, data = dataset\_name, type = “statistic”)  e.g.  hist( ~ age | gender age, data = classroom\_stats, ylab = “age”, type = “percent”, breaks = c(10, 20, 30, 40))  This second plot will do histogram based on two different factor variables. Saying type tells what statistic to use to present the data. It can also be “density” or “count”. Breaks determines the bins |  |
|  | qq plot | qqPlot(dataset\_name$variable)  Plots the data against the theoretical quantiles of a normal distribution | This is a way to assess whether data are normally distributed.  This plot is part of the “car” package, which must be installed and called |
|  | Boxplot | Boxplot(dataset\_name$variable)  If you want to do conditional plot using lattice package version, use:  Bwplot(factor\_variable~numeric\_variable) |  |
|  | Plot data generically | Plot(vara |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | ALL STATA COMMANDS BELOW HERE |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Saving output |  |  |  |
|  | Results to text | translate @Results mylog.txt | Captures everything you can see by scrolling |
|  | Opening log | log using c:\mylog.log | Starts capturing moving forward |
|  | Closing log | log close |  |
|  | Converting log to text | translate mylog.scml mylog.txt |  |
|  | Commenting | \* means ignore line, // means ignore rest of line |  |
|  | Get data out of memory | Clear | Need to do this before loading new dataset |
|  | Clear results window | cls |  |
| Data manage-ment |  |  |  |
|  | Operations | = (for equations)  == (for identifying values for classifying, this is like saying  “is equal to”)  != (is not equal to)  & (and, used for including things in evaluation) |  |
|  | Calculations with repeated measures | Use \_n to indicate row-specific operation  e.g. bys pidn: generate mmsechange = (mmsetot[\_n] – mmsetot[n-1])  This will calculate change score between mmse in a given row and the mmse in the row preceding it within each group of pidns | Have to sort by variable that ensures organized relationship between successive rows |
|  | Autofill a value to all subsequent lines of that id | xfill variable, i(id)  If you have a pidn with four rows and you want to transfer information from the first row to other rows, you can use xfill. This is useful for longitudinal data. |  |
|  | Create variable | Generate myvariablename = default value  Or  Generate myvariablename = default value, before(variablename) | Can use a “. “ after the equal sign to indicate that all values should initially be labeled as missing data  Add option before to place the column where you want it |
|  | Replace values | Replace myvariablename = new value  OR  Replace myvariablename = new value if(othervariable==value) | New value can be an operation on another variable, like log transform, square root. Can also replace with condition based on other variable using if statement, as shown to left |
|  | Create and define label | Label define mylabelname 1 “Label 1” 2 “Label 2” 3 “Label 3” etc | Create label description |
|  | List created labels and their values | Label list |  |
|  | Remove a label | Label drop mylablename |  |
|  | Apply labels | Label values myvariablename mylabelname | Apply labels to values in a variable |
|  | Do a mathematical operation and see result onscreen | Display mathematicaloperation  e.g. display log10(4.5) \* log10(3.7) | Way to do an operation and show result on the screen |
|  | Center a variable around its mean | egen meanvar1 = mean(var1)  gen cvar1 = var1 = meanvar1 | Often helps with interpretation of coefficients for continuous variables |
|  | Convert a variable to standardized units | Egen svar1 = std(var1) if var1 != . | This command will exclude observations with missing data |
|  | Cut a continuous variable into  discrete categories | Egen categoricalvariable = cut(continuousvariable), at (value1, value2, value2…)  e.g. egen agecat = cut(age), at(10, 20, 30, 40)  or  egen agecat = cut(age), group(5) | Can designate cut points or cut into X equal groups |
|  | Move repeated data between rows and columns | Reshape wide timechangingvariable1 timechangingvariable2, i(clusteringvariable) j(repeatervariable)  -undo using:  reshape long | This will take data from rows and move them into columns. Can undo with simple command |
|  | Reorder variables | Order \*, sequential  This will order variables alphabetically (and numerically if numbers are involved). | Makes it easier to refer to subsets of variables that you might want to refer to as a group |
|  | Refer to multiple variables in a command | 1. Can use a prefix that is included in several variable names  Command prefix\*  e.g. list cog\* will list values for all variables that start with the prefix ‘cog’.  2. Can specify a range, which will include the variables falling within that range in order they are recorded in dataset)  Command variableX-variableY  e.g. list bnttot-gds will list values for all variables between bnttot and gds. |  |
|  | Make a date into a useable date in Stata | Date(stringvariable, “MDY”)  Dates entered into other programs like  For example, let’s say you have a column with dates called in a variable called stringdate, and they are formatted like this: MM/DD/YYYY  e.g. generate statadate = date(stringdate, “MDY”)  will create the variable statadate, take the dates in that column, and interpret as month, date, then year, and create the proper number and place them in the corresponding location in statadate  or, you could take a certain specific date and get stata to format it properly:  e.g. date = date(“11/10/16”, “MDY”), in 3  here, you are giving it a string that it needs to interpret as a date, so you need to put the date in quotation marks | Dates entered into other programs like excel are imported into stata as strings. You need to use the date function to take that string and have stata convert it into a number, which will be written as the number of days since January 1st, 1960.  If you import an excel spreadsheet with dates in it, stata will interpret and import it as a date (at least with version 14). |
|  | Displaying stata dates in more readable format | format datevariable %d |  |
|  | Process to MAKE INTERVALS from a SERIES OF DATES for longitudinal data (assuming you have visits labeled in order) | First make a variable with a stata date in it  Generate datevariable date(stringvariable, “MDY”)  Then, make sure you have a variable that has the visit number for each visit for each pidn, can call it “visit”  Then sort your data by idnumber and visit number  Sort idvariable visitvariable  Then make a variable that has the baseline date. This will take two steps:  First, put the baseline date in the first row for the new variable for each subject (visit 1)  Generate basedate if (visit==1) = datevariable  Then put the baseline date in the rest of the rows for each subject by grabbing the data in the first row for each  by idvariable: replace basedate = basedate[1]  Now, you can subtract the baseline date from the visit date to get an interval, in days  Generate elapsedays = datevariable – basedate  You can convert this to years by dividing by 365  Generate elapseyrs = elapsedays/365 | You would use most of these commands to create change scores for a variable given a bunch of scores that might change over time |
|  | Process to MAKE CHANGE SCORES from a SERIES OF SCORES for longitudinal data ARRANGED IN ROWS (assuming you have visits labeled in order) | First sort your data by idnumber and visit number  Sort idvariable visitvariable  Then make a variable that has the baseline value. This will take two steps:  First, put the baseline value from a variable like testscore in the first row for the new variable for each subject (visit 1)  Generate baseval if (visit==1) = testscore  Then put the baseline value in the rest of the rows for each subject by grabbing the data in the first row for each  by idvariable: replace baseval = baseval[1]  Now, you can subtract the baseline score from the visit test score to get a change  Generate testscorechange = testscore – baseval  You can annualize this by dividing by the interval if you have that calculated (see above)  Generate annlzdtestchange = testscorechange/elapsedyrs\_variable |  |
| Data Description/review |  |  |  |
|  | Summarize whole dataset | Describe | List numbers of observations, lists variables, |
|  | Examine metadata about each variable | Codebook | For each variable, will provide range of values, frequencies of values, number missing, etc. Can use to know if variable is best as categorical, continuous, etc |
|  | List values for specific variables | List variable1 variable 2 |  |
|  | Review basis stats for specific variable | Summarize variable1, detail   * or   Summarize | Summarize with no variablename lists variables in the dataset, and basic stats for each. Option detail gives more stats |
|  | Organize table 1 variable per line | Tabstat myvariable1 myvariable2, stats(stats) columns(stats) | Default is column by variable. If you have few stats for a lot of variables, it’s better to say columns stats, will be easier to read. If you have a lot of stats for a few variables, column variables is better. |
|  | Summarize data for a categorical variable | tab myvariable  tab categoricalvariable, sum(continuousvariable)  can also use term “tabulate” instead of “tab” | Will give frequencies and percent for each value of the variable.  Option sum will produce means, sd for the designated variable acc to categories |
|  | Summarize variable statistics in a table | Tabstat myvariablename1 myvariablename1 etc., stats(mean n sd etc) | Stats is an option telling it what statistics to summarize, can use by option to split data according to another categorical variable |
|  | Examine skewness/kurtosis | Sktest variablename |  |
|  | Make formatted table splitting variable data by grouping variables | Table groupingvariable1 groupingvariable2, c(statistic variable3 statistic variable3)  e.g. table visit sex, c(mean age n age) | C indicates what data you want in the columns (c for contents) |
|  | Summarize data by categories of another ordinal variable | sort ordinalvariablename  by ordinalvariablename: summ variable  (e.g.  sort agecategory  by agecategory: sum brainvolume)  -or-  by ordinalvariablename: summ variable, detail  -or, can combine the sort and by command-  bysort ordinalvariablename: summ variable, detail | Always have to sort dataset on the ordinal variable before giving the by/summ command  Option detail provides additional variable detail (percentile, descriptions) |
|  | Sample size calculations | Sampsi mean1 mean2 p(p-value) r(ratio) sd(sd1) sd(sd2)  e.g. sampsi 0 0.878 p(0.8) r(1) sd1(5.07) sd2(5.07)  This will calculate sample size in each group when comparing two equally sized groups where one is changing 0.878 units relative to the other, and the standard deviation for the change is 5.07 | r is the ratio in sample sizes between groups being compared |
|  | Effect size estimates | esize |  |
|  | Effect size estimates | esizei n\_observation1 mean1 sd1 n\_observations2 mean2 sd2 | Esizei calculates effect from input means, sds you give it. This is called the immediate version, using STATA as a calculator. Can also calculate effect size from raw observations in a variable using esize |
| General Analysis Graphs |  |  |  |
|  | Keep a graph on screen for comparison with other graphs | Graphcommand variable, name(graphname, replace)  e.g. scatter age height, name(agebyheight, replace) | Obvious |
|  | Look at distribution | Histogram variable, bin(#) | Bin option specifies number of bars, size of group for each bar |
|  | See if variable distribution looks normal by comparing it with normal distribution | Kdensity variable1, normal | This is really a smoothed histogram |
|  | See if variable distribution looks normal by comparing it with normal distribution | qnorm variablename | Q-Q plot compares actual values to quantiles form normal dist with same mean, sd |
|  | Look at distribution of a variable using boxplot | Graph box variablename, over(grouping variable)  -or-  Graph box variablename, over(grouping variable) name(plotname replace) | Upper/lower ends of box are 25th 75th percentiles (height is interquartile range, or IQR), line is median, whiskers at 1.5 IQRs, then have outliers  Option name keeps this plot when others requested, replace allows you to replace it with plot of the same name |
|  | Examine whether outcome y is linear with respect to x-values | Lowess outcomevar predictorvar, bw(0.4) | Bw option specifies smoothing; higher, more smoothing |
|  | Another way to examine whether outcome y is linear with respect to x-values using residuals | Predict residualvariablename, residuals  Scatter residualvariablename predictorvar  e.g.  regress age time  predict ageresid, resid  scatter ageresid time | This is another good way to examine whether the change in a variable is liner over time, to decide whether you should add another term to assess for curvature |
|  | Plot residuals vs predicted values | Predict residualvariablename, residuals  Predict predictedvariablename, xb  Scatter residualvariable predictedvariable | This is a way to verify the variance in the outcome variable is normally distributed  Would run this after regression model. First have to create values for predicted y value at each x (that’s held in the variable taking the xb data) and residual for that observation (that’s held in the variable taking the residuals data) , and then use a scatterplot to examine the relationship |
|  | Examine pattern of mean values across multiple categories (e.g. time) using line chart | Table category1 category2, c(statistic variable)  The command above give you a table to tell you how the variable you are interested looks across categories  Bysort category1 category2: egen summaryvariable=mean(variable)  The command above sorts by categories and creates a summary statistic for each category value  Twoway (connect summaryvariable category1 if catgory2=value1) (connect summaryvariable category1 if catgory2=value2)  The command above will make a line graph of the summary variable you created for all values of category1 at a specific value for category 2 and a separate line graph for all values of category 1 at another value for category2.  e.g.  Table visit sex, c(mean age n age)  bysort visit sex: egen meanage=mean(age)  twoway (connect meanage visit if sex==0) (connect meanage visit if sex==1)  The above set of commands make a table showing mean age and n for each visit separately for males and females, then makes a mean age for each visit for each sex and then graphs the mean age across visits separately for men and women. | Can use same connect graph to graph predicted values. Would first use sort command to properly order the variable you want to graph by (e.g. sort visit) and then you would use the connect command (e.g. connect predictedvalue visit if sex==0) |
|  | Spaghetti plot (individual values) | Spagplot yvariable xvariable, id(idvariable) options  Spagplot will make individual plots, one for each id, graphing y as a function of x. With no options, it will plot the lines based on the fitted value. If you use the option nofit, it will graph the actual datapoints  Spagplot doesn’t come with Stata, you have to download it  e.g.  Spagplot cdrbox visit, id(pidn) nofit  This will graph cdrbox for each visit for each pidn, connecting observations for each pidn with a line. |  |
|  | Calculate p-value from standard error or CI | From SE  Calculate z-score: z = Estimate/SE  P-value calculation: p = exp(-0/717xz – 0.416xz^2)  From CI  Calculate SE first: SE = (upper limit – lower limit)/(2x1.96) |  |
|  | Calculate standard deviation from standard error or confidence interval | From standard error  SD = SE x √N  From confidence interval  SD = (upper limit – lower limit)/3.92 x √N |  |
| Analysis: Regression |  |  |  |
|  | Regression (standard) | Regress outcomevariable predictorvariable |  |
|  | Regression with no output | Quietly regress outcomevariable predictorvariable |  |
|  | Regression with beta values instead of CIs | Regress outcomevariable predictorvariable, beta | Beta represents the change in sd units in the outcome for a one sd change in predictor. Useful for comparing effect sizes for multiple predictors |
|  | Compute values based on model | Predict  e.g.  predict yfit, xb will produce fitted y values in a new variable called yfit  predict yres, res will produce a residual values (difference between actual and predicted) in a new variable called yres  or, can calculate residuals from predicted mean | Run these after run regression model. Xb, which graphs predicted y values for model, is default  Will work after xtmixed  For xtgee have to calculate residuals |
|  | Poisson regression | poisson outcomevariable predictorvariable | Used when outcome variable is a “count” variable (e.g. number of ED visits). |
|  | Bootstrap regression | Bootstrap “regress outcomevariable predictorvariable” \_b, reps (1000) | Used when assumptions for regression violated |
|  | Bootstrap regression | Regress outcomevariable predictorvariable, vce(bootstrap, reps(1000) bca seed(1)))  Above command runs a bootstrapped regression  Estat bootstrap, all  Above command will output alternative confidence intervals with denotation of (N)- normal confidence interval, (P) percentile confidence interval, (BC) bias-corrected confidence interval, (BCa) bias corrected and accelerated confidence interval | Reps specifies number of repetitions. Bca allows Stata to calculate acceleration for each statisitic. See option sets starting point of pseudorandom number generator, which you will have to re-use if you want same bias-corrected estimates of CIs next time.  P values are based on bootstrap standard errors |
|  | Declare a variable as categorical | Put ‘i’ in front of it, as in i.variablename | Automatically makes lowest category number as the reference in the regression/model. Coefficient of the constant represents the mean of the base value, coefficients of other levels are relative to base level |
|  | Declare a categorical variable as having no base | Regress outcome Ibn.variablename, noconstant | All levels will be included in the regression model. Coefficient for each level of variable should be mean for that level. Have to use noconstant option if you are not creating a base level |
|  | Change reference categories for a predictor | Regress outcome ibl.categoryvariable |  |
|  | Examine predicted means for each level of categorical variable | Margins categoricalvariable |  |
|  | Estimate effects or compare effects of variables within a multivariate model | Lincom myvariablename1 – myvariablename2  This assesses whether the predicted values for the outcome at the mean value of each variable are significantly different, and will give that predicted difference  -or-  Lincom myvariable1 + myvariable1#myvariable2  In situations where there’s an interaction, this will look at additive effect one variable holding all others constant plus additional effect of that variable per unit of the other per unit of the interacting variable)  -or-  lincom \_cons + X.variable1  This commands would give predicted value for the outcome at the requested level of variable 1 by adding the effect of the variable to the intercept value  e.g. lincom \_cons + sex would give the predicted value for the outcome as sex=1 (e.g. male, assuming coded as 0 for female, 1 for male) and evaluate the significance of that prediction  -or-  lincom \_cons + X \* variable1  e.g. lincom \_cons +10 \* age would give predicted value of outcome associated with a 10 unit change in age. If this was, for instance a study of newborns and outcome was weight and age was measured in days, the \_cons would give predicted weight at birth and the lincom command would give predicted weight at 10 days and significance of that prediction with CI  -or-  lincom i1.catigoricalvariable – i2.categoricalvariable  This will directly compare two levels of a categorical variable | Options (after comma) include statistic (e.g. HR for Hazard Ratio) default would be coefficient |
|  | Look at effects of one or multiple variables with a larger multivariate analysis at once | Testparm myvariable | Provides overall test, does not assume trend. ? Default Wald test? |
|  | Compare fits for different models by comparing log likelihoods  Uses likelihood ratio test | lrtest A B  e.g.  regress weight height  est store A  regress weight height age  est store B  lrtest A B  This set of commands (run each line, then do next) will make a regression examining whether height predicts weight, and store that estimate in A, then run a second regression also including age as a potential predictor, and store that estimate in B, and then compare estimates A and B with the lrtest. If the lrtest is significant, then the additional variable significantly improves the model fit | First you have to run one model, then you run a second model with an additional term you are interested in using. A and B are stored log-likelihoods from two different cox or logistic regression models |
|  | Examine whether there is a trend across levels for an ordinal variable | Test i.myvariablename | Must use appropriate linear combination of levels based on number of variables. In VGSM book, given by table 4.8 |
|  | Another way to examine for trend across ordinal levels (STATA 12) | Contrast p.myvariablename  Or  Contrast q.myvariablename | p will use actual values, e.g. if it’s coded 1,2,3,5, it will assume twice the effect between 3 and 5 as between 2 and 3.  q will not assume that, will just use order. |
|  |  |  |  |
| Analysis:  Repeated measures regression |  |  |  |
|  | Regression with repeated measures outcome | Xtgee outcomevariable predictorvariable, i(groupingvariable) | Stands for cross-sectional time series general estimating equation  Xtgee  1. Can only handle one level of clustering  2. Cannot generate predicted values for each cluster  3. Can incorporate robust SEs  xtgee assumes relationship that holds on average over all subjects in a population. Doesn’t allow modeling of subject specific intercepts and slopes |
|  | Declare dataset as clustered | Xtset clusteringvariable repeatedvariable  e.g. xtset id visit |  |
|  | Fixed vs. Random Factors | Random factor: Distribution is assumed for the effects associated with levels of the factor  Are we willing to assume that the effects at each level of a factor are a random sample from a distribution.  Meaning, there is no hypothesized predictable relationship a-priori between different levels of our factor and the population of interest.  Fixed factor: values are unknown constants for levels of a factor  Here, we are recognizing that there could be a relationship between the different levels of a factor and the population of interest. If the variable is categorical, it would be fixed if the values of the variable represented some important aspects of the population, they are not just random values from that population.  e.g. unrelated patient id list is likely a random factor  e.g. In fecal fat data from course:  Pill type is fixed because there is a structure behind what pill types were chosen  Gender is fixed because there is clearly a structure behind choosing them (they represent two major possibilities of humans)  Subject is a random factor because they could be seen as chosen from a larger group  e.g. For back pain data  Physician is random variable, cause physicians can be seen as chosen from a larger distribution of physicians  Practice style is fixed because there is structure behind the choice of these styles, and there is a limited set of styles  Visit time is fixed, because these visit times were not chosen at random, may not represent all visit times, may be influenced by specific factors, like symptoms, practice style, etc.  Patients is random cause each patient can be seen as representing a randomly chosen patient. That patient’s name, ID number, do not tell you anything else about them relative to all other patients  Age is fixed (continuous)  Gender is fixed  CONTINUOUS variables are NEVER random because different levels of the predictor are not likely to have a random set of outcomes relative to other levels nearby | Observations that share same level of random effect are modeled as correlated.  Most of thinking is around categorical variables because continuous are always NOT random  For a factor that has a relatively small number of clusters that may not be representative, might consider keeping the factor fixed (e.g. only studied 4 to 5 patients, or doctor’s offices) |
|  | Mixed effects regression | Xtmixed outcomevariable fixed\_effect1 fixed\_effect2 || random\_effect1: || random\_effect2, reml  e.g. xtmixed bweight birhord initage || momid:, reml  -or-  xtmixed cost i.practicestyle i.education || doctor: ||patient: reml  The second command is an example with two levels of clustering, both being random effects. Start with higher level of cluster, then give next level | Reml option uses restricted maximum likelihood, which is less affected by outliers  Outcome and predictor variables can both be varying or constant with patients  Xtmixed:  1. Only good for numerical, normally distributed outcomes  2. Can handle multiple levels of clustering  3. Can generate predicted values for each cluster  4. Can partition variability within/across clusters  5. Requires you to have some idea of correlation structure  Colon after a random effect says to allow separate intercept for each level of random factor, meaning that each cluster has its own baseline and that the variability in baseline values across the clusters is random. |
|  | Mixed effects regression for numerical outcome with interaction | Xtmixed outcomevariable timevariable##timecontant\_fixed\_factor || random\_variable1: || random\_variable2: , reml  e.g. xtmixed weight height decade##sex || country: || idnumber  Will run model looking at the effects of sex and height on weight across decades (for instance if measured weight at 3 decade intervals) taking into account measures within people and within countries. Include interaction with sex cause if you’re interested in sex as a variable, want to know if that effect varies over time. | NEED to include interaction if predictor variable is a time-invariant predictor (like sex, which doesn’t change across time, or any baseline variable you want to use to predict)  DO NOT NEED to include interaction of predictor is time-varying (like age, which varies with time) |
|  | Mixed effects regression allowing separate slope as well as it’s own intercept | Xtmixed outcomevariable timevariable##timecontant\_fixed\_factor || random\_variable1: timevariable, cov(un) reml  e.g. xtmixed weight i.group day i.group#day || mouse: day, cov(un) reml | Putting the time variable after the colon says you would like to treat the time variable as a random effect as well. In the example, you would be saying that each mouse has its own relationship between weight and day, and that the slopes of these relationships across mice are random. |
|  | Total variance | Total variance = (Patient SD)2 + (residual SD)2  ICC = (patient SD)2 / total variance |  |
|  | Describe clustering and repeated variables for clustered dataset | Xtdescribe | Use after giving xtset command  Will indicate range of values for clustering and repeated variables |
|  | Provide basic statistics for clustered dataset | xtsum | Use after giving xtset command  Will give means, sd’s etc for all variables, including sd’s within and between subjects. \*\*Note that variables with non-zero within subject sd’s are time-varying. |
|  | Use robust option for xtgee clustered models | Xtgee outcomevariable predictorvariable, i(groupingvariable) corr(ex) robust | Robust option uses correlation structure to calculate standard errors. Affects the standard errors not the coefficient. Uses variation from cluster to cluster to do the SE estimation.  good default for xtgee when have a lot of clusters relative to observations per cluster (e.g. 100 centers with 10 observations each is a good kind of ratio). Allows us to estimate standard errors even when the assumed correlation structure is wrong. Particularly useful if you are using a specified structure like exchangeable. Less useful when you assume unstructured and therefore estimating all correlations between levels of predictor. It is better than unstructured cause it doesn’t actually estimate all those correlations.  Rule of thumb, at least 50 subjects, and number of clusters should not be on the order of the number of subjects.  Don’t want to use robust when lots of observations per center. If can’t use robust, have to be comfortable that you know the correlation structure you choose is accurate. Would use xtmixed and compare models with different correlation structures. |
|  | Things you can specify for repeated measures analyses (e.g. xtgee)  1. Family | Family (distribution)  e.g. xtgee bweight birhord initage, family(gaussian) link(identity) corr(exchangeable) i(momid)  This is using the default for family and other choices. If using all defaults you don’t have to specify them at all.  Can be shortened:  e.g. xtgee bweight birhord initage, family(gau) link(i) corr(exch) i(momid) | Specifies hypothesized distribution of outcome variable.  Choices:  Gaussian (default): normally distributed  Binomial: binary outcome (e.g. yes/no), can also be multiple catetorical outcomes  Nbinomial: negative binomial, also count, doesn't make assumption that …  Poisson: For count variable where outcome is non-negative integer. useful for rates/ratios/events. (E.g. number of ER visits per year)  Gamma: For numeric outcomes that are positive and skewed to right (e.g. reaction times, creatinine, blood chemistries, costs). Useful when you don’t want to log-transform non-normal data cause it would be difficult to interpret |
|  | Things you can specify for repeated measures analyses (e.g. xtgee)  2. Link | Link (outcome form)  e.g. xtgee bweight birhord initage, family(gaussian) link(identity) corr(exchangeable) i(momid)  This is using the default for link and other choices | Specifies how connect average value to predictor.  Choices are:  Identity (default for Gaussian): will use mean of raw outcome values to model relationships  Log (default for poission): will model log of average value for outcome. Not same as log-transforming the outcome variable  Logit (default for binomial): models log of odds for outcome  Power  probit |
|  | Things you can specify for repeated measures analyses (e.g. xtgee)  3. Correlation structure  Specifying correlation structure for a mixed model regression | Xtmixed outcomevariable predictorvariable || clustervariable: , corr(un) reml | Default is exchangeable  Choices for correlations structures:  Exchangeable (ex): This is default. all pairs have same correlation  Unstructured (un): each pair can have a different correlation  Independent : observations are not correlated  Autoregressive (ar): means things are less correlated the farther apart they are. Have to specify the degree to which things fall off |
|  | Things you can specify for repeated measures analyses (e.g. xtgee)  Can be prespecified using xtset  4. Clustering variable | i (variable linking observations as a cluster)  e.g. xtgee bweight birhord initage, family(gaussian) link(identity) corr(exchangeable) i(momid) | Clustering variable  Cluster can be by person, location (e.g. hospital, state) whatever data you think would be correlated because of a common factor |
|  | Things you can specify for repeated measures analyses (e.g. xtgee)  Can be prespecified using xtset  5. Time | t | Time variable |
|  | Examine correlations between variables in a model with clustered data | Xtcorr | This will give you correlations after factoring out other variables in the model. **RUN AFTER YOU’VE RUN THE REGRESSION** |
|  | Mixed effect model for binomial (including binary) outcome using xtgee | Xtgee outcomevariable predictorvariable, i(groupingvariable), family(binomial) link(logit) corr(ex) robust  -or-  Xtgee outcomevariable predictorvariable, i(groupingvariable), family(binomial) link(logit) corr(ex) robust ef | Family has to be binomial  Link has to be logit  Can use exchangeable correlation if use robust  Coefficient is log-odds of output  If use ef option (exponential function) then output is odds ratio |
|  | Mixed effect model for count outcome | xtmepoisson | Applies to mixed model where outcome is not numeric or normally distributed. Same syntax as xtmixed. |
|  | Mixed effect model for binomial (including binary) outcome using xtmixed | xtmelogit | Applies to mixed model where outcome is not numeric or normally distributed. Same syntax as xtmixed |
|  | Mixed effect model modeling log of mean for numeric outcome instead of the mean of raw values | Xtgee outcomevariable predictorvariable, i(groupingvariable), family(Gaussian) link(log) corr(ex) robust  -or-  Xtgee outcomevariable predictorvariable, i(groupingvariable), family(Gaussian) link(log) corr(ex) robust ef | Doing exp of coefficient from this output will give relative change associated with the predictor rather than absolute.  Can add ef at end so Stata will exponentiate for you. |
|  | Mixed effect model modeling log of odds ratio for outcome instead of the mean of raw values | Xtgee outcomevariable predictorvariable, i(groupingvariable), family(binomial) link(log) | Gives log of probability of outcome associated with predictor relative to the baseline value of the predictor  translates to relative risk estimate instead of log odds. |
|  | Mixed effect model modeling log of odds ratio for outcome instead of the mean to produce best estimate of relative risk | Xtgee outcomevariable predictorvariable, i(groupingvariable), family(poisson) robust | Recommended approach for relative risk calculation with mixed models  Poisson model use log of mean of outcome in link as default. Has advantage of working better than binomial when probabilities are large, like above 1.  Need robust to override assumption about correlation structure inherent in poisson |
| Survival |  |  |  |
|  | Declare data as survival dataset | Stset timevariable, failure(outcome indicator variable) |  |
|  | Graph survival data | Sts graph | Default graphs Kaplan-Meier curve. Option hr will graph hazard function |
|  | Graph predicted survival effect for specific predictors | Stcurve, survival at (variable1 = value1) at (variable1 = value2)  e.g. stcurve, survival at1(sex=0) at2(sex=1)  -or  e.g. stcurve, survival at1(sex=0, age=30) at2(sex=0 age=60) | Can put multiple parameters for adjustment |
|  | Graph to compare observed and predicted survival for given predictor | Stcoxkm, by(variable) |  |
|  | Graph to examine proportional hazards assumption for predictor | Stphplot, by(variable) nonegative nolntime | This creates log minus log curves for values of a categorical predictor. Evaluate whether curves for levels are constant distance apart |
|  | Another Graph to examine proportional hazards assumption for predictor | Estat phtest, plot(variable) | Generates scatterplot with line through data. Line should be flat |
|  | Another Graphic way to examine prop hazards assumption for predictor | Stcox variable1, schaledsch(sch\_var1)  Lowess sch\_var1 timevariable | First command generates scaled schoenfeld residuals for the variable of interest and saves in new variable  Second command generates scatterplot of the schoenfeld residuals vs. the timevariable along with smoothed line estimate of predicted relationship |
|  | Formal testing of proportional hazards assumption | Stcox variable1, schaledsch(sch\_var1)  Estat phtest, detail | First command generates scaled schoenfeld residuals for the variable of interest and saves in new variable  Second command does test of proportional hazards using the schoenfeld residuals. Significant p-value means hazards are NOT proportional (test looks for correlation between residuals and time, expressed as rho) |
|  | Test effect of variable on survival | Sts test variablename | Default is log rank test. Option Wilcoxon will use that test instead |
|  | Test effects of multiple predictors on survival using Cox regression | Stcox variable1 variable2 etc. | Default is to produce hazard ratio. Uses Wald test to give p-value for each variable, uses likelihood ratio for overall effect. If you want to see beta coefficients instead of hazard ratios, use option ‘nohr’ |
|  | Test effects of a predictors on survival controlling for a variable that violates the proportional hazards assumption | Stcox variable1, strata(variable2) | Stratifies the data according to variable 2 and gives hazard ratio only for variable1 |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models for two factor regression with interaction (helps to plan lincom comparisons) | | | | |
| Group | Cond 1 (y/n) | Cond 2 (y/n) | Intxn | Effect |
| 1 | 0 | 0 | 0 | B0 |
| 2 | 1 | 0 | 0 | B0 + B1 |
| 3 | 0 | 1 | 0 | B0 + B2 |
| 4 | 1 | 1 | 1 | B0 + B1 + B2 + B3 |

|  |  |
| --- | --- |
| Number of categories | Linear combinations for assessing trends |
| 3 | β3 = 0 |
| 4 | -β3 + β3 + 2β4 = 0 |
| 5 | -β2+ β4 + 2β5 = 0 |
| 6 | -3β2 – β3 + β4 + 3β5 + 5β6 = 0 |