|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fns | | getwd() | | | | | | | | | | | setwd() | | | | | | |
| list.files("../data/") | | | | | | | | | | | install.packages("stringr") | | | | | | |
| ?fn: to get info on fn | | | | | | | | | | | library(stringr) | | | | | | |
| ??fn: return list of search results containing the word fn | | | | | | | | | | | help(package="stringr"): list all available fns in package | | | | | | |
| R Data Struc-tures | | Reserved names: FALSE Inf NA NaN NULL TRUE break else for function if in next repeat while c q s t C D F I T  letters: return a to z LETTERS: return A to Z  R uses 1 indexing (i.e. 1st index starts from 1) & no negative indexing | | | | | | | | | | | | | | | | | |
| R has no scalars. Basic building block for storing data is a vector  vector <- c(1,2,3) OR vector <- 1:3  + - \* / ^ %% (modulo) will perform component ops on ea element in vector  vector[-1] (print all elems except for 1st; - : everything except what is after -)  names(vector) <- c(): to give names to elems of vector  vector['name1']: access elem with name 'name1'  vector[order(vector)]: sort vector (order returns index elem shld be in) | | | | | | | | | | | | | | | head(vector), tail(vector, n = 3)  length(vector)  %/% (integer division), %\*% (matrix multiply)  sum(vector), mean(vector)  vector[1]: access 1st elem  v[c(1,2)] OR v[1:2]: access 1st & 2nd elem  |: or, &: and | | |
| mat <- matrix(1:9, nrow=3, byrow=TRUE)  dim(mat)  mat[4]: returns 4th elem  mat[3,2]: returns int of elem in 3rd row, 2nd col  mat[3,2, drop=FALSE]: returns matrix of elem in 3rd row, 2nd col  mat[c(1,3),]: returns matrix of elems in 1st and 3rd rows  rowSums(mat): sum across rows  colSums(mat): sum across cols  cbind(mat1, mat2): add new cols  rbind(mat1, mat2): add new rows | | | | | | | | | | | | rownames(mat) <- c(): give names to rows; rownames(mat) just return rownames  colnames(mat) <- c(): give names to cols; colnames(mat) just return colnames  mat["b", "iii"]: indexing elem in matrix using named rows and cols  mat <- matrix(..., dimnames=list(rownames, colnames))  %\*%: matrix multiply | | | | | |
| Dataframe: cols can be of different classes, not for matrix  A <- c("a", "b", "c"); B <- c(1,2,3); df <- data.frame(A, B)  df[, "B"] or df[c(3,2), ]  df[c(TRUE, TRUE, FALSE), ]: select rows that are true  df[order(df$col1), ]: sort df by col1 | | | | | | | | | | df$A or df$A[1:2]  names(df)  head(df, n=3)  subset(df, subset=col2 <10): filter by condition | | | | | | | |
| lst <- list(A=seq(1,5,by=2), B=seq(1,5,length = 4), C=mat)  lst[[1]][1] == lst$A[1] | | | | | | | | | | | | | | | | lst[[1]]: returns A as a vector  lst[1]: returns a list with A in it | |
| category <- factor(vector): convert vector to factor  nominal categorical factor: no ordering (male, female)  ordinal categorical factor: has ordering (high, medium, low)  levels(factor\_vector) <- c('Male', 'Female')  summary(factor\_vector) | | | | | | | | factor(c('male', 'female'))  factor(vector, ordered = TRUE, levels = c('low', 'medium', 'high'))  factor\_vector originally has levels 'M', 'F' -> now 'Male', 'Female'  output num of elems of ea factor | | | | | | | | | |
| Data types | | | | character, numeric, integer, logical, complex | | | | | | | | | | | | | | class(X) | |
| Fns | edit(var): open GUI to edit data  View(var): to view data  str(data, max.level=1): view structure of data and only show 1st level of nested data  sample(1:6, size=2, replace=TRUE) # select 2 observations with replacement | | | | | | | | | | | | | | | | | saveRDS(data, file="data.rds")  data <- readRDS("data.rds")  rm("x"): remove var x  rm(list = ls()): remove all objects | |
| args(function) #show all args of function, if no default value -> must supply value when caling function | | | | | | | | | | | | | | | | | | |
| x <- seq(from=1, to=5, by=2) == seq(from=1, to=5, length=3)  rep(x, times=3) #1 3 5 1 3 5 1 3 5  rep(x, times=c(1,2,3)) #1 3 3 5 5 5 | | | | | | | | paste(): return concatenated vectors, won't print  print(): return & print vector, cannot concat  cat(): return NULL, print vector | | | | | | | | | | |
|  | unlist()  as.\*()  is.\*()  rev() | | | | | | | | flatten list to produce vector  convert to \* class  check if is \* class  reverse elems | | | | | | | | | | |
|  | nrow()  NROW() | | | | | | | | number of rows  number of rows (return 0 instead of NULL if no rows) | | | | | | | | | | |
| Recycling Rule | | | x <- 5:10; y <-1:2 (shorter vector are recycled multiple times until they match length of longest vector)  i.e. x + y returns 6 8 8 10 10 12 | | | | | | | | | | | | | | | <, <=, >, >=, ==, !=, %in%  &, |: to combine logical vectors elem wise  &&, ||: combine logical elem into 1 value | |
| Fns in R | | | fn <- function() {  if () {return()}  else if () {}  else {}  } | | | | | | | | repeat {} #will run at least once; need include break statement  while {} #might not run due to condition  for (i in seq\_along(vector)) SAME AS for (i in 1:length(vector)) | | | | | | | | |
| Repeated Applications of Functions | | | | | apply(), sapply(), lapply(), vapply(), mapply(), tapply(), mcapply() | | | | | | | | | | | | | | |
| apply(data, margin, function, ...) usually for matrix  row\_means <- apply(X, 1, fn) #1 for row, 2 for col  apply(X, 2, fn, arg=0.1)  apply(X, 2, function(x) sum(x>0)/nrow(X)) | | | | | | | | | | X <- matrix(...)  round(row\_means, digits=2) #round to 2dp  #if fn has additional args, can just add in apply params  #using lambda function | | | | |
| sapply() returns vector or matrix  sapply(data, function(x) x$datapoint2)  sapply(data, function, ...) | | | | | | | | | | | #user-friendly wrapper of lapply  #... = can put additional args to supply to function | | | |
| lapply() (necessary if output of fn call on ea elem is not a vector of the same length) | | | | | | | | | | | | | | lapply returns list of same length |
| use sapply or lapply when repeating a function  BUT if function we are using is a base R function like log(), then no need as can just log(X)  Use for loop if there are many statements and need to run for loop once only OR if result from previous iteration needed for next iteration | | | | | | | | | | | | | | |
| tapply(data, grouping, function) #apply function to groups of data according to grouping | | | | | | | | | | | | | | |
| replicate(n, expr) #repeat expr for n times | | | e.g. expr is invoking a function, or something that will give a result, not just a function name | | | | | | | | | | | |
| vapply(X, function, function.value, ..., use.names=TRUE) | | | function.value to specify what is the output type of function | | | | | | | | | | | |
| String & Regex | | | library(stringr)  str\_length(string)  str\_c(string1, string2)  str\_c('x', c('a','y'), 'z', sep=',')  str\_sub(vector, start, end) | | | | | | | | get length of vector of strings  combine strings  'x,a,z' 'x,y,z'  subset vector of string, from char start to char end | | | | | | | | |
| Functions  #fixed for special chars | | | str\_view(vec, pattern)  str\_view\_all(x, pattern)  str\_detect(x, pattern)  str\_which(x, pattern)  str\_replace(x, pattern, new\_pattern)  str\_split(x, pattern=fixed(""))  str\_extract  str\_match  str\_subset  str\_count | | | get 1st match of pattern for ea elem in vector  get all matches of pattern for ea elem in vector  get logical vector of which string matches  get indexes of strings which matches  replace pattern with new pattern  split string according to pattern (returns list OR do simplify=TRUE)  return only pattern  extract matching groups  return strings containing pattern  return num of times pattern occur in ea string | | | | | | | | | | |
| Basic Regex  ?about\_search\_regex  vignette('stringr')  vignette('regular-expressions') | | | | "^a"  "a$"  "[ae]$"  [^ae]  ".a."  "a|b"  \\w  \\d  ()  \\b | | | | match 'a' at beginning of string  match 'a' at end of string  match 'a' or 'e' at end of string  not a or e  match 3 char with 'a' in the middle (. = any char)  match 'a' or 'b'  alphanumeric, not whitespace, not special chars  digits (equivalent to [0-9])  grp of chars  define word boundary | | | | | | | | |
| Modifiers | | | | +  ?  {2}  {1, 2}  {2,} | | | | 1 or more time  0 or 1 time  exactly 2 time  exactly 1 or 2 times  2 or more time | | | | | | | | |
| R specific char classes  ?base::regex | | | | [:alpha:]  [:punct:]  [[:digit:][:space:]] | | | | alphabets  punctuation  digit or space | | | | | | | | |
| Groups  use round paranthesis to define groups | | | | '\\b(\\w+)\\b \\1' | alphanumeric ≥1 time in word, \1: same grp of char  str\_replace('the the shop' , "\\b(\\w+)\\b \\1", "\\1") returns 'the shop'  If have more than 1 grp, i.e. () ()..., \1: all grps, \2: 1st grp, \3: 2nd grp | | | | | | | | | | | |
| Look-ahead and Look-behind ops | | | | [0-9](?=pattern)  (?<=pattern).+ | return num before pattern  return rest of string after pattern | | | | | | | | | | | |
| Factors | | | fac <- factor(x1)  levels(fac)  fac <- factor(x1, levels=c()) | | | | | | | | convert to factor  levels of a factor  vector don't change, but levels displayed now in order (i.e. vector itself still not ordered) | | | | | | | | |
| Datetime | | | (%a/%A: weekday, %b/%B: month in letters, %H: 24hr, %I: 12hr, %M: minutes, %S: sec, %p: AM/PM, ) | | | | | | | | | | | | | | | | |
| ?strftime, Sys.Date()  d1 <- as.Date('2022/08/22', '%Y/%m/%d')  weekdays(d1, abbreviate=FALSE)  months(d1, abbreviate=FALSE)  seq(d1-30, d1, by='1 week') | | | | | | | | convert to date object (strftime(date, "%Y-%m")  extract which day of the week  extract which month  create seq starting from d1-30 days, increment by 1 week  (can do 1 quarter: 3mths and more, ?seq.Date) | | | | | | | | |
| Plotting | | | plot(x,y, xlab='', ylab='',main='', pch=2, col='blue', bty='n', type='l')  abline(h=0.4,col='') #add horizontal line, v for vertical line  points(x,y) #add point at (x,y) to existing plot  lines(c(x1,x2), c(y1,y2)) #add line to existing plot | | | | | | | | main: title, pch: plotting character, cex: character expansion, cex.axis, cex.main, bty: type of box surrounding plot (for points), type: type of points, lty: line type  example(pch) to see all plotting characters  colours() to see all colors | | | | | | | | |
| Barchart | | | library(stringr)  barplot(df$col2, border=NA, names.arg=str\_to\_title(df$col1)  barplot(matrix, beside=TRUE) #create grouped bar charts | | | | | | | | heights proportional to values of variables | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Import Data | | read.csv(file, header=FALSE, skip=2, col.names=c())  unique(data$col)  apply(data, 2, function(x) sum(is.na(x)))  readLines()  library(tidyverse)  read\_csv() | | #skip 1st 2 rows, #na.strings="-" (specify "-" in csv to be treated as NA)  #colClasses=c("character", "integer", "factor") (specify what classes each col should be)  #get unique values in col  #get sum of NA values  #used if no other choice  #better formatting for data | | | | | | |
| Histo-gram | | hist(data$col, main, xlab, freq, col, border, breaks)  #freq=FALSE to get density plot  #e.g. breaks=seq(0,200,by=10) to get bins of 10  - bar charts for categorical data, histogram for cts data | | | Things to look out in histogram:  - skewed/symmetric  - unimodal/multimodal? (if multimodal, is there a subpopulation?)  - outliers | | | | | |
| Excel files | library(readxl)  read\_excel(file, skip=, col\_names=FALSE)  excel\_sheets(fname) | | | | vignette('sheet\_geometry), vignette('cell-and-column-types')  #range="C7:E9" (use cells C7 to E9 for data)  #sheet=3 (use 3rd sheet; if file has multiple sheets)  #get sheet names | | | | | |
| library(stringr)  all\_data <- NULL  for (sn in sheet\_names) {  tmp <- read\_excel(fname, sheet=sn) #get data for 1 sheet  sn2 <- str\_trim(str\_split(sn, ",")[[1]] #splitting sheet name  tmp$col1 <- sn2[3]  all\_data <- rbind(all\_data, tmp)  } | | | | OR get\_data <- function(sn, fname2) {  tmp <- read\_excel(fname2, sheet=sn)  sn2 <- str\_split(sn, ", ")[[1]]  tmp$col1 <- str\_extract(sn2[3], "^\\d+-\\d+"  #OR "^.+(?= years)" for regex to get '1-14' from '1-14 years'  tmp }  out <- lapply(sheet\_names, get\_data, fname2=fname)  do.call(rbind, out) #equivalent to rbind(out[[1]], out[[2]], ..., out[[n]]) | | | | | |
| JSON files | | - can contain strings (""), num, obj, array, true, false, null  library(jsonlite)  fromJSON(txt) #or can input file | | | | Object: unordered collectiion of key/value pairs (dictionary)  Array: ordered list of values (python list) | | | | |
| string <- readLines(file)  prettify(string)  minify(string) | | | | # include indentation  # output all in 1 line | | | | |
| lines <- readLines(file)  json\_list <- lapply(lines, fromJSON) | | | | #for multiple JSON objs in file  OR stream\_in(file("..")) | | | | |
| OOP in R | | S3 Object  class(X) <- "fooS3"  summary.fooS3 <- function() {}  plot.fooS3 <- function() {}  summary(X); plot(X)  methods(generic.function="plot")  methods(class="fooS3") | | | | | | | | #make X into a class of "fooS3"  #make summary work for class "fooS3"  # make plot work for class "fooS3"  #see which classes plot works for  #see which methods are defined for fooS3 |
| S4 Object  fooS4 <- setClass("fooS4", slots=c(X="numeric", msg="character"))  X <- fooS4(X=rnorm(20), msg="test")  setMethod("summary", signature(object="fooS4"),  definition=function(object,...) {})  setMethod("plot", signature(object="fooS4"),  definition=function(object,...) {})  isS4(X)  slotNames(X)  showMethods(X) | | | | | | | | -obj has slots/attributes  #creating constructor  #instantiating fooS4 class  #creating summary method for fooS4  #creating plot method for fooS4  #check if is an S4 obj  #check slots of X  #see all methods defined for fooS4 |
| RC obj  fooRC <- setRefClass("fooRC", fields=..., methods=...)  X <- fooRC$new(...)  X$plot() | | | | | #same as python obj (methods belong to the class)  #defining constructor  #instantiating fooRC  #calling method of fooRC | | | |
| Working w Databases | | | library(mongolite); library(jsonlite)  credentials <- paste0(readLines("mongo\_user\_pwd.txt"), collapse=":")  connection\_string <- paste0("mongodb://", credentials, "172.20.28.29:2717/test") | | | | | | | |
| con2 <- mongo(verbose=TRUE, collection="table", url=connection\_string)  con2$count()  con2$index() | | | | | | #Mongo collection=SQL table  #num of records in collection  #check what is indexed in DB | |
| q1 <- toJSON(list(col1="", col3 = ""), auto\_unbox=TRUE)  q2 <- toJSON(list(col1 = list(`$in`=c("",""))), auto\_unbox=TRUE)  q3 <- toJSON(list(`$text`=list(`$search$`="")), auto\_unbox=TRUE)  con2$find(query=q1, fields='{"col2":1, "col5": 1}') | | | | | | #get JSON to be passed as query  #check if value IN query  #search text indices in DB  #queries and returns matching records, (get col2 and col5 where record match query q1) | |
| it <- con2$iterate(query=q2, fields='{"col2":0, "col5":1}')  it$batch(10)  repeat {  x <- try(it$batch(10))  if(inherits(x, "try-error")) {  message("Exiting cleanly.."); break;  }  res <- c(res, x)  } | | | | | | #create iterator and return only col5  #return first(next) 10 matching records  #keep getting records in batch of 10  #try catch clause  #concatenate all results tgt | |
| con2$export()  con2$insert()  rm(con2) | | | | | | #queries & write matching records to file  #insert records into collection  #close connection | |
| Data from the Web | | Downloading file:  file\_url <- "https://....gov.sg/..." (e.g.)  return\_val <- download.file(file\_url, "../data/fname.zip")  con <- unz("../data/fname.zip", "file1.csv")  data <- read.csv(con, header=TRUE)  library(tidyverse)  youths <- filter(data, col1==, col2==)  %>% mutate(pct = as.numeric(col3))  %>% arrange(desc(col3))  oldpar <- par(mar=c(5.1,5.8,2.1,1.1))  ?par #to find out more graphical params  barplot(youths$col3, names.arg=youths$col4, horiz=TRUE, cex.names=0.6, las=1, border=NA)  par(oldpar) | | | | #if return\_val = 0: success, 2nd arg to specify where to place file  #open connection to csv file within zip file  #coerce data type  #arrange by col3  #par returns current parameter settings while setting new parameters. mar: margins for BOT, LEFT, TOP, RIGHT  #las: make labels horizontal  #reset params back to default | | | | |
| API:  library(httr); set\_config(verbose())  base\_url <- "https://api...."  response <- GET(base\_url, query=list(date\_time="..."))  data <- content(response)  bin <- content(response, "raw")  writeBin(bin, "file.png")  x\_coords <- sapply(data$col, '[[', 1) | | | | | | # use if data sent back is json  # use if data sent back is a file (to get binary data)  # write binary data to file in local  #apply data$col[[1]] on ea elem | | |
| Web Scraping  libray(rvest); library(xml2)  page <- read\_html("url")  nodes <- html\_nodes(page, "tag to search for")  html\_attr(nodes, name="attribute1")  html\_text()  html\_table()  html\_structure() | | | | | | #typical html have tags w class name, type, id,...  #extract nodes  #extract specific attributes  #extract text  #extract table in node  #inspect structure of node | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Manipu-lation | library(tidyverse)  as\_tibble() #convert to tibble obj; tibble() #create tibble obj  glimpse(df) | | | | | | | | | | | | #tibble similar to dataframe  #transpose df |
| filter(df, col1 == x, col3 == y, between(col2, -z, z), col4 %in% c(w,w+1))  filter(df, is.na(col3) | col3 > y) | | | | | | | | | | | | #pick rows according to values in col  #by default filter will drop row w NA |
| select(df, col1, col2, col4) OR select(df, col2:col5) OR select(df, !(col1:col3))  select(df, starts\_with("")) OR ends\_with() OR contains("")  select(df, matches(""))  select(df, where(is.numeric)) | | | | | | | | | | | | #pick cols by names  #available conditions for col names  #for regex  #cols that return TRUE when fn applied |
| mutate(df, newcol = ..., newcol2 = ..., .before=col2)  #fn that works in mutate: basic arithmetic, log, log10, lead, lag  lag(vector, n=3)  lead()  cumsum, cumprod, cumin, cumax  minrank(vector), minrank(desc(vector)): for descending order  mutate(df, id=row\_number())  mutate(col2 = recode(col2, "Brunei Darussalam", "Brunei")) | | | | | | | | | #create new vars, default add new cols to end of df  #.before so new col before col2 OR .after  #compare w previous value  #compare w future value  #cumulative fns  #assign rank according to value in vector  #create id column  #if value is ..., change to ... | | | |
| arrange(df, col2, col1, col3)  arrange(df, desc(col2)) | | | | #reorder rows according to col, if tie, then use next col given  #reorder in descending order (missing values always placed at end) | | | | | | | | |
| d2 <- group\_by(df, col2, col3, col1)  filter(d2, row\_number() > 2) OR filter(d2, col1 <= median(x))  mutate(d2, newcol=cumsum(col1)) | | | | | | | | | | | #group rows  #works according to group, not whole dataset | |
| summarise(df, newcol=mean(col2,na.rm=TRUE))  group\_by(df, col1, col2, col3) %>%  summarise(newcol=..., .groups = "drop") | | | | | | | #show single col, unlike mutate which also shows old col  #drop grouping created, so fns after this apply to whole df | | | | | |
| Reorder | | | | df %>% mutate(col1 = reorder(col1, -col2) | | | | | #order categorical col1 in descending order of col2 | | | | |
| Pipe operator  %>% | | | | df %>% filter(col2 == ) %>% group\_by(col1) %>%  summarise(count=n(), newcol=..., .groups="drop") %>%  filter(col3>0, col5!="") | | | | | | | | | n(): count number of data pts in a grp  #try to put 1 %>% per line |
| Summary fns | | | | df2 <- df %>% filter(!is.na(col1), !is.na(col2))  df2 %>% group\_by(col3, col4)  %>% summarise(first = min(col5),  latest = last(col5, order\_by = col6), .groups="drop")  df2 %>% group\_by() %>% summarise(newcol = n\_distinct(col6),  .groups="drop") %>% arrange(desc(col6))  df2 %>% group\_by() %>% summarise(freq=n(), col = col[1])  df2 %>% group\_by() %>% add\_tally() | | | | | | | | | #other fns are mean(), median(), sd(), IQR(), mad(), min(), quantile(), max()  #first(col, n, order\_by): get col of row in 2nd place after order\_by,  #last(), nth() similar to first()  #n\_distinct() similar to n()  #same as df2 %>% group\_by() %>% count()  #creates new col w/o summarising data |
| Across columns | #across has to be called within mutate() or summarise()  mutate(df, across(col1:col5, fn))  mutate(df, across(last\_col()-2:last\_col(), fn)  mutate(df, across(col1:col5, .fns = list(a1 = fn1, a2 = function(x) x^2, b2 = ~.x^2) | | | | | | | | | | | | #apply fn across col1 to col5  #apply fn to 3rd last col to last col  #apply fns a1, a2, b2 to col1 to col5 (Note a2 same as b2) |
| Across rows | df %>% rowwise() %>% mutate(new\_col = col2 + col3)  df %>% rowwise() %>% mutate(new\_col = sum(c\_across(col1:col3)) | | | | | | | | | | | | #similar idea |
| Tidy data | 1) Each var form a col. 2) Each observation form a row. 3) Each type of observational unit forms a table  Untidy: 1. Column headers are values -> pivot\_longer(). 2. Multiple vars stored in 1 col -> pivot\_wider()  df %>% slice\_head(n==3) %>% ... #try out on 1st 3 rows | | | | | | | | | | | | |
| 1. df %>% pivot\_longer(!country, names\_to="year", values\_to="cases")   |  |  |  | | --- | --- | --- | | country | year | Result | | A | 1999 | 123 | | B | 1999 | 456 | | A | 2000 | 789 | | B | 2000 | 012 |  |  |  |  | | --- | --- | --- | | country | 1999 | 2000 | | A | 123 | 789 | | B | 456 | 012 | | | | | | | | | | | names\_to=c("col1", "col2"), names\_sep=-1  #split original col name into [:-1] and [-1:] and insert into col1 and col2 respectively  names\_to=c("col1", ".value"), names\_sep=-1  #SEE L3S82 | | |
| 2. df %>% pivot\_wider(id\_cols = country:year, names\_from="type", values\_from="count")   |  |  |  |  | | --- | --- | --- | --- | | country | year | type | count | | A | 1999 | s | 123 | | A | 1999 | t | 456 | | A | 2000 | s | 789 | | A | 2000 | t | 012 |  |  |  |  |  | | --- | --- | --- | --- | | country | year | s | t | | A | 1999 | 123 | 456 | | A | 2000 | 789 | 012 | | | | | | | | | | | | | |
| separate | df %>% separate(col3, into=c("col3", "col4"), convert=TRUE) | | | | | | - separate col by splitting where a separator char appears.  - convert to convert data type | | | | | | |
| unite | df %>% unite(col3:col5, sep="/") | | | | | | merge multiple cols into one, w separator char | | | | | | |
| Relational data | | | Primary key: uniquely identifies observation in its own table  Foreign key: uniquely identifies observation in another table  Usually 1 primary key to many foreign key | | | | | | | | | | To check if key is unique:  df %>% count(col1) %>% filter(n>1) |
| Tribble | | | tribble(~col1, ~col2, x1, "y1", x2, "y2", x3, "y3") | | | | | | | | | | create tibble row wise |
| Mutating joins | | | Add new vars to a df from matching observations in another  df %>% inner\_join(df2, by = "key")  Outer join: left join, right join, full join  df %>% left\_join(df2, by=c("col1" = "col2"))  If 1 table has duplicate keys, the matching row is duplicated as well  If both tables have duplicate keys, then cartesian product of keys is created | | | | | | | | | | intersection on col "key" in both df  #join by col1 in df & col2 in df2 |
| Filtering joins | | | Filter observations from 1 df based on whether or not they match an observation in other table | | | | | | | | | | |
| semi\_join(x,y) #keeps all observations in x that have a match in y  anti\_join(x,y) #drops all observations in x that have a match in y | | | | | | | | | |  |
| Example | | top\_dest <- flights %>% count(dest, sort=TRUE) %>% head(10)  flights %>% semi\_join(top\_dest) | | | | flights %>% count(dest) %>% slice\_max(order\_by=n, n = 10)  %>% semi\_join(flights, .) | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Good graphs | 1) show the data  2) induce viewer to think about the substance rather than about methodology, graphic design, software used, ...  3) avoid distorting what the data have to say  4) present many numbers in a small space  5) make large data sets coherent | | | | 6) encourage the eye to compare diff pieces of data  7) reveal the data at several levels of detail, from broad overview to the fine structure  8) serve a reasonable clear purpose: description, exporation, tabulation or decoration  9) be closely integrated w the statistical & verbal descriptions of a dataset | | |
| Types of graphical displays | Data maps: superimposed attributes onto geographical data (study clusters, hotspots)  Time series  Narrative graphics of space and time: add spatial dimensions to time series displays  Relational data: more abstract, put 1 var on x-axis and another on y-axis | | | | | | |
| Bad graphs | | Inconsistent basis of comparison, distortion in data graphics, unnecessary chart,  Not always bad: dual axis, axis not starting w 0 | | | | | |
| Graphical Integrity | Representation of nums should be directly proportional to the numerical quantities represented  Clear, detailed and thorough labeling should be used  In time series, use deflated and standardised units of monetary measurements  Num of dims in graphic should not exceed num of dimensions in data | | | | | | |
| Data-ink ratio | Data-ink ratio = data-ink/total ink to print graphic. Should try to maximise ratio  Erase non-data ink, or redundant data-ink | | | | | | |
| Chart junks | Distracting patterns/cartoons, unnecessary 3D, | | | For tables, can remove vertical lines, or rearrange rows and cols to show insights (provided no actual ordering to rows and cols) | | | |
| Small multiples | | | Design of each frame is constant | | | | E.g. heat map over time |
| Friendly graphics | Words are spelt out  Words run from left to right  Little messages on the graphic explain the data  Labels on the graphic; no legend required  Graphics provokes curiosity of viewer | | | | | UNFRIENDLY: contains abbreviations  Words run vertically  Viewer need to keep referring to text to understand graphic  Legend diff to rmb  Filled w chart junk | |
| Bias | Patternicity bias: tend to find pattern  Storytelling bias  Confirmation bias | | | | | | Treat each "finding" as a conjecture  Confirm it in multiple ways  Ask someone else to review |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ggplot | library(tidyverse)  vignette('ggplot2-specs')  ggplot(data = df) + geom\_point(mapping = aes(x=col1, y = col2),  size=0.5,  position = "jitter")  + scale\_<aesthetic>\_manual(values=1:7, name='', labels=c())  + labs(title='', x='', y='') | | | | # can map color, size, shape, fill, alpha, group, stroke  #arguments outside aes are applied to all points  #OR position = position\_jitter(width=.., height=0)  #specify how to map variable | | |
| size expects variable to be a ordered vector  color expect factors, so need convert as.factor(col3)  shape expects only ≤6 factors, can override this | | | | geom\_point(): more for 2 cts variables  geom\_histogram, geom\_freqpoly: more for 1 cts variable | | |
| General data exploration steps | | | Look for general trend. Look for points which deviate the most  Check for overplotting (multiple points overlay on one another) | | | | |
| Histogram | x-axis 1st divided into bins. Then num of observations in each bin is counted  geom\_histogram display counts in each bin w bars  geom\_freqpoly display counts w lines (more appropriate when comparing dist conditioned on a categorical variable) | | | | | #aesthetics are x-coordinate, alpha, colour, fill  #other args: width of bins, num of bins, location of bins | |
| ggplot(df) + geom\_histogram(aes(x=col1), binwidth=10, boundary=0,  colour='', fill='')  ggplot(df) + geom\_histogram(aes(x=col1, y=after\_stat(density)), binwidth=10,  boundary=0, fill='')  + coord\_cartesian(xlim=c(100,200))  + xlim(c(100,200)) #filter out rows according to xlim (not what we want)  ggplot(df) + geom\_histogram(aes(x=col1, y=after\_stat(density)), binwidth=10,  boundary=0, fill=col3, position='dodge')  + scale\_fill\_discrete(labels=c("",""), name="")  ggplot(df) + geom\_freqpoly(aes(x=col10, y=after\_stat(density)), binwidth=10,  boundary=0, colour=col3) | | | | | #boundary specify x-axis shld start at 0  #after\_stat: use statistics computed by ggplot  #zoom into specific part of plot  #dodge place bars side by side  #adjust legend  #if want to get a smooth curve; use geom\_density | |
| Things to look out for: symmetric? bimodal/unimodal? outliers? | | | | | | |
| Line plot | geom\_line()  #aesthetics are x,y- coordinates, alpha, colour  #other args: lty, linewidth | | | | | #more for time series | |
| ggplot(df, aes(x=col1, y=col2, group=col3)) + geom\_line() + geom\_point()  + geom\_text(data=df2, aes(x=col1, y=col2, label=col3)),  hjust="left", nudge\_x=0.1, size=3.5) | | | | | #since the aes same for both line & point  #add text to plot  #add justification, shift text to right by 0.1 | |
| Reference line | geom\_vline(aes(xintercept=100), lty=2, size=0.3): vertical line  geom\_hline(): horizontal line  geom\_abline(): straight line defined by slope and intercept | | | | |  | |
| Parallel Coordinates Plot | | ggplot(df) + geom\_line(aes(x=col1, y=col2, group=col5, colour=col4), alpha=)  library(GGally)  ggparcoord(data=df, columns=1:4, groupColumn=5, order="anyclass", showPoints=TRUE, alphaLines=0.3) | | | | #To compare r/s btw variables, and to compare that r/s across groups  #same graph but w GGally package | |
| Bar Charts | geom\_col: height of bar represent values in data  geom\_bar: height of bar proportional to num of cases in each group  +coord\_flip() | | | | | # Aesthetics: x,y,alpha, colour, fill, linetype, size  #flip x and y coordinates | |
| Facets | Typically categorical variable is used for facet (1 subplot created for each level of factor) | | | | | | |
| geom\_...(aes(), show.legend=FALSE)  + facet\_wrap(~ col3, scales="free\_x")  + theme(legend.position='bottom')  facet\_grid() | | | #to not show legend  #for facet by 1 variable; so x-axis not fixed across facets  #if legend present  #facet by > 1 variable | | | |
| Geom Smooth | Draws line through the points. To better see trends in time series data/ r/s btw variables. Some e.g. of smoothers are linear regression models, loess smoother  + geom\_smooth(method="lm", se=FALSE)  geom\_smooth(span=0.6) #loess default for <1000 points; span=range of points  to give more importance to  Things to check: is variance of pts at each x constant | | | | | # Aesthetics: x,y,alpha, colour, fill, linetype  #Linear model; se: whether to have CI  #locally weighted regression smoother | |
| Rug Plots | Supplement scatter plot w marginal distributions at both axis  ggplot(df, aes(x=col1, y=col2)) + geom\_point() + geom\_rug(position="jitter") | | | | | # Aesthetics: x,y,alpha, colour, group, linetype, size | |
| Box Plots | Visualization of lower whisker, lower quartile, median, upper quartile, upper whisker  Outliers is observation if < lower quartile – 1.5\*IQR OR > upper quartile + 1.5\*IQR  Boxplot does not portray certain features of dist, such as distinct mounds, and possible gaps in data  If dist is unimodal, then boxplot can give an indication about skew of dist.  Boxplots useful for identifying outliers, and comparing groups w.r.t their center and spread  ggplot(df, aes(x=category, y=col2), outlier.shape=NA) + geom\_boxplot()  boxplot(df$col2, plot=FALSE)$out | | | | | | # aes(x, lower, upper, middle, ymin, ymax, colour, fill, linetype  #to not show outliers  #to get coord of outliers |
| 2D histogram | Study dist of 2 variables at a time, w fill = count  + geom\_bin2d(aes(x=col1, y=col2), binwidth=c(0.7,5) +  scale\_fill\_gradient(name="Count", low="white", high="red")  geom\_hex()  Another way to view joint dist is through a contour plot: geom\_densit\_2d  ggplot(df, aes(x=col1, y=col2, z=col3)) +  stat\_summary\_hex(fun="mean", binwidth=c(0.7,5)) | | | | | #0.7 for col1, 5 for col2 as bins  #use scale\_fill\_gradient2 for 3 colours  #use hexagons instead of squares for bins  #to use another col instead of count  #use mean of col3 in each bin | |
| Grid of plots | library(gridExtra)  grid.arrange(p1,p2,nrow=1) OR grid.arrange(grobs=list(p1,p2), nrow=1) | | | | | #1 x 2 plots | |
| Themes | +theme\_bw() , classic, dark, minimal, ... | | | | | #can modify ticks, axis, .... | |
|  |  | | | | |  | |
| ggplot | library(ggplot)  ggplot(df, aes(x = col1, y = col2, color=col3, size=col4)) + geom\_point() + scale\_x\_log10()  ggplot(df, aes(x=col1, y=col2, color=col3)) + geom\_point()  + expand\_limits(y = 0) + ggtitle("")  geom\_line(): line plot (for data over time)  geom\_col() : bar plot (comparing values across categories)  geom\_histogram(bins = ): histogram (1 var, freq)  geom\_boxplot() : box plot (dist of 1 var across categories)  geom\_text(label = rownames(df)): text instead of points on graph  geom\_point(alpha = 0.4) + geom\_smooth()  linetype, labels, shape, size, fill, alpha,... | | | | | #geom\_point for scatterplot  #scale\_x\_log10: to scale x-axis by log  #include y = 0 in graph  #can do factor(col1) to treat col 1 as factor  # alpha adds transparency, smooth(): add 95% CI | |
| overlap | identity, dodge, stack, fill, jitter, jitterdodge, nudge | | | | | geom\_point(position = "jitter")  position\_dodge2(): more for cols | |
| Axis | + labs(x="", y = "") + ylim() | | | | |  | |
|  | + geom\_errorbar(aes(ymin = avg-std, ymax = avg+std)) | | | | |  | |
| Theme | + theme(axis.title = element\_text(color=""), line = element\_blank(), legend.position = "none") | | | | | margin(T,R,B,L,unit), unit(num, unit) | |
| statistics | stat\_bin() same as geom\_histogram() OR geom\_frequency()  stat\_count() same as geom\_bar() | | | | |  | |
| facets | facet\_grid(rows = vars(col1), cols = col2)  OR facet\_grid(col1 ~ col2)  + facet\_wrap(~ col4) | | | | | vars: coerce to factor  col1 ~ . for row only/ . ~ col2 for cols only  #facet\_wrap: create subplots based on col4 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Visualisa-tion | When inspecting data: what type of variation occurs within each variables?  What type of covariation occurs between diff variables?  Which values are most common? Why? Which values are outliers? Any unusual patterns in data? | | | | | | | Comparison based on addition  Comparison based on multiplication (can use log to convert \* to +) | | |
| Contingency tables | Display for 2 categorical variables. Each entry in table is num of observations. Can use stacked bar plot to visualise  2 variables are indep if the conditional proportions for one of them are identical at each category of the other | | | | | | | | | |
| Difference of proportion = | | | | Ranges from -1 to 1  0 -> no association | | | | | |
| Relative risk = | | | | Ranges from 0 to ∞  1 -> no association  Preferable to use relative risk when both proportions are close to 0 | | | | | |
| Let p be propability of success, 1-p be probability of failure  The odds of success is =  If proportion is close to 0 or 1 -> use log transformation  Log-odds = log  Odds ratio = (for 2 groups) | | | | Odds = 0 -> P(success) = 0  Odds = 1 -> P(success) = 0.5  Odds = ∞ -> P(success) = 1  Log-odds = -∞ -> P(success) = 0  Log-odds = 0 -> P(success) = 0.5  Log-odds = ∞ -> P(success) = 1 | | | | | |
| ggplot(df) + geom\_bar(aes(x=col1, fill=col2), position=position\_fill(reverse=TRUE))  library(ggmosaic)  ggplot(df) + geom\_mosaic(aes(x=product(col1), fill=col2)) | | | | | | #to stack plots  #mosaic charts similar to stacked bar charts, but size of rectangles = counts instead of proportions | | | |
| Under independence | | library(GGally)  ggtable(df, "col1", "col2", cells="prop", fill="std.res") #heatmap displaying proportions. Color based on whether cells have higher/lower counts than expected under assumption of indep  ### to calculate proportions  observed\_table <- df %>% count(col1, col2) %>% pivot\_wider(names\_from = "col1", values\_from="col3") %>% arrange(desc(col2))  col\_tmp <- colSums(observed\_table[,-1]); row\_tmp <- rowSums(observed\_table[,-1])  expected\_proportion <- data.frame(row\_tmp/sum(row\_tmp) %\*% t(col\_tmp/sum(col\_tmp))) \* 100 | | | | | | | | |
| Pivot tables/ heatmap | ggplot(df, aes(x=col1, y = col2)) + geom\_tile(aes(fill= col3)) = geom\_text(aes(label = round(col3, 0)) + scale\_fill\_gradient | | | | | | | scale\_fill\_gradient/ scale\_fill\_gradient2/ scale\_fill\_gradientn: for cts color palette | | |
| Color Palette | df %>% mutate(new\_col = cut(col3, c(2000, 3000, 5000, 8000)) + gg[;pt + geom\_tile + scale\_fill\_brewer(palette = 2) #OR palette = "Oranges" | | | | | | | scale\_fill\_brewer for discrete colors | | |
| RColorBrewer  Color separated by: hue (red, green, blue), value (light, dark), saturation (dull, vivid)  If want readers to identify which values are higher/lower than other values, then should use sequential values  if want viewer to identify ordering in 2 directions, should use diverging pattern w diff hues  Table on right show sequential colors, pastel, diverging | | | Background pattern  Description automatically generated with medium confidence | | | | | | |
| Heatmap | data(Cars93, package="MASS"); library(Hmisc); describe(Cars93)  cor <- select(Cars93, which(!sapply(Cars93, is.factor))) %>% cor(. , use="pairs")  cor\_df <- as.data.frame(cor, row.names=NULL) %>% mutate(var1 = row.names(cor)) %>% pivot\_longer(col1:col9, names\_to="var2", values\_to="correlation")  ggplot(cor\_df) + geom\_tile(aes(x=var1, y=var2, fill=correlation)) | | | | | | | | #compute correlation matrix  #add 2nd col for row names  #heatmap of correlation | |
| Hierarchical clustering | Do not require prior specification of num of clusters (unlike K-means)  Output of hierarchical clustering is a dendrogram  Can cluster variables (find similar variables) or cluster observations | | | | | | |  | | |
| Dissimilarity Measures | Btw individual observations/columns Let d be a binary fn of 2 observations. More common e.g. is Euclidean dist d(xi, xj) = , or L1-norm: d(xi, xj) = | | | | | | | | | |
| Btw groups: Need a linkage method to compute dissimilarity btw groups, G and H  Single linkage: closest pair of observation btw both groups, dS(G,H) =  Complete linkage: furthest pair of observation btw both groups, dC(G,H) =  Average linkage: average of all pairwise dissimilarities btw groups, dA(G,H) =  Ward linkage: minimise variance within groups. Suppose G is formed by merging G1 and G­2  Then dW(G,H) = | | | | | | | | | |
| Dendrogram | If 2 points merge at a smaller vertical dist -> 2 points are less dissimilar to one another  i.e. height of each split is proportional to the value of the intergroup dissimilarity btw its 2 child nodes | | | | | | | | | |
| hc <- hclust(as.dist((1 - cor)/2))  ord <- order.dendrogram(as.dendrogram(hc))  cor\_df2 <- mutate(cor\_df, var1 = factor(var1, levels=row.names(cor)[ord]),  var2 = factor(var2, levels=row.names(cor)[ord])  ggplot(cor\_df2) + geom\_tile(aes(x=var1, y = var2, fill=correlation)) +  scale\_fill\_gradient2() +  theme(axis.text.x = element\_text(angle=90, vjust=0,hjust=1)) | | | | | #use dist for original df, as.dist for correlation matrix | | | | |
| Multi-Dimensional Scaling | Suppose we have computed all pairwise dissimilarities d(xi, xj) btw our N high-dimensional vectors.  W choice of k, we seek values, z1,...,zN s.t. following fn is minimised: S(z1,...,zN) = | | | | | | | | | MDS not same as PCA  PCA maximises variance  Principal components are ordered and LC of original vectors |
| cars\_dist <- as.dist((1-cor)/2); mds2 <- MASS::sammon(cars\_dist, k = 2))  grps <- as.factor(cuttree(hc, k=5))  mds\_df <- data.frame(mds2$points) %>% mutate(label=row.names(mds2$points),  Cluster=grps) %>% rename("Var.1"="X1", "Var.2"="X2")  ggplot(mds\_df) + geom\_text(aes(x=Var.1, y=Var.2, label=label, col=Cluster, show.legend=FALSE) | | | | | | | | | # reduce observation to  #get 5 groups |
| Ladder of transformation | | Can transform x or y variable or both  Consider -y-2, -y-1, log y, y1/4, y1/2, y, y2, y3  geom\_point(aes(x=log10(x), y=y^2))  OR  geom\_point(aes(x=x,y=y)) + scale\_x\_log10() + scale\_y\_sqrt()  scale\_...: more restrictive | Diagram  Description automatically generatedDiagram  Description automatically generated | | | | | | | |
| Pairs of Plots | library(GGally)  select(df, `col1`:`col8`) %>% ggpairs | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |
|  |  | | | | | | |  | | |