R Notes

Ty Darnell

Contents

Part I

The Language

Chapter 1

Vectors

1.1 Vectors

Recycling: The automatic lengthening of vectors in certain settings

Filtering: The extraction of subsets of vectors

Vectorization: Where functions are applied element-wise to vectors

You cannot insert or delete elements of a vector, you have to reassign the vector to accomplish this.

1.2 Creating Vectors

create vector: c(1,2,3,...)

seq(from,to,by)

length.out instead of by to specify how many numbers you want

along.with takes the length from the length of this argument

rep(x,times,each=1)

sort(x,decreasing=FALSE)

1.3 Subsetting Vectors

use - remove indexes

$vec[-1]$ select all but first index

$vec[-length(vec)]$ select all but last index

1.4 Vectorized Operations

Vectorized: a function applied to a vector is actually applied individually to each element.
The code can take a vector of values as input and manipulate each value in the vector at the same time.

Vectorized operations can be simpler than a for loop

a*b
a[1:7]*b[1:7]

How to write vectorized code:

- Use vectorized functions to complete the sequential steps in your program
- Use logical subsetting to handle parallel cases.
  Try to manipulate every element in a case at once.

Chapter 2

Matrices and Arrays

2.1 Matrix

$matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3, byrow = FALSE)$

Row and Column Bindings: Use rbind or cbind to create a matrix

Matrix Dimensions: dim nrow ncol

use *diag* to find the values along the diagonal in a square matrix

use square brackets and a - to omit elements

Transpose of Matrix: $t(A)$

use **diag** to create an identity matrix: $diag(3)$ is a 3 by 3 identity matrix

Matrix Multiplication: $A\% * \%B$

***if a is $m \times n$ and B is $p \times q$, where $n = p$. The result will be size $m \times q$.

Matrix Inversion: $solve(A)$ A must be square, nonsingular

Determinant: $det(A)$

Cross Product: $crossprod(A)$

Outer Product: $outer(x, y, "FUN")$

Eigen Values and Vectors: $eigen(A)$

2.2 Array

A 3d array has 3 dimensions: row,column,layer

$array(data = 1:10, dim = c(r, c, l))$

each **layer** is a matrix with the row and column dimensions

Chapter 3

Non-Numeric Values

3.1 Logical Values

any returns TRUE if any logicals are TRUE

all returns TRUE if all logicals are TRUE

& and element-wise operator

&& and single comparison

| or element-wise operator

|| or single comparison

! not

Short versions are meant for element-wise: return multiple logicals

Long versions are meant for single comparison: return a single logical value

The result of any logical operator is a logical value

An **and** statement has a higher precedence than an **or** statement

which takes a logical vector and returns the indexes of the TRUE entries. $myvec[-which(myvec < 0)]$ omits the negative entries

$which(A > 25, arr.ind = T)$ gives row and column positions for a matrix

arr.ind treats the object as a matrix or an array rather than a vector

3.2 Characters

R treats a string as a single entity.

Use $nchar(x)$ to return the number of characters

concatenate: use *cat* or *paste*

cat outputs directly to the console but doesn't return anything

paste returns the concatenated string as an object

$substr(x, start, stop)$ takes a string and extracts the part between two character positions (inclusive)

$sub(pattern, replacement, x)$ searches string for a smaller string pattern and replaces first instance

gsub replaces every instance

3.3 Factors

Are used in categorical data

use $factor$ to convert to a factor

$factor(x, levels, ordered = T/F)$

levels: possible values of the factor

use *levels* to extract the levels as vector

use *cut* to **bin** (group) data into categories

the right argument determines if boundary levels are on the left or right

include.lowest includes the highest or lowest value, depending on whether right is T or F

labels argument labels the categories

$cut(x, breaks, right = T/F, include.lowest = T/F, labels)$

Chapter 4

Lists and Data Frames

4.1 Lists

Member Reference: Use double brackets to retrieve items from a list

List Slicing: single brackets, allows you to select multiple items at once

Names: use $names(namevector)$ to name the elements of a list

Use the *dollar operator* $ to preform member referencing

4.2 Data Frames

All vectors in a data frame must be of equal length

use $data.frame()$ to create

each row in a DF is called a **record**

each column is a **variable**

R's default behavior for character vectors passed to a DF is to convert each variable into a factor

set $stringsAsFactors$ argument to FALSE to prevent this

add rows to DF using rbind add columns using cbind or dollar operator

subsetting records: $mydata\$sex == "M"$ returns T or F

$mydata[mydata\$sex == "M",]$ returns data for all variables for Males

use negative index to omit entries: $mydata[mydata\$sex == "M", -3]$

also can use character vector of variable names to pick returned columns

Chapter 5

Special Values, Classes and Coercion

5.1 Special Values

Infinity: Inf

$is.infinite()$ and $is.finite()$

NaN: Not a number

$is.nan()$

cannot use relational operators with NaN

Which: use $which()$ to convert logical values into index positions

use arr.ind argument of which to return values as an array

5.2 NA

Not Available: used for missing entries

$is.na()$

identifies NA and NaN

$na.omit()$ deletes all NAs, also applies to NaNs if elements are numeric

5.3 NULL

$NULL$ is used to explicitly define an empty entity different than a missing entity (NA)

$is.null$ use to check if the whole argument is null. True or False returned once.

5.4 Attributes

attributes: $attributes(x)$ returns a list of the explicit attributes of an object

can use the dollar operator to retrieve contents of attributes list

use dimnames argument with a matrix to name the rows and columns, this is an attribute

$attr(x, which = "dim")$ to obtain specific attribute.

5.5 Classes

An object's **class** is an attribute. All objects identify with at least one class

Class identification is called **inheritance**

Elementary R objects such as vectors, matrices, and arrays are implicitly classed, meaning the class is not identified with $attributes()$

You can always use $class()$to find the class of any object

$typeof()$ reports the type of data contained within an object

use *is-dot* functions to determine if object is a specific class or data type. Operates on object itself, returns a single logical value

5.6 Coercion

Converting from one object or data type to another is called **coercion**

Implicit coercion occurs automatically when elements need to be converted to another type in order for an operation to complete.

Explicit coercion can be achieved using *as-dot* functions

ex: $as.logical()$ or $as.numeric()$

Chapter 6

Base R Plotting

6.1 Plot

Plot: takes in two vectors and opens a textitgraphics device where it displays the result. If already open, R's default behavior is to refresh the device, overwriting with the new plot.

$plot(x, y)$ or $plot(mat)$ where mat is an nx2 matrix

6.2 Graphical Parameters

- **type**: tells R how to plot the coordinates (ex: stand-alone points or joined by lines)
- **main, xlab, ylab**: Options to include plot title, horizontal and vertical axis label
- **col**: colors to use for plotting points and lines. You can type in a number for the *col* argument or type in the corresponding character string. Use $colors()$ to dsiplay color choices
- **pch**: point character, what character to use for plotting individual points, takes an integer value between 1 and 25 or you can specify a single character to use for each point.
- **cex**: character expansion, controls size of plotted characters, takes integer value, 1 is default.
- **lty**: line type, specifies the type of line to use to connect the points. Takes an integer value between 1 and 6.
- **lwd**: line width, controls thickness of plotted lines, takes integer value, 1 is default.

- **xlim, ylim**: provides limits for the horizontal and vertical range of the plotting region. Each requires a vector of length 2. $c(lower, upper)$

6.3 Automatic Plot Types

To control plot type, specify a single character-valued option for the argument *type*

- "**p**" points only, the default value
- "**l**" lines
- "**b**" both points and lines
- "**o**" overplotting the points with lines, eliminates the gapes between points and lines
- "**n**" no points or lines, creating and empty plot. This can be useful for complicated plots that must be constructed in steps.

6.4 Adding Points, Lines and Text to an Existing Plot

These functions will add to a plot without refreshing or clearing the window:

- **points** Adds points $points(x[y >= 5], y[y >= 5])$
- **lines,abline,segments** Adds lines:
  - **lines** $lines(x, y)$ Draws lines connecting coordinates in x and y.
  - **abline** $abline(h = c(-5, 5), col = "red")$ Use h and v for horizontal and vertical.
  - **segements**
    $segments(x0 = c(5, 15), y0 = c(-5, 5), x1 = c(5, 15), y1 = c(5, 5))$
- **text** Writes text $text(x, y, labels = "")$ text is centered on the coordinates provided
- **arrows** Adds arrows $arrows(x0, y0, x1, y1)$
- **legend** Adds a legend
  $legend(placement, legend = c(labels), pch = c(1, 2), col = c(1, 2)$

Chapter 7

Plotting with ggplot2

7.1 Quick Plot

$qplot(x, y)$ Similar to base r $plot()$

ggplot2 plots are stored as objects, they have an underlying, static representation until you change them.

7.2 Geoms

Geometric Modifiers: geoms

$qplot(x, y, geom = "blank") + geom_point() + geom_line()$

Some arguments you can supply to geoms are: color, shape, linetype

ggplot2 is compatible with many of the base r graphical parameters.

type $??geom$ to obtain a list of geoms

7.3 Aesthetic Mapping with Geoms

A factor used to split a data set into categories is called a variable. You can use ggplot2 to map the variable to aesthetic values.
ex: $qplot(x, y, color = var, shape = var)$

use aes inside a geom to override default mapping

7.4 ggplot

$ggplot(data = mpg) + geom_point(aes(x = displ, y = hwy))$

Chapter 8

Reading and Writing Files

8.1 Reading Files

use *data*() to load a built in dataset

Read Table $read.table(file, header = T/F, sep = "", na.strings = " * ", stringsAsFactors = T/F)$

Use *setwd*(). Then all you need is the file name and extension as the file argument.

Use $list.files()$ to view textual output of the contents of any folder

$file.choose()$ opens up your folder and returns the path of the file you click on. Can use it for the file argument in read.table

Read CSV
$read.csv(file, header = T/F, na.strings =, stringsAsFactors = T/F)$

Use *scan*() and *readLines*() to parse files

$readLInes(con = connection, n = maxlinestoread)$

8.1.1 Web-Based Files

R can read files from a website with *read.table*()
Use the URL address for the file argument

8.1.2 Other File Formats

.dat files can be read using *read.table*, however they may contain extra information at the top that must be skipped using the *skip* argument.

skip: number of lines at the top of the file that should be ignored

8.2 Writing Files

write.table($x = datafile, file = newfile, sep = "", na = "", quote = T/F, row.names = T/F$)

If you only supply a file name, it will be created in the working directory.

quote determines whether to encapsulate each non-numeric entry in double quotes.

row.names asks whether to include the row names of the data source.

write.csv is a shortcut version designed for *.csv* files.

8.2.1 Plots and Graphics FIles

Use $jpeg()$ or $pdf()$ to create a file

Default width and height for pdf is inches, pixels in jpeg.

Write a plot to a file:

1. Create file: $jpeg(filename, height, width)$
2. Create plot
3. Close the file device: $dev.off()$

8.2.2 ggsave

$gg.save(filename)$ saves plot to a file.
Put the extension in the filename ex: "*myplot.png*"

8.3 Ad Hoc Object Read/Write Operations

use *dput* to write and *dget* to read other kinds of R objects.

$dput(x, file)$ $dget(file)$

Part II

Programming

Chapter 9

Calling Functions

9.1 Environments

R enforces scoping rules with virtual **environments**- separate compartments where data structures and functions are stored.

Global Environment: The compartment set aside for user-defined objects.

$ls()$ lists all the objects, variables, and user-defined functions in the current global environment.

Package Environment: Each package environment represents several subenvironments that control different aspects of a search for a given object.

Local Environments: Each time a function is called, a new environment is created, called the local environment or lexical environment. This contains all the objects and variables created in and visible to the function.

Partial Argument Matching: You can abbreviate argument names to avoid typing out the full argument.

9.2 Ellipsis

You can make a function more flexible in the number of arguments it can accept by using the ellipsis (...)

Two cases for ellipsis:

1. The ellipsis is the first argument for functions like data.frame, c, and list. The ellipsis represents the main ingredients in this case. The contents of the ellipsis is used in the resulting object or output.

2. It is the last argument for functions like plot. The ellipsis in this case is meant as a supplementary or potential repository of optional arguments. Used when the function calls other subfunctions that require additional arguments depending upon the originally supplied items.

Chapter 10

Conditions and Loops

10.1 Conditions

10.1.1 if Statements

An **if** statement runs a block of code only if a certain condition is true

The **condition** is placed in parentheses after the if keyword. The code is in braces after the condition.

```
if(condition) {
  code to run
}
```

highlight code use command + enter to run selection

use option + command + (b,e,r) to run from the beginning to the current line, the current line to the end, run all code

|| or statement that produces a single logical result
&& and statement that produces single logical result

10.1.2 else Statements

use an **else** statement if you want something different to happen when the condition is FALSE

```
if(condition){
  code to run if condition is TRUE
} else {
  code to run if condition is FALSE
}
```

10.1.3 Using ifelse for Element-wise Checks

ifelse can perform vector-oriented check. Checks each element

$ifelse(test, yes, no)$

- **test** takes a logical-valued data structure
- **yes** provides the element to return if the condition is satisfied
- **no** gives the element to return if the condition is FALSE

10.1.4 Nesting and Stacking Statements

You can nest if statements by putting them inside braces of preceding if

You can stack if statements using **else if**

10.1.5 The switch Function

switch: EXPR is the object of interest, the remaining arguments provide the values or operations to carry out based on the value of EXPR. The final untagged value is the result if EXPR doesn't match any of the preceding items.

$switch(EXPR, val1, val2, elseval)$

Integer version of *switch* works slightly differently, instead of using tags, the outcome is determined by positional matching

10.2 for Loops

```
for(loopindex in loopvector) {
    do any code in here
}
```

10.2.1 Nesting for Loops

When a for loop is nested in another for loop, the inner loop is executed in full before the outer loop loopindex is incremented, at which point the inner loop is executed all over again.

10.3 while Loops

```
while(loopcondition) {
    do any code in here
}
```

A *while* loop uses a single logical-valued loopcondition to control how many times it repeats. If the condition is TRUE, the code is executed then the loop condition is checked again. The loop terminates immediately once the condition is FALSE.

10.4 Implicit Looping with apply

apply takes a function and applies it to each margin of an array.

$apply(X, MARGIN, FUN)$

MARGIN = integer value of margin of X to operate on.
The margin index follows the positional order of the dimensions for matrices and arrays. 1 always refers to row, 2 to columns, 3 to layers, 4 to blocks and so on. $margin = c(1, 2)$ would apply to both rows and columns.

10.4.1 Other apply Functions

tapply: performs operations on subsets of the object of interest, where those subsets are defined in terms of one of more factor vectors.
$tapply(factorvector, INDEX, function)$

lapply: operates member by member on a list. Returns a list.
$lapply(list, function)$

sapply: returns same result as lapply but in array form.
$sapply(list, function)$

vapply: similar to *sapply*

mapply: operates on multiple vectors or lists at once.

All of the apply functions allow for additional arguments to be passed to FUN, most do this via an ellipsis.

10.5 Other Control Flow Mechanisms

10.5.1 Declaring break or next

Use **break** to preemptively terminate a loop.

Use **next** to advance to the next iteration and continue execution.

10.5.2 repeat

```
repeat{
    do any code in here
}
```

To stop repeating code inside the bracs you must use *break* inside the braced area (usually with an if statement).

Chapter 11

Writing Functions

11.1 The function Command

functionname < − function(arg1,arg2,arg3,...) {
    do any code in here when called
    return(returnobject)
}

When R encounters a **return** statement during execution, the function exits

If there's no *return* statement inside a function, the function will end when the last line has been run and will return the most recently assigned object.

11.2 Arguments

lazy evaluation: Expressions are evaluated only when they are needed. Arguments are accessed and used only at the point they appear in the function body.

missing checks the arguments of a function to see if all required arguments have been supplied. It takes a single argument tag and returns TRUE if the argument isn't found.

11.3 Specialized Functions

11.3.1 Helper Functions

Helper function: functions written and used specifically to facilitate the computations carried out by another function.

Can be **internally** or **externally** defined.

11.3.2 Disposable Functions

Disposable or Anonymous functions: function intended for use in a single instance without explicitly creating a new object in your global environment.

You can pass in a short, simple function as an argument.

11.3.3 Recursive Functions

Recursion is when a function calls itself.

An accessible stopping rule is critical to any recursive function.

Recursion is a good option when you don't know ahead of time how many times a function needs to be called to complete a task. Useful for sort and search algorithms.

Chapter 12

Exceptions, Timings, and Visibility

12.1 Exception Handling

When there's an unexpected problem during the execution of a function, R will notify you with either a **warning** or an **error**.

12.1.1 Formal Notifications: Errors and Warnings

Error forces the function to immediately terminate at the point it occurs.

Warning indicates that the function is being run in an atypical way but tries to work around the issue and continue executing.

Use *warning*("*message*") issue a warning.

Use *stop*("*message*") to throw an error

12.1.2 Catching Errors with try Statements

Use a **try** statement to attempt a function call and check whether it produces an error. This prevents halting execution.

$try(function, silent = T/F)$

12.2 Progress and Timing

Sys.time() tells you the current time

proc.time() is more detailed

system.time() times a single expression

12.3 Masking

Masking: one object or function will take precedence over the other and assume the object or function name, while the masked function must be called with an additional command.

You have to include the name of the package of the masked function in the call with a double colon.
$base :: sum(foo)$

12.3.1 Data Frame Variable Distinction

You can use:
$attach(dataframe)$
$detach(dataframe)$

To attach and detach the dataframe to the search path. This saves you from typing dateframe$var every time you want to access the variable.

Part III

Statistics, Probability, Math

Chapter 13

Elementary Statistics

13.1 Summary Statistics

$table(data)$ lists the frequency of the data. Use this to find the mode(s).

mean, median, min, max, range all do exactly what you expect.

Set **na.rm** argument to TRUE to remove NAs and NaNs from data

Use **tapply** function to compute statistics group by a specific categorical variable. You could find the mean by category for example.

$table(data)/nrow(data)$ calculates proportions

$round(data, digits)$

quantile: value computed from a collection of numeric measurements that indicates an observation's rank when compared to all the other observations.

$quantile(data, prob)$

$summary(data)$ gives you the five-number summary

Spread: var, sd, IQR functions

13.1.1 Covariance and Correlation

covariance expresses how much two numeric variables change together and whether the relationship is positive or negative.

correlation identifies the direction and strength of any association.

$cov(xdata, ydata)$
$cor(xdata, ydata)$

Chapter 14

Basic Data Visualization

14.1 Barplots

stacked: bars are split vertically
dodged: bars are broken up and placed beside each other

$data.freq \rightarrow table(data\$var)$
$barplot(data.freq,)$

$qplot(factor(data\$var), geom = "bar")$

14.2 Histogram

$hist(data\$var)$

$qplot(data\$var)$

14.3 Boxplots

$boxplot(data\$var)$

14.4 ggpairs

Use to create matrix of plots
$ggpairs(data, mapping = aes(col = var))$

Chapter 15

Probability

$cumsum(data)$ cumulative sum

$sample(x, size, replace = T/F, prob)$

15.1 Common PMFs

Each distribution has four core R functions tied to it:

- **d**-function: (*density*) provides specific mass or density function values
- **p**-function: (*probability*) provides cumulative distribution probabilities
- **q**-function: (*quantile*) provides quantiles
- **r**-function: (*random*) provides random variate generation

15.1.1 Binomial Distribution

dbinom, *pbinom*, *qbinom*, *rbinom*

- **dbinom**: provides mass function probabilities, $P(X = x)$
  $dbinom(x, size, prob)$
- **pbinom**: provides cumulative probability distribution, $P(X \leq x)$
  $pbinom(q, size, prob)$
- **pbinom**: provides inverse cumulative probability distribution (*quantile function*) gives x such that $P(X \leq x) = p$
  $qbinom(p, size, 1/6)$
- **rbinom**: used to generate any number of realizations of X given a specific binomial distribution.
  $rbinom(n, size, prob)$

15.1.2 Poisson Distribution

λ is the mean number of occurrences

- **dpois**: Provides the individual Poisson mass function probs, $P(X = x)$
  $dpois(x, lambda)$
- **ppois**: Provides the left cumulative probs, $P(X \leq x)$
  $ppois(q, lambda)$
- **qpois**: inverse of $ppois$
  $qpois(p, lambda)$
- **rpois**: produces random variates
  $rpois(n, lambda)$

15.1.3 Other Mass Functions

- **geometric**: *dgeom, pgeom, qgeom, rgeom*
  parameters: prob
- **negative binomial**: *dnbinom, pnbinom, qnbinom, rnbinom*
  parameters: size, prob
- **hypergeometric**: *dhyper, phyper, qhyper, rhyper*
  parameters: m,n,k
- **multinomial**: *dmultinom, rmultinom*
  parameters: size, prob

15.2 Common PDFs

15.2.1 Uniform

- **dunif**: returns heights for any value within the defined interval
  $dunif(x, min, max)$
- **punif**: returns areas under the function
  $punif(q, min, max)$
- **qunif**: $qunif(p, min, max)$
- **runif**: $runif(n, min, max)$

15.2.2 Normal

- **dnorm**: returns value of the normal curve at any x,
  $dnorm(xvals, mean, sd)$

- **pnorm**: returns left-side probabilities under the normal curve, $pnorm(q, mean, sd)$
- **qnorm**: returns quantile value, $qnorm(p, mean, sd)$
- **rnorm**: random variates of normal distribution, $rnorm(n, mean, sd)$

Use $qqnorm(data)$ to create a normal quantile-quantile plot

$qqline(data, col)$ adds the "optimal" line that the coordinates would lie along if the data were perfectly normal.

15.2.3 Student's t-distribution

dt, pt, qt, rt
first argument is x,q,p,n respectively. Second argument is df.

15.2.4 Exponential

λ is the **rate** parameter of the distribution

- $dexp(x, rate)$
- $pexp(q, rate)$
- $qexp(p, rate)$

15.2.5 Other Density Functions

- **chi-squared**: models sums of squared normal variates
  dchisq, pchisq, qchisq, rchisq
  first argument is x,q,p,n respectively. Second argument is df.
- **F-distribution**: used to model ratios of two chi-squared random variables
  df, pf, qf, rf
  first argument is x,q,p,n respectively. Then df1, df2.
- **gamma distribution**: generalization of both the exponential and chi-squared distributions
  dgamma, pgamma, qgamma, rgamma
  first argument is x,q,p,n respectively. Then shape and scale.
- **beta distribution** used in Bayesian modeling
  dbeta, pbeta, qbeta, rbeta
  first argument is x,q,p,n respectively. Then shape1 and shape2

Chapter 16

Math

16.1 Probability and Functions

16.1.1 Functions

f <- $function(x)\ x^2$
defines the function f by $f(x) = x^2$

$if\ (x > 0)\ x^2\ else\ x^3$
piecewise function

Plotting a Function

1. **Provide dummy dataset**
   p <- $ggplot(data.frame(x = c(0, 10)), aes(x))$
2. **Define Function**
3. $p + stat\_function(fun = fun.1)$

Plotting Multiple Functions

$p + stat\_function(fun = fun.1, aes(color = "fun.1")) + stat\_function(fun = fun.2, aes(color = "fun.2")) + scale\_color\_manual(values = c("red", "blue"))$

Put function names as colors inside $aes()$ and
use $scale\_color\_maual(values = c(color1, color2))$
to create the legend with the desired colors.

16.1.2 Probability

$pbirthday(k)$ solves the birthday problem for k people

$lfactorial(n)$ gives the $log(n!)$

use $prod(25 : 21)$ to multiple all items in a vector

16.2 Set Operations

$union(x, y)$

$intersection(x, y)$

$setdiff(x, y)$ set difference, consisting of all elements of x that are not in y

$setequal(x, y)$ test for equality between x and y

c %in%in y membership, testing whether c is an element of the set y

$chose(n, k)$ Number of possible subsets of size k chosen from a set of size n

$is.element(el, set)$

16.3 Sample

$sample(c(x1, x2, ...), size, replace = T/F)$

Sample randomly reorders the elements passed as the first argument.

$replicate(n, experiment)$ simulates n runs of experiment

16.4 Calculus

$optimize(f, lower, upper, maximum = T/F)$ maximizes f numerically over an interval

$uniroot(f, lower, upper)$ searches numerically for a zero in f over an interval

16.4.1 Derivative

$D(expression(x^n), "x")$

use *expression* to convert your function to an expression.

16.4.2 Integration

$integrate(f, lower, upper)$

$integrate(function(x)\{1/((x + 1) * sqrt(x))\}, 0, Inf)$

Chapter 17

ggfortify

17.1 ggdistribution

ggdistribution is a helper function to plot distributions using ggplot2

$ggdistribution(func, data, distribution\ specfic\ arguments)$

$func$ is the distribution argument.

ex: for normal distribution:
$ggdistribution(func = distributiondnorm, data, mean, sd)$

ex: binomial distribution:
$ggdistribution(dbinom, x = seq(0, 5), size = 5, prob = 1/2)$

17.2 autoplot density

$autoplot(density(rnorm(1:50)), fill =' green')$

17.3 autoplot Diagnostics for Linear Models

$autoplot(lm(data.x\ data.y, dataset))$

Part IV

Data Transformation

Chapter 18

dplyr

18.1 dplyr Basics

use $View()$ to see all o the columns

Five key dplyr functions:

1. **filter**: pick observations by their values
2. **arrange**: reorder the rows
3. **select**: pick variables by their names
4. **mutate**: create new variables with functions of existing variables
5. **summarize**: collapse many values down to a single summary

These can all be used with $group\_by()$ which causes the function to operate group by group instead of on the whole dataset

All verbs work similarly, the first argument is a data frame, the subsequent arguments describe what to do with the data, and the result is a new data frame

18.2 Filter

$filter(flights, month == 1, day == 2)$

second and subsequent arguments are the expressions that filter the data frame.

the %in% operator
month %in% c(11,12)

$between(x, left, right)$ shortut for $x \geq$ left & $x \leq$ right

18.3 arrange

$arrange(flights, year, month, day)$

sorts the data frame

Changes the order of of the rows, each additional column is used to break ties. Missing values are always sorted to the end.

use $desc()$ to reorder column in descending order

18.4 select

$select(flights, year, month, day)$

allows you to zoom in on the variables you're interested in

subsets the data frame

$select(flights, -(year : day))$ exclude columns

use $rename(flights, tail\_num = tailnum)$ to rename a variable

18.4.1 select Helper Functions

$starts\_with("abc")$ matches names that begin with "abc"

$ends\_with("xyz")$ matches names that end with "xyz"

$contains("ijk)$ matches names that contain "ijk"

$matches("")$ selects variables that match a regular expression

$num\_range("x", 1 : 3)$ matches x1, x2, and x3

$everything()$ all variables, use to move variables to the start of the data frame

$one\_of(flights, "arr\_delay")$ variables in character vector

18.5 mutate

$mutate(flights, gain = arr\_delay - dep\_delay)$

Add new variables

use $transmute(flights, gain = arr\_delay - dep\_delay)$ if you only want to keep the new variables

18.5.1 Creation Functions

Functions must be vectorized to use with $mutate()$

arithmetic operators, modular arithmetic, logs, and logical comparisons all work

$lead()$ and $lag()$ allow you to refer to leading or lagging values

ranking functions such as $min\_rank()$

18.6 summarize

$summarize(flights, delay = mean(dep_delay, na.rm = T))$

collapse a data frame into a single row

use $group\_by()$ with $summarize()$ to get **grouped summaries**

$group\_by(flights, year, month, day)$ gives you the average delay per date

use the pipe %>% to combine multiple operations

count $n()$ takes no arguments and returns the size of the current group

$sum(!is.na(x))$ count of nonmissing values

Measures of position: $first(x)$, $nth(x, 2)$, $last(x)$

$count(var)$

use $ungroup()$ to remove grouping