Introduction to R

Introduction to R Objectives:

* Become familiar with the RStudio IDE, R syntax, and R data types
* Understand how to get help with R
* Become familiar with the R reference card
* Become familiar with Rmarkdown to generate reports

There are many software programs designed to store, summarize, visualize, or analyze data:

* Spreadsheets: Microsoft Excel, Google Sheets, etc.
* Data Analysis and Statistical programs: SAS, SPSS, Minitab
* Programming languages: Python, C, C++, Fortran, Java
* Database software: Microsoft Access, SAP, SQL databases

R is statistical software which can read data, manipulate data, summarize data, fit statistical models, create quality plots, and more. It is used by companies like Google, Twitter, and Microsoft, as well as in government agencies and at national labs.

# Setting Up

This guide will help set up R and RStudio on your machine. It is included for reference.

## Installing R

On Windows or OS X:

* Go to <http://www.r-project.org/>
* Click the CRAN link on the left, and pick a download site (0-Cloud is a good choice)
* Choose link based on your OS
* On Windows, choose the "base" subdirectory to install R.
* On OS X, choose the .pkg file to install R.

## Installing RStudio

* Browse to <https://www.rstudio.com/>
* Mouse over Products and click RStudio
* Choose RStudio Desktop
* Click Download RStudio Desktop
* Choose the installer appropriate for your platform

# Example: Tips data

## Goals

* Explore a real dataset using R
* Get the "flavor" of R for data management and exploration
* Don't focus on the code - it will be explained later and in much more detail

## Tips Dataset

A server recorded the tips they received over about 10 weeks, including several variables:

* Amount they were tipped
* Cost of the total bill
* Characteristics about the party (# people, gender, etc.)

Primary Question: How do these variable influence the amount being tipped?

Follow along using [Tips-Example.R](code/Tips-Example.R)

## First look: Reading in the data

Use R to look at the top few rows of the tips data set. First, load tip using read.csv:

tips <- read.csv("http://bit.ly/2iNqvKM")

Now, use the head function to look at the first 6 rows of the data:

head(tips)  
## total\_bill tip sex smoker day time size  
## 1 16.99 1.01 Female No Sun Dinner 2  
## 2 10.34 1.66 Male No Sun Dinner 3  
## 3 21.01 3.50 Male No Sun Dinner 3  
## 4 23.68 3.31 Male No Sun Dinner 2  
## 5 24.59 3.61 Female No Sun Dinner 4  
## 6 25.29 4.71 Male No Sun Dinner 4

How big is this data set and what types of variables are in each column?

str(tips)  
## 'data.frame': 244 obs. of 7 variables:  
## $ total\_bill: num 17 10.3 21 23.7 24.6 ...  
## $ tip : num 1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ...  
## $ sex : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 2 2 2 2 2 ...  
## $ smoker : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...  
## $ day : Factor w/ 4 levels "Fri","Sat","Sun",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ time : Factor w/ 2 levels "Dinner","Lunch": 1 1 1 1 1 1 1 1 1 1 ...  
## $ size : int 2 3 3 2 4 4 2 4 2 2 ...

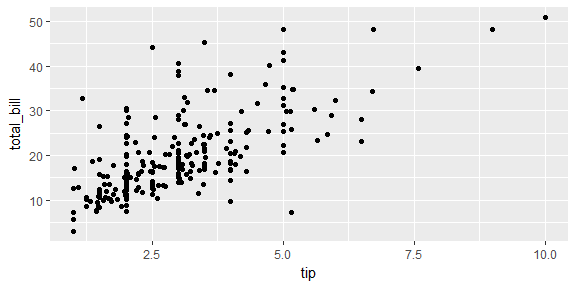
Get a summary of the values for each variable in tips:

summary(tips)  
## total\_bill tip sex smoker day   
## Min. : 3.07 Min. : 1.000 Female: 87 No :151 Fri :19   
## 1st Qu.:13.35 1st Qu.: 2.000 Male :157 Yes: 93 Sat :87   
## Median :17.80 Median : 2.900 Sun :76   
## Mean :19.79 Mean : 2.998 Thur:62   
## 3rd Qu.:24.13 3rd Qu.: 3.562   
## Max. :50.81 Max. :10.000   
## time size   
## Dinner:176 Min. :1.00   
## Lunch : 68 1st Qu.:2.00   
## Median :2.00   
## Mean :2.57   
## 3rd Qu.:3.00   
## Max. :6.00

What is the relationship between total bill and tip value? First, install and load ggplot2 (a graphics library):

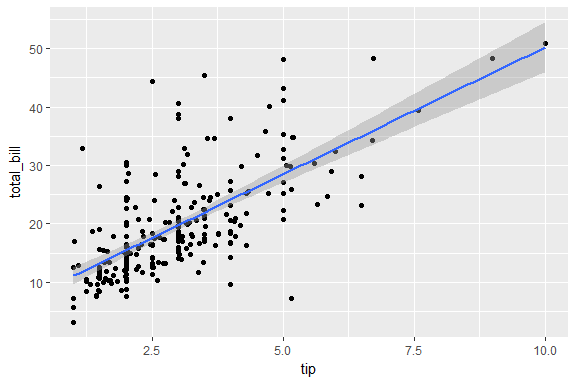
install.packages("ggplot2")  
library(ggplot2)

qplot(x = tip, y = total\_bill, geom = "point", data = tips)



Add linear regression line to the plot:

qplot(x = tip, y = total\_bill, geom = "point", data = tips) +   
 geom\_smooth(method = "lm")

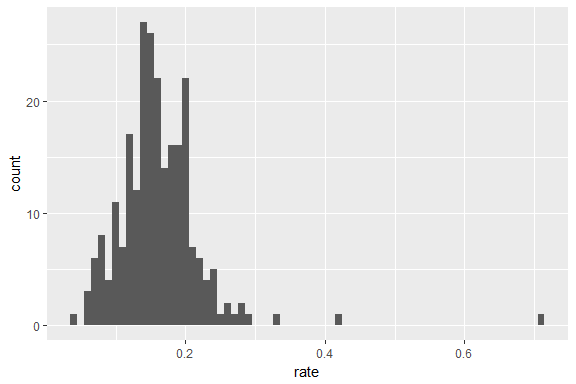


Tipping is generally done using a rule of thumb based on a percentage of the total bill. Make a new variable in the data set for the tipping rate = tip / total bill

tips$rate <- tips$tip / tips$total\_bill  
  
summary(tips$rate)  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.03564 0.12910 0.15480 0.16080 0.19150 0.71030

What is the distribution of tip rate in this dataset? Use R to create a histogram:

qplot(x = rate, data = tips, binwidth = .01)



One person tipped over 70%, who are they?

tips[which.max(tips$rate),]  
## total\_bill tip sex smoker day time size rate  
## 173 7.25 5.15 Male Yes Sun Dinner 2 0.7103448

Calculate the average tipping rate for men and women separately:

mean(tips$rate[tips$sex == "Male"])  
## [1] 0.1576505  
mean(tips$rate[tips$sex == "Female"])  
## [1] 0.1664907

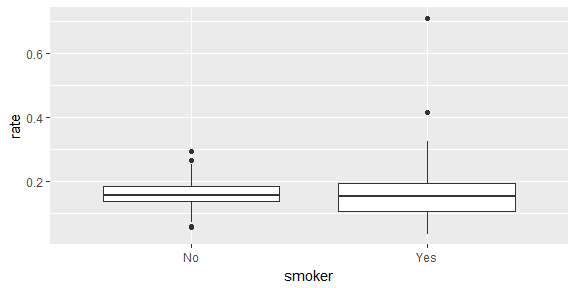
Is the difference between the two groups statistically significant? Calculate a t-test to examine whether the average tip rate for men is the same as the average tip rate for women.

t.test(rate ~ sex, data = tips)  
##   
## Welch Two Sample t-test  
##   
## data: rate by sex  
## t = 1.1433, df = 206.76, p-value = 0.2542  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.006404119 0.024084498  
## sample estimates:  
## mean in group Female mean in group Male   
## 0.1664907 0.1576505

The p-value is greater than 0.05, and the confidence interval contains 0: there is no evidence of a significant difference in tipping rate between men and women.

Compare two groups using a side-by-side boxplot. For instance, do smokers tip at a different rate than non-smokers?

qplot(x = smoker, y = rate, geom = "boxplot", data = tips)



##### Your Turn!

Try playing with chunks of code shown above to further investigate the tips data:

1. Get a summary of the total bill values
2. Make side by side boxplots of tip rates for different days of the week
3. Find the average tip value for smokers

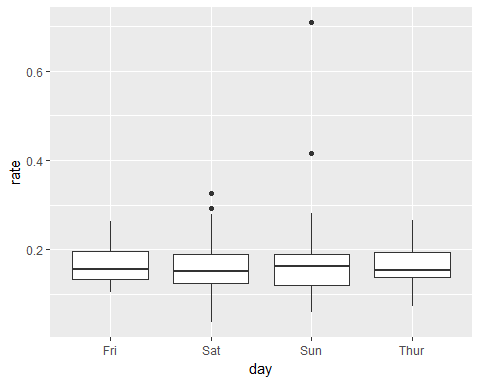
##### Solutions

1. Get a summary of the total bill values

summary(tips$total\_bill)  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 3.07 13.35 17.80 19.79 24.13 50.81

1. Make side by side boxplots of tip rates for different days of the week

qplot(day, rate, geom = "boxplot", data = tips)



1. Find the average tip value for smokers

mean(tips$tip[tips$smoker == "Yes"])  
## [1] 3.00871

# R Basics

## Getting Help in R

The help() function is useful for getting help with a function:

help(head)

The ? function also works:

?head

When searching for results online, it is helpful to use R + CRAN + <query> to get good results.

## R Reference Card

A copy of the R reference card is available at:

<http://cran.r-project.org/doc/contrib/Short-refcard.pdf>

This card contains short versions of the most common functions used in R.

## R as an Overgrown Calculator

R can perform simple mathematical operations.

# Addition and Subtraction  
2 + 5 - 1  
## [1] 6  
  
# Multiplication  
109\*23452  
## [1] 2556268  
  
# Division  
3/7  
## [1] 0.4285714

Here are a few more complex operations:

# Integer division  
7 %/% 2  
## [1] 3  
  
# Modulo operator (Remainder)  
7 %% 2  
## [1] 1  
  
# Powers  
1.5 ^ 3  
## [1] 3.375

# Exponentiation  
exp(3)  
## [1] 20.08554  
  
# Logarithms  
log(3)  
## [1] 1.098612  
log(3, base = 10)  
## [1] 0.4771213  
  
# Trig functions  
sin(0)  
## [1] 0  
cos(0)  
## [1] 1  
tan(pi/4)  
## [1] 1  
  
asin(0)  
## [1] 0  
acos(1)  
## [1] 0  
atan(1)  
## [1] 0.7853982

## Variables in R

Variables in R are created using the assignment operator, <-:

x <- 5  
R\_awesomeness <- Inf  
MyAge <- 21 #Haha

These variables can then be used in computation:

log(x)  
## [1] 1.609438  
MyAge ^ 2  
## [1] 441

### Rules for Variable Names

* Variable names can't start with a number
* Names are case-sensitive
* Common letters are used internally by R and should be avoided as variable names  
  c, q, t, C, D, F, T, I
* There are reserved words that R won't allow as variable names.  
  for, in, while, if, else, repeat, break, next
* R will let you use the name of a predefined function.  
  Try not to overwrite those!

#### Error messages

# Variable starts with a number  
1age <- 3  
## Error: <text>:2:2: unexpected symbol  
## 1: # Variable starts with a number  
## 2: 1age  
## ^

# Case Sensitive  
Age <- 3  
age  
## Error in eval(expr, envir, enclos): object 'age' not found

# Special Words can't be variable names  
for <- 3  
## Error: <text>:2:5: unexpected assignment  
## 1: # Special Words can't be variable names  
## 2: for <-  
## ^

#### A Cautionary Tale

# This is a VERY bad idea:  
  
T <- FALSE  
F <- TRUE  
  
T == FALSE  
## [1] TRUE  
F == TRUE  
## [1] TRUE  
  
rm(T, F) # Fix it!  
  
T == FALSE  
## [1] FALSE

Note: In R, T and F are shorthand for TRUE and FALSE respectively.

## Vectors

A variable can contain more than one value.  
A *vector* is a variable which contains a set of values of the same type.  
The c() (combine) function is used to create vectors:

y <- c(1, 5, 3, 2)  
z <- c(y, y)

R performs operations on the entire vector at once:

y / 2  
## [1] 0.5 2.5 1.5 1.0  
z + 3  
## [1] 4 8 6 5 4 8 6 5

Vectors can be created using the c() or rep() function. To create a vector of consecutive values, use the : function:

a <- 10:15  
a  
## [1] 10 11 12 13 14 15

### Modifying Vectors

Vectors can be modified using indexing:

# Get the total bill out of the tips dataset  
bill <- tips$total\_bill  
  
x <- bill[1:5]  
x  
## [1] 16.99 10.34 21.01 23.68 24.59  
x[1] <- 20  
x  
## [1] 20.00 10.34 21.01 23.68 24.59

Elements of a vector must all be the same type:

head(bill)  
## [1] 16.99 10.34 21.01 23.68 24.59 25.29  
bill[5] <- ":-("  
head(bill)  
## [1] "16.99" "10.34" "21.01" "23.68" ":-(" "25.29"

By changing a value to a string, all the other values were changed to strings as well.

##### Your Turn

Using the R Reference Card (and the Help pages, if needed), do the following:

1. Find out how many rows and columns the `iris' data set has. Figure out at least 2 ways to do this.  
   *Hint: "Variable Information" section on the first page of the reference card!*
2. Use the rep function to construct the following vector: 1 1 2 2 3 3 4 4 5 5  
   *Hint: "Data Creation" section of the reference card*
3. Use rep to construct this vector: 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

##### Solutions

1. Find out how many rows and columns the `iris' data set has. Figure out at least 2 ways to do this.

data(iris)  
  
# first way:   
nrow(iris)  
## [1] 150  
ncol(iris)  
## [1] 5  
  
# second way:   
dim(iris)  
## [1] 150 5  
  
# third way:   
str(iris) # look at the top line  
## 'data.frame': 150 obs. of 5 variables:  
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
## $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

1. Use the rep function to construct the following vector: 1 1 2 2 3 3 4 4 5 5

rep(c(1:5), each = 2)  
## [1] 1 1 2 2 3 3 4 4 5 5

1. Use rep to construct this vector: 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

rep(c(1:5), times = 3)  
## [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

###### End Solutions

Vectors can be manipulated by indexing.

Different elements of a vector can be extracted using brackets:

a[1]  
## [1] 10  
a[2]  
## [1] 11  
a[5]  
## [1] 14

Indexes can also be more complicated:

a[c(1, 3, 5)]  
## [1] 10 12 14  
a[1:5]  
## [1] 10 11 12 13 14

## Logical Values

* R has built in support for logical values
* TRUE and FALSE are built in. T (for TRUE) and F (for FALSE) are supported but can be modified
* Logicals can result from a comparison using

### Indexing with Logicals

Logical vectors can be used for indexing as well:

x <- c(2, 3, 5, 7)  
x[c(TRUE, FALSE, FALSE, TRUE)]  
## [1] 2 7  
x > 3.5  
## [1] FALSE FALSE TRUE TRUE  
x[x > 3.5]  
## [1] 5 7

#### Examples

# Get the rate variable out of the tips dataset  
rate <- tips$rate   
  
head(rate)  
## [1] 0.05944673 0.16054159 0.16658734 0.13978041 0.14680765 0.18623962  
  
sad\_tip <- rate < 0.10  
  
rate[sad\_tip]  
## [1] 0.05944673 0.07180385 0.07892660 0.05679667 0.09935739 0.05643341  
## [7] 0.09553024 0.07861635 0.07296137 0.08146640 0.09984301 0.09452888  
## [13] 0.07717751 0.07398274 0.06565988 0.09560229 0.09001406 0.07745933  
## [19] 0.08364236 0.06653360 0.08527132 0.08329863 0.07936508 0.03563814  
## [25] 0.07358352 0.08822232 0.09820426

## Data Frames

A collection of vectors, similar to a table in an Excel spreadsheet

* A data set is stored in a data frame
* Each column is a vector of the same length
* Each column can be a different type of data
* Each element in the vector/column has to have the same type of data
* columns can be accessed using $

tips is a data frame:

head(tips)  
## total\_bill tip sex smoker day time size rate  
## 1 16.99 1.01 Female No Sun Dinner 2 0.05944673  
## 2 10.34 1.66 Male No Sun Dinner 3 0.16054159  
## 3 21.01 3.50 Male No Sun Dinner 3 0.16658734  
## 4 23.68 3.31 Male No Sun Dinner 2 0.13978041  
## 5 24.59 3.61 Female No Sun Dinner 4 0.14680765  
## 6 25.29 4.71 Male No Sun Dinner 4 0.18623962

tips$sex shows the sex column of tips

tips$sex[1:20]  
## [1] Female Male Male Male Female Male Male Male Male Male   
## [11] Male Female Male Male Female Male Female Male Female Male   
## Levels: Female Male  
# Show the first 20 items in the sex column of tips

##### Your Turn

1. Find out how many people tipped over 20%.  
   *Hint: use the sum function on a logical vector to calculate how many TRUEs are in the vector:*

sum(c(TRUE, TRUE, FALSE, TRUE, FALSE))  
## [1] 3

1. **More Challenging**: Calculate the sum of the total bills of anyone who tipped over 20%

##### Solutions

1. How many people tipped over 20%?

sum(tips$rate > .2)  
## [1] 39

1. Sum of the total bills where the tip was over 20%

sum(tips$total\_bill[tips$rate > .2])  
## [1] 619.23

###### End Solutions

## Data Types in R

* Can use mode or class to find out information about variables
* str is useful to find information about the structure of your data
* Many data types: numeric, integer, character, Date, and factor most common

str(tips)  
## 'data.frame': 244 obs. of 8 variables:  
## $ total\_bill: num 17 10.3 21 23.7 24.6 ...  
## $ tip : num 1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ...  
## $ sex : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 2 2 2 2 2 ...  
## $ smoker : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...  
## $ day : Factor w/ 4 levels "Fri","Sat","Sun",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ time : Factor w/ 2 levels "Dinner","Lunch": 1 1 1 1 1 1 1 1 1 1 ...  
## $ size : int 2 3 3 2 4 4 2 4 2 2 ...  
## $ rate : num 0.0594 0.1605 0.1666 0.1398 0.1468 ...

Use mode and class to get more general information:

class(tips)  
## [1] "data.frame"  
mode(tips)  
## [1] "list"

tips is a data frame, which is a list of vectors that have the same length.

### Converting Between Types

Convert variables to a different type using the as series of functions:

size <- head(tips$size)  
size  
## [1] 2 3 3 2 4 4  
as.character(size)  
## [1] "2" "3" "3" "2" "4" "4"  
as.numeric("2")  
## [1] 2

## Some useful functions

There are a whole variety of useful functions to operate on vectors.

tip <- tips$tip  
x <- tip[1:5]  
length(x) # Number of elements of a vector  
## [1] 5  
sum(x) # Sum of elements in a vector  
## [1] 13.09

Using the basic functions it wouldn't be hard to compute some basic statistics.

# Homemade Statistical Functions  
(n <- length(tip))  
## [1] 244  
(meantip <- sum(tip) / n)  
## [1] 2.998279  
(standdev <- sqrt(sum((tip - meantip)^2) / (n - 1)))  
## [1] 1.383638

But these functions are already built in to R.

# Built in statistical functions  
mean(tip)  
## [1] 2.998279  
sd(tip)  
## [1] 1.383638  
summary(tip)  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 2.000 2.900 2.998 3.562 10.000  
quantile(tip, c(.025, .975))  
## 2.5% 97.5%   
## 1.1760 6.4625

## Element-wise Logical Operators

* & (elementwise AND)
* | (elementwise OR)

c(T, T, F, F) & c(T, F, T, F)  
## [1] TRUE FALSE FALSE FALSE  
c(T, T, F, F) | c(T, F, T, F)  
## [1] TRUE TRUE TRUE FALSE  
# Which are big bills with a poor tip rate?  
id <- (bill > 40 & rate < .10)  
tips[id,]  
## total\_bill tip sex smoker day time size rate  
## 103 44.30 2.5 Female Yes Sat Dinner 3 0.05643341  
## 183 45.35 3.5 Male Yes Sun Dinner 3 0.07717751  
## 185 40.55 3.0 Male Yes Sun Dinner 2 0.07398274

##### Your Turn

1. Read up on the dataset (?diamonds)
2. Plot price by carat (use qplot - go back to the motivating example for help with the syntax)
3. Create a variable ppc for price/carat. Store this variable as a column in the diamonds data
4. Make a histogram of all ppc values that exceed $10000 per carat.
5. Explore any other interesting relationships you find

##### Solutions

1. Plot price by carat (use qplot - go back to the motivating example for help with the syntax)

qplot(carat, price, data = diamonds)

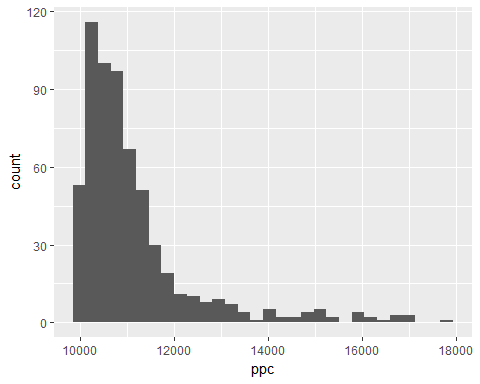


1. Create a variable ppc for price/carat. Store this variable as a column in the diamonds data

diamonds$ppc <- diamonds$price / diamonds$carat

1. Make a histogram of all ppc values that exceed $10000 per carat.

qplot(ppc, geom = "histogram", data = diamonds[diamonds$ppc > 10000,])  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



1. Explore any other interesting relationships you find

# Data Structures

## Data Frames

* Data Frames are the work horse of R objects
* Structured by rows and columns and can be indexed
* Each column is a specified variable type
* Columns names can be used to index a variable
* Advice for naming variable applies to editing columns names
* Can be specified by grouping vectors of equal length as columns

Data frames can be indexed in several ways:

* Elements indexed similar to a vector using [ ]
* df[i,j] will select the element in the row and column
* df[ ,j] will select the entire column and treat it as a vector
* df[i ,] will select the entire row and treat it as a vector
* Logical vectors can be used in place of i and j used to subset the row and columns

New variables can be added to a data frame: 1. Create a new vector that is the same length as other columns 2. Append new column to the data frame using the $ operator 3. The new data frame column will adopt the name of the vector

### Example

The following demonstration will use Edgar Anderson's Iris Data:

flower <- iris

Select Species column (5th column):

head(flower[,5])  
## [1] setosa setosa setosa setosa setosa setosa  
## Levels: setosa versicolor virginica

Select Species column with the $ operator:

head(flower$Species)  
## [1] setosa setosa setosa setosa setosa setosa  
## Levels: setosa versicolor virginica

Logical indexing:

head(flower$Species == "setosa")  
## [1] TRUE TRUE TRUE TRUE TRUE TRUE  
  
head(flower[flower$Species=="setosa", ])  
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa

### Creating a Data Frame

Create a data frame using data.frame function

mydf <- data.frame(NUMS = 1:5,   
 lets = letters[1:5],  
 vehicle = c("car", "boat", "car", "car", "boat"))  
mydf  
## NUMS lets vehicle  
## 1 1 a car  
## 2 2 b boat  
## 3 3 c car  
## 4 4 d car  
## 5 5 e boat

Column names can be accessed with the names function. They can also be changed by assigning values to names(df), as shown below.

# Use the `names` function to set that first column to lowercase:  
names(mydf)[1] <- "nums"  
mydf  
## nums lets vehicle  
## 1 1 a car  
## 2 2 b boat  
## 3 3 c car  
## 4 4 d car  
## 5 5 e boat

##### Your Turn

1. Make a data frame with column 1: 1,2,3,4,5,6 and column 2: a,b,a,b,a,b
2. Select only rows with value "a" in column 2 using logical vector
3. mtcars is a built in data set like iris: Extract the 4th row of the mtcars data.

##### Solutions

1. Make a data frame with column 1: 1,2,3,4,5,6 and column 2: a,b,a,b,a,b

mydf <- data.frame(col1 = 1:6, col2 = rep(c("a", "b"), times = 3))  
  
mydf  
## col1 col2  
## 1 1 a  
## 2 2 b  
## 3 3 a  
## 4 4 b  
## 5 5 a  
## 6 6 b

1. Select only rows with value "a" in column 2 using logical vector

mydf[mydf$col2 == "a",]  
## col1 col2  
## 1 1 a  
## 3 3 a  
## 5 5 a

1. Extract the 4th row of the mtcars data.

data(mtcars)  
  
mtcars[4,]  
## mpg cyl disp hp drat wt qsec vs am gear carb  
## Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1

###### End Solutions

## Lists

* Lists are a structured collection of R objects
* R objects in a list need not be the same type
* Create lists using the list function
* Lists indexed using double square brackets [[ ]] to select an object

### Example

Create a list containing a matrix and a vector:

mylist <- list(matrix(letters[1:10], nrow = 2, ncol = 5),  
 seq(0, 49, by = 7))  
mylist  
## [[1]]  
## [,1] [,2] [,3] [,4] [,5]  
## [1,] "a" "c" "e" "g" "i"   
## [2,] "b" "d" "f" "h" "j"   
##   
## [[2]]  
## [1] 0 7 14 21 28 35 42 49

Use indexing to select the second list element:

mylist[[2]]  
## [1] 0 7 14 21 28 35 42 49

##### Your Turn

1. Create a list containing a vector and a 2x3 data frame
2. Use indexing to select the data frame from your list
3. Use further indexing to select the first row from the data frame in your list

##### Solutions

1. Create a list containing a vector and a 2x3 data frame

mylist <- list(vec = 1:6, df = data.frame(x = 1:2, y = 3:4, z = 5:6))

1. Use indexing to select the data frame from your list

mylist[[2]]  
## x y z  
## 1 1 3 5  
## 2 2 4 6

1. Select the first row from the data frame in your list

mylist[[2]][1,]  
## x y z  
## 1 1 3 5

###### End Solutions

## Examining Objects

R objects can get very complicated very quickly (nesting lists inside of lists, for example). The following functions are useful to figure out the structure of objects in R.

* head(x) - View top 6 rows of a data frame
* tail(x) - View bottom 6 rows of a data frame
* summary(x) - Summary statistics
* str(x) - View structure of object
* dim(x) - View dimensions of object
* length(x) - Returns the length of a vector

### Example

Examine the first two values of an object by passing the n parameter to the head function:

head(diamonds, n = 2) # first 2 rows of diamonds data frame  
## # A tibble: 2 × 11  
## carat cut color clarity depth table price x y z ppc  
## <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl>  
## 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43 1417.391  
## 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31 1552.381  
tail(diamonds, n = 2) # last 2 rows of diamonds data frame  
## # A tibble: 2 × 11  
## carat cut color clarity depth table price x y z ppc  
## <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl>  
## 1 0.86 Premium H SI2 61.0 58 2757 6.15 6.12 3.74 3205.814  
## 2 0.75 Ideal D SI2 62.2 55 2757 5.83 5.87 3.64 3676.000

What's the structure of the object?

str(diamonds) # structure of diamonds data frame  
## Classes 'tbl\_df', 'tbl' and 'data.frame': 53940 obs. of 11 variables:  
## $ carat : num 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...  
## $ cut : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...  
## $ color : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...  
## $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...  
## $ depth : num 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...  
## $ table : num 55 61 65 58 58 57 57 55 61 61 ...  
## $ price : int 326 326 327 334 335 336 336 337 337 338 ...  
## $ x : num 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...  
## $ y : num 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...  
## $ z : num 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...  
## $ ppc : num 1417 1552 1422 1152 1081 ...  
str(mylist) # structure of mylist list  
## List of 2  
## $ vec: int [1:6] 1 2 3 4 5 6  
## $ df :'data.frame': 2 obs. of 3 variables:  
## ..$ x: int [1:2] 1 2  
## ..$ y: int [1:2] 3 4  
## ..$ z: int [1:2] 5 6

How does R summarize objects?

summary(diamonds) # summarize each column in diamonds  
## carat cut color clarity   
## Min. :0.2000 Fair : 1610 D: 6775 SI1 :13065   
## 1st Qu.:0.4000 Good : 4906 E: 9797 VS2 :12258   
## Median :0.7000 Very Good:12082 F: 9542 SI2 : 9194   
## Mean :0.7979 Premium :13791 G:11292 VS1 : 8171   
## 3rd Qu.:1.0400 Ideal :21551 H: 8304 VVS2 : 5066   
## Max. :5.0100 I: 5422 VVS1 : 3655   
## J: 2808 (Other): 2531   
## depth table price x   
## Min. :43.00 Min. :43.00 Min. : 326 Min. : 0.000   
## 1st Qu.:61.00 1st Qu.:56.00 1st Qu.: 950 1st Qu.: 4.710   
## Median :61.80 Median :57.00 Median : 2401 Median : 5.700   
## Mean :61.75 Mean :57.46 Mean : 3933 Mean : 5.731   
## 3rd Qu.:62.50 3rd Qu.:59.00 3rd Qu.: 5324 3rd Qu.: 6.540   
## Max. :79.00 Max. :95.00 Max. :18823 Max. :10.740   
##   
## y z ppc   
## Min. : 0.000 Min. : 0.000 Min. : 1051   
## 1st Qu.: 4.720 1st Qu.: 2.910 1st Qu.: 2478   
## Median : 5.710 Median : 3.530 Median : 3495   
## Mean : 5.735 Mean : 3.539 Mean : 4008   
## 3rd Qu.: 6.540 3rd Qu.: 4.040 3rd Qu.: 4950   
## Max. :58.900 Max. :31.800 Max. :17829   
##   
summary(mylist) # summarize mylist - # values in each item in the list  
## Length Class Mode   
## vec 6 -none- numeric  
## df 3 data.frame list

What are the dimensions of the object?

dim(diamonds) # dimensions of diamonds data frame  
## [1] 53940 11  
dim(mylist) # mylist doesn't have dimensions because it isn't a rectangular object  
## NULL  
  
length(diamonds) # diamonds is a data frame with 10 columns (or really, a list with 10 vectors that are the same length)  
## [1] 11  
length(mylist) # mylist has 2 objects  
## [1] 2

##### Your Turn

1. View the top 8 rows of mtcars data
2. What type of object is the mtcars data set?
3. How many rows are in iris data set? (try finding this using dim or indexing + length)
4. Summarize the values in each column in iris data set

##### Solutions

1. View the top 8 rows of mtcars data

head(mtcars, n = 8)  
## mpg cyl disp hp drat wt qsec vs am gear carb  
## Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
## Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
## Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
## Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
## Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2  
## Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
## Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4  
## Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2

1. What type of object is the mtcars data set?

str(mtcars)  
## 'data.frame': 32 obs. of 11 variables:  
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...  
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...  
## $ disp: num 160 160 108 258 360 ...  
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...  
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...  
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...  
## $ qsec: num 16.5 17 18.6 19.4 17 ...  
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...  
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...  
## $ gear: num 4 4 4 3 3 3 3 4 4 4 ...  
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...

1. How many rows are in iris data set? (try finding this using dim or indexing + length)

dim(iris)  
## [1] 150 5

1. Summarize the values in each column in iris data set

summary(iris)  
## Sepal.Length Sepal.Width Petal.Length Petal.Width   
## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100   
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300   
## Median :5.800 Median :3.000 Median :4.350 Median :1.300   
## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199   
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800   
## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500   
## Species   
## setosa :50   
## versicolor:50   
## virginica :50   
##   
##   
##

###### End Solutions

## Working with Output from a Function

* Can save output from a function as an object
* Object is generally a list of output objects
* Can use items from the output for further computing
* Examine object using functions like str(x)

### Demo: Saving Output

Conduct a t-test using iris data to see if petal lengths for setosa and versicolor are the same. The t.test function can only handle two groups, so subset out the virginica species.

iris.subset <- iris[iris$Species != "virginica", ]  
t.test(Petal.Length ~ Species, data = iris.subset)  
##   
## Welch Two Sample t-test  
##   
## data: Petal.Length by Species  
## t = -39.493, df = 62.14, p-value < 2.2e-16  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -2.939618 -2.656382  
## sample estimates:  
## mean in group setosa mean in group versicolor   
## 1.462 4.260

Save the output of the t-test to an object:

tout <- t.test(Petal.Length ~ Species, data = iris.subset)

Look at the structure of the t-test object:

str(tout)  
## List of 9  
## $ statistic : Named num -39.5  
## ..- attr(\*, "names")= chr "t"  
## $ parameter : Named num 62.1  
## ..- attr(\*, "names")= chr "df"  
## $ p.value : num 9.93e-46  
## $ conf.int : atomic [1:2] -2.94 -2.66  
## ..- attr(\*, "conf.level")= num 0.95  
## $ estimate : Named num [1:2] 1.46 4.26  
## ..- attr(\*, "names")= chr [1:2] "mean in group setosa" "mean in group versicolor"  
## $ null.value : Named num 0  
## ..- attr(\*, "names")= chr "difference in means"  
## $ alternative: chr "two.sided"  
## $ method : chr "Welch Two Sample t-test"  
## $ data.name : chr "Petal.Length by Species"  
## - attr(\*, "class")= chr "htest"

Since this is simply a list, use regular indexing to access the p-value.

tout$p.value  
## [1] 9.934433e-46  
tout[[3]]  
## [1] 9.934433e-46

## Importing Data

It is generally necessary to import data in to R rather than just using built-in datasets.

* Tell R where the data is saved using setwd() (or an appropriate file path)
* Read in data using R functions such as:
  + read.table() for reading in .txt files
  + read.csv() for reading in .csv files
  + the readr package has "smarter" versions of these functions and may be more useful
* Assign the data to new R object when reading in the file

### Importing Data Demo

First, create a csv file. Use a text editor, excel... Then load it in:

littledata <- read.csv("PretendData.csv")

##### Your Turn

* Make 5 rows of data in an excel spreadsheet and save it as a *tab-delimited txt file*.
* Import this new .txt file into R with read.table. You may need to look at the help page for read.table in order to properly do this.

##### Solutions

[Excel Spreadsheet](./data/Fun%20Webcomics.xlsx) (also at <https://bit.ly/2hOYYYT>)

webcomics <- read.table("./data/FunWebcomics.txt", header = T)  
webcomics  
## Fun.Webcomics URL  
## 1 xkcd http://xkcd.com/  
## 2 sarah's scribbles http://www.gocomics.com/sarahs-scribbles  
## 3 the oatmeal http://theoatmeal.com/  
## 4 dinosaur comics http://www.qwantz.com/  
## 5 hyperbole and a half http://hyperboleandahalf.blogspot.com/

# Packages and Basic Programming

## R Packages

* Commonly used R functions are installed with base R
* R packages containing more specialized R functions can be installed freely from CRAN servers using function install.packages()
* After packages are installed, their functions can be loaded into the current R session using the function library()

### Finding R Packages

* How do I locate a package with the desired function?
* Google ("R project" + search term works well)
* R website task views to search relevent subjects: <http://cran.r-project.org/web/views/>
* ??searchterm will search R help for pages related to the search term
* sos package adds helpful features for searching for packages related to a particular topic

Handy R Packages:

* ggplot2: Statistical graphics
* dplyr/tidyr: Manipulating data structures
* knitr: integrate LaTeX, HTML, or Markdown with R for easy reproducible research

## Creating Functions

Code Skeleton:

foo <- function(arg1, arg2, ...) {  
 # Code goes here  
 return(output)  
}

Example:

mymean <- function(data) {  
 ans <- sum(data) / length(data)  
 return(ans)  
}

## If/Else Statements

Skeleton:

if (condition) {  
 # Some code that runs if condition is TRUE  
} else {  
 # Some code that runs if condition is FALSE  
}

Example:

mymean <- function(data) {  
 if (!is.numeric(data)) {  
 stop("Numeric input is required")  
 } else {  
 ans <- sum(data) / length(data)  
 return(ans)  
 }  
}

## Looping

* Reduce the amount of typing
* Let R do repetitive tasks automatically
* R offers several loops: for, while, repeat.

for (i in 1:3) {  
 print(i)  
}  
## [1] 1  
## [1] 2  
## [1] 3

### For Loops

tips <- read.csv("https://bit.ly/2iNqvKM")  
  
id <- c("total\_bill", "tip", "size")  
for (colname in id) {  
 print(colname)  
}  
## [1] "total\_bill"  
## [1] "tip"  
## [1] "size"  
  
for(colname in id) {  
 print(paste(colname, mymean(tips[, colname])))  
}  
## [1] "total\_bill 19.7859426229508"  
## [1] "tip 2.99827868852459"  
## [1] "size 2.56967213114754"

### While Loops

i <- 1  
while (i <= 5) {  
 print(i)  
 i <- i + 1  
}  
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5

##### Your Turn

1. Create a function that takes numeric input and provides the mean and standard deviation for the data (sd may be useful)
2. Add checks to your function to make sure the data is either numeric or logical. If it is logical convert it to numeric.
3. Loop over the columns of the diamonds data set and apply your function to all of the numeric columns.

##### Solutions

1. Create a function that takes numeric input and provides the mean and standard deviation for the data (sd may be useful)

myfun <- function(x) {  
 m <- mean(x)  
 s <- sd(x)  
 return(c(mean = m, sd = s))  
}

1. Add checks to your function to make sure the data is either numeric or logical. If it is logical convert it to numeric.

myfun <- function(x) {  
 if (is.logical(x)) {  
 x <- as.numeric(x)  
 }  
 if (!is.numeric(x)) {  
 warning("x is not logical or numeric. Cannot compute a mean or std. deviation.")  
 return(c(mean = NA, sd = NA))  
 }  
 m <- mean(x)  
 s <- sd(x)  
 return(c(mean = m, sd = s))  
}

1. Loop over the columns of the diamonds data set and apply your function to all of the numeric columns.

data(diamonds)  
diamondStats <- matrix(0, nrow = ncol(diamonds), ncol = 2,   
 dimnames = list(names(diamonds),   
 c("mean", "sd")))  
  
for(i in 1:ncol(diamonds)) {  
 diamondStats[i,] <- myfun(diamonds[[i]])  
}  
  
diamondStats  
## mean sd  
## carat 0.7979397 0.4740112  
## cut NA NA  
## color NA NA  
## clarity NA NA  
## depth 61.7494049 1.4326213  
## table 57.4571839 2.2344906  
## price 3932.7997219 3989.4397381  
## x 5.7311572 1.1217607  
## y 5.7345260 1.1421347  
## z 3.5387338 0.7056988

# R Markdown Basics

## Hello R Markdown!

Choose your output format!

### *Why* R Markdown?

* It's **simple**. Focus on writing, rather than copy/paste and formatting
* It's **flexible**. Markdown was created to simplify writing HTML, but thanks to [pandoc](http://pandoc.org/), Markdown converts to many different formats!
* It's **dynamic**. Find a critical error? Get a new dataset? Regenerate a report without copy/paste problems!
  + Automating reports made easy!
* Encourages **transparency**. Collaborators (including future you) will appreciate having the analysis & report integrated.
* Enables **interactivity/reactivity**. Allow your audience to explore the analysis (rather than passively read it).

### *What* is Markdown?

* Markdown is a particular type of markup language.
* Markup languages are designed to produce documents from plain text.
* Some of you may be familiar with *LaTeX*. This is another (less human friendly) markup language for creating pdf documents.
* *LaTeX* gives greater control, but it is restricted to pdf and has a **much** steeper learning curve.
* Markdown is becoming a standard. Many websites will generate HTML from Markdown (e.g. GitHub, Stack Overflow, reddit).

### *Who* is using R Markdown, and *for what*?

* The [R Markdown website](http://rmarkdown.rstudio.com/) is built with R Markdown.
* The [new edition of the ggplot2 book](https://github.com/hadley/ggplot2-book/) uses it.
* R package vignettes (for example, [I made this for plotly](http://cpsievert.github.io/plotly/pipe-dsl/)).
* [Many](http://lincolnmullen.com/projects/dh-r/index.html) great [tutorials](http://renkun.me/learnR/) use R markdown add-ons such as [Rgitbook](http://jason.bryer.org/Rgitbook/).
* People are [blogging with it](http://cpsievert.github.io/knitr-jekyll//2015/04/jekyll-tufte-servr.html).
* This course is built with markdown

## *R* Markdown

* Straight from the [R Markdown home page](http://rmarkdown.rstudio.com/):

R Markdown is an authoring format that enables easy creation of dynamic documents, presentations, and reports from R. It combines the core syntax of **markdown** (an easy-to-write plain text format) **with embedded R code chunks** that are run so their output can be included in the final document. R Markdown documents are fully reproducible (they can be automatically regenerated whenever underlying R code or data changes).

##### Your Turn

Study the first page of the [R Markdown Reference Guide](https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf).

Yes, the *entire* markdown syntax can be described in one page!

Can you think of anything that is missing from the syntax (that you might want when creating documents)?

## RMarkdown details

Markdown doesn't natively support...

* Stuff in formal publications:
  + Figure/table referencing  
    (there are addins for this functionality)
  + Picture resizing (for word docs)
* Many, many appearance related things
  + image/figure alignment
  + coloring
  + fonts

There is hope...

* Complex formatting using HTML/LaTeX markup, but don't expect it to convert between output formats.
* There are many efforts to extend Markdown  
  (but, then again, keeping it simple is the point!)
* More features are being added daily
* Create or use templates for better control over formatting

##### Your Turn

Have a look at R Markdown presentations and templates.

**Pro tip**: run devtools::install\_github("rstudio/rticles") to get more templates

### Format Options

The stuff at the top of the .Rmd file (called *yaml front matter*) tells **rmarkdown** what output format you want.

---  
title: "Untitled"  
date: "May 16, 2016"  
output: html\_document  
---

In this case, when "Knit HTML" is clicked, RStudio calls rmarkdown::render("file.Rmd", html\_document()). Default values can be changed (see the [source of this presentation](https://raw.githubusercontent.com/srvanderplas/NPPD-Analytics-Workshop/master/01.Introduction/01.slides.Rmd)).

## Code Chunks

A code chunk is a concept borrowed from the [knitr](http://yihui.name/knitr/) package (which, in turn, was inspired by [literate programming](http://en.wikipedia.org/wiki/Literate_programming)). In .Rmd files, start/end a code chunk with three back-ticks.

```{r chunk1}  
1 + 1  
```

Want to run a command in another language?

```{r chunk2, engine = 'python'}  
print "a" + "b"  
```

### Options

There are a plethora of [chunk options](http://yihui.name/knitr/options/) in **knitr** (engine is one of them). Here are some that I typically use:

* echo: Show the code?
* eval: Run the code?
* message: Relay messages?
* warning: Relay warnings?
* fig.width and fig.height: Change size of figure output.
* cache: Save the output of this chunk (so we don't have to run it next time)?

##### Your Turn

Study the second page of the [R Markdown Reference Guide](https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf) and go back to the Hello R Markdown example we created.

* **Easy**: Modify the figure sizing and alignment.
* **Medium**: Add a figure caption.
* **Hard**: Can you create an animation? (Hint: look at the fig.show chunk option -- you might need to the **animation** package for this)

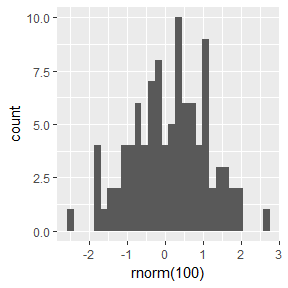
**Pro Tip**: Don't like the default chunk option value? Change it at the top of the document:

```{r setup2, verbatim = TRUE}  
knitr::opts\_chunk$set(message = FALSE, warning = FALSE)  
```

##### Solutions

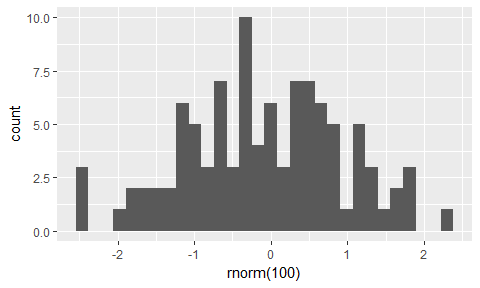
```{r, fig.align = "right", fig.width = 3, fig.height = 3, out.width = "50%"}  
qplot(rnorm(100))  
```

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



```{r, fig.cap = "Histogram of 100 samples from a normal distribution"}  
qplot(rnorm(100))  
```

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



Histogram of 100 samples from a normal distribution

```{r, fig.show = 'animate', ffmpeg.format = 'mp4'}  
samples <- seq(100, 500, 50)  
for (i in samples) {  
 print(  
 qplot(rnorm(i)) + ggtitle(sprintf("%d Samples from a Normal Dist", i))  
 )  
}  
```

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

video of chunk unnamed-chunk-96

## Formatting R output

Ugly:

m <- lm(mpg ~ disp, data = mtcars)  
summary(m) # output isn't very attractive  
##   
## Call:  
## lm(formula = mpg ~ disp, data = mtcars)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -4.8922 -2.2022 -0.9631 1.6272 7.2305   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 29.599855 1.229720 24.070 < 2e-16 \*\*\*  
## disp -0.041215 0.004712 -8.747 9.38e-10 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 3.251 on 30 degrees of freedom  
## Multiple R-squared: 0.7183, Adjusted R-squared: 0.709   
## F-statistic: 76.51 on 1 and 30 DF, p-value: 9.38e-10

Pretty:  
[pander](http://rapporter.github.io/pander/) is one great option.

library(pander)  
pander(m)

Fitting linear model: mpg ~ disp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Estimate | Std. Error | t value | Pr(>|t|) |
| **disp** | -0.04122 | 0.004712 | -8.747 | 9.38e-10 |
| **(Intercept)** | 29.6 | 1.23 | 24.07 | 3.577e-21 |

Ugly:

a <- anova(m)  
a  
## Analysis of Variance Table  
##   
## Response: mpg  
## Df Sum Sq Mean Sq F value Pr(>F)   
## disp 1 808.89 808.89 76.513 9.38e-10 \*\*\*  
## Residuals 30 317.16 10.57   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Pretty:

pander(a)

Analysis of Variance Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
| **disp** | 1 | 808.9 | 808.9 | 76.51 | 9.38e-10 |
| **Residuals** | 30 | 317.2 | 10.57 | NA | NA |

### Pander knows about a lot of stuff!

methods(pander)  
## [1] pander.anova\* pander.aov\*   
## [3] pander.aovlist\* pander.Arima\*   
## [5] pander.call\* pander.cast\_df\*   
## [7] pander.character\* pander.clogit\*   
## [9] pander.coxph\* pander.cph\*   
## [11] pander.CrossTable\* pander.data.frame\*   
## [13] pander.Date\* pander.default\*   
## [15] pander.density\* pander.describe\*   
## [17] pander.evals\* pander.factor\*   
## [19] pander.formula\* pander.ftable\*   
## [21] pander.function\* pander.glm\*   
## [23] pander.Glm\* pander.gtable\*   
## [25] pander.htest\* pander.image\*   
## [27] pander.irts\* pander.list\*   
## [29] pander.lm\* pander.lme\*   
## [31] pander.logical\* pander.lrm\*   
## [33] pander.manova\* pander.matrix\*   
## [35] pander.microbenchmark\* pander.mtable\*   
## [37] pander.name\* pander.nls\*   
## [39] pander.NULL\* pander.numeric\*   
## [41] pander.ols\* pander.orm\*   
## [43] pander.polr\* pander.POSIXct\*   
## [45] pander.POSIXlt\* pander.prcomp\*   
## [47] pander.randomForest\* pander.rapport\*   
## [49] pander.rlm\* pander.sessionInfo\*   
## [51] pander.smooth.spline\* pander.stat.table\*   
## [53] pander.summary.aov\* pander.summary.aovlist\*  
## [55] pander.summary.glm\* pander.summary.lm\*   
## [57] pander.summary.lme\* pander.summary.manova\*   
## [59] pander.summary.nls\* pander.summary.polr\*   
## [61] pander.summary.prcomp\* pander.summary.rms\*   
## [63] pander.summary.survreg\* pander.summary.table\*   
## [65] pander.survdiff\* pander.survfit\*   
## [67] pander.survreg\* pander.table\*   
## [69] pander.tabular\* pander.ts\*   
## [71] pander.zoo\*   
## see '?methods' for accessing help and source code

##### Your Turn

* Look through the list of pander methods. Can you apply any of the methods that we haven't discussed? We just saw pander.lm and pander.anova.