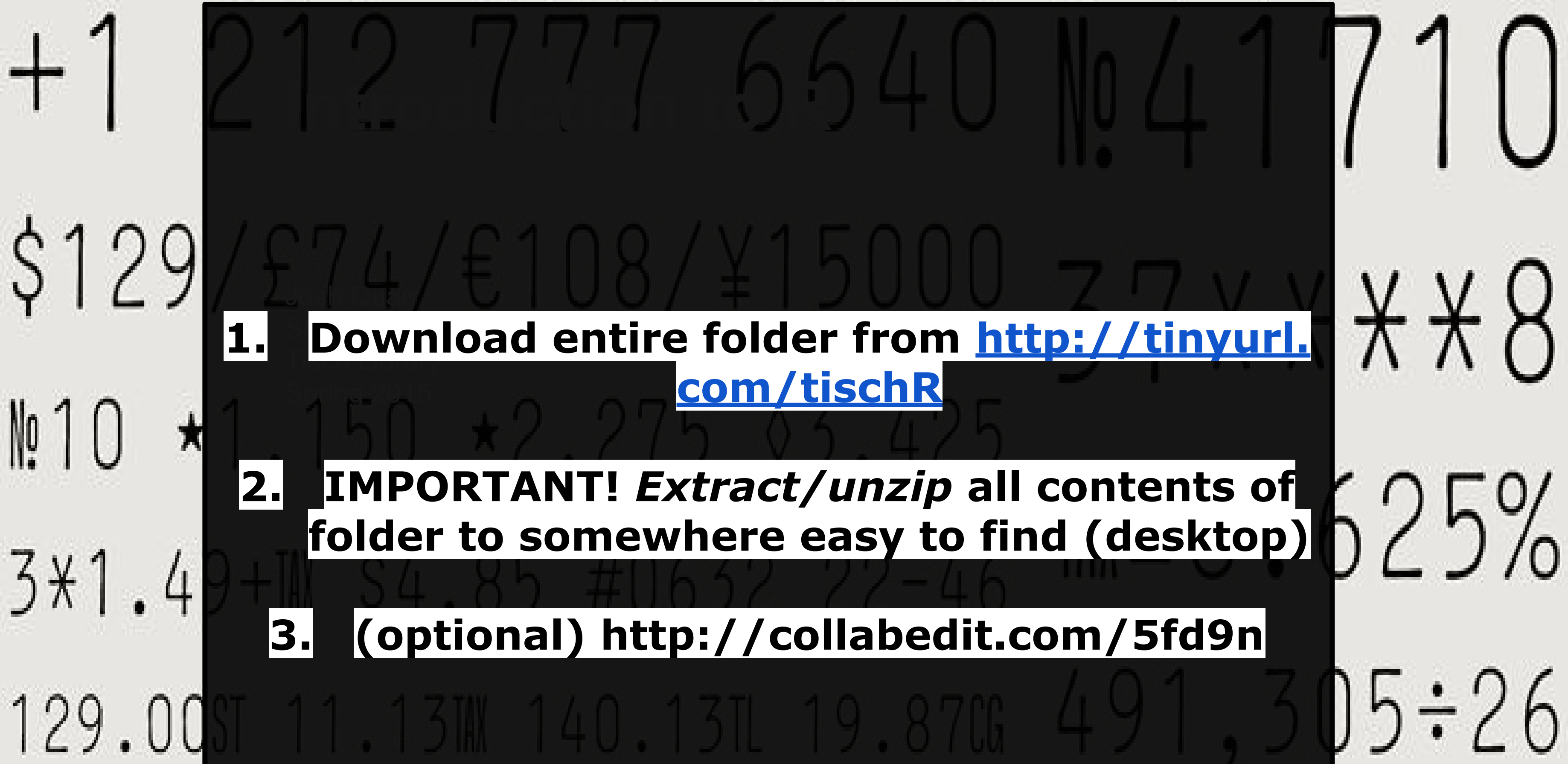

+1 212 777 6640 №41710
$129/£74/€108/¥15000
**8
№10 ★1,150 ★2,275 ◊3,425
3*1.49+TAX $4.85 #0632 22-46
625%
129.00ST 11.13TAX 140.13TL 19.87CG 491,305÷26

1. **Download entire folder from http://tinyurl.com/tischR**

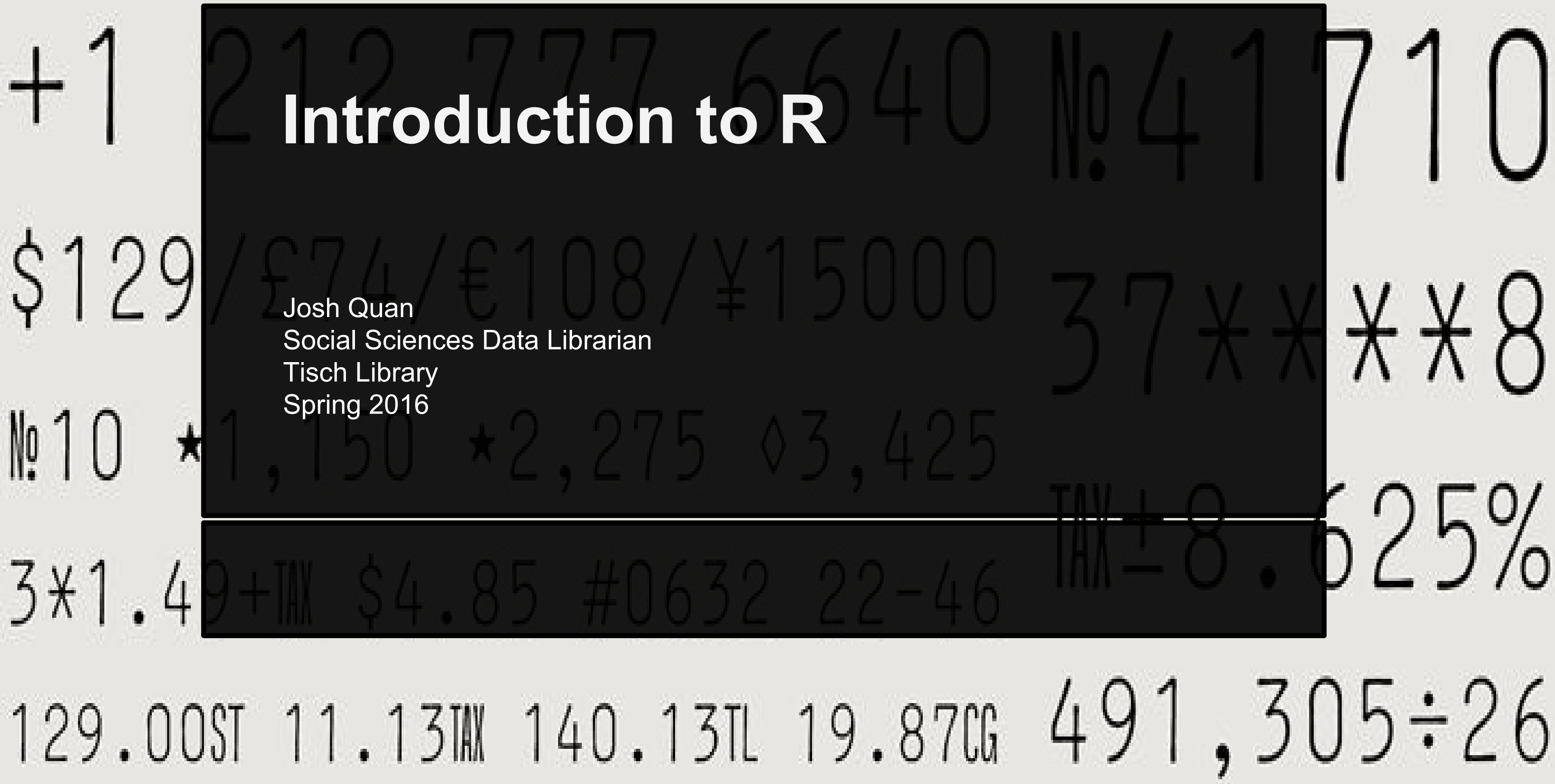
2. **IMPORTANT! *Extract/unzip* all contents of folder to somewhere easy to find (desktop)**

3. **(optional) http://collabedit.com/5fd9n**

Source: http://www.typography.com/images/whatsInsidePageImages/numbers-revenue.png

Introduction to R

Josh Quan
Social Sciences Data Librarian
Tisch Library
Spring 2016


+1 212 777 6640 №41710
$129/£74/€108/¥15000 37****8
№10 ★1,150 ★2,275 ◊3,425 TAX±8.625%
3*1.49+TAX $4.85 #0632 22-46
129.00ST 11.13TAX 140.13TL 19.87CG 491,305÷26

Source: http://www.typography.com/images/whatsInsidePageImages/numbers-revenue.png

What we will cover in the hour:

R Studio Environment, Workflow, and Basics

Objects, Structures and Data types

Subsetting

R packages

Baby names dataset

Linear Modeling

Getting help

R

Why learn R?

- Second most commonly used scripting language, after SQL[1]

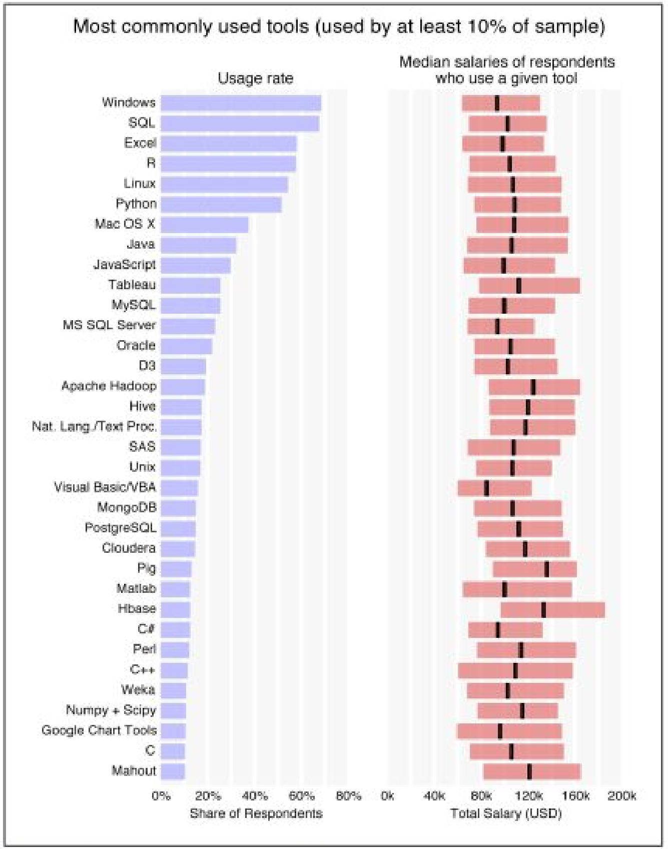


Figure 1-10. Most commonly used tools

[1] 2014 O'Reilly Data Science Salary Survey (http://www.oreilly.com/data/free/files/2014-data-science-salary-survey.pdf)

Why learn R?

- Second most commonly used scripting language, after SQL[1]
  - Third most commonly used data tool after SQL and Excel[1]

- R is popular in both industry and academia

- R is open source and has a large, active community

- Reproducible code = Reproducible workflow = Transparency

- In addition to advancing the needs of mathematicians/statisticians in ways that other tools can’t, R is a bridge to “Data Science” competencies
  - Data manipulation, visualization, and machine learning

[1] 2014 O’Reilly Data Science Salary Survey (http://www.oreilly.com/data/free/files/2014-data-science-salary-survey.pdf)

Back to earth…

- R is not a data vault or spreadsheet. The easiest way to enter data in R is to enter it somewhere else, then import it. To visually inspect tabular data, SPSS or Excel are still better options.

- R makes some ordinary tasks more difficult right out of the box.

- The learning curve for R is nontrivial. Being a data analyst != being a programmer.

- Being open source is a boon and curse.

- R is not meant to be a “general” programming language like Python.

To be successful with R

- Like learning any programming language, take your time and try to run the code in your head before you run it on your machine. Try to predict what will happen.

- Be patient.

- Think of a fun project that you actually would like to do and do it in R.

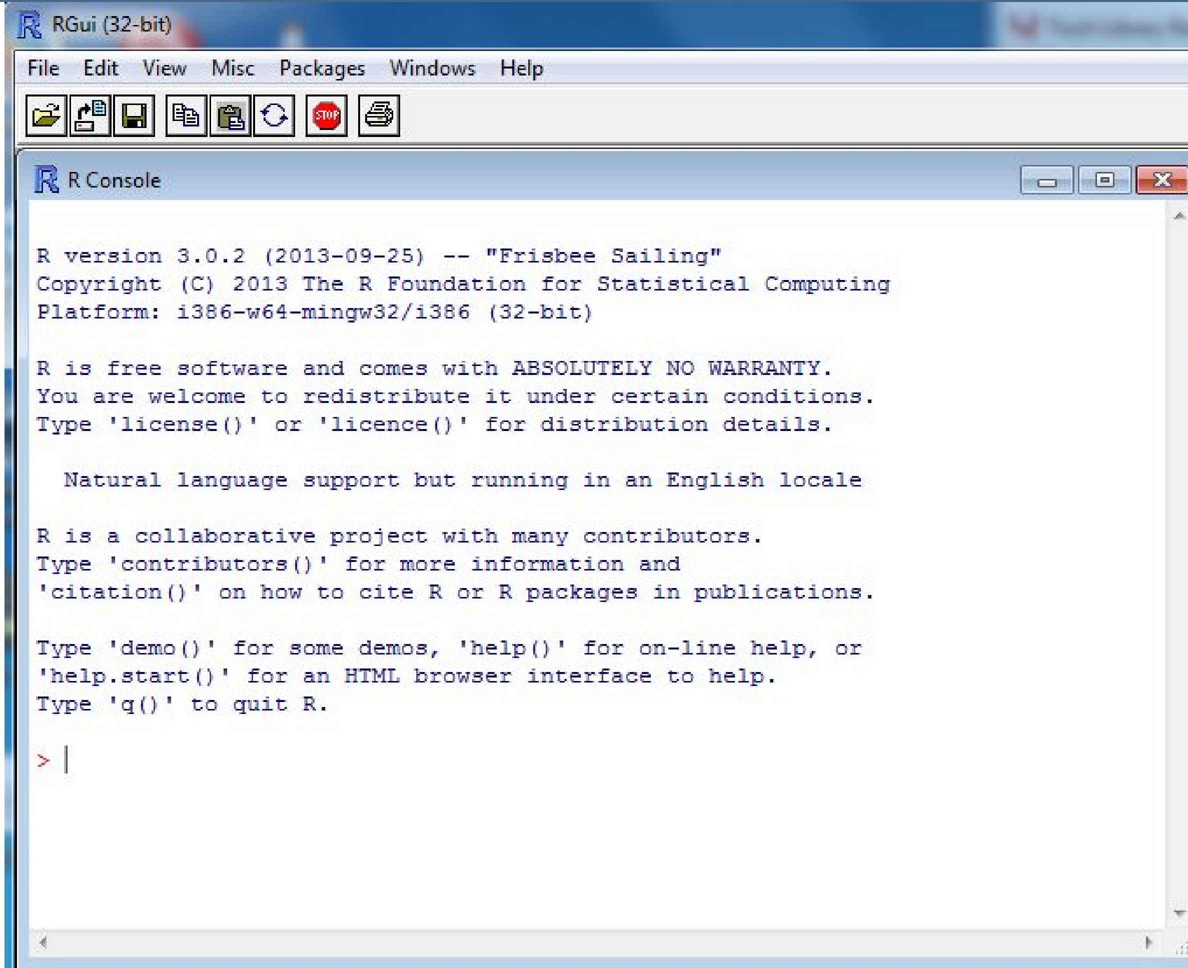
- Ask your friendly librarian about useful resources.

- Understand that unlike excel, there are many paths to the same solution in R. You need not learn them all but to troubleshoot effectively and ask for help it is worthwhile to understand how others might work with R (ie., subsetting)

- If you master the techniques and concepts in this workshop you've mastered 80% of R. The rest is identifying specific packages/methodologies that are relevant to your domain.

R Console

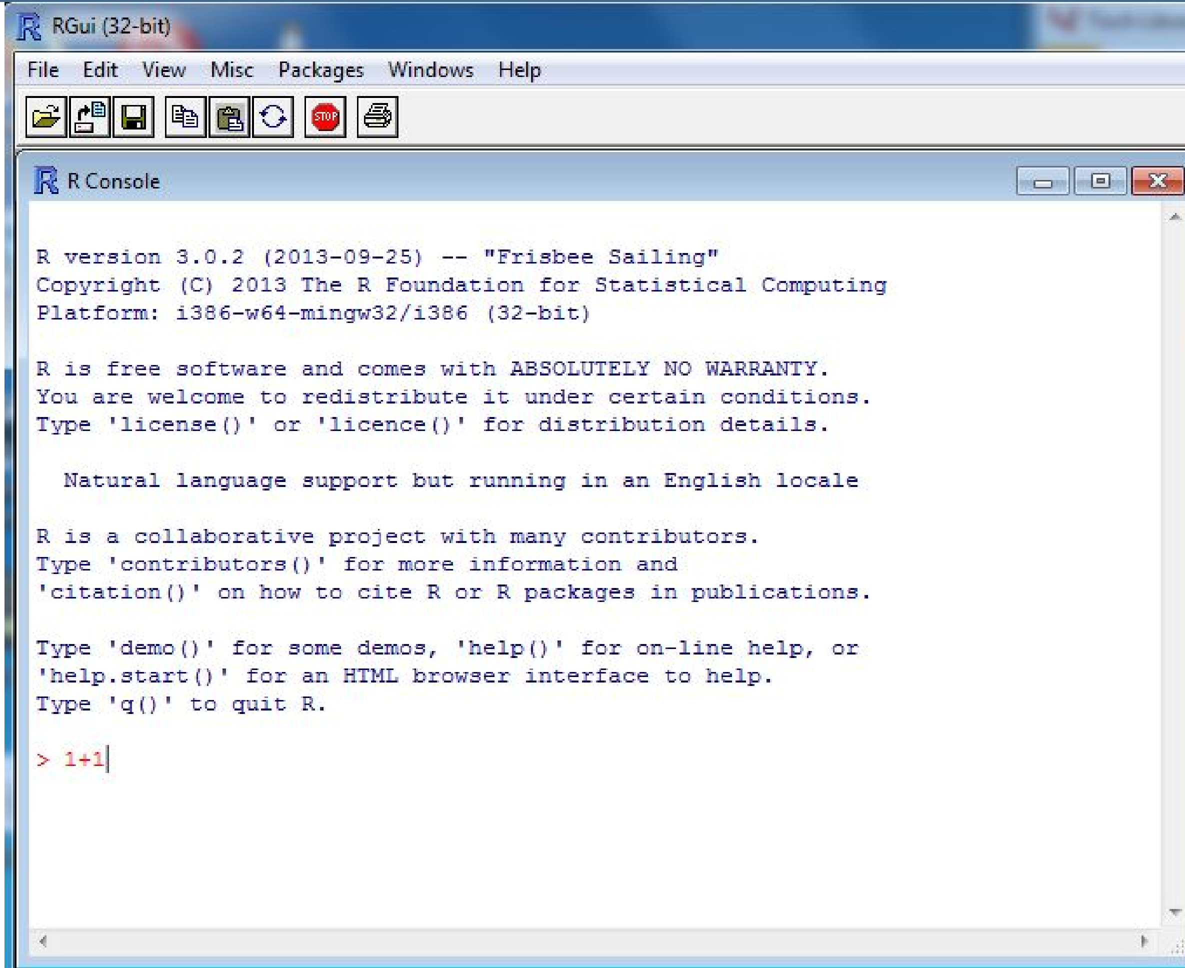
The console gives you a place to execute commands written in the R computer language.



```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

R Console

Type commands on the line that begins with a > sign (known as the prompt)



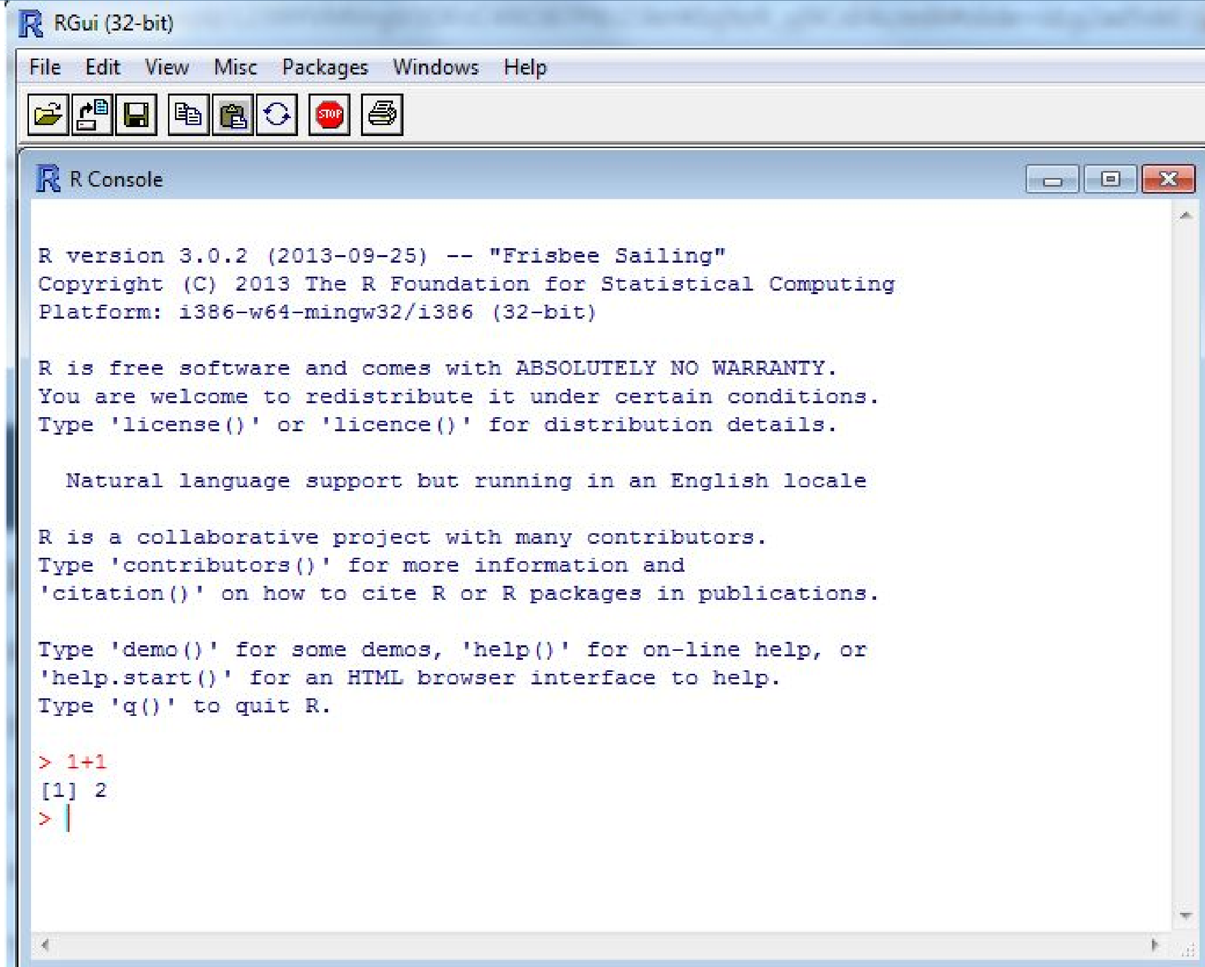
```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1+1
```

R Console

When you hit enter,
R will run your command
and display any output
below it

Output

New Prompt



```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1+1
[1] 2
> 
```

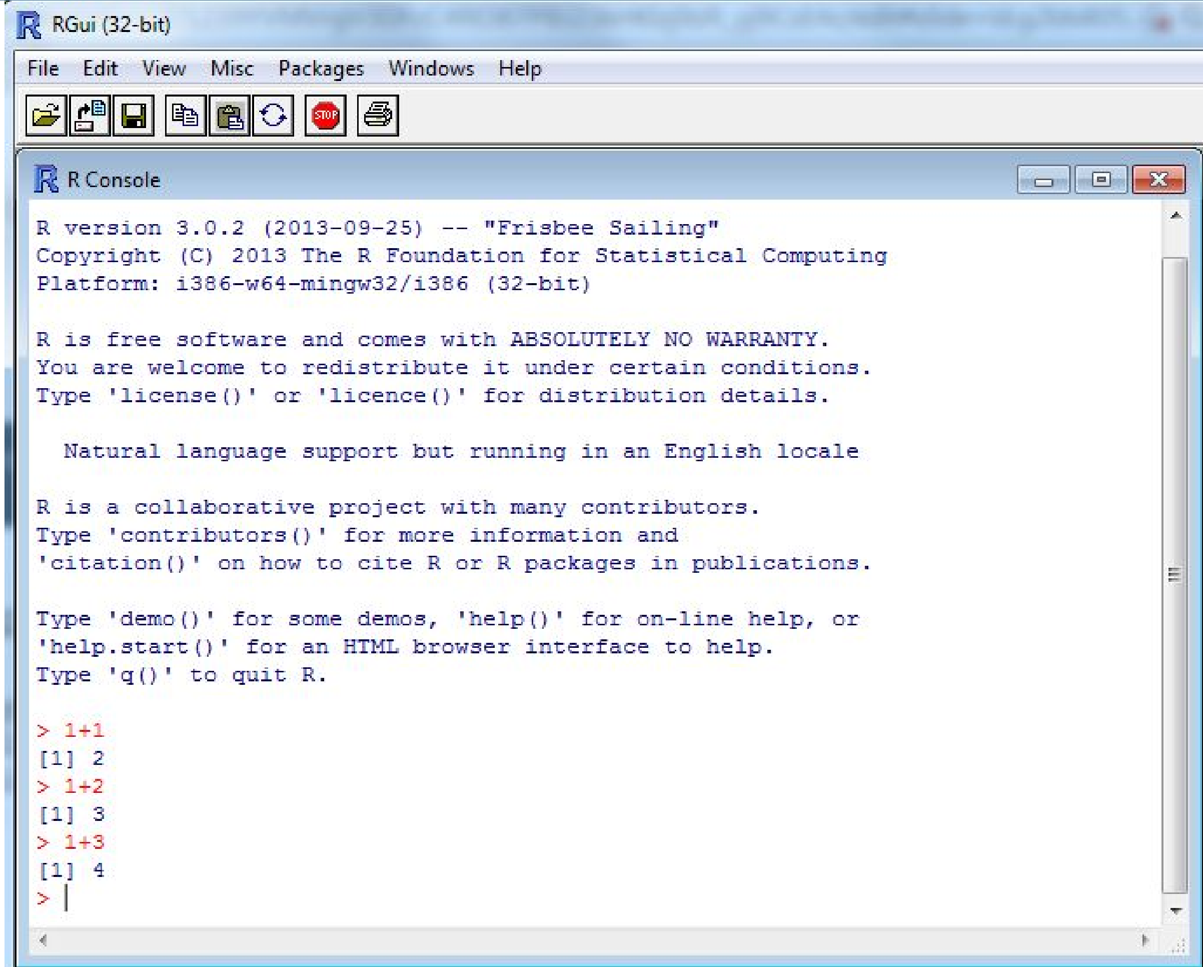
R Console

As you enter commands,
you accumulate a history
of past commands.

You can scroll through past commands by using the up arrow key on your keyboard.

R displays an index [1] next to the output.

Just ignore this.



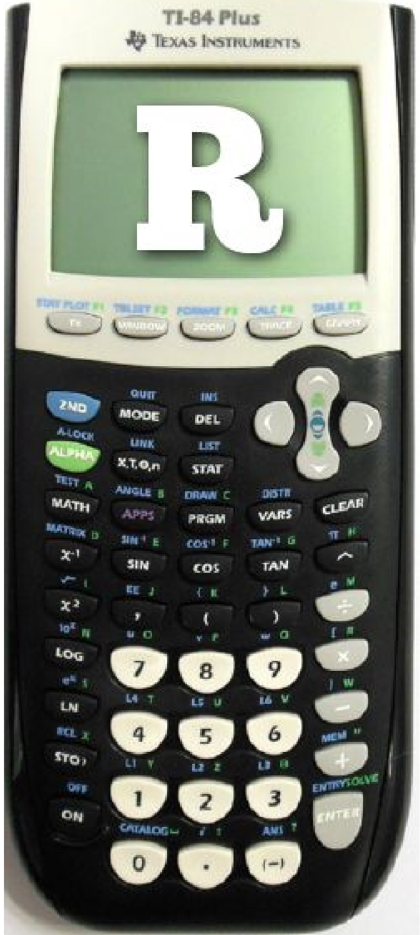
```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1+1
[1] 2
> 1+2
[1] 3
> 1+3
[1] 4
> 
```

R is like a fancy calculator

```
5 + 5
```

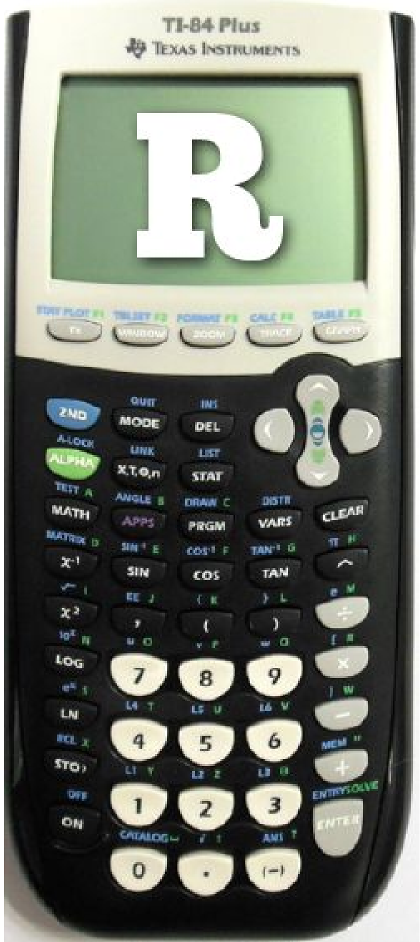
```
1 * 2
```

```
4 ^ 2
```



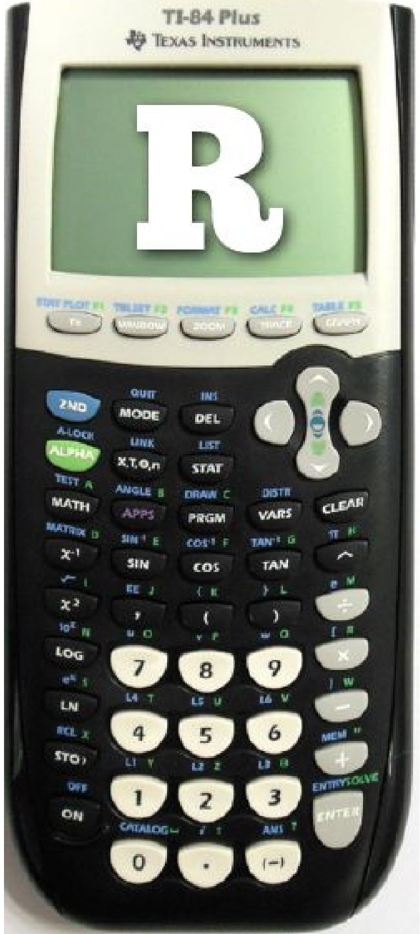
R is like a fancy calculator

```
5 + 5
# 10

1 * 2
# 2

4 ^ 2
# 16
```



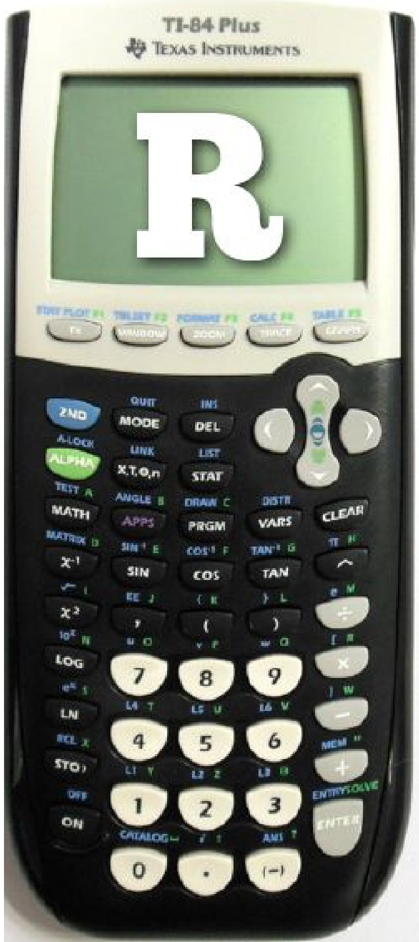
R can do algebra

```
a <- 1
b <- 2

a + b
```



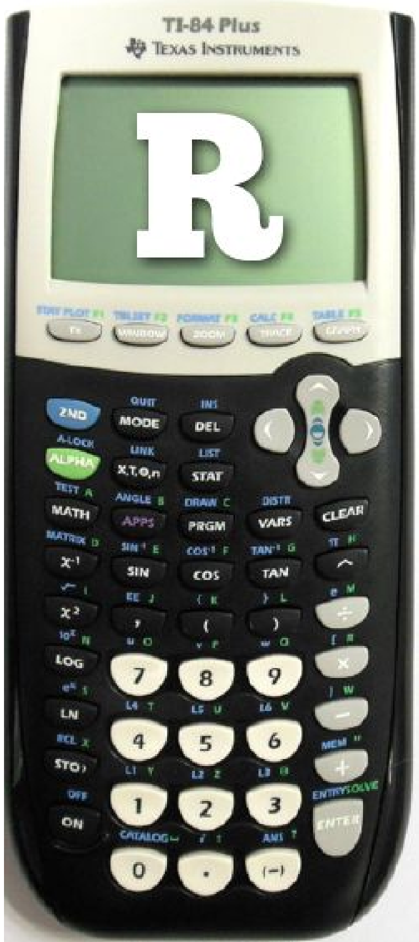
R can do algebra

```
a <- 1
b <- 2

a + b

# 3

A <- 3

a + b - A
```



R can do algebra

```
a <- 1
b <- 2

a + b

# 3

A <- 3

a + b - A

# 0
```



R Environment, Workflow, and Basics

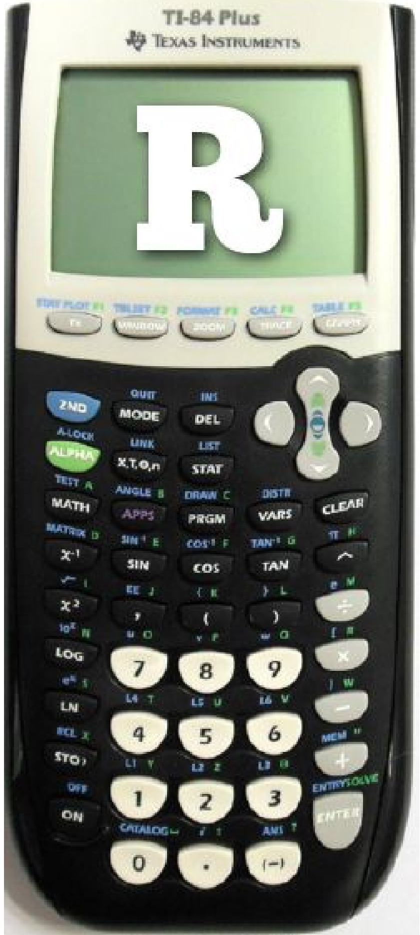
It has built in functions that let you do more sophisticated manipulations

```
round(3.145)
```

```
factorial(3)
```

```
sqrt(9)
```

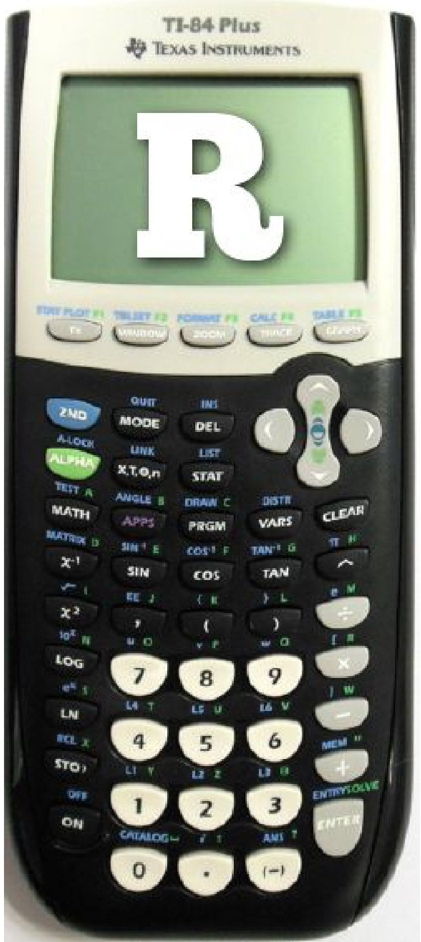
```
mean(c(7.5,8.2,3.1,5.6,10.9,4.6))
```



R Environment, Workflow, and Basics

It has built in functions that let you do more sophisticated manipulations

```
round(3.145)
# 3

factorial(3)
# 6

sqrt(9)
# 3

mean(c(7.5,8.2,3.1,5.6,10.9,4.6))
# 6.65
```

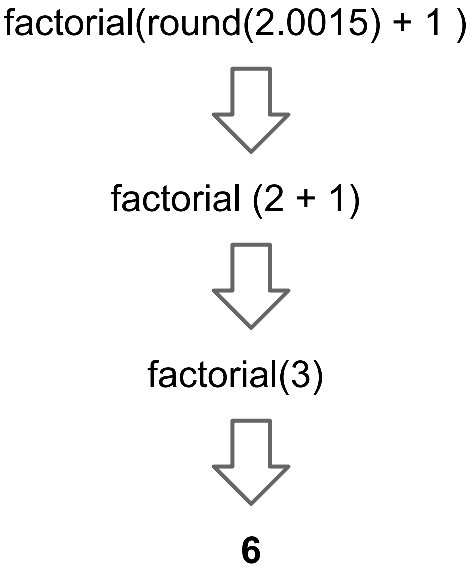


Your Turn

What do you think this will return?

```
factorial(round(2.0015) + 1)
```

R always works from the innermost parenthesis to the outermost (just like a calculator)



factorial(round(2.0015) + 1)

factorial (2 + 1)

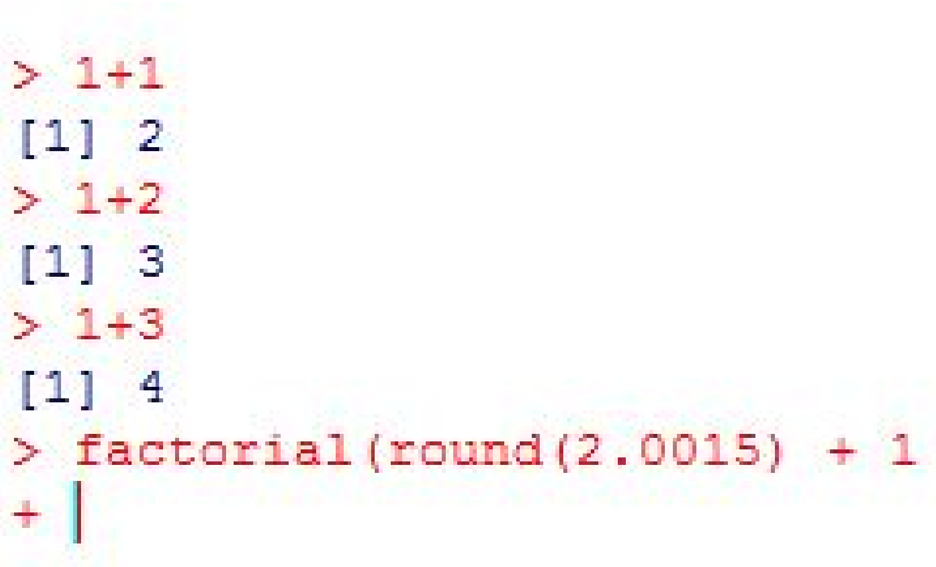
factorial(3)

6

+ prompt

if your prompt turns into a “+”,
R thinks you haven’t finished your previous command.

Either finish the command or press escape to start over.

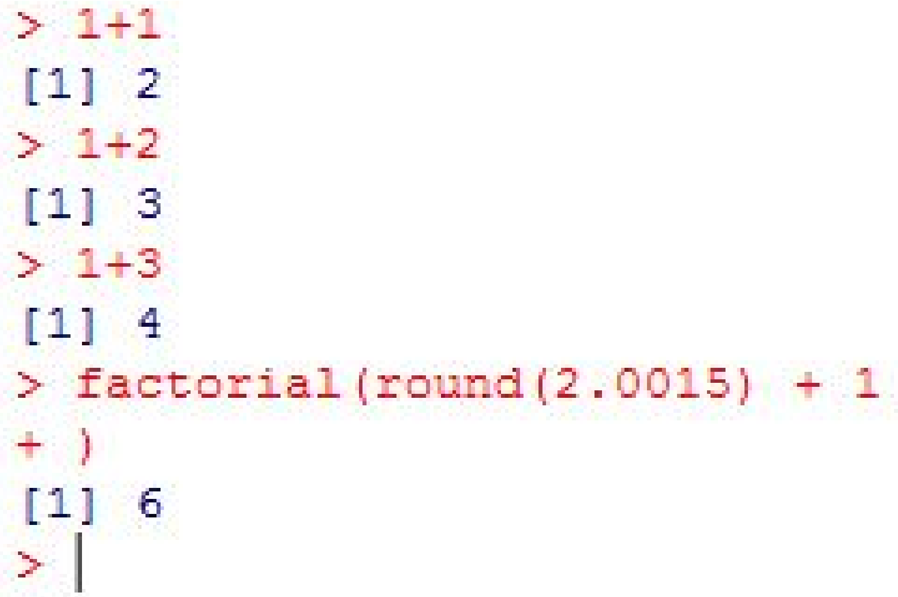


```
> 1+1
[1] 2
> 1+2
[1] 3
> 1+3
[1] 4
> factorial(round(2.0015) + 1
+ |
```

+ prompt

if your prompt turns into a “+”,
R thinks you haven’t finished your previous command.

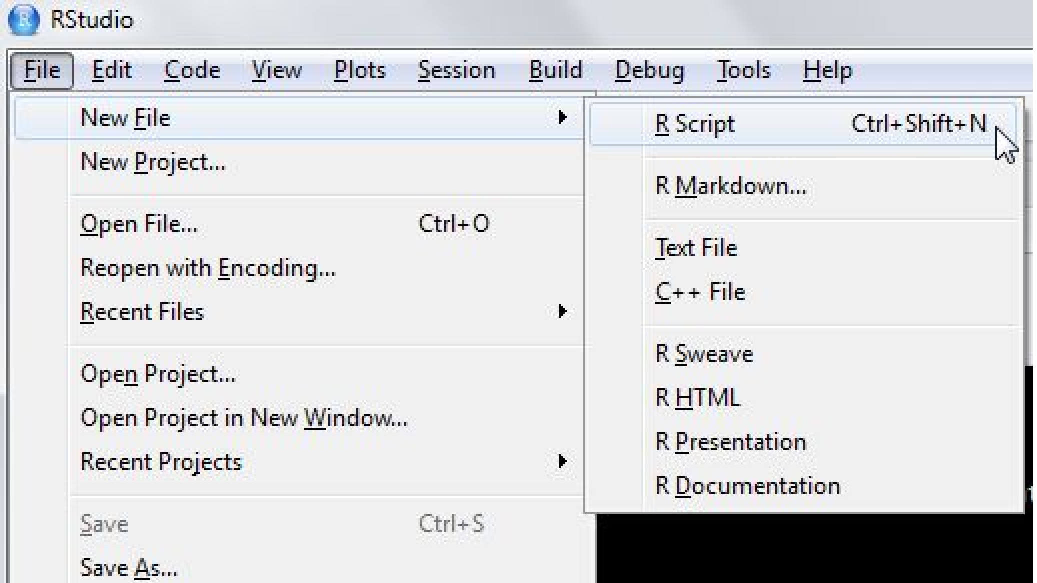
Either finish the command or press escape to start over.



```
> 1+1
[1] 2
> 1+2
[1] 3
> 1+3
[1] 4
> factorial(round(2.0015) + 1
+ )
[1] 6
> 
```

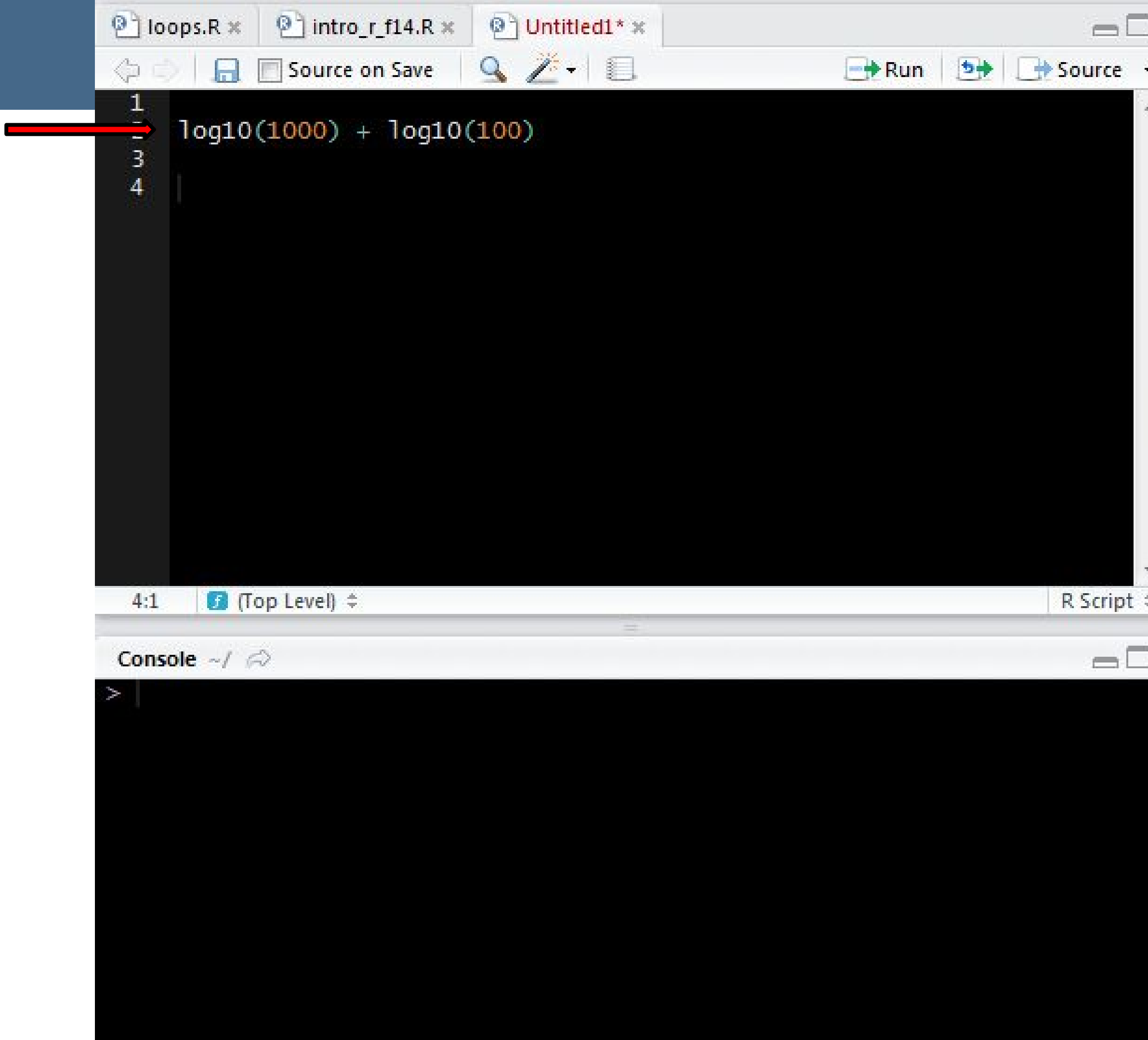
R Scripts

It is much easier to compose your code in an R script than in the command line.
To open a script, go to File > New script in the toolbar



Common Workflow

1. Write code in an R script

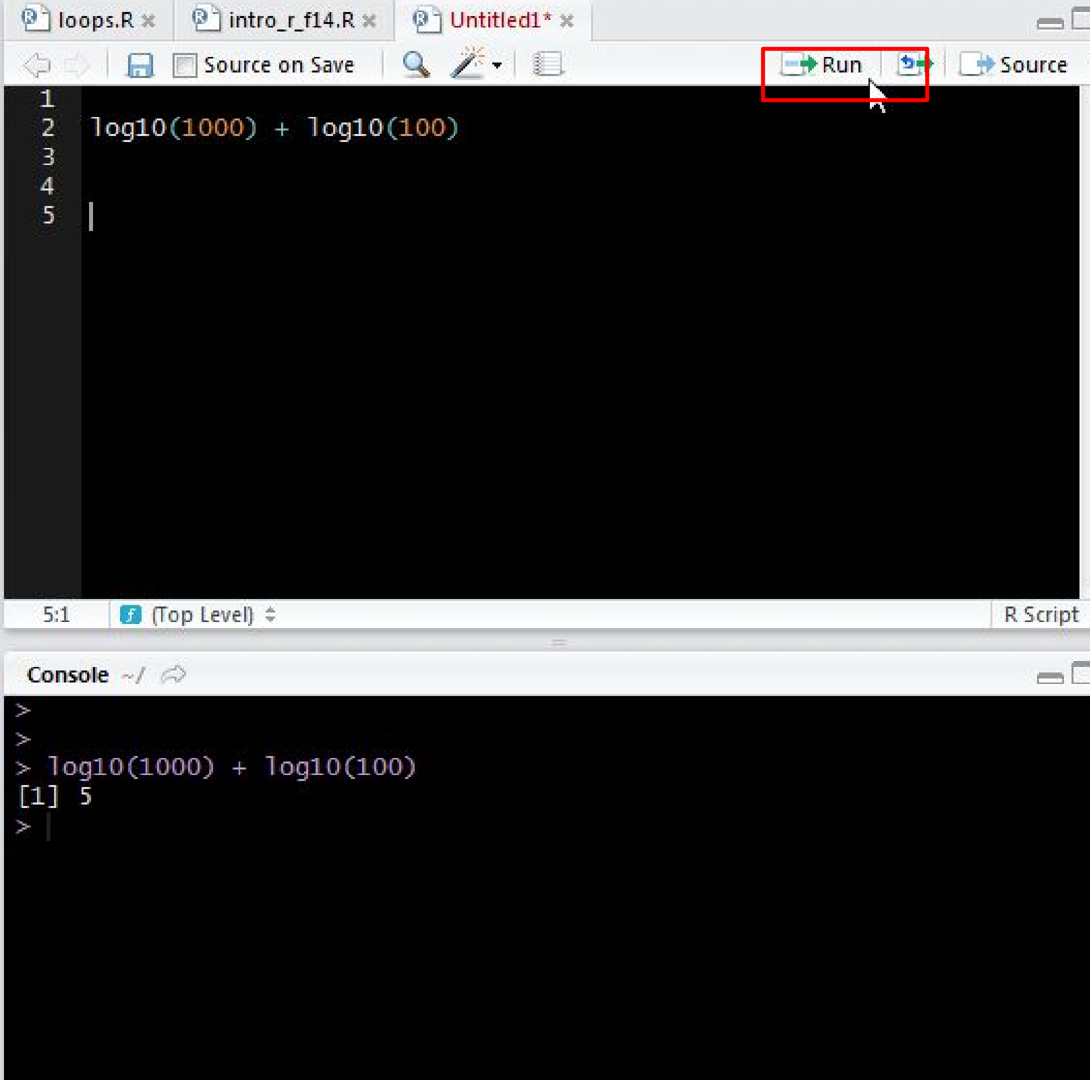


```
log10(1000) + log10(100)
```

Common Workflow

1. Write code in an R script

2. Run code in console with run icon or from script (Ctrl- r) or Ctrl-Enter



```
log10(1000) + log10(100)
```

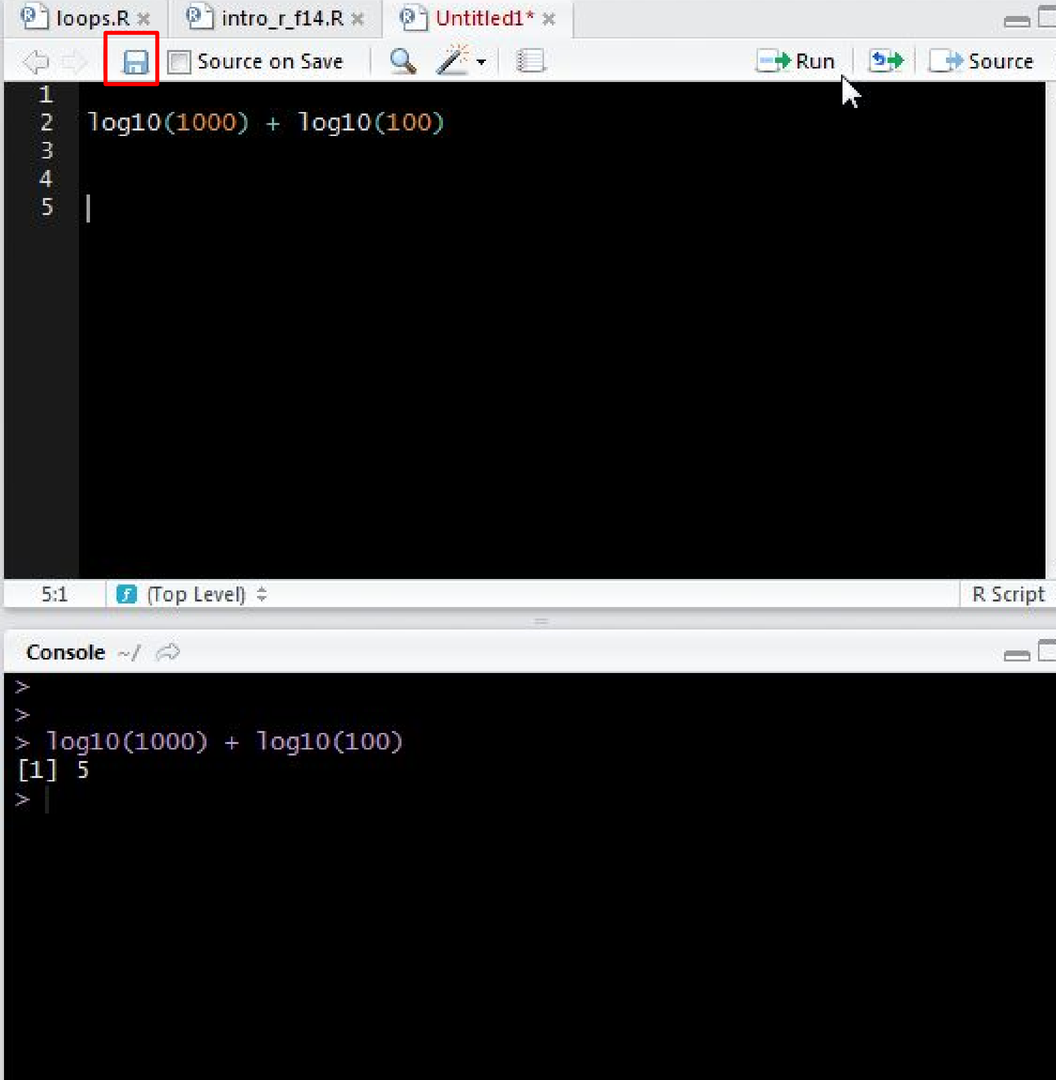
```
>
>
> log10(1000) + log10(100)
[1] 5
>
```

Common Workflow

1. Write code in an R script

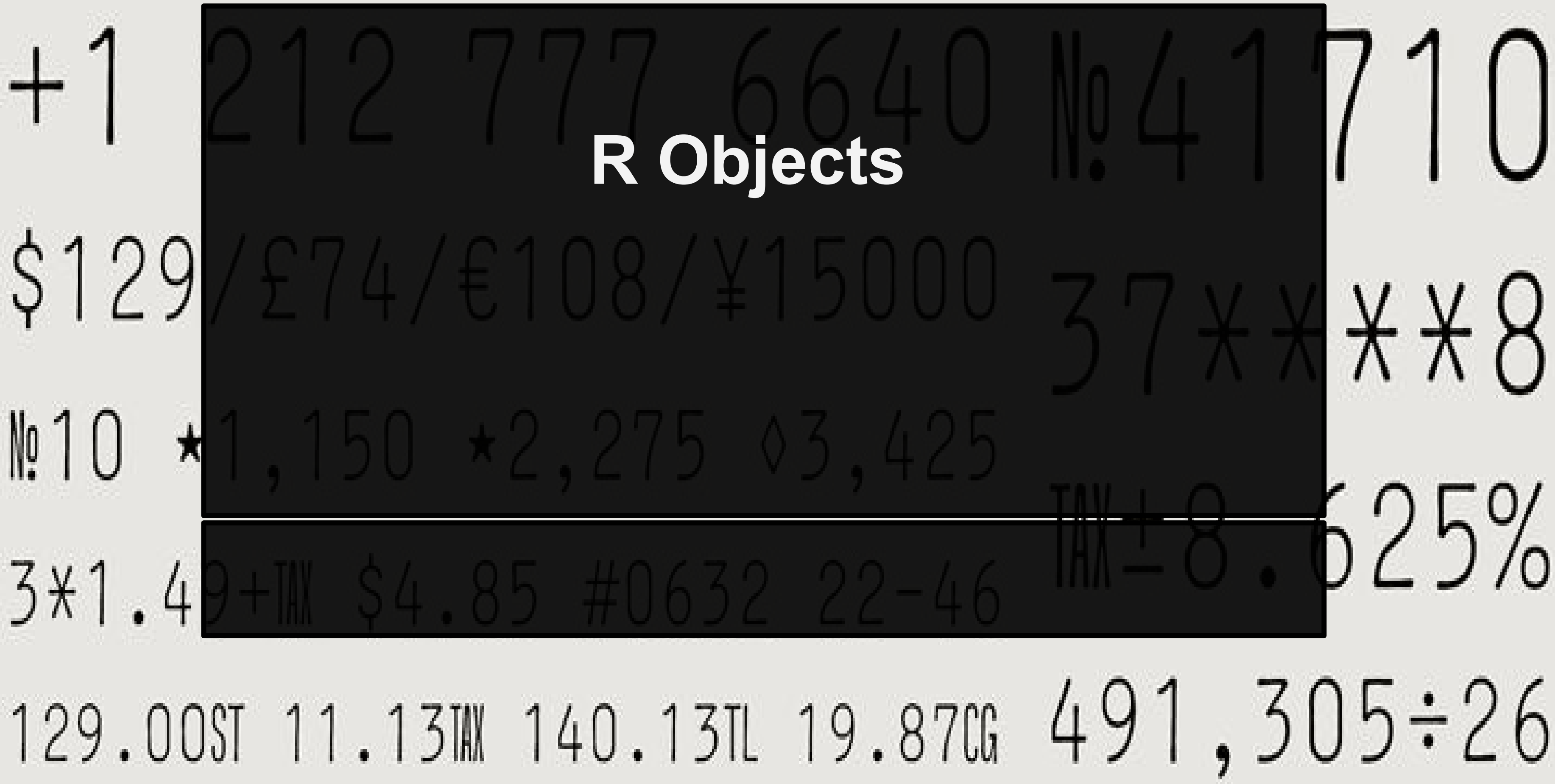
2. Run code in console with run icon or from script (Ctrl- r) or Ctrl-Enter

3. Save R Script when finished



```
log10(1000) + log10(100)
```

```
>
>
> log10(1000) + log10(100)
[1] 5
>
```


+1 212 777 6640 №41710
$129/£74/€108/¥15000 37***8
№10 ★1,150 ★2,275 ◊3,425 TAX±8.625%
3*1.49+TAX $4.85 #0632 22-46
129.00ST 11.13TAX 140.13TL 19.87CG 491,305÷26

R Objects

Source: http://www.typography.com/images/whatsInsidePageImages/numbers-revenue.png

Objects

Save information as an R object with the less than sign followed by a minus, e.g, an arrow: <-

```
foo <- 42
```

Objects

Save information as an R object with the greater than sign followed by a minus, e.g, an arrow: <-

name of new object

```
foo <- 42
```

Objects

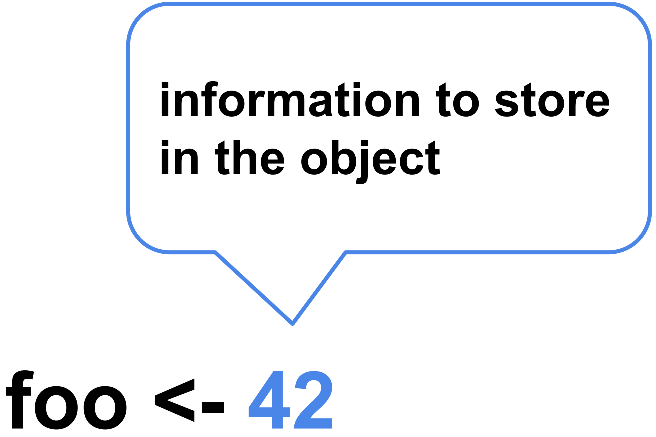
Save information as an R object with the greater than sign followed by a minus, e.g, an arrow: <-

assignment operator, “gets”

```
foo <- 42
```

Objects

Save information as an R object with the greater than sign followed by a minus, e.g, an arrow: <-



information to store in the object

```
foo <- 42
```

Common R Workflow

Save output of one function as an R object to use in a second function

```
foo <- round(3.1415) +1

foo
```

Common R Workflow

Save output of one function as an R object to use in a second function

```
foo <- round(3.1415) +1

foo

# 4
```

Common R Workflow

Save output of one function as an R object to use in a second function

```
foo <- round(3.1415) +1

foo

# 4

factorial(foo)
```

Common R Workflow

Save output of one function as an R object to use in a second function

```
foo <- round(3.1415) +1

foo

# 4

factorial(foo)

# 24
```

rm

You can remove an object with rm

```
foo

# 4

rm(foo)
foo
```

rm

You can remove an object with rm

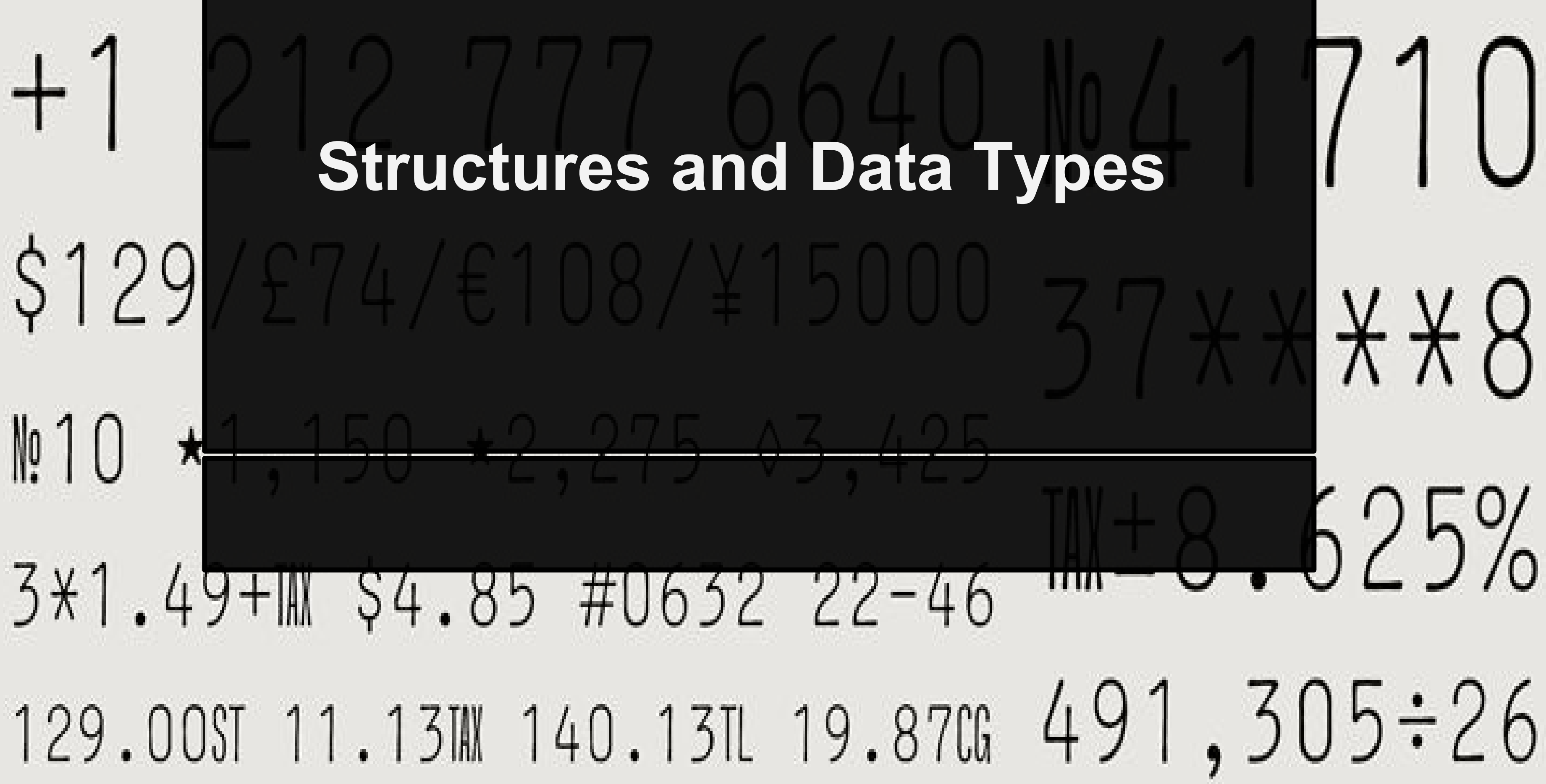
```
foo

# 4

rm(foo)
foo

# Error: object ‘foo’ not found
```

Objects

This can be useful if you overwrite an object that comes with R

```
pi

# 3.141593

pi <- 1
pi

# 1

rm(pi)
pi

# 3.141593
```

Structures and Data Types


+1 212 777 6640 №41710
$129/£74/€108/¥15000 37**¥*8
№10 ★1,150 ★2,275 ◊3,425 TAX±8.625%
3*1.49+TAX $4.85 #0632 22-46
129.00ST 11.13TAX 140.13TL 19.87CG 491,305÷26

Source: http://www.typography.com/images/whatsInsidePageImages/numbers-revenue.png

Data Types

Like Excel, R can recognize different types of data

Four Basic Types:

Numeric

Character string (text)

Logical

Factor

Numeric

Any number, no quotes. Appropriate for math.

```
1 + 1

3000000

class(0.00001)
```

Numeric

Any number, no quotes. Appropriate for math.

```
1 + 1

3000000

class(0.00001)

# "numeric"
```

Character

Any symbols surrounded by quotes.

Appropriate for words, variable names, messages, any text.

```
"hello"
```

```
class("hello")
```

Character

Any symbols surrounded by quotes.

Appropriate for words, variable names, messages, any text.

```
“hello”

class(“hello”)

# "character"
```

Structures and Data types

```
“hello” + “world”

# Error

nchar("hello")

# 5

paste("hello", "world")

# “hello world”
```

Logical

TRUE or FALSE. R’s form of binary data. Useful for logical tests.

```
3 > 4

# FALSE

class(TRUE)

# “logical”

class (T)

# “logical”
```

Factor

R’s form of categorical data. Saved as an integer with a set of labels (e.g. levels)

```
fac <- factor(c("a", "b", "c"))
fac
# a b c
# Levels: abc



class(fac)

# factor
```

Structures and Data types

Type	Examples
numeric	0, 1, -2, 3.14, 0.0005
character	“gender”, “date”, “31”
logical	TRUE, FALSE, T, F
factor	a c c b Levels: abc

Structures

Five types of structures in R:

Vectors

Matrices

Arrays

Lists

Data frames

Structures

Five types of structures in R:

Vectors

Matrices

Arrays

Lists

Data frames

Data frame = “Dataset”

Vectors

Combine multiple elements into a one dimensional array

Create with the c function

```
vec <- c(1, 2, 3, 10, 100)
```

Vectors

Combine multiple elements into a one dimensional array

Create with the c function

```
vec <- c(1, 2, 3, 10, 100)
vec

# 1 2 3 10 100
```

Matrices

Multiple elements stored in a two dimensional array

Create with the matrix function

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
```

Matrices

Multiple elements stored in a two dimensional array

Create with the matrix function

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
mat

#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```

Matrices

Multiple elements stored in a two dimensional array

Create with the matrix function

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
mat

#     [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```

Structures and Data types

vector of elements to go in the matrix

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
mat

#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```

Structures and Data types

number of rows for matrix

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
mat

#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```

Structures and Data types

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3)
```

Structures and Data types

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3)
mat
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Structures and Data types

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, byrow = TRUE)
mat
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Structures and Data types

```
vec <- c(1, 2, 3, "10", 100)
```

Structures and Data types

```
vec <- c(1, 2, 3, "10", 100)

class(vec)

# "character"
```

Structures and Data types

Vectors and Matrices only allow one data type.

So what do we do if we want to work with multiple data types?

Data frame

A data frame is a two dimensional group of R objects.

Each column in a data frame can be a different type

```
df <- data.frame(c(1, 2, 3),
c("R","S","T"), c(TRUE, FALSE, TRUE))

class (df)
```

Data frame

A data frame is a two dimensional group of R objects.

Each column in a data frame can be a different type

```
df <- data.frame(c(1, 2, 3),
c("R","S","T"), c(TRUE, FALSE, TRUE))

class (df)

# “data.frame”
```

names

You can name the elements of a vector, list or data frame when you create them.

```
nvec <- c(one = 1, two = 2, three = 3)

nvec

#  one   two three
#    1     2     3
```

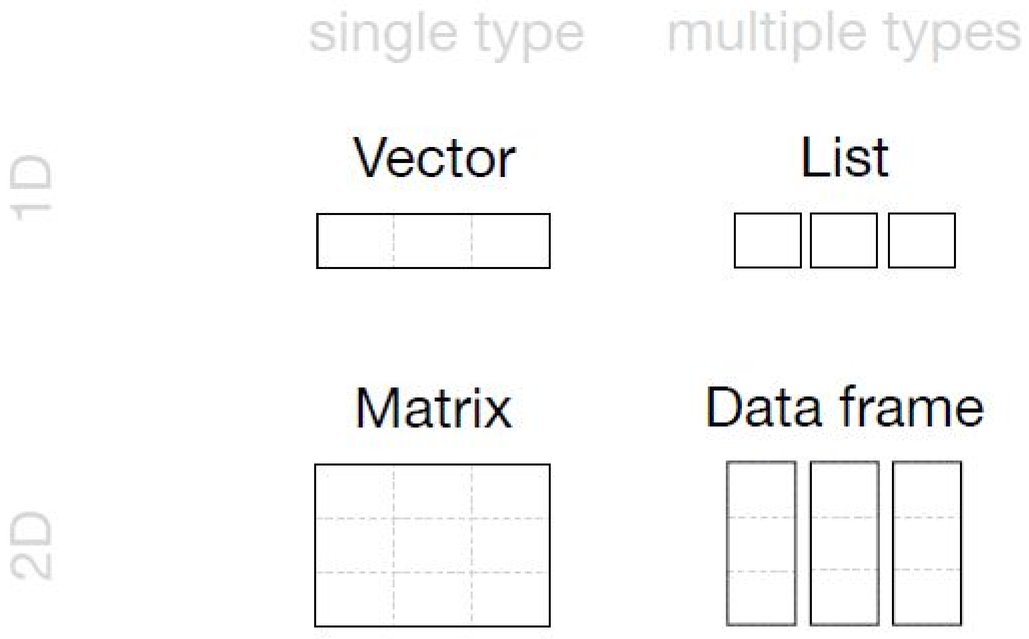
Structures and Data types

```
ndf <- data.frame(numbers = c(1, 2, 3),
letters = c("R","S","T"),
logic = c(TRUE, FALSE, TRUE))




ndf
```

Structures and Data types

```
ndf <- data.frame(numbers = c(1, 2, 3),
letters = c("R","S","T"),
logic = c(TRUE, FALSE, TRUE))



ndf

# numbers letters logic
# 1 1 R TRUE
# 2 2 S FALSE
# 3 3 T TRUE
```

Structures and Data types



single type

multiple types

1D

Vector

List

2D

Matrix

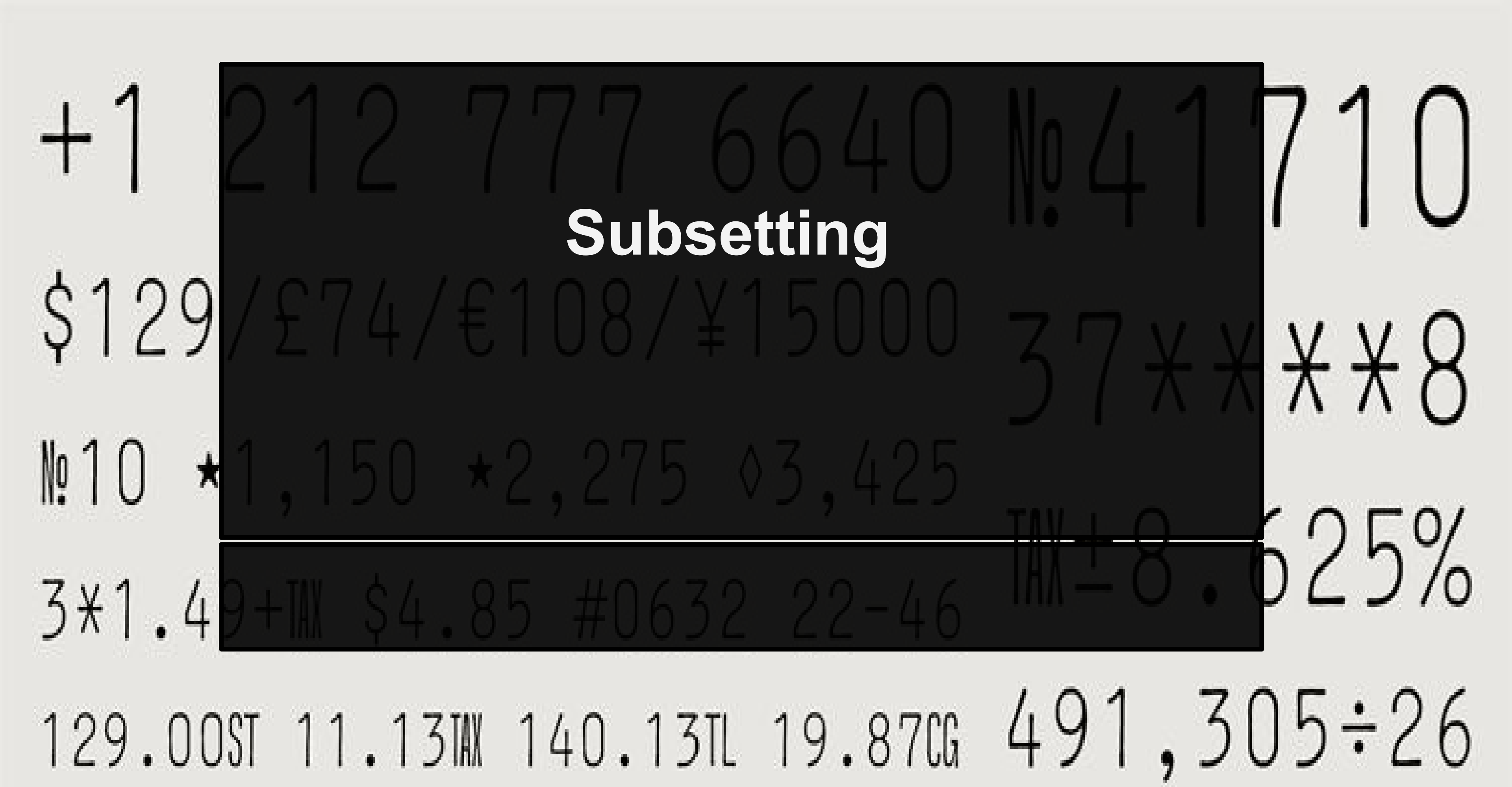
Data frame

```
x <- c(0, 0, 0, 0, 1, 0 ,0)
y <- x
y
# 0 0 0 0 1 0 0
```

How can you save just the fifth element of x to y?

How can you change the fifth element of x to a 0?

Subsetting


+1 212 777 6640 №41710
$129/£74/€108/¥15000 37****8
№10 ★1,150 ★2,275 ◊3,425 TAX±8.625%
3*1.49+TAX $4.85 #0632 22-46
129.00ST 11.13TAX 140.13TL 19.87CG 491,305÷26

Source: http://www.typography.com/images/whatsInsidePageImages/numbers-revenue.png

Subset notation

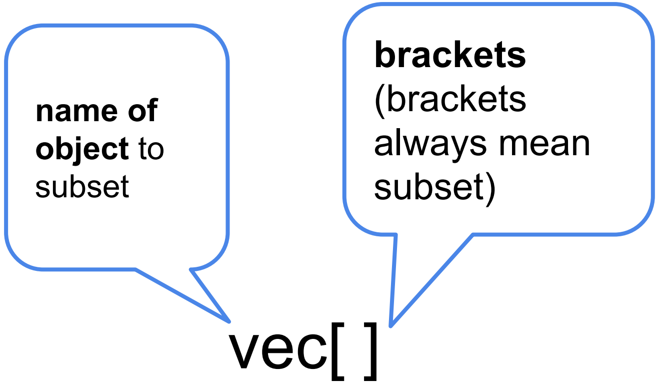
```
vec[ ]
```

Subset notation

name of object to subset

```
vec[ ]
```

Subset notation

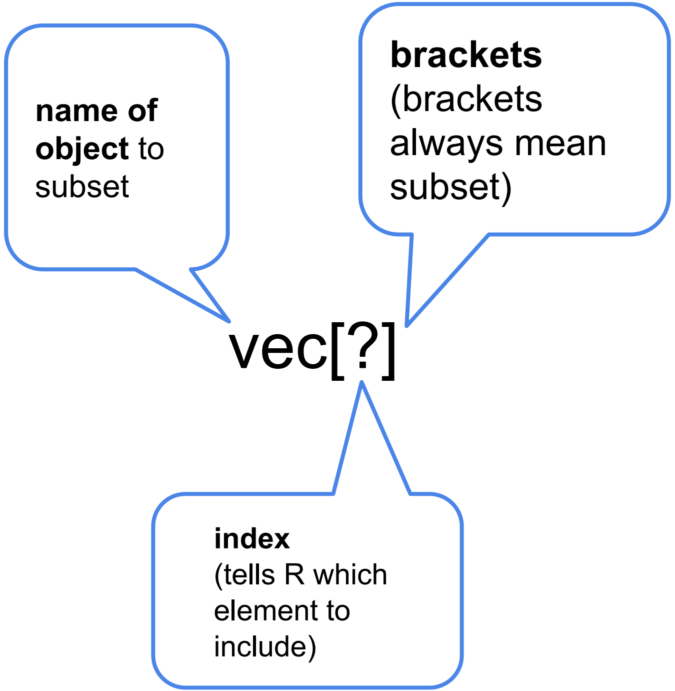


name of object to subset

brackets (brackets always mean subset)

vec[ ]

Subset notation



name of object to subset

brackets (brackets always mean subset)

vec[?]

index (tells R which element to include)

Each dimension needs its own index

vec [?]

6	1	3	6	10	5

Each dimension needs its own index

vec [?]

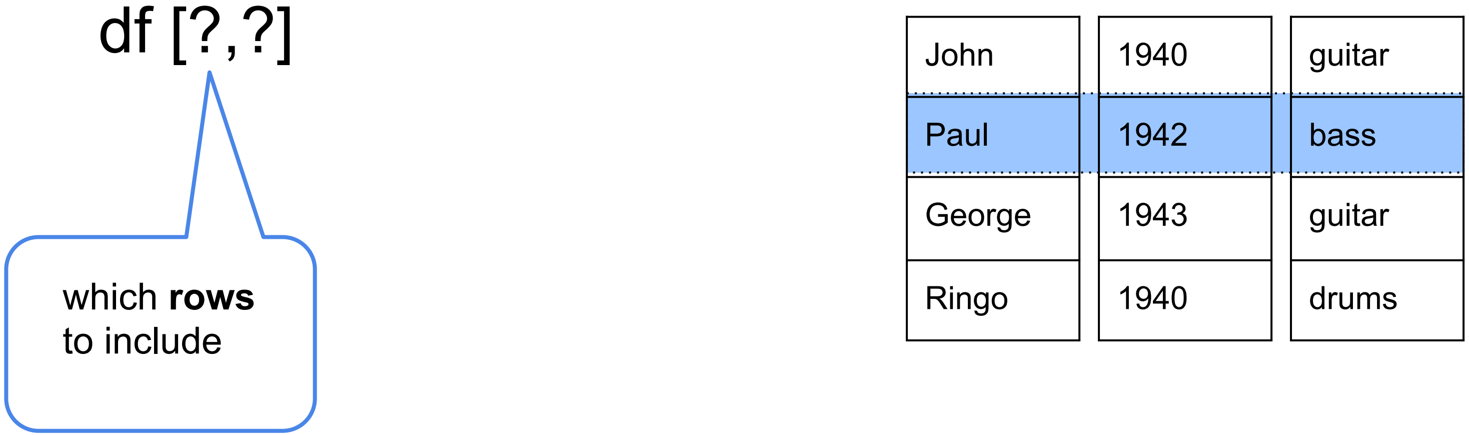
6	1	3	6	10	5

Each dimension needs its own index

df [?,?]

John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Each dimension needs its own index



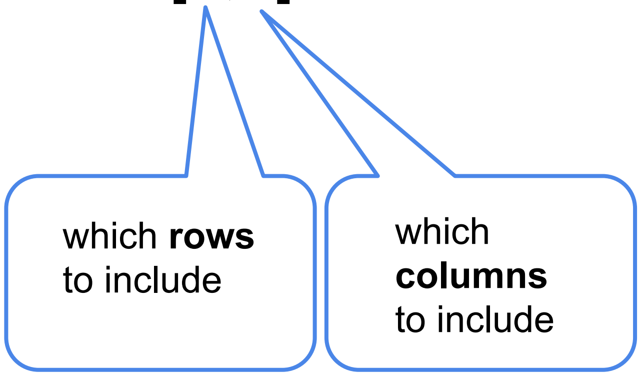
df [?,?]

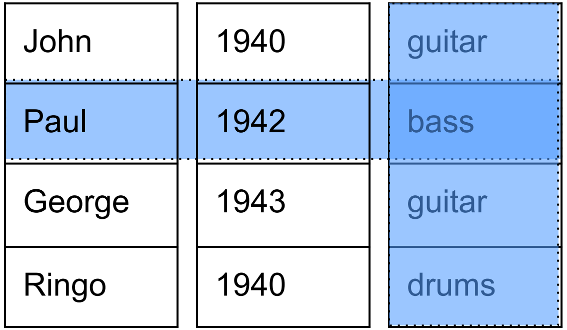
which **rows** to include

John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Each dimension needs its own index

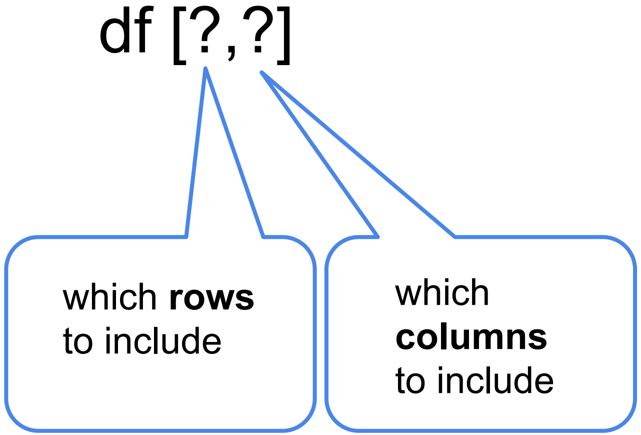
df [?,?]





John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Each dimension needs its own index

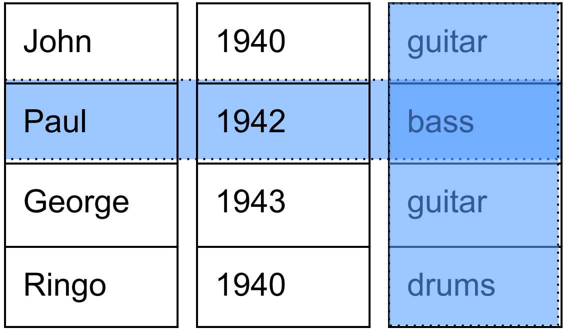


df [?,?]

which **rows** to include

which **columns** to include

separate with comma



John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Subsetting and Summaries

```
vec <- c(6, 1, 3, 6, 10, 5)
```

```
df <- data.frame(
name = c("John", "Paul", "George", "Ringo"),
birth = c(1940, 1942, 1943, 1940),
instrument = c("guitar", "bass", "guitar", "drums")
)
```

run the code on the following slide **IN YOUR HEADS**

Subsetting and Summaries

vec

6	1	3	6	10	5

df

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

```
# Predict what the following code will do
# DON'T RUN IT

vec[2]
vec[c(5, 6)]
vec[-c(5,6)]
vec[vec > 5]

df[c(2, 4), 3]
df[ , 1]
df[ , "instrument"]
df$instrument
```

Four ways to subset using [ ]

1. Integers

2. Blank spaces

3. Names

4. Logical

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

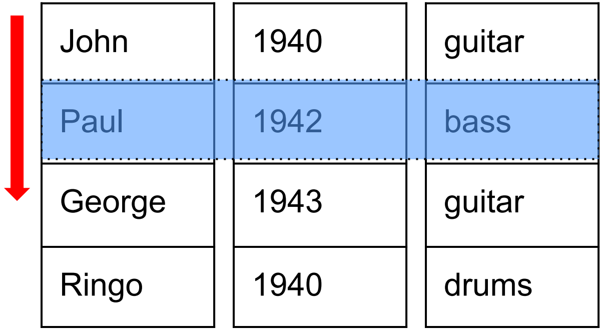
df[?,?]

John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

df[2,?]

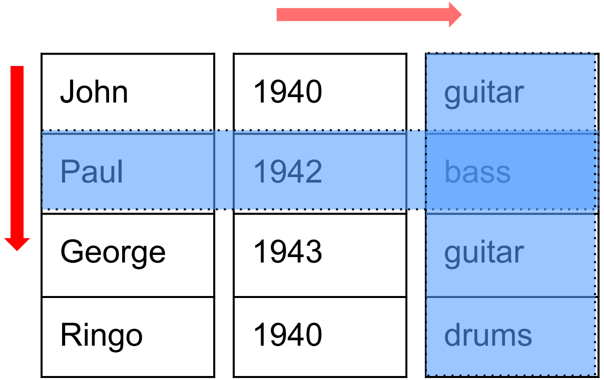


John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

```
df[2,3]
```

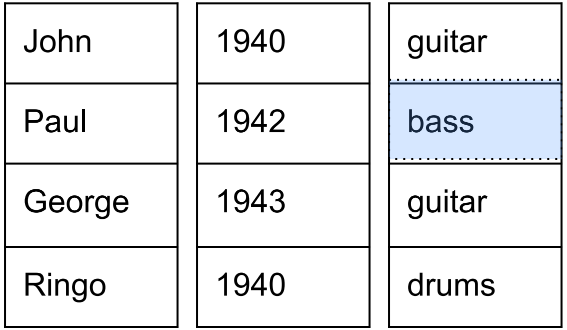


John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

```
df[2,3]
```

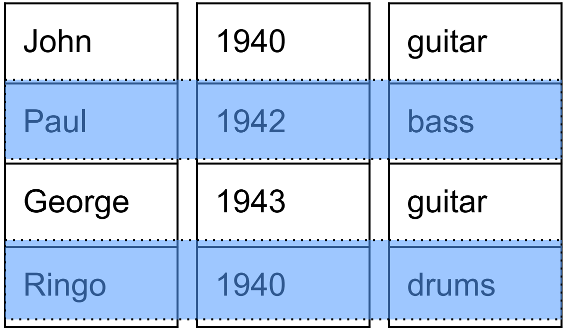


John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

```
df[c(2,4), ?]
```

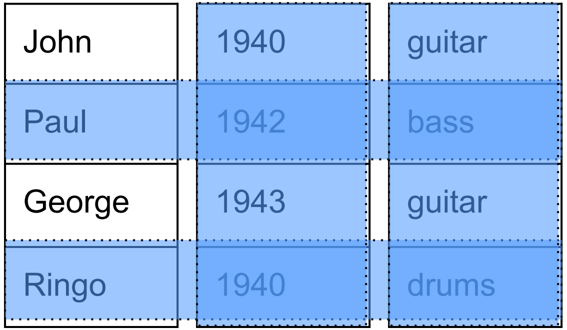


John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

```
df[c(2,4), c(2,3)]
```

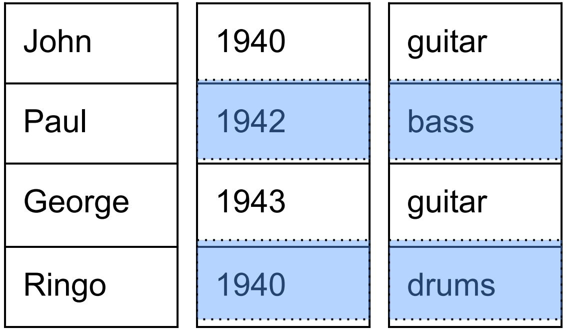


John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

```
df[c(2,4), c(2,3)]
```



John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

```
df[c(2,4), 3]
```

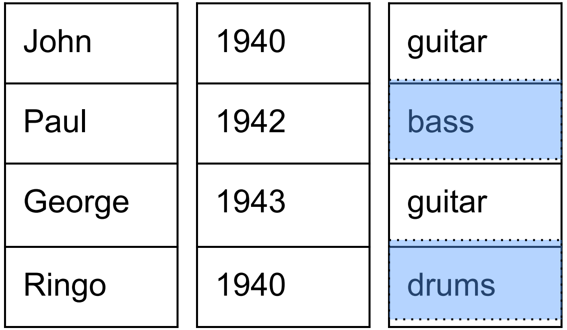
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

?

Integers (positive)

positive integers behave just like *ij* notation in linear algebra

```
df[c(2,4), 3]
```

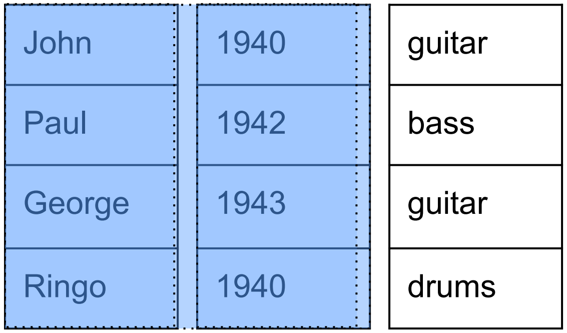


John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

Colons are a useful way to create vectors and to return results

```
df[1:4,1:2]
```



John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers
in the same direction

vec[-c(5,6)]

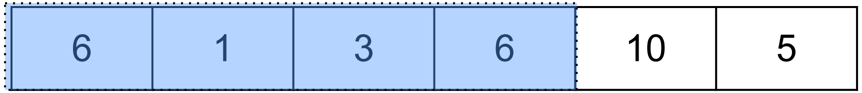
6	1	3	6	10	5

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers
in the same direction

```
vec[-c(5,6)]
```



6	1	3	6	10	5

Blank spaces

Blank spaces return **everything**

(i.e., no subsetting occurs on that dimension)

vec[ ]

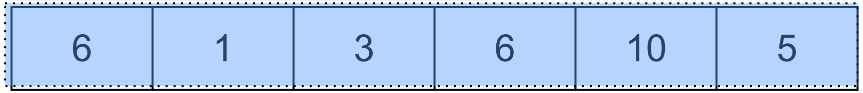
6	1	3	6	10	5

Blank spaces

Blank spaces return **everything**

(i.e., no subsetting occurs on that dimension)

vec[ ]



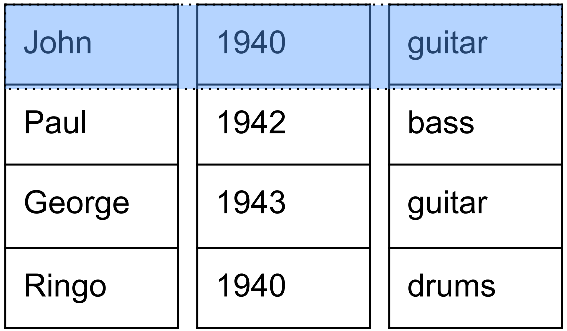
6	1	3	6	10	5

Blank spaces

Blank spaces return **everything**

(i.e., no subsetting occurs on that dimension)

df[1,]



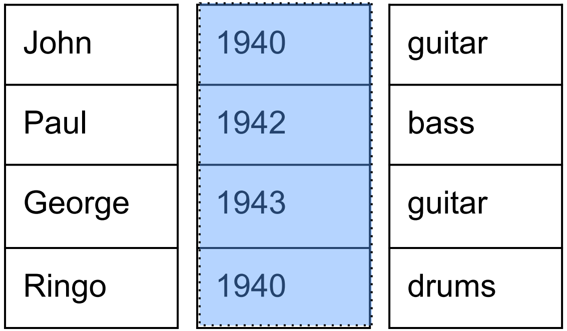
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Blank spaces

Blank spaces return **everything**

(i.e., no subsetting occurs on that dimension)

df[,2]



John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Names

If your object has names, you can ask for elements or columns back by bame.

```
vec[    ]
```

6	1	3	6	10	5

Names

If your object has names, you can ask for elements or columns back by bame.

```
names(vec) <- c("a","b","c","d","e","f")
```

```
vec[    ]
```

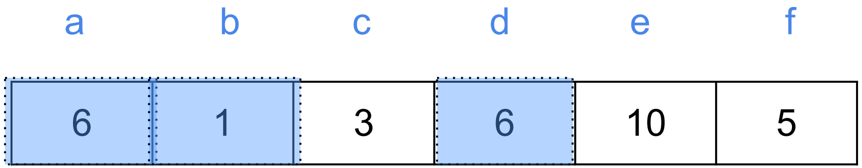
a	b	c	d	e	f
6	1	3	6	10	5

Names

If your object has names, you can ask for elements or columns back by bame.

```
names(vec) <- c("a","b","c","d","e","f")
```

```
vec[c("a","b","d")]
```

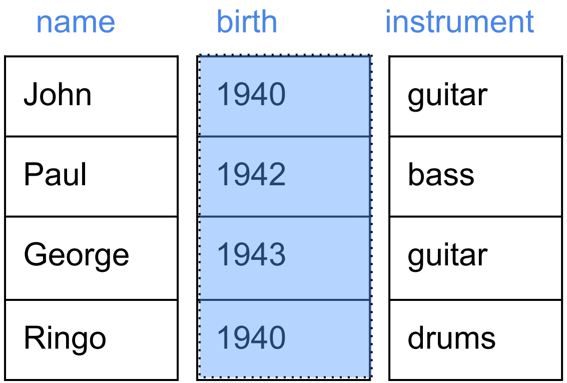


a	b	c	d	e	f
6	1	3	6	10	5

Names

If your object has names, you can ask for elements or columns back by bame.

```
df[ ,"birth"]
```

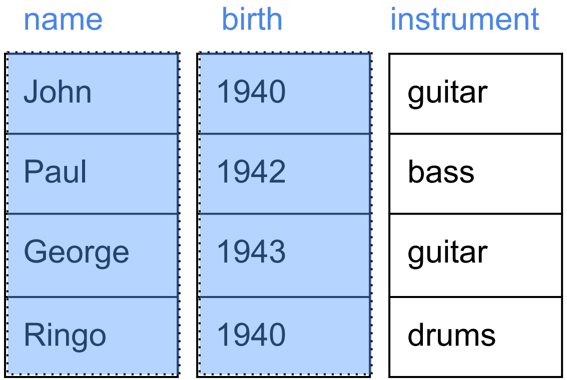


name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Names

If your object has names, you can ask for elements or columns back by bame.

```
df[ ,c("name","birth")]
```



name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Names $

A common syntax for subsetting lists and data frames with names is the dollar sign.

```
df$birth
```

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Names $

A common syntax for subsetting lists and data frames with names is the dollar sign.

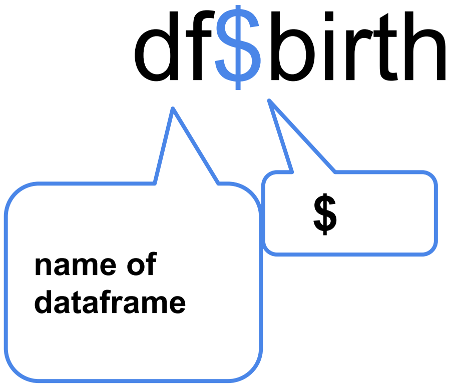
```
df$birth
```

name of dataframe

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Names $

A common syntax for subsetting lists and data frames with names is the dollar sign.



df$birth

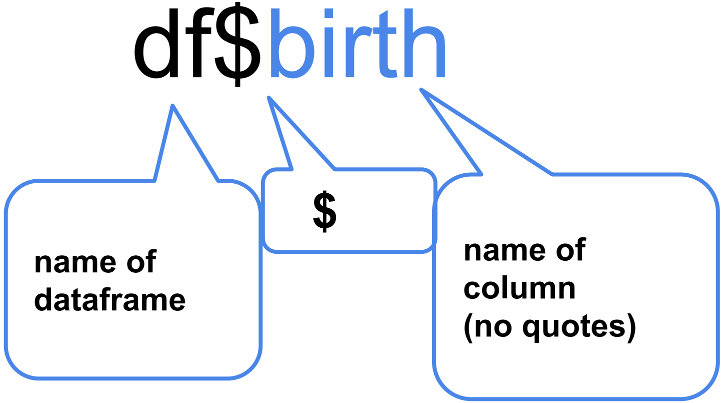
$

name of dataframe

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Names $

A common syntax for subsetting lists and data frames with names is the dollar sign.



df$birth

name of dataframe

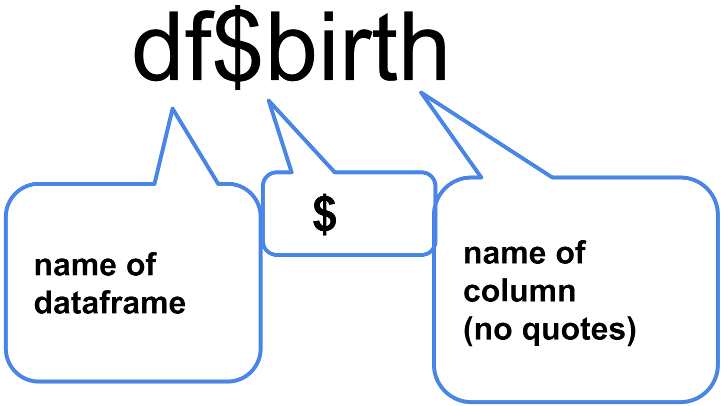
$

name of column (no quotes)

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Names $

A common syntax for subsetting lists and data frames with names is the dollar sign.



name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Logical

You can subset with a logical vector of the same length as the dimension you are subsetting. Each element that corresponds to a **TRUE** will be returned.

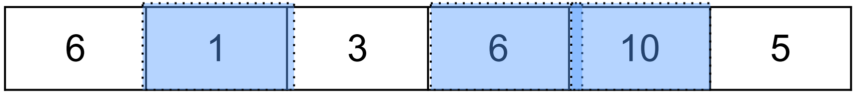
```
vec[c(FALSE,TRUE,FALSE,TRUE,TRUE,FALSE)]
```

6	1	3	6	10	5

Logical

You can subset with a logical vector of the same length as the dimension you are subsetting. Each element that corresponds to a **TRUE** will be returned.

```
vec[c(FALSE,TRUE,FALSE,TRUE,TRUE,FALSE)]
```



6	1	3	6	10	5

Logical operators

operator	tests
x > y	is x greater than y?
x >= y	is x greater than or equal to y?
x < y	is x less than y?
x <= y	is x less than or equal to y?
x == y	is x equal to y?
x != y	is x not equal to y?
x %in% c(y,z)	is x in the set c(y, z)?

Boolean operators

operator	tests
a & b	both a and b are TRUE
a | b	at least one of a and b is TRUE (or)
xor(a,b)	a is TRUE or b is TRUE, but not both
!(a)	not a (TRUE goes to FALSE, FALSE goes to TRUE)
any(a,b,c)	at least one of a, b , or c is TRUE
all(a,b,c)	each of a, b, and c is TRUE

Logical tests(standard and boolean operators)

Combining logical tests with subsetting is a very powerful technique!

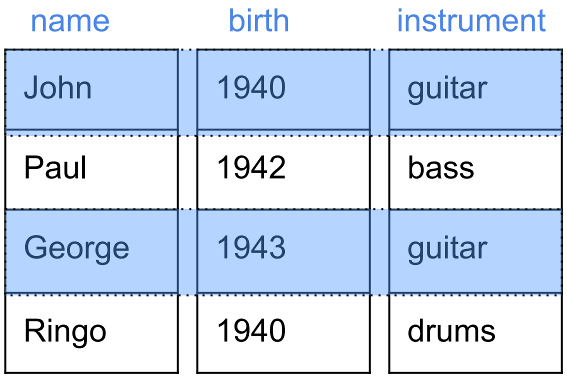
```
df[df$instrument == "guitar", ]
```

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Logical tests(standard and boolean operators)

Combining logical tests with subsetting is a very powerful technique!

```
df[df$instrument == "guitar", ]
```



name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Logical tests(standard and boolean operators)

Combining logical tests with subsetting is a very powerful technique!

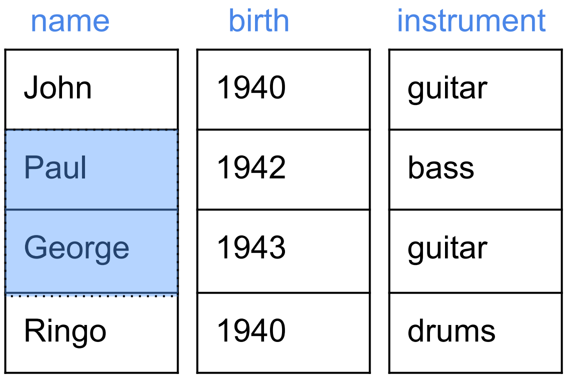
```
df[df$birth > 1940, 1]
```

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Logical tests(standard and boolean operators)

Combining logical tests with subsetting is a very powerful technique!

```
df[df$birth > 1940, 1]
```



name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Logical tests(standard and boolean operators)

Combining logical tests with subsetting is a very powerful technique!

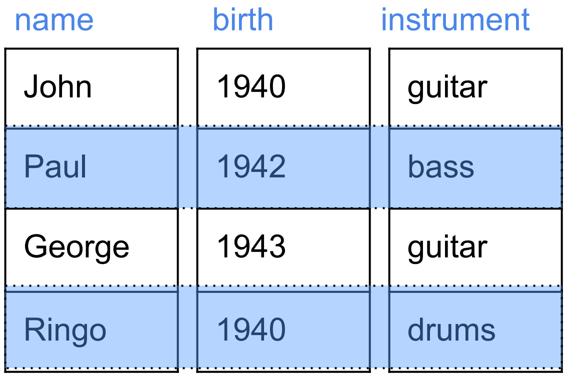
```
df[df$birth < 1943 & df$instrument != "guitar", ]
```

name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Logical tests(standard and boolean operators)

Combining logical tests with subsetting is a very powerful technique!

```
df[df$birth < 1943 & df$instrument != "guitar", ]
```



name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

BUT WAIT
THERE'S MORE
Baking Soda
quickmeme.com

Subset() Function

The easiest way to subset is to use the subset() function. Notice we don’t use brackets [ ]. The form of the subset function is,

subset(*x*, *subset, select*)

```
subset(df, birth > 1940, select = name)
```

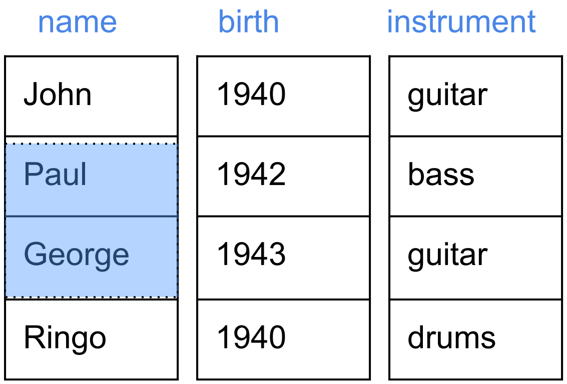
name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Subset() Function

The easiest way to subset is to use the subset() function. Notice we don’t use brackets [ ]. The form of the subset function is,

subset(*x*, *subset, select*)

subset(df, birth > 1940, select = name)



name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Subset() Function

The easiest way to subset is to use the subset() function. Notice we don’t use brackets [ ]. The form of the subset function is,

subset(*x*, *subset, select*)

subset(df, name == “Paul”)

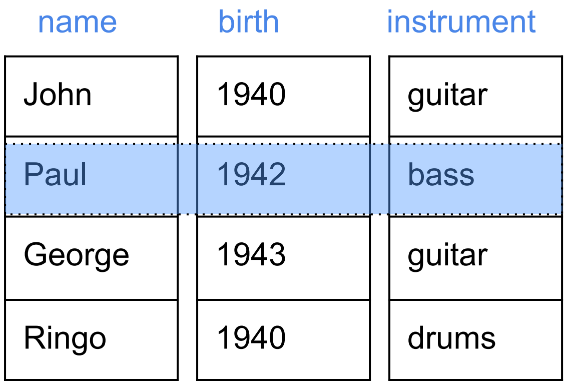
name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Subset() Function

The easiest way to subset is to use the subset() function. Notice we don't use brackets [ ]. The form of the subset function is,

subset(*x*, *subset, select*)

subset(df, name == "Paul")



name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Subset() Function

The easiest way to subset is to use the subset() function. Notice we don’t use brackets [ ]. The form of the subset function is,

subset(*x*, *subset, select*)

subset(df, select = birth)

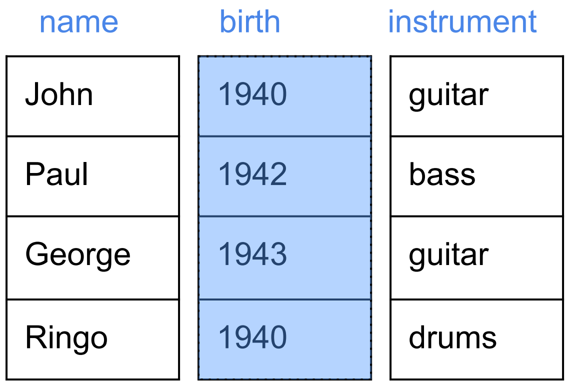
name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Subset() Function

The easiest way to subset is to use the subset() function. Notice we don't use brackets [ ]. The form of the subset function is,

subset(*x*, *subset, select*)

subset(df, select = birth)

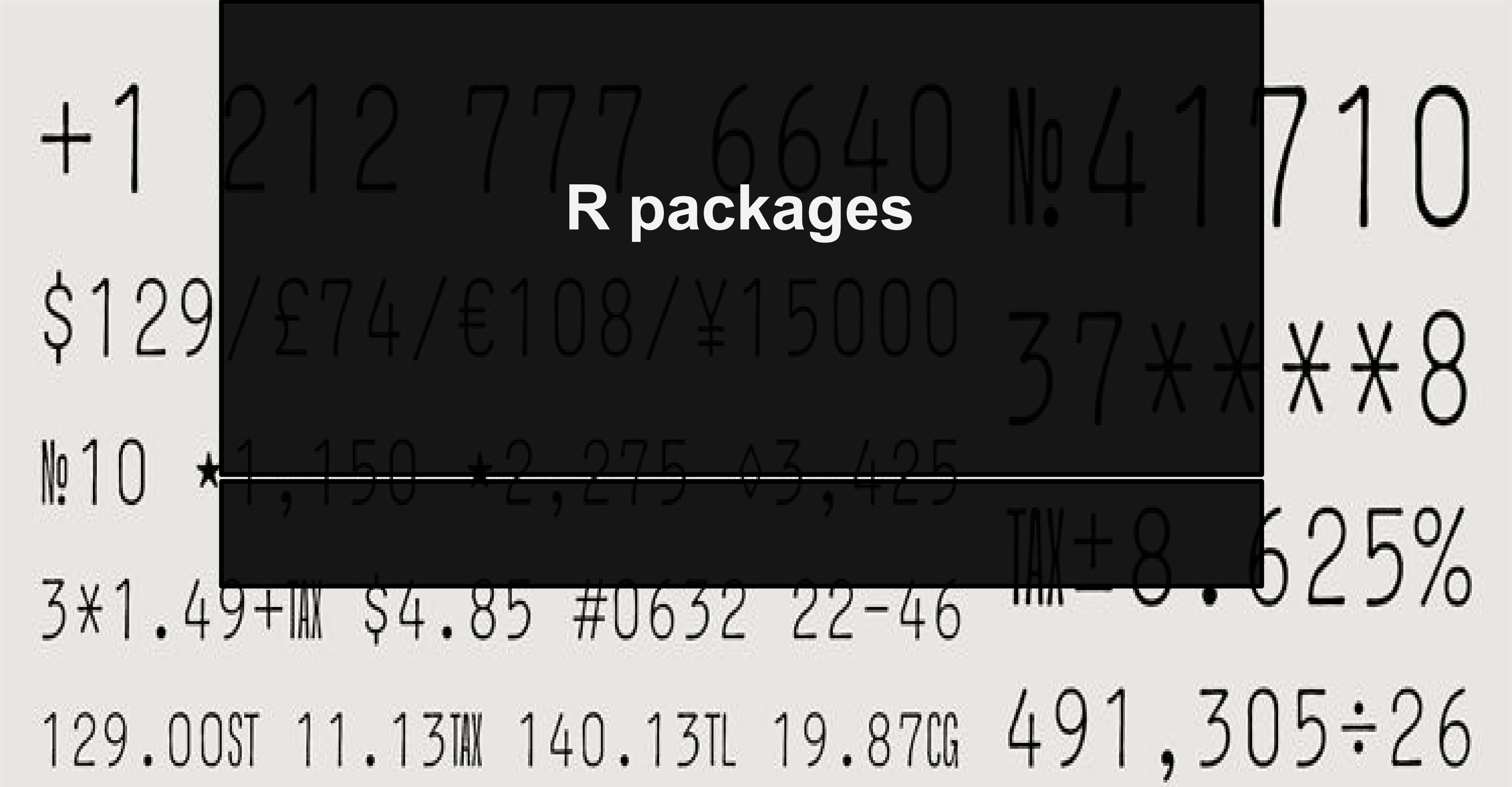


name	birth	instrument
John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

Subsetting and Summaries

	effect
integers	positive: returns specified elements negative: returns everything but the specified elements
blank spaces	returns everything
names	returns elements or columns with the specified names
logicals	returns elements that correspond to TRUE. Can use logical tests with booleans/standard operators
subset()	returns what is specified in subset and/or select arguments

R packages



Source: http://www.typography.com/images/whatsInsidePageImages/numbers-revenue.png

R Packages

A collection of functions written for the R language.

Usually focuses on a specific task or problem.

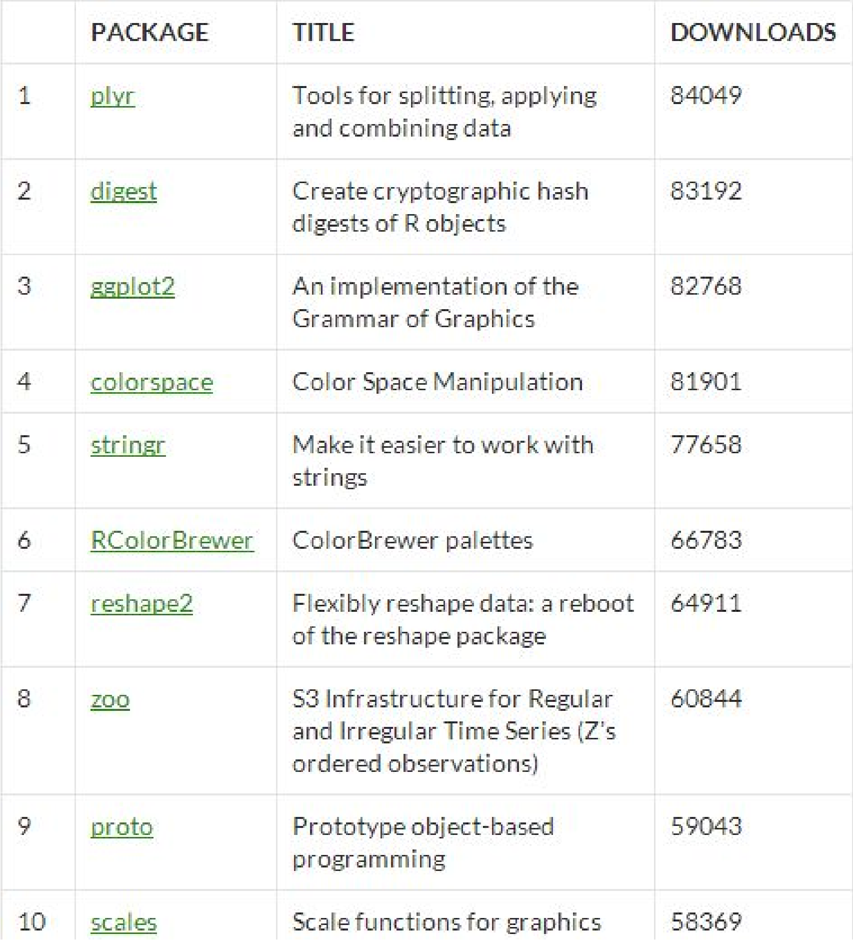
Most of the useful R applications appear in packages.

There are over 5100 R packages

Packages

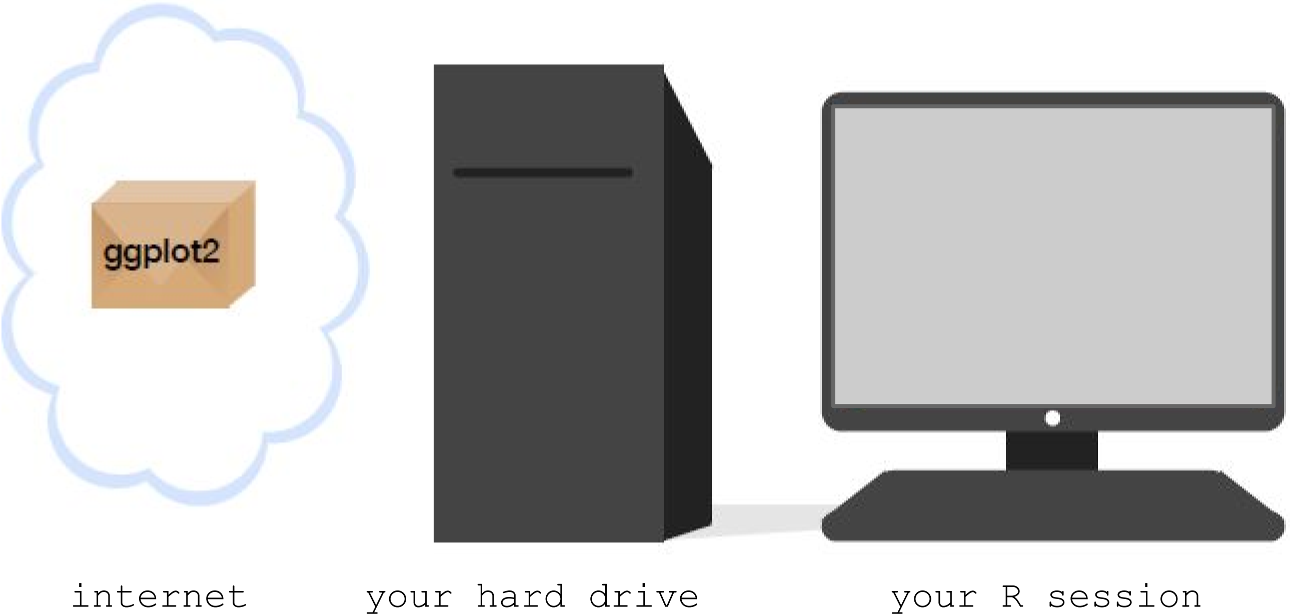
Top 10 R packages from Jan-May 2013

source: http://www.r-statistics.com/



	PACKAGE	TITLE	DOWNLOADS
1	plyr	Tools for splitting, applying and combining data	84049
2	digest	Create cryptographic hash digests of R objects	83192
3	ggplot2	An implementation of the Grammar of Graphics	82768
4	colorspace	Color Space Manipulation	81901
5	stringr	Make it easier to work with strings	77658
6	RColorBrewer	ColorBrewer palettes	66783
7	reshape2	Flexibly reshape data: a reboot of the reshape package	64911
8	zoo	S3 Infrastructure for Regular and Irregular Time Series (Z's ordered observations)	60844
9	proto	Prototype object-based programming	59043
10	scales	Scale functions for graphics	58369

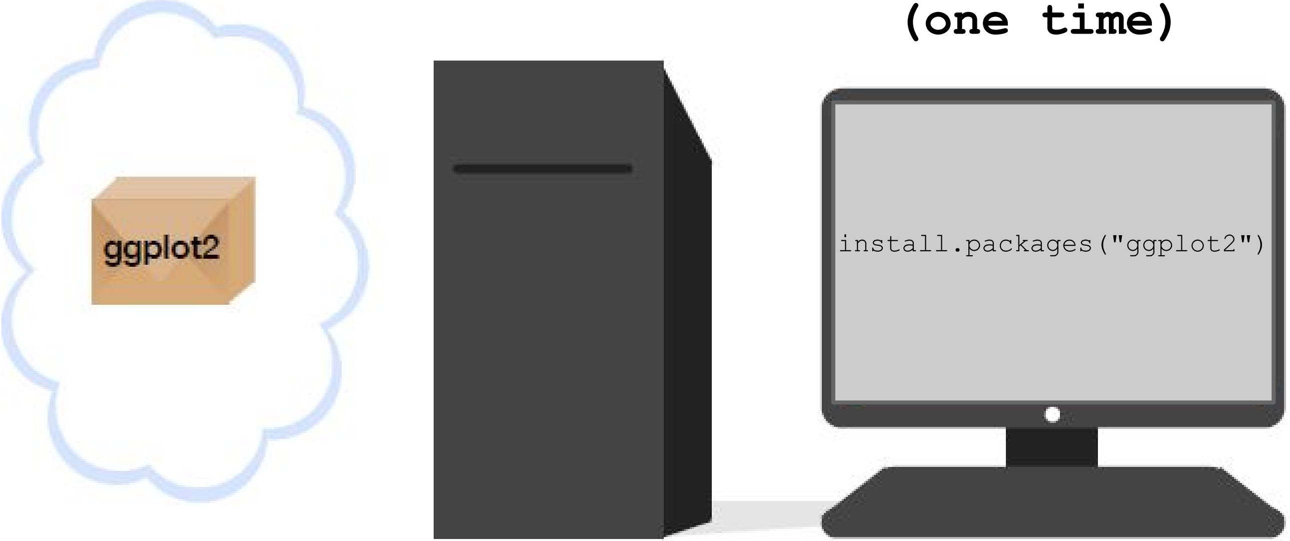
Packages



Packages

install your package with
`install.packages("ggplot2")`

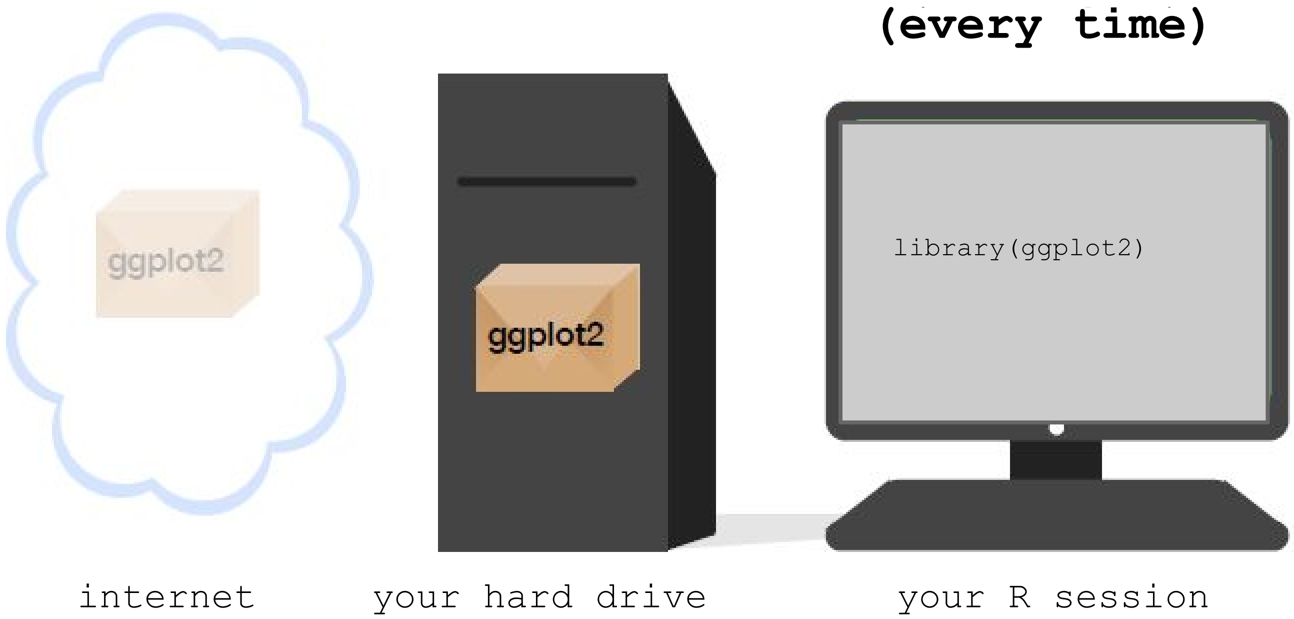
(one time)



internet your hard drive your R session

Packages

Load your package with `library(ggplot2)`



(every time)

```
library(ggplot2)
```

Baby names dataset

Top 1000 female and male baby names in the US, from 1880 to 2008.

258,000 records (1000 * 2 * 129)

Five variables: year, name, soundex, sex, prop

Image Source: http://www.flickr.com/photos/the_light_show/2586781132http://www.flickr.com/photos/the_light_show/2586781132

Baby names

```
library(plyr)
library(ggplot2)
```

```
options(stringsAsFactors = FALSE)
```

prevent R from reading in strings as factors (the default)

```
bnames <- read.csv("bnames.csv")
```

make sure the data sets are in your working directory

Baby names

library(plyr)

Function	Package
subset	base
mutate	plyr
arrange	plyr

They all share similar syntax. The first argument is a data frame, and all other arguments are interpreted in the context of that data frame. Each returns a data frame.

And yes...another way to subset.

Baby names

head(bnames)

```
  X   v1      v2       v3  v4   v5
1 1 1880    John 0.081541 boy J500
2 2 1880 William 0.080511 boy W450
3 3 1880   James 0.050057 boy J520
4 4 1880 Charles 0.045167 boy C642
5 5 1880  George 0.043292 boy G620
6 6 1880   Frank 0.027380 boy F652
```

Baby names

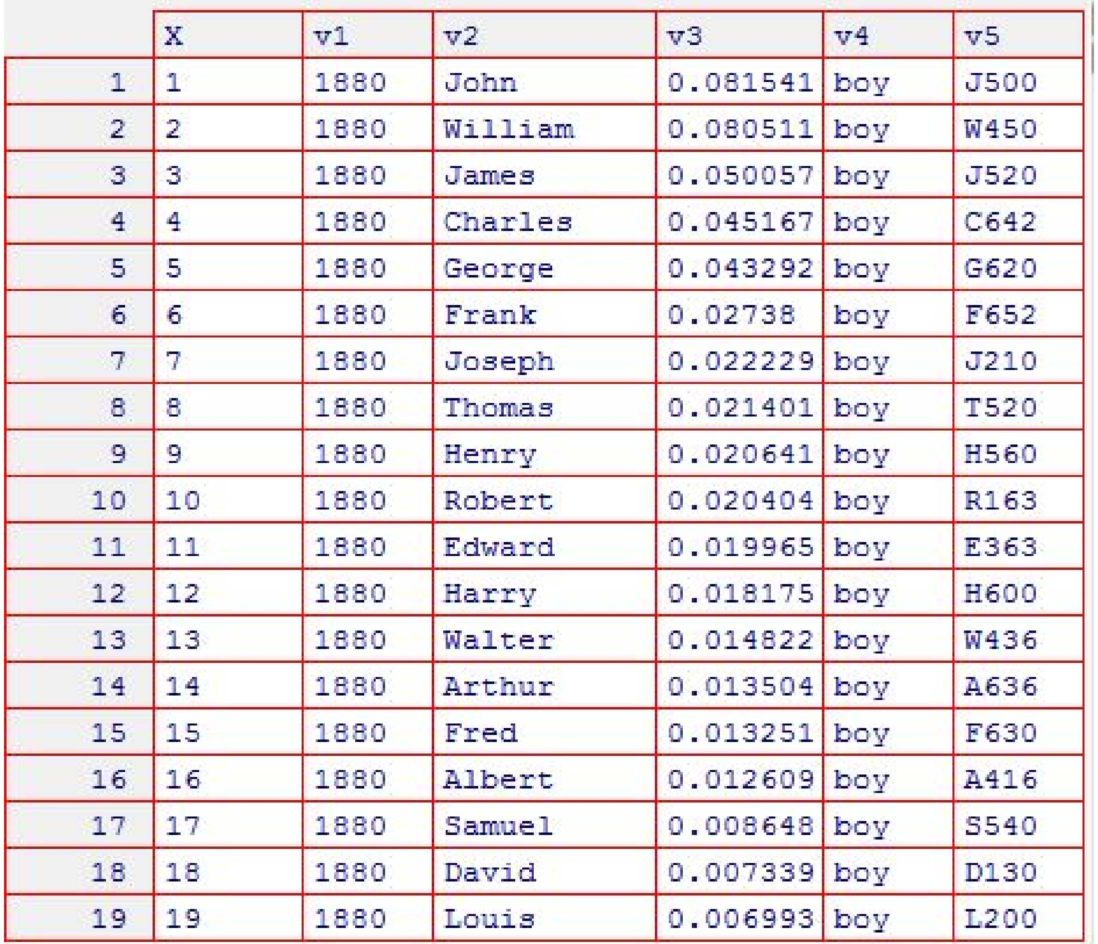
tail(bnames)

```
           X   v1       v2       v3   v4   v5
257995 257995 2008     Diya 0.000128 girl D000
257996 257996 2008 Carleigh 0.000128 girl C642
257997 257997 2008    Iyana 0.000128 girl I500
257998 257998 2008   Kenley 0.000127 girl K540
257999 257999 2008   Sloane 0.000127 girl S450
258000 258000 2008  Elianna 0.000127 girl E450
```

Baby names

View(bnames)

fix(bnames)



	X	v1	v2	v3	v4	v5
1	1	1880	John	0.081541	boy	J500
2	2	1880	William	0.080511	boy	W450
3	3	1880	James	0.050057	boy	J520
4	4	1880	Charles	0.045167	boy	C642
5	5	1880	George	0.043292	boy	G620
6	6	1880	Frank	0.02738	boy	F652
7	7	1880	Joseph	0.022229	boy	J210
8	8	1880	Thomas	0.021401	boy	T520
9	9	1880	Henry	0.020641	boy	H560
10	10	1880	Robert	0.020404	boy	R163
11	11	1880	Edward	0.019965	boy	E363
12	12	1880	Harry	0.018175	boy	H600
13	13	1880	Walter	0.014822	boy	W436
14	14	1880	Arthur	0.013504	boy	A636
15	15	1880	Fred	0.013251	boy	F630
16	16	1880	Albert	0.012609	boy	A416
17	17	1880	Samuel	0.008648	boy	S540
18	18	1880	David	0.007339	boy	D130
19	19	1880	Louis	0.006993	boy	L200

Baby names

bnames$X <- NULL #drop extraneous X variable
head(bnames)

```
  v1      v2        v3   v4    v5
1 1880    John 0.081541 boy J500
2 1880 William 0.080511 boy W450
3 1880   James 0.050057 boy J520
4 1880 Charles 0.045167 boy C642
5 1880  George 0.043292 boy G620
6 1880   Frank 0.027380 boy F652
```

Baby names

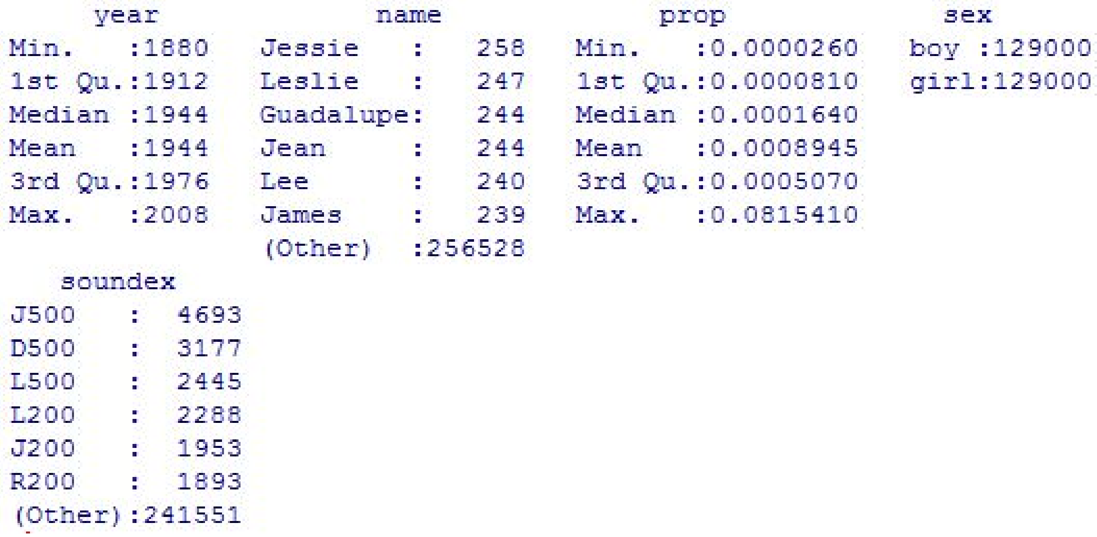
#rename variables. limitation here is that you have to enter all of them in order. to rename just a single variable, the rename function is faster (reshape package)

```
names(bnames) <- c("year", "name", "prop", "sex", "soundex")
head(bnames)
```

```
  year    name     prop sex soundex
1 1880    John 0.081541 boy    J500
2 1880 William 0.080511 boy    W450
3 1880   James 0.050057 boy    J520
4 1880 Charles 0.045167 boy    C642
5 1880  George 0.043292 boy    G620
6 1880   Frank 0.027380 boy    F652
```

Baby names

summary(bnames)



```
      year                 name               prop                  sex
 Min.    :1880   Jessie    :   258   Min.    :0.0000260   boy :129000
 1st Qu.:1912   Leslie    :   247   1st Qu.:0.0000810   girl:129000
 Median :1944   Guadalupe:   244   Median :0.0001640
 Mean    :1944   Jean      :   244   Mean    :0.0008945
 3rd Qu.:1976   Lee       :   240   3rd Qu.:0.0005070
 Max.    :2008   James     :   239   Max.    :0.0815410
                 (Other)   :256528
    soundex
 J500    :   4693
 D500    :   3177
 L500    :   2445
 L200    :   2288
 J200    :   1953
 R200    :   1893
 (Other):241551
```

Baby names

```
joshua <- subset(bnames, name == "Joshua")
head(joshua)
```

```
     year   name     prop sex soundex
212  1880 Joshua 0.000481 boy    J200
1249 1881 Joshua 0.000369 boy    J200
2236 1882 Joshua 0.000410 boy    J200
3202 1883 Joshua 0.000489 boy    J200
4266 1884 Joshua 0.000334 boy    J200
5254 1885 Joshua 0.000371 boy    J200
```

Baby names

```
joshua <- mutate(joshua, perc = prop * 100)     #create new variable
head(joshua)
```

```
  year   name     prop sex soundex   perc
1 1985 Joshua 0.021946 boy    J200 2.1946
2 1984 Joshua 0.021468 boy    J200 2.1468
3 1988 Joshua 0.021324 boy    J200 2.1324
4 1989 Joshua 0.021047 boy    J200 2.1047
5 1981 Joshua 0.020980 boy    J200 2.0980
6 1987 Joshua 0.020525 boy    J200 2.0525
```

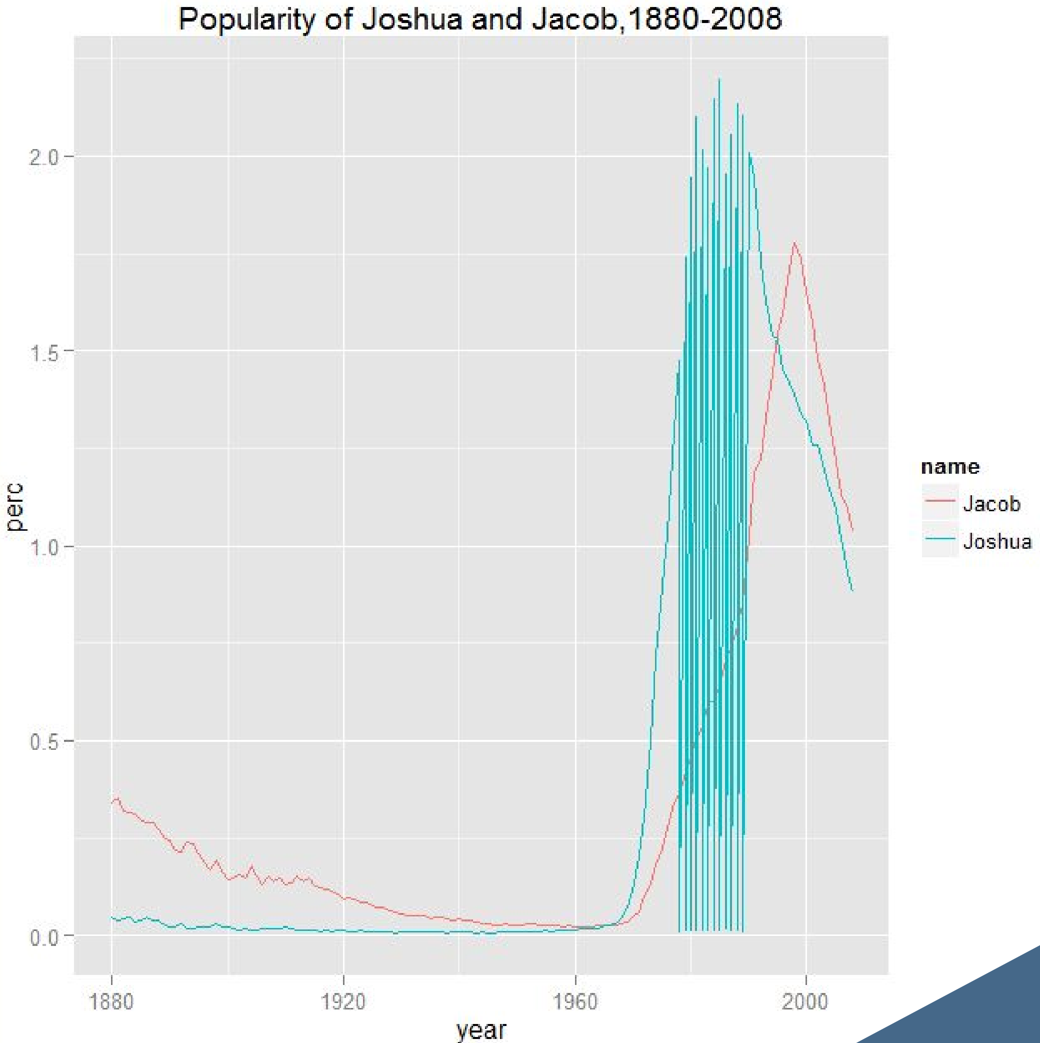
plyr

Baby names

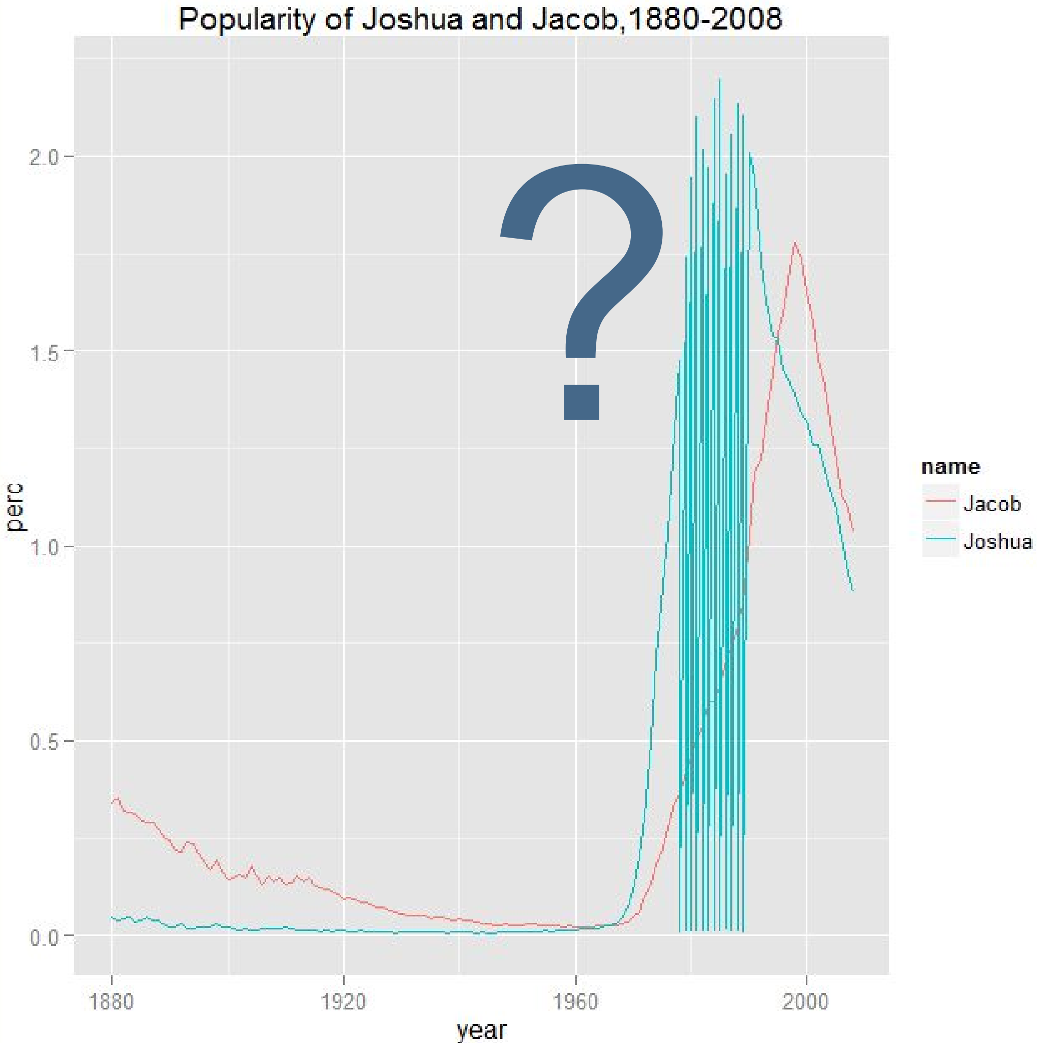
```
joshua <- arrange(joshua, desc(perc))
head(joshua, n=10)
```

```
   year   name     prop sex soundex   perc
1  1985 Joshua 0.021946 boy     J200 2.1946
2  1984 Joshua 0.021468 boy     J200 2.1468
3  1988 Joshua 0.021324 boy     J200 2.1324
4  1989 Joshua 0.021047 boy     J200 2.1047
5  1981 Joshua 0.020980 boy     J200 2.0980
6  1987 Joshua 0.020525 boy     J200 2.0525
7  1982 Joshua 0.020160 boy     J200 2.0160
8  1990 Joshua 0.020095 boy     J200 2.0095
9  1983 Joshua 0.019708 boy     J200 1.9708
10 1986 Joshua 0.019550 boy     J200 1.9550
```

plyr

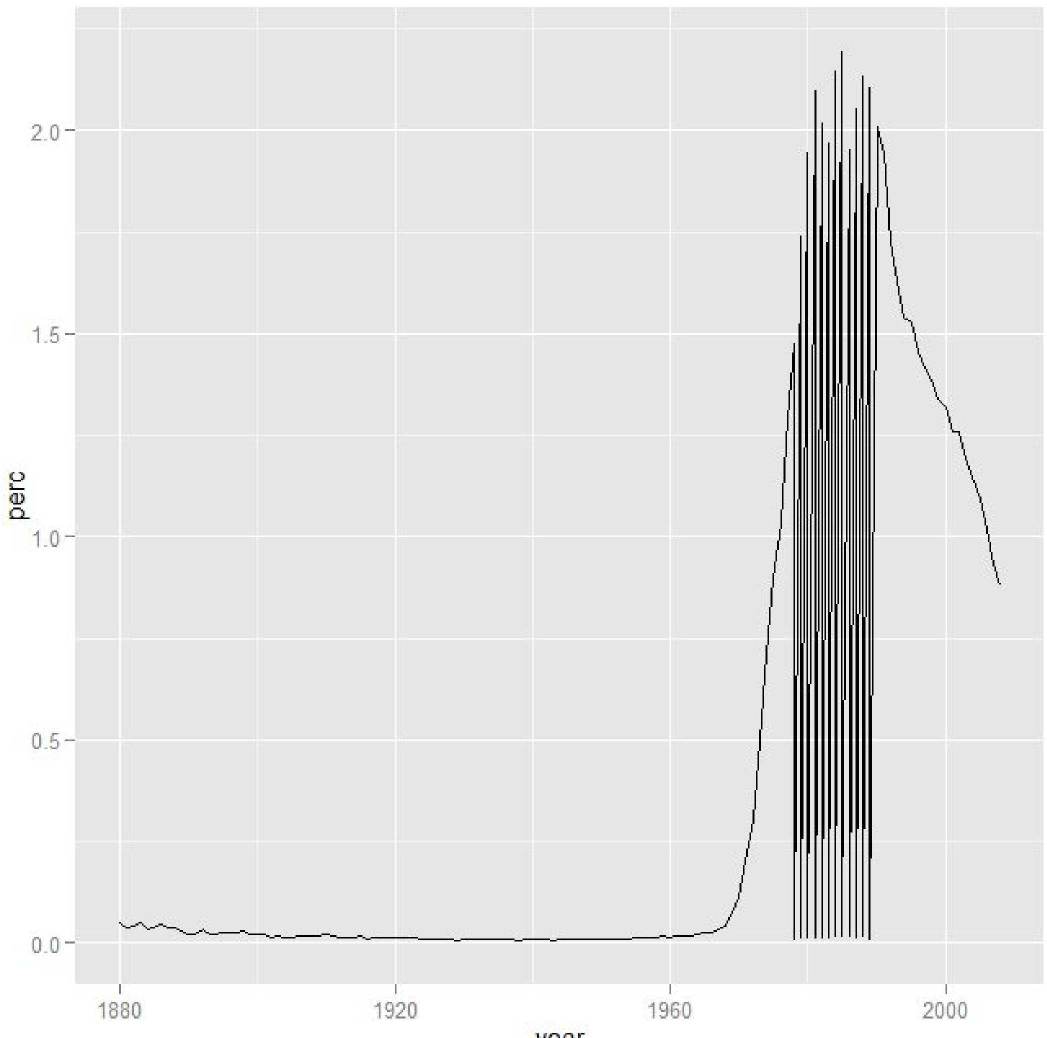

Popularity of Joshua and Jacob,1880-2008
perc
2.0
1.5
1.0
0.5
0.0
1880
1920
1960
2000
year
name
Jacob
Joshua

ggplot2


Popularity of Joshua and Jacob,1880-2008
?
perc
2.0
1.5
1.0
0.5
0.0
1880
1920
1960
2000
year
name
Jacob
Joshua

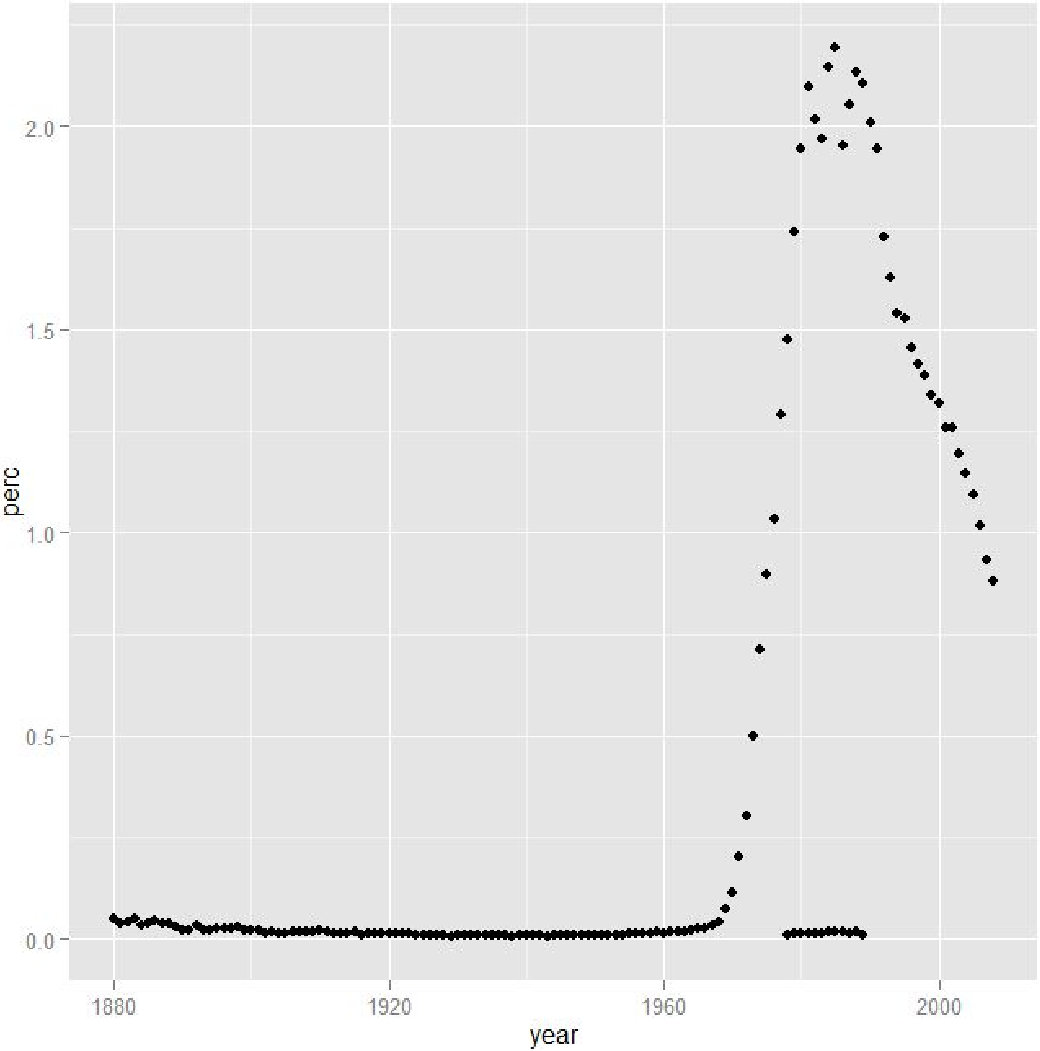
Baby names

```
joshua <- subset(bnames, name == "Joshua") #subset Joshua

qplot(year, perc, data = joshua, geom = "line") #quick plot
```


perc
2.0
1.5
1.0
0.5
0.0
1880
1920
1960
2000
year

Baby names

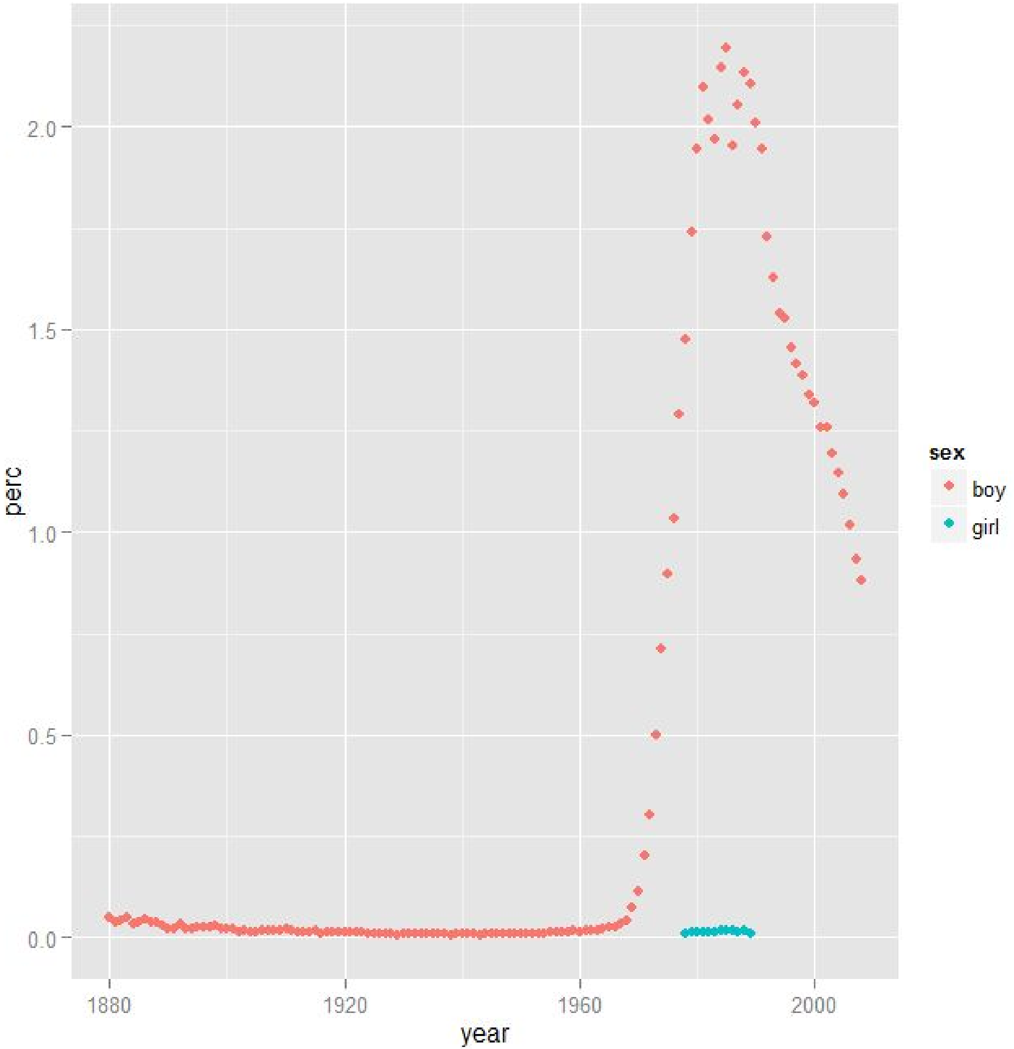
```
qplot(year, perc, data = joshua, geom = "point")    #quick plot
```


perc
2.0
1.5
1.0
0.5
0.0
1880
1920
1960
2000
year

Baby names

```
qplot(year, perc, data = joshua, geom = "point", color = sex)
```

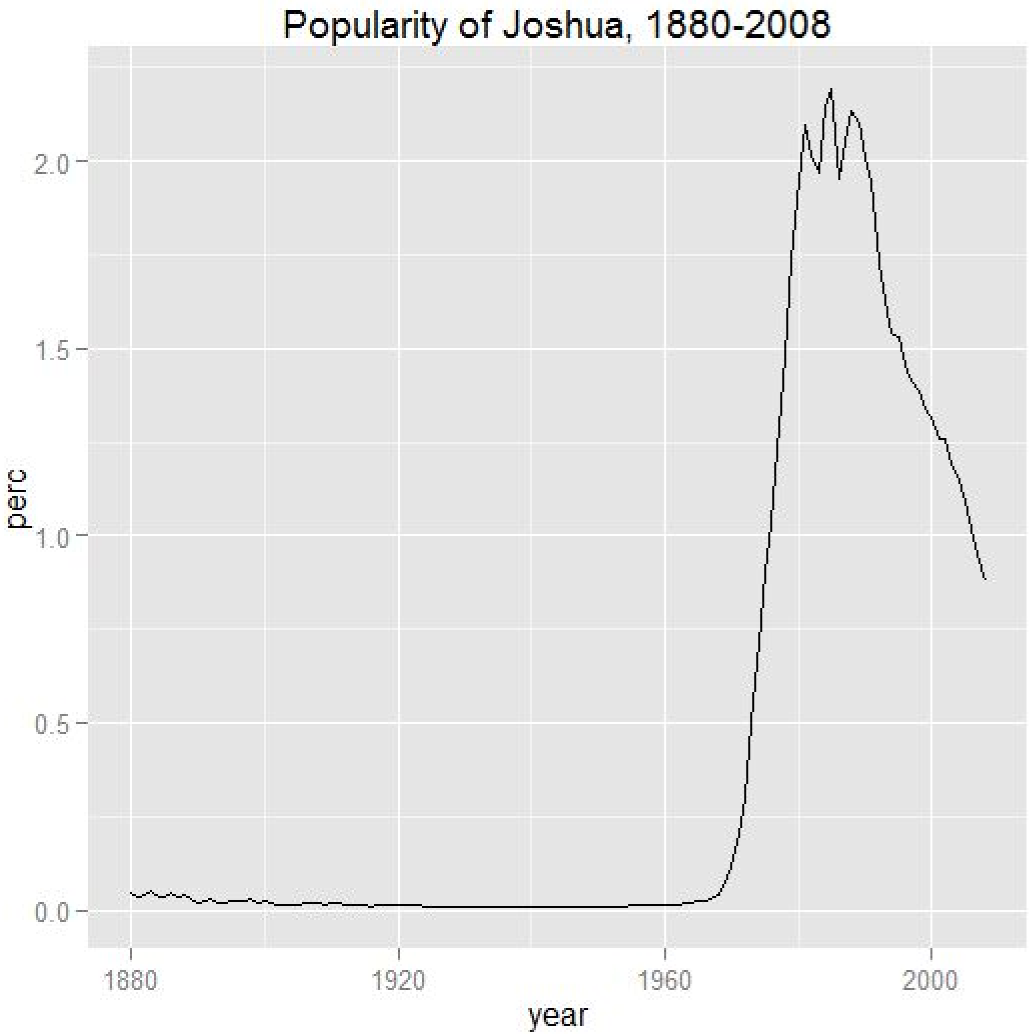
creates a different colored set of points for each group of sex (male, female)


2.0
1.5
1.0
0.5
0.0
perc
1880
1920
1960
2000
year
sex
● boy
● girl

Baby names

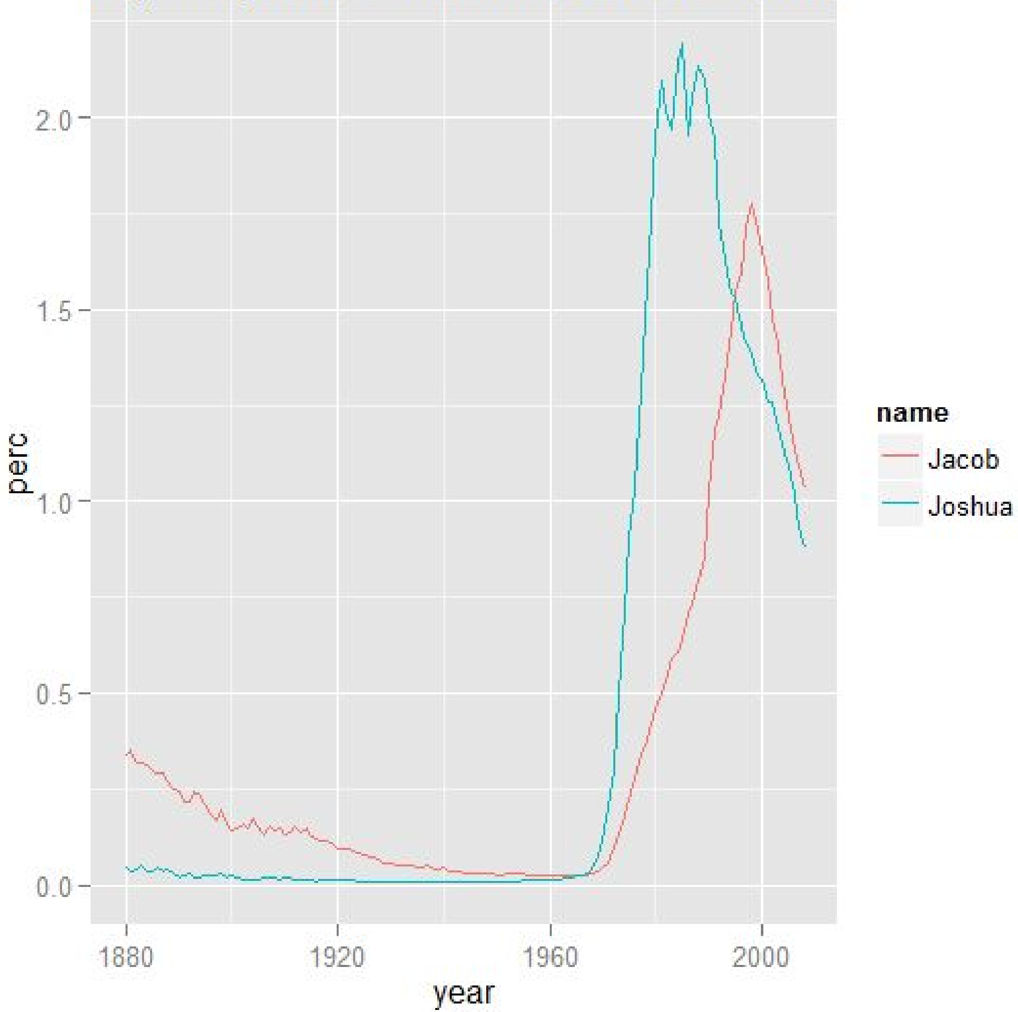
```
joshua <- subset(joshua, name == "Joshua" & sex == "boy")
```

```
qplot(year, perc, data = joshua, geom = "line") + ggtitle("Popularity of Joshua, 1880-2008")
```



Popularity of Joshua, 1880-2008
perc
2.0
1.5
1.0
0.5
0.0
1880
1920
1960
2000
year

Popularity of Joshua and Jacob, 1880-2008



Your Turn

1. Create an object containing a data frame that is a subset of your name.

2. Create a new percentage variable ‘perc’, where prop * 100 *hint: mutate()*

3. Create a plot of the popularity of your name over time. Weird trends? Do you need to subset again?

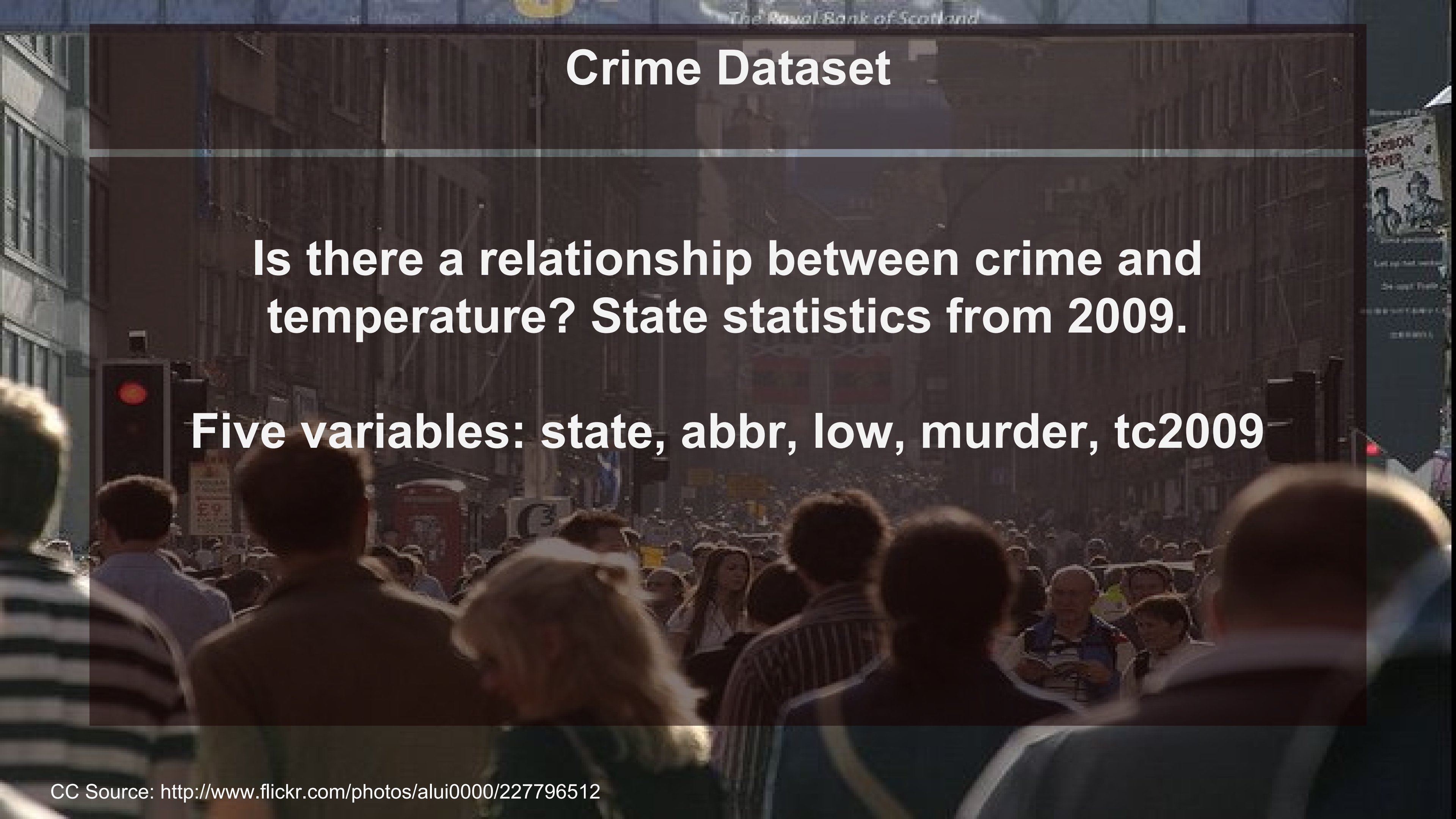
4. Reorder the rows from descending from highest to lowest by perc. What year was most popular for your name? *hint: arrange()*

5. Reorder by year. What were the most popular names in your birth year?

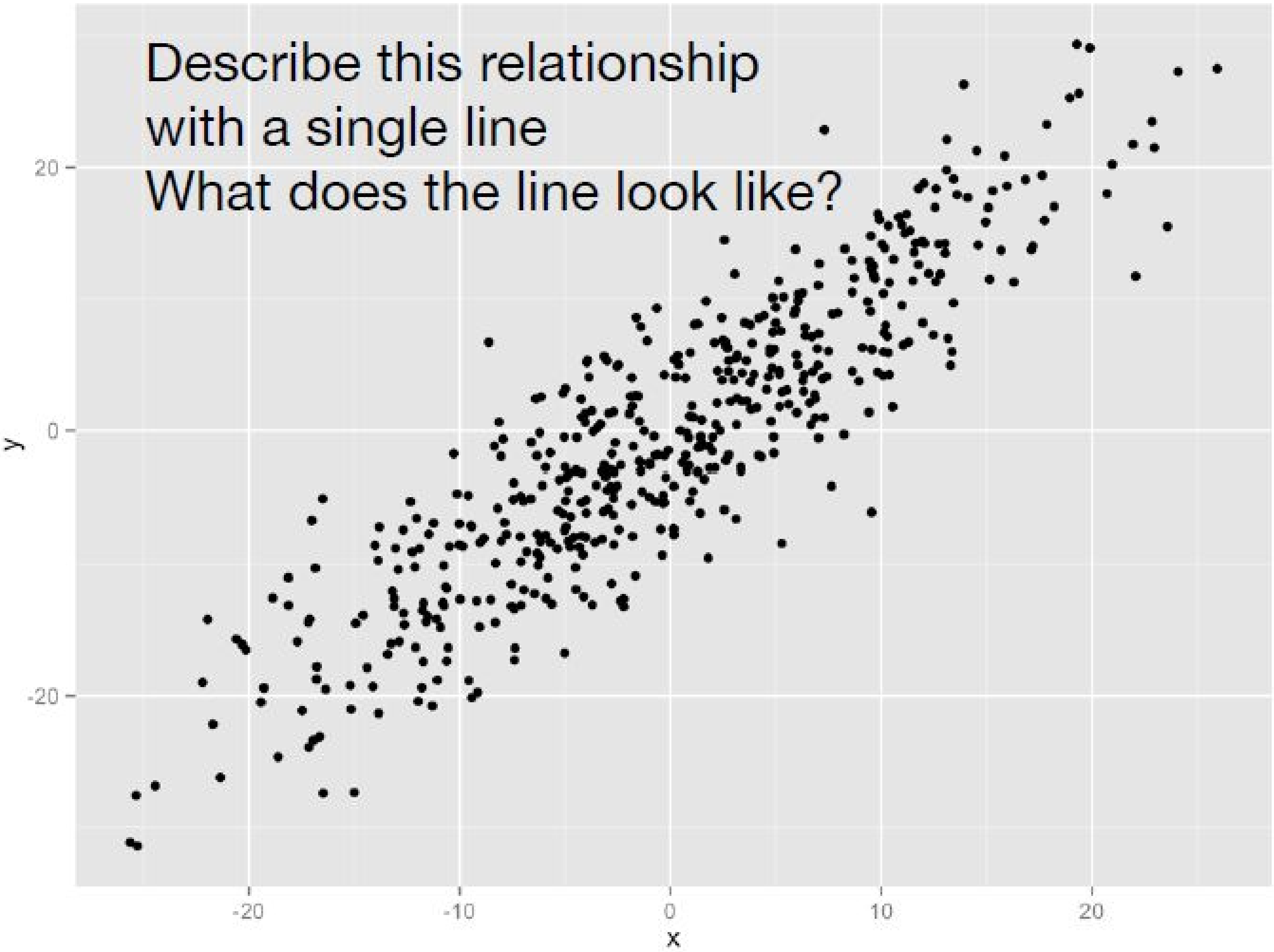
Crime Dataset

Is there a relationship between crime and temperature? State statistics from 2009.

Five variables: state, abbr, low, murder, tc2009

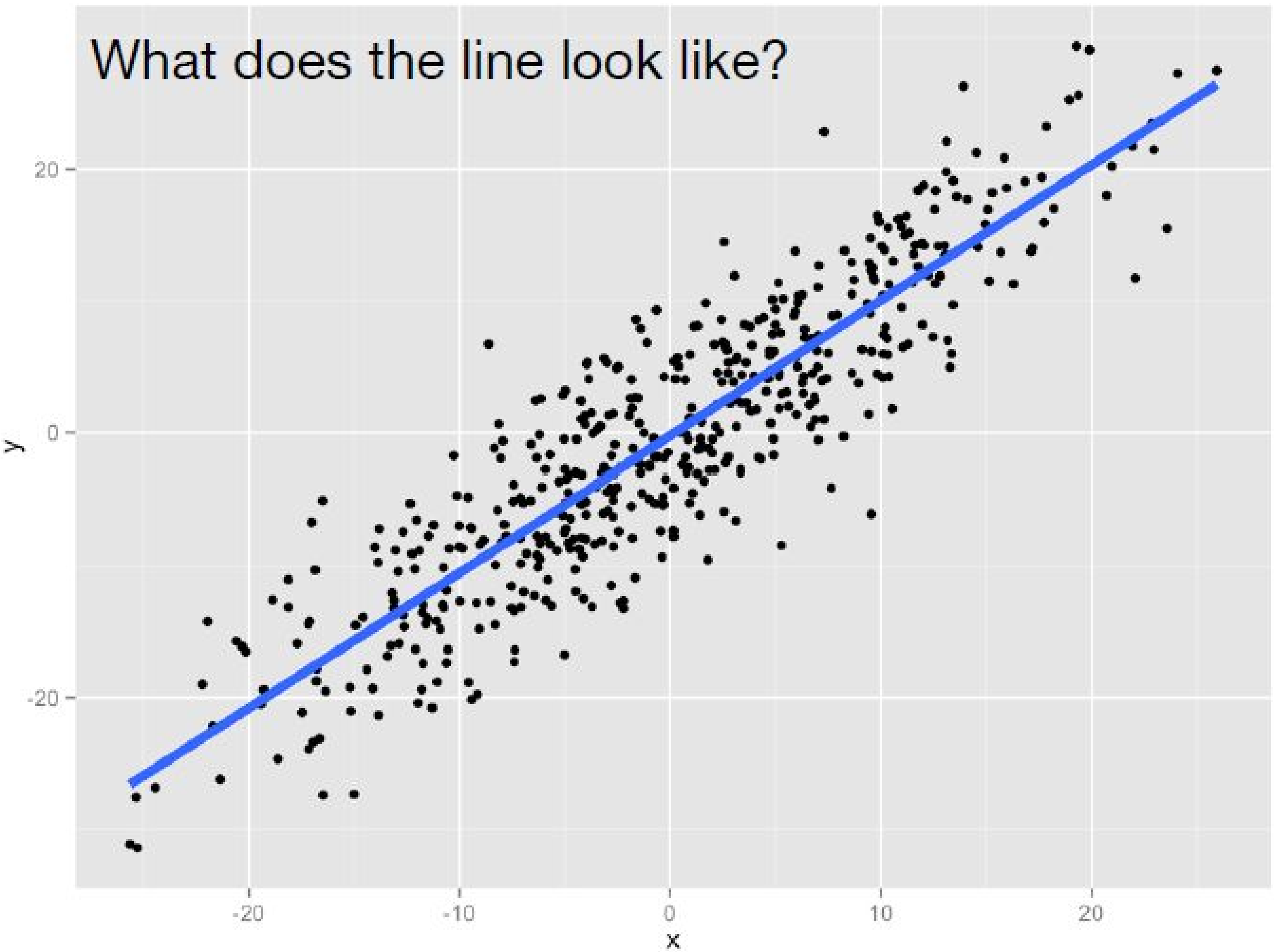


CC Source: http://www.flickr.com/photos/alui0000/227796512



Describe this relationship
with a single line
What does the line look like?

What does the line look like?



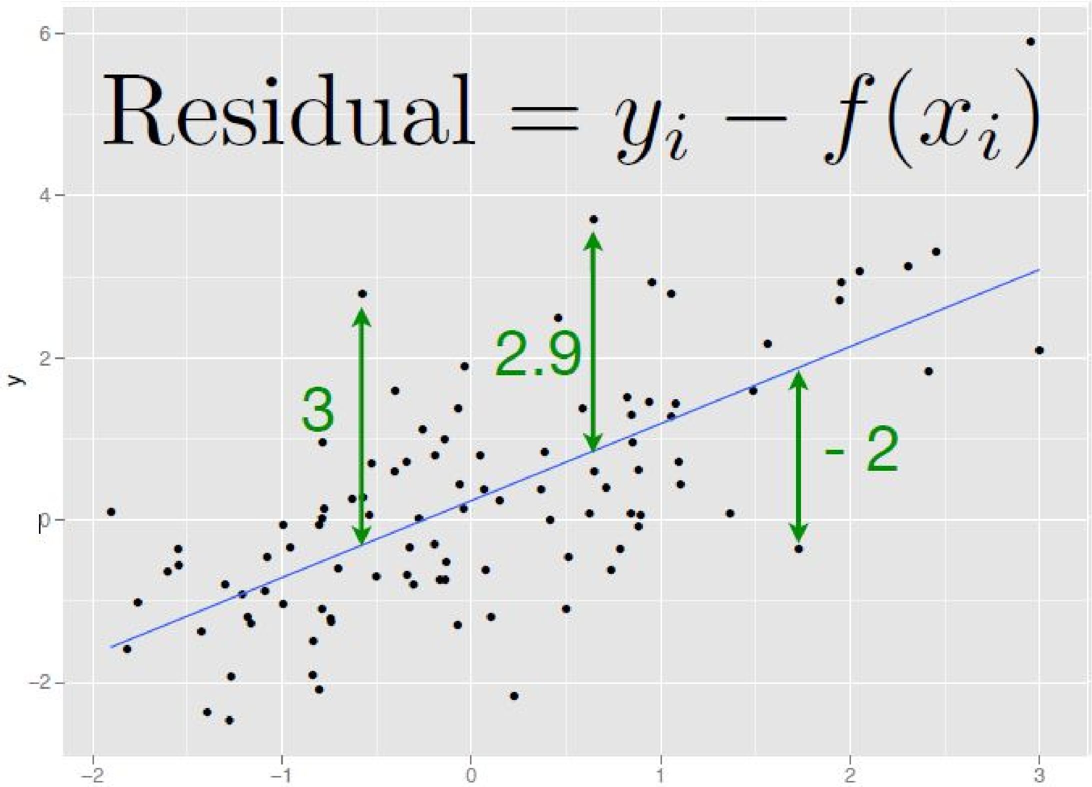
Linear Modeling

The linear regression algorithm constrains $f(\hat{x})$ to have the form,

$$f(\hat{x}) = \alpha + \beta x + \epsilon$$

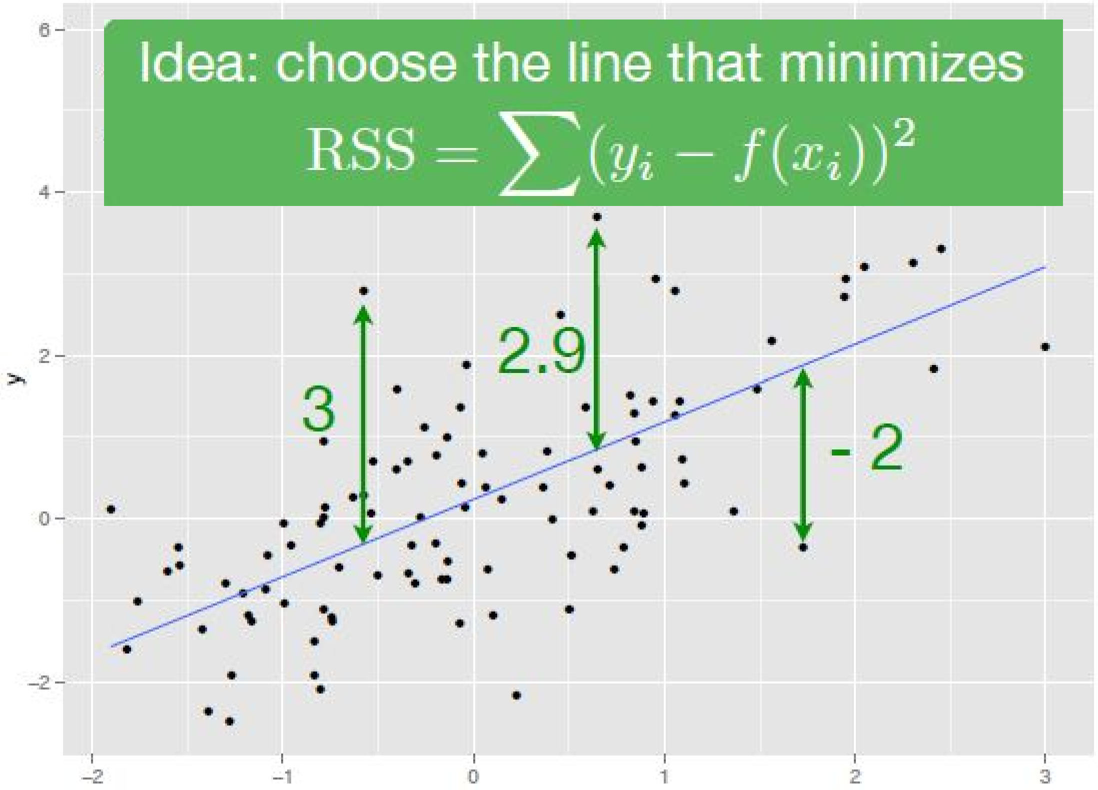
e.g., $f(\hat{x})$ will be a straight line in x .

How do we fit the best line?



Linear Modeling

How do we fit the best line?



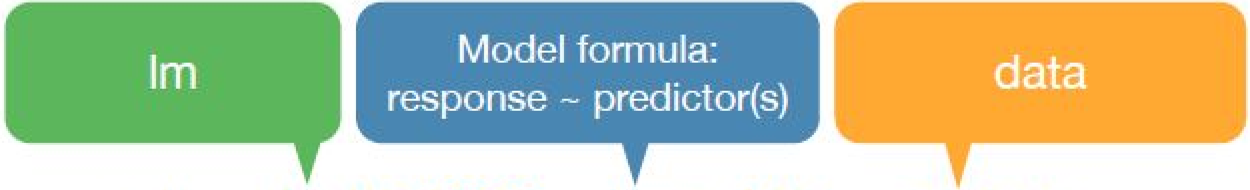
Idea: choose the line that minimizes

$$\mathrm{RSS} = \sum (y_i - f(x_i))^2$$

Linear Modeling

```
#read in crime dataset

crime <- read.csv("crime.csv")

head(crime)
```

Linear Modeling



lm

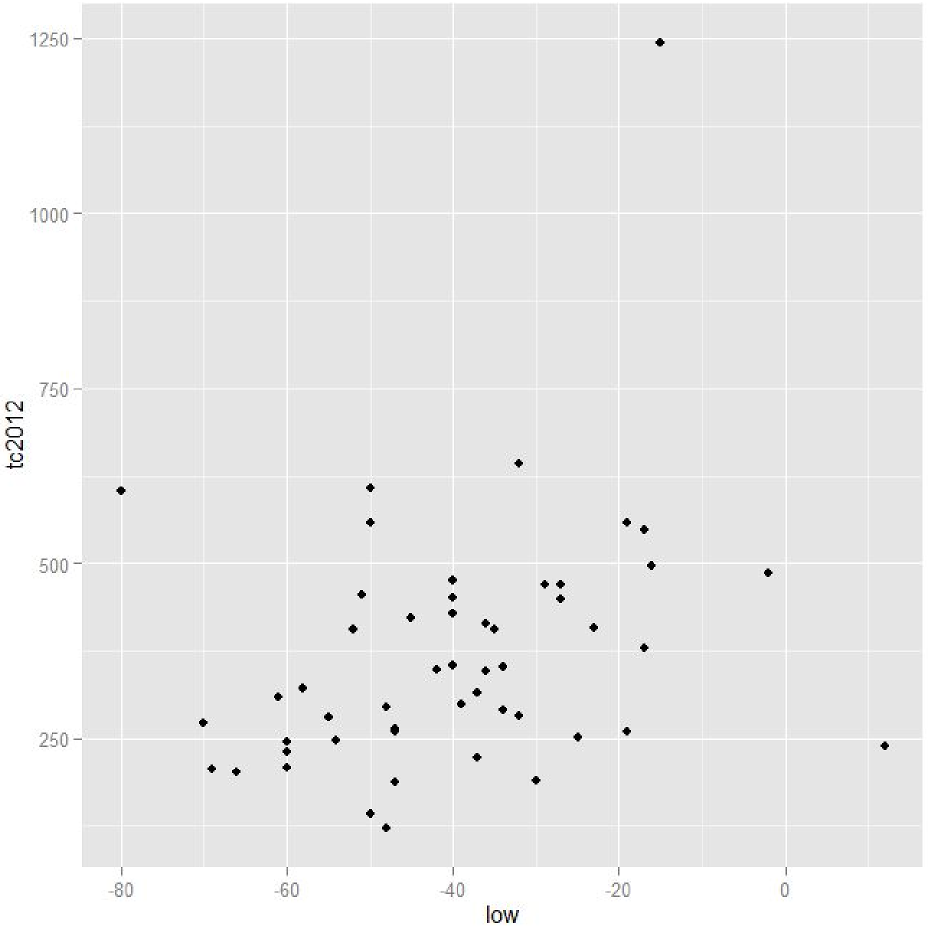
Model formula:
response ~ predictor(s)

data

```
mod <- lm(tc2009 ~ low, data = crime)
```

Linear Modeling

qplot(low, tc2009, data=crime)



Linear Modeling

```
mod <- lm(tc2009 ~ low, data=crime)
mod

## Call:
## lm(formula = tc2009 ~ low, data = crime)

## Coefficients:
## (Intercept)       low
##   491.135       3.002
```

Linear Modeling

$$y = \alpha + \beta x + \epsilon$$

α is the expected value of y when x is 0.

β is the expected increase in y associated with a one unit increase in x

Linear Modeling

```
coefficients(mod)


(Intercept)     low
 491.13        3.002
```

Linear Modeling

```
coefficients(mod)


(Intercept)     low
 491.13       3.002
```

α β

Linear Modeling

```
coefficients(mod)


(Intercept)    low
 491.13       3.002
```

α β

The best estimate of tc2009 for a state with low = -10 is,

491.13 + 3.002 * **(-10)** = 461.11

Extracting info

A common pattern for R models: store and explore

1. Create model object

2. Run function(s) on model object

summary()
predict()
resid()
plot()

Linear Modeling

summary(mod)

```
Call:
lm(formula = tc2009 ~ low, data = crime)

Residuals:
    Min      1Q  Median      3Q     Max
-287.96  -88.37  -16.44   62.13  797.60

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  491.135     59.588   8.242 8.16e-11 ***
low            3.002      1.365   2.200   0.0326 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 172.2 on 49 degrees of freedom
Multiple R-squared:  0.08986,   Adjusted R-squared:  0.07129
F-statistic: 4.838 on 1 and 49 DF,  p-value: 0.03259
```

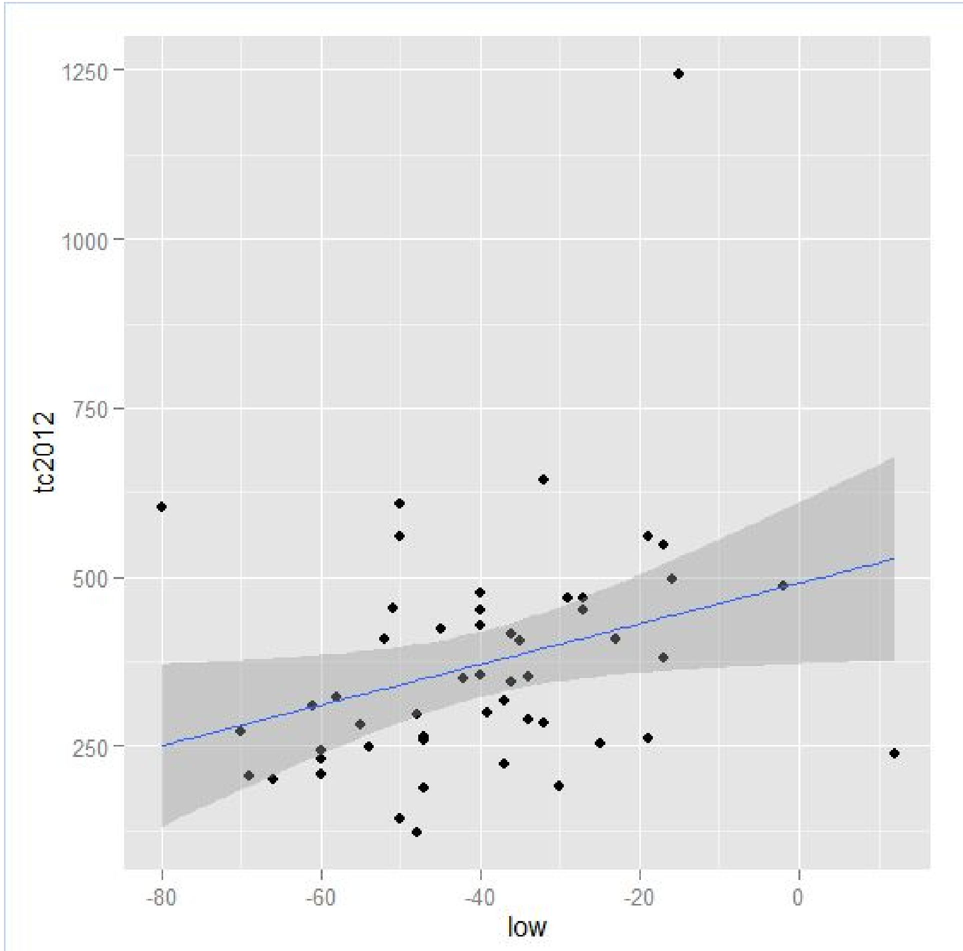
Linear Modeling

summary(mod)

```
Call:
lm(formula = tc2009 ~ low, data = crime)

Residuals:
    Min      1Q  Median      3Q     Max 
-287.96  -88.37  -16.44   62.13  797.60 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  491.135     59.588   8.242 8.16e-11 ***
low            3.002      1.365   2.200   0.0326 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 172.2 on 49 degrees of freedom
Multiple R-squared:  0.08986,	Adjusted R-squared:  0.07129 
F-statistic: 4.838 on 1 and 49 DF,  p-value: 0.03259
```

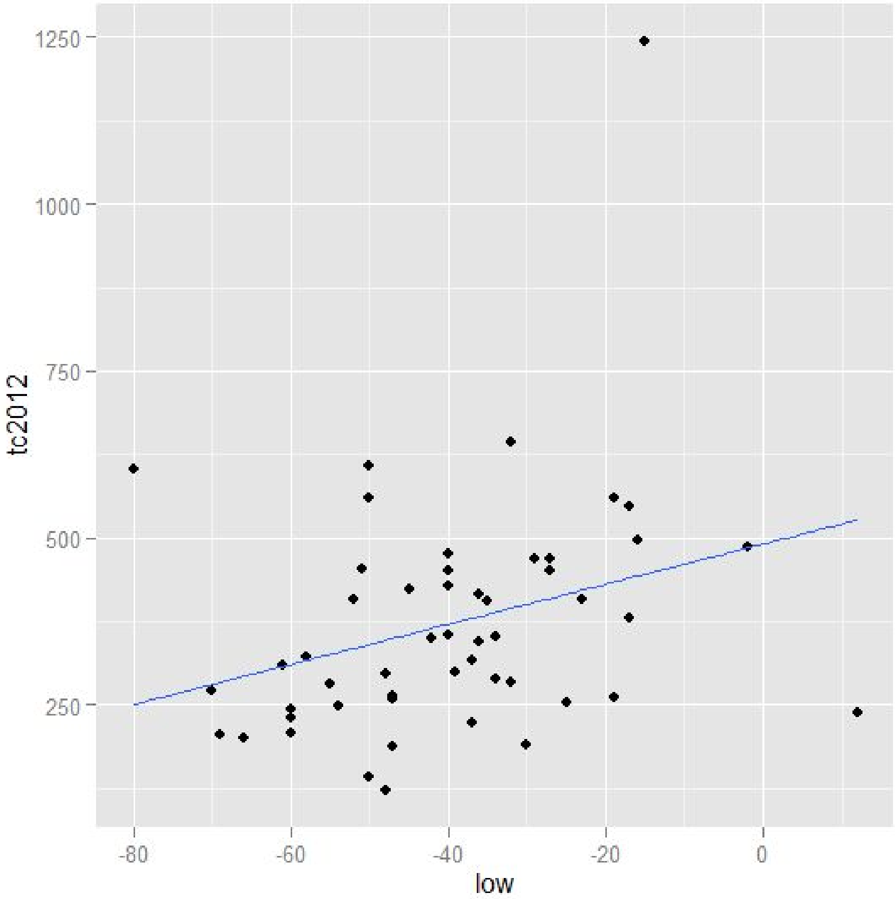
Does the model do better at predicting Y than random chance?

Linear Modeling



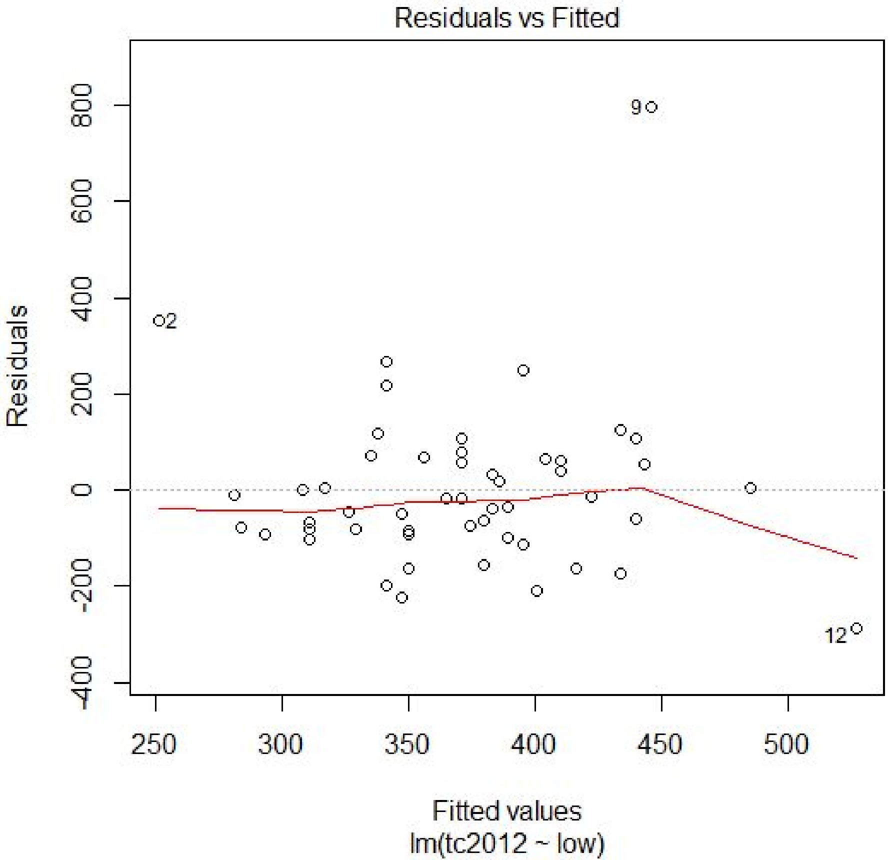
```
qplot(low, tc2009, data = crime) +
geom_smooth(method = “lm”)
```

Linear Modeling



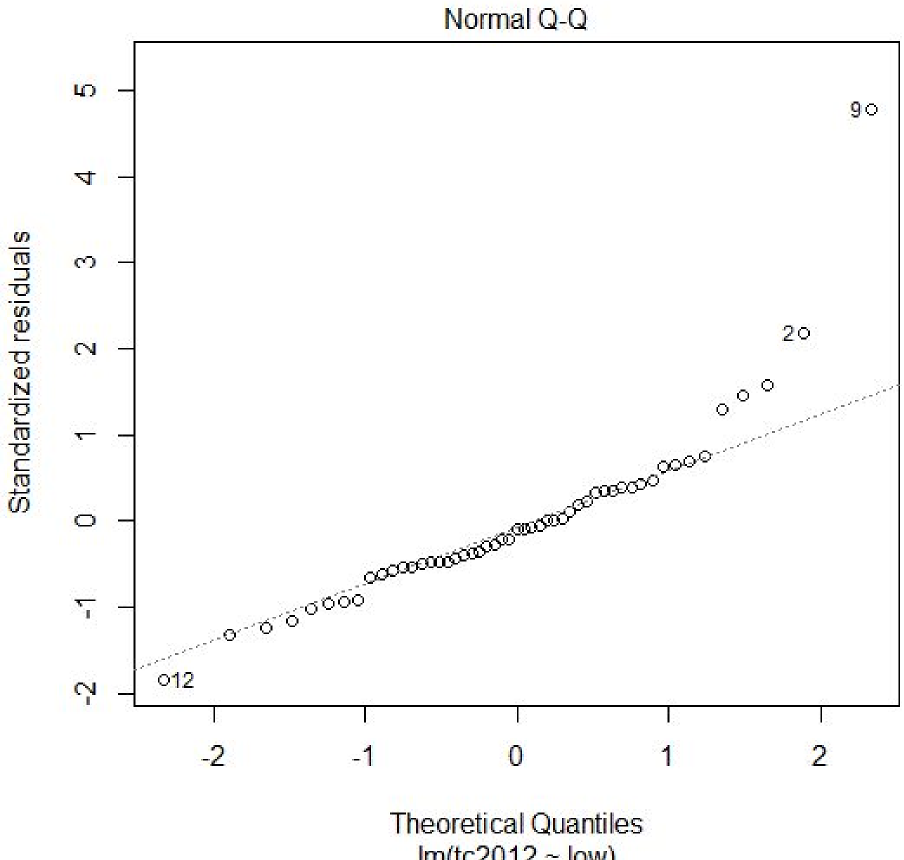
```
qplot(low, tc2009, data = crime) +
geom_smooth(se = FALSE, method = “lm”)
```

Linear Modeling



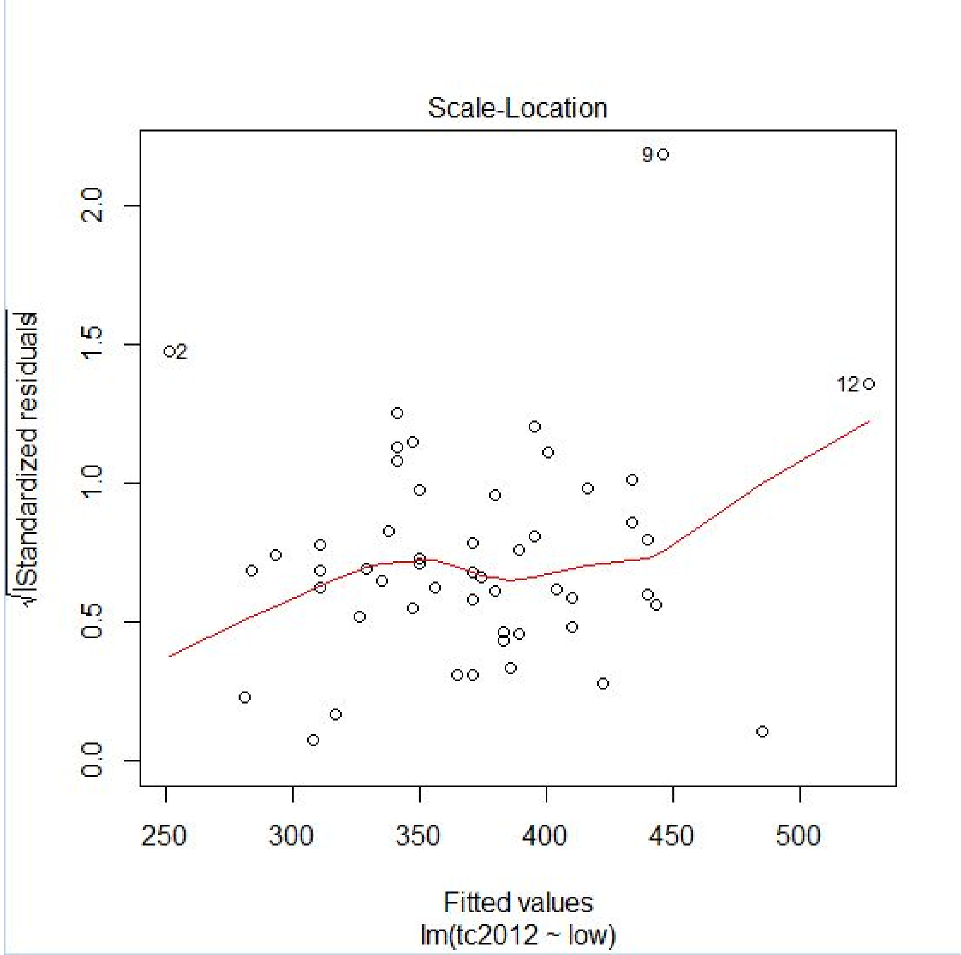
```
plot(mod)
```

Linear Modeling



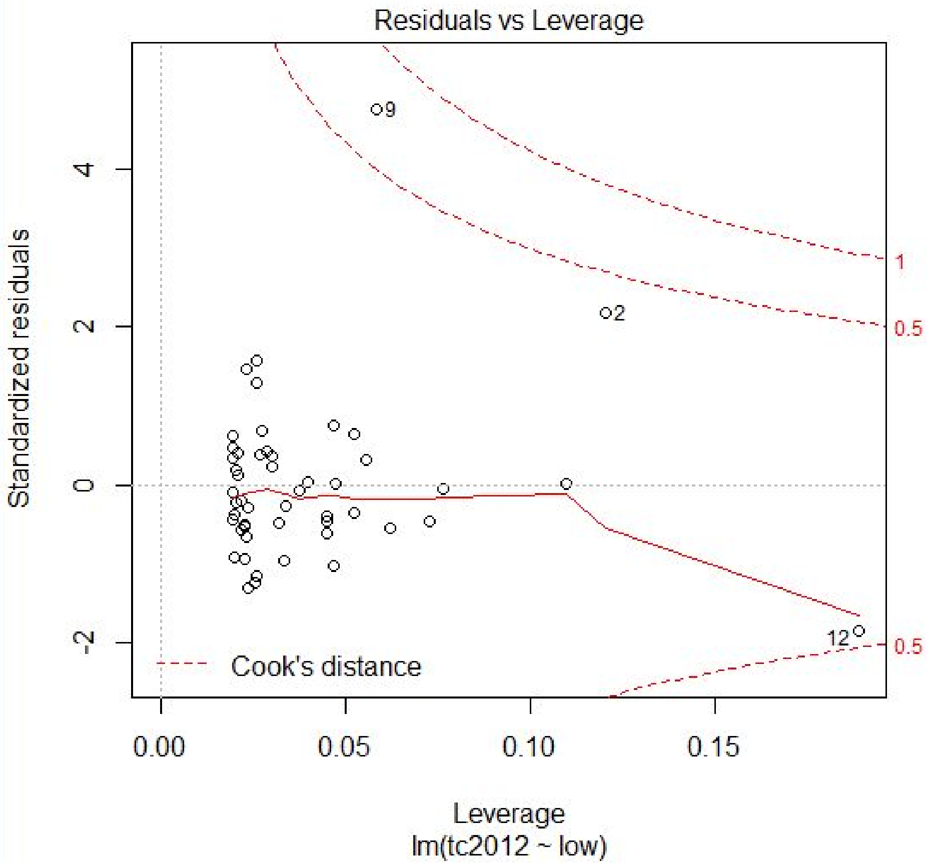
```
plot(mod)
```

Linear Modeling



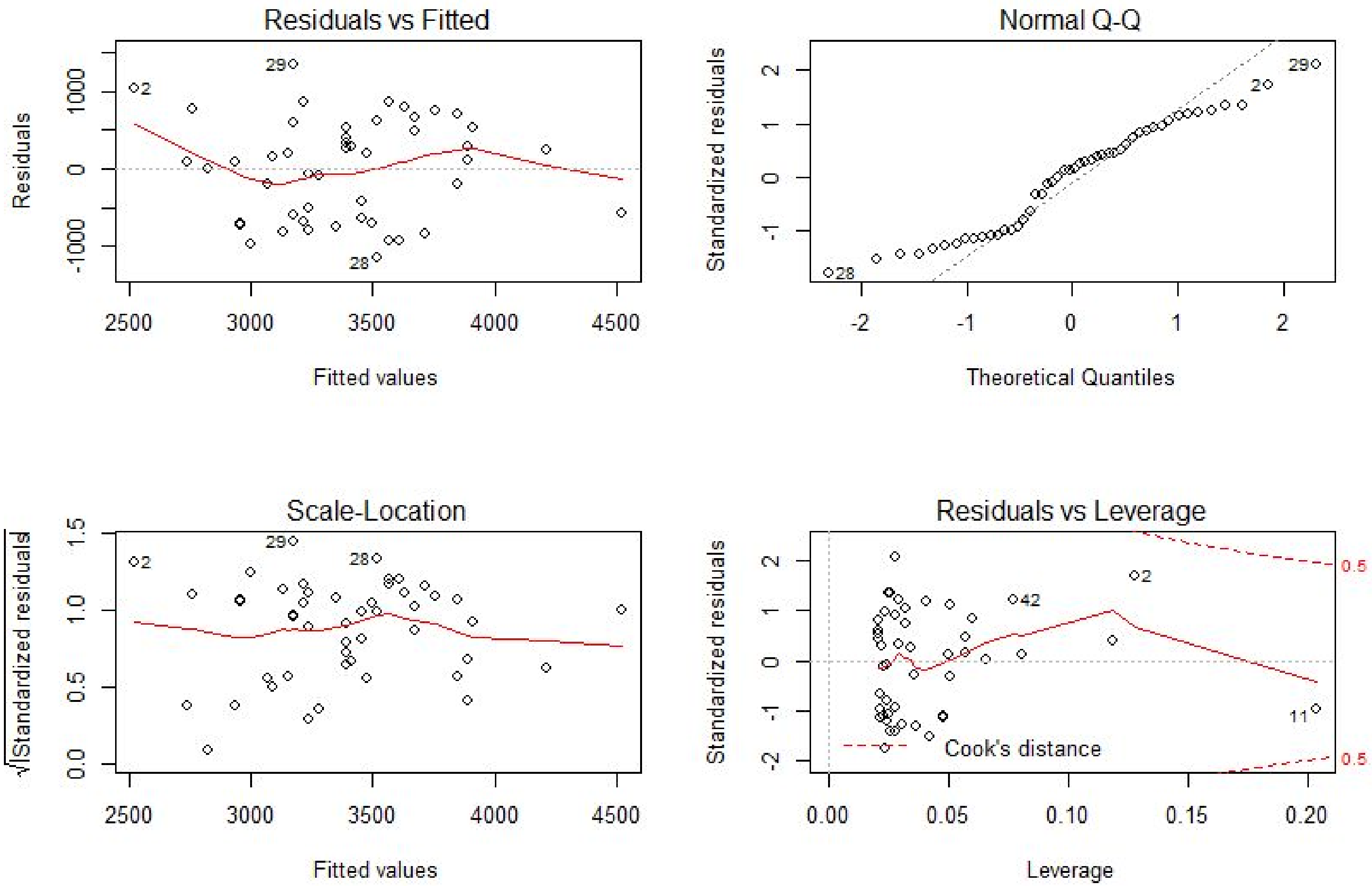
```
plot(mod)
```

Linear Modeling

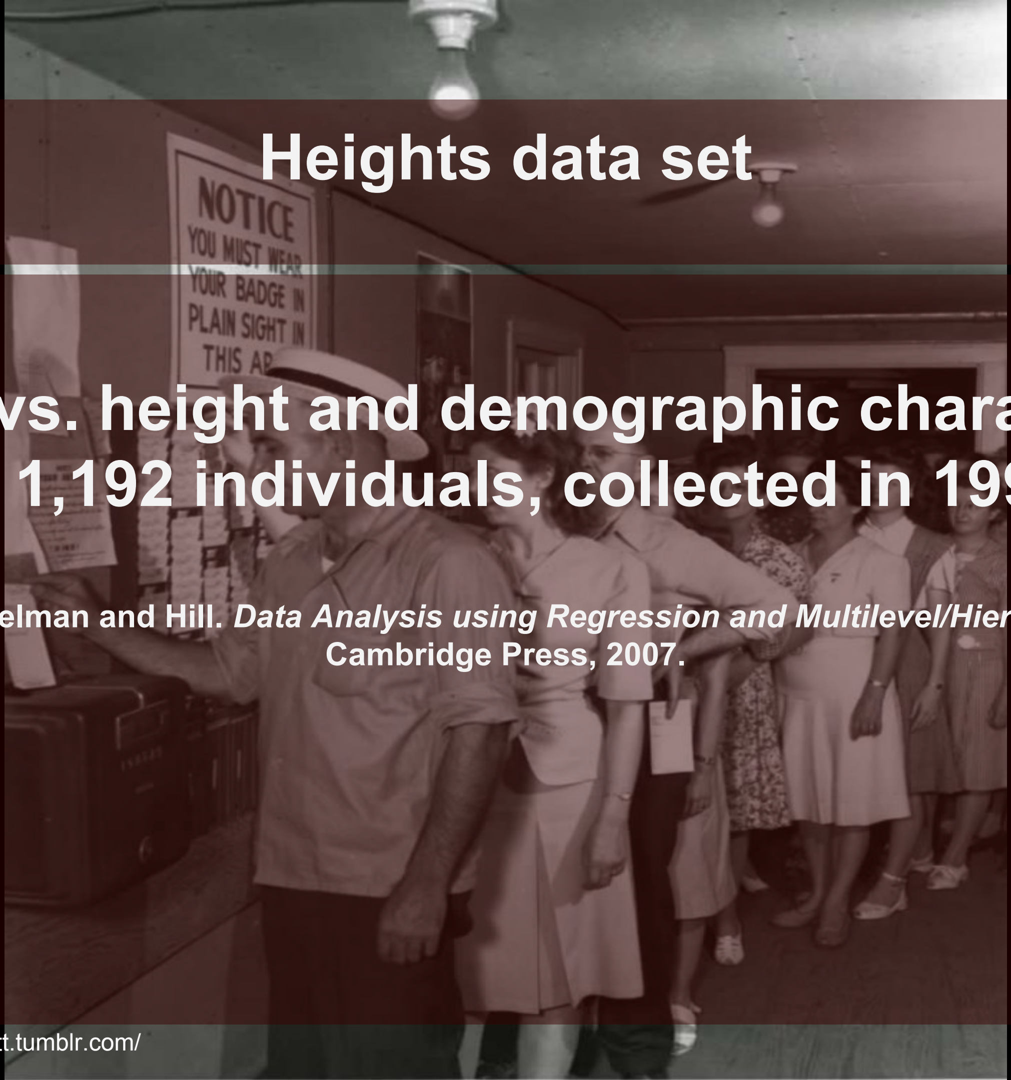


```
plot(mod)
```

Linear Modeling



```
par(mfrow = c(2,2))
plot(mod)
```



Heights data set

Earnings vs. height and demographic characteristics of 1,192 individuals, collected in 1994

Sampled from Gelman and Hill. *Data Analysis using Regression and Multilevel/Hierarchical Models.* Cambridge Press, 2007.

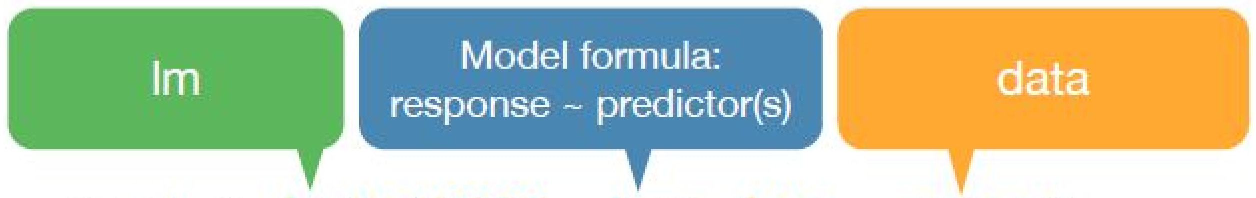
CC Source: http://photosofedwestcott.tumblr.com/

Your Turn

Read in the “heights.csv” file as an object.
Regress earn on height and name this linear model object *m1*

Discuss your interpretation of the height coefficient with your neighbor

Linear Modeling



lm

Model formula:
response ~ predictor(s)

data

```
m1 <- lm(earn ~ height, data = heights)
```

Linear Modeling

```
m1 <- lm(earn ~ height, data = heights)
summary(m1)
```

```
Residuals:
   Min     1Q Median     3Q    Max
-30043 -11422  -3608   6443 173488

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -58611.9     9525.6  -6.153 1.04e-09 ***
height        1221.9      142.1   8.598  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18900 on 1190 degrees of freedom
Multiple R-squared:  0.05849, Adjusted R-squared:  0.0577
F-statistic: 73.93 on 1 and 1190 DF,  p-value: < 2.2e-16
```

Linear Modeling

```
m1 <- lm(earn ~ height, data = heights)
summary(m1)
```

```
Residuals:
   Min     1Q Median     3Q    Max
-30043 -11422  -3608   6443 173488

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -58611.9     9525.6  -6.153 1.04e-09 ***
height        1221.9      142.1   8.598  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18900 on 1190 degrees of freedom
Multiple R-squared:  0.05849, Adjusted R-squared:  0.0577
F-statistic: 73.93 on 1 and 1190 DF,  p-value: < 2.2e-16
```

Linear Modeling

m1 <- lm(earn ~ height, data = heights)
summary(m1)

```
Residuals:
   Min     1Q Median     3Q    Max 
-30043 -11422  -3608   6443 173488 

Coefficients:
            Estimate Std. Error t value  Pr(>|t|)    
(Intercept) -58611.9     9525.6  -6.153  1.04e-09 ***
height        1221.9      142.1   8.598   < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18900 on 1190 degrees of freedom
Multiple R-squared:  0.05849, Adjusted R-squared:  0.0577 
F-statistic: 73.93 on 1 and 1190 DF,  p-value: < 2.2e-16
```

Linear Modeling

m1 <- lm(earn ~ height, data = heights)
summary(m1)

```
Residuals:
   Min      1Q  Median      3Q     Max 
-30043 -11422   -3608    6443  173488 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -58611.9     9525.6  -6.153 1.04e-09 ***
height        1221.9      142.1   8.598  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18900 on 1190 degrees of freedom
Multiple R-squared:  0.05849, Adjusted R-squared:  0.0577 
F-statistic: 73.93 on 1 and 1190 DF,  p-value: < 2.2e-16
```

Each 1” increase in height is associated with a $1,221.90 increase in earnings

The best estimate of earn for someone 68 inches tall is,

-58611.9 + 1221.95 * (68) = 24,477.3

Linear Modeling

```
m2 <- lm(earn ~ height + sex, data = heights)
summary(m2)
```

Linear Modeling

```
m2 <- lm(earn ~ height + sex, data = heights)
summary(m2)
```

```
Call:
lm(formula = earn ~ height + sex, data = heights)

Residuals:
   Min     1Q Median     3Q    Max 
-30018 -11127  -3260   6080 170360 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -5732.9    12665.4  -0.453   0.6509    
height         371.7      195.7   1.899   0.0578 .  
sexmale       9479.7     1526.0   6.212 7.21e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18610 on 1189 degrees of freedom
Multiple R-squared:  0.08809, Adjusted R-squared:  0.08656 
F-statistic: 57.43 on 2 and 1189 DF,  p-value: < 2.2e-16
```

Linear Modeling

```
m2 <- lm(earn ~ height + sex, data = heights)
summary(m2)
```

Regression with categorical variables creates a “reference” category. In this example the reference or baseline is female and the coefficient (sexmale) is rate of change relative to that baseline.

How can we change this to look at it the other way around?

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -5732.9    12665.4  -0.453   0.6509
height         371.7      195.7   1.899   0.0578 .
sexmale       9479.7     1526.0   6.212 7.21e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘
’ 1

Residual standard error: 18610 on 1189 degrees of freedom
Multiple R-squared:  0.08809, Adjusted R-squared:  0.08656
F-statistic: 57.43 on 2 and 1189 DF,  p-value: < 2.2e-16
```

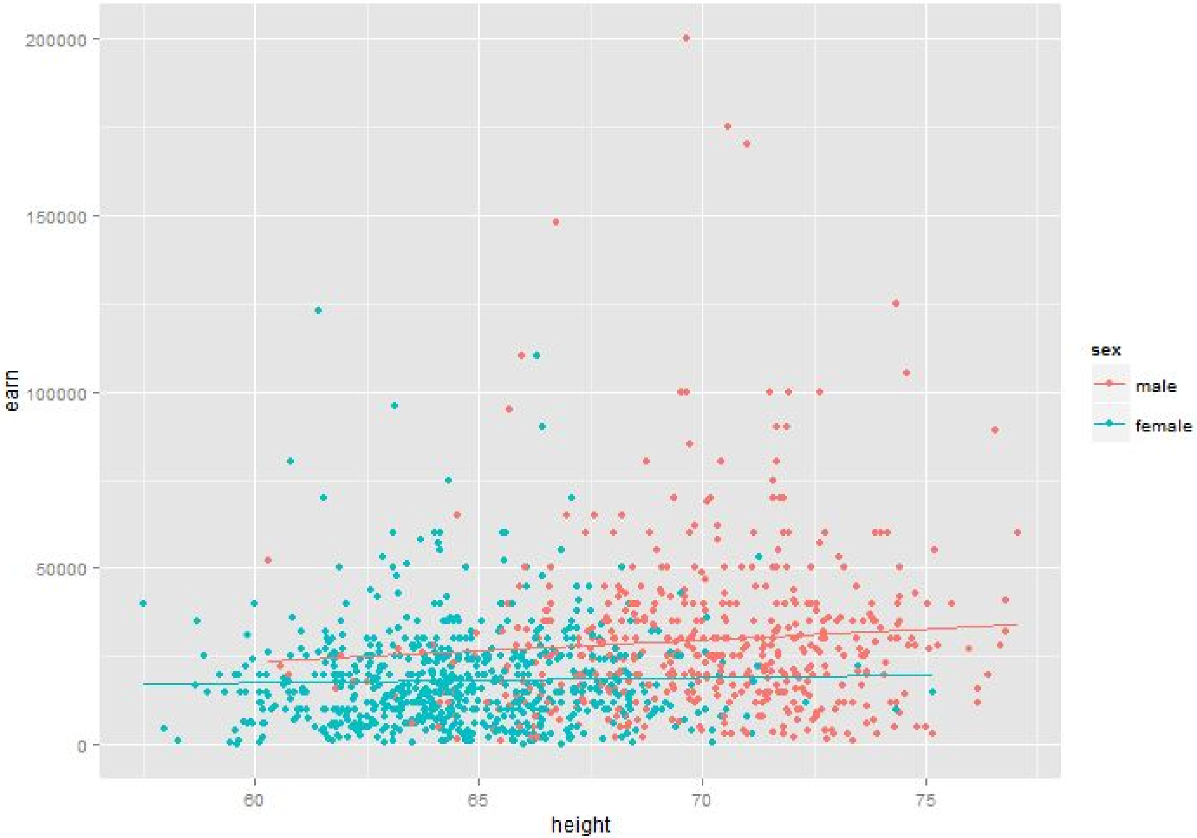
Linear Modeling

```
heights$sex <- factor(heights$sex, levels = c("male", "female"))
m3 <- lm(earn ~ height + sex, data = heights)
summary(m3)
```

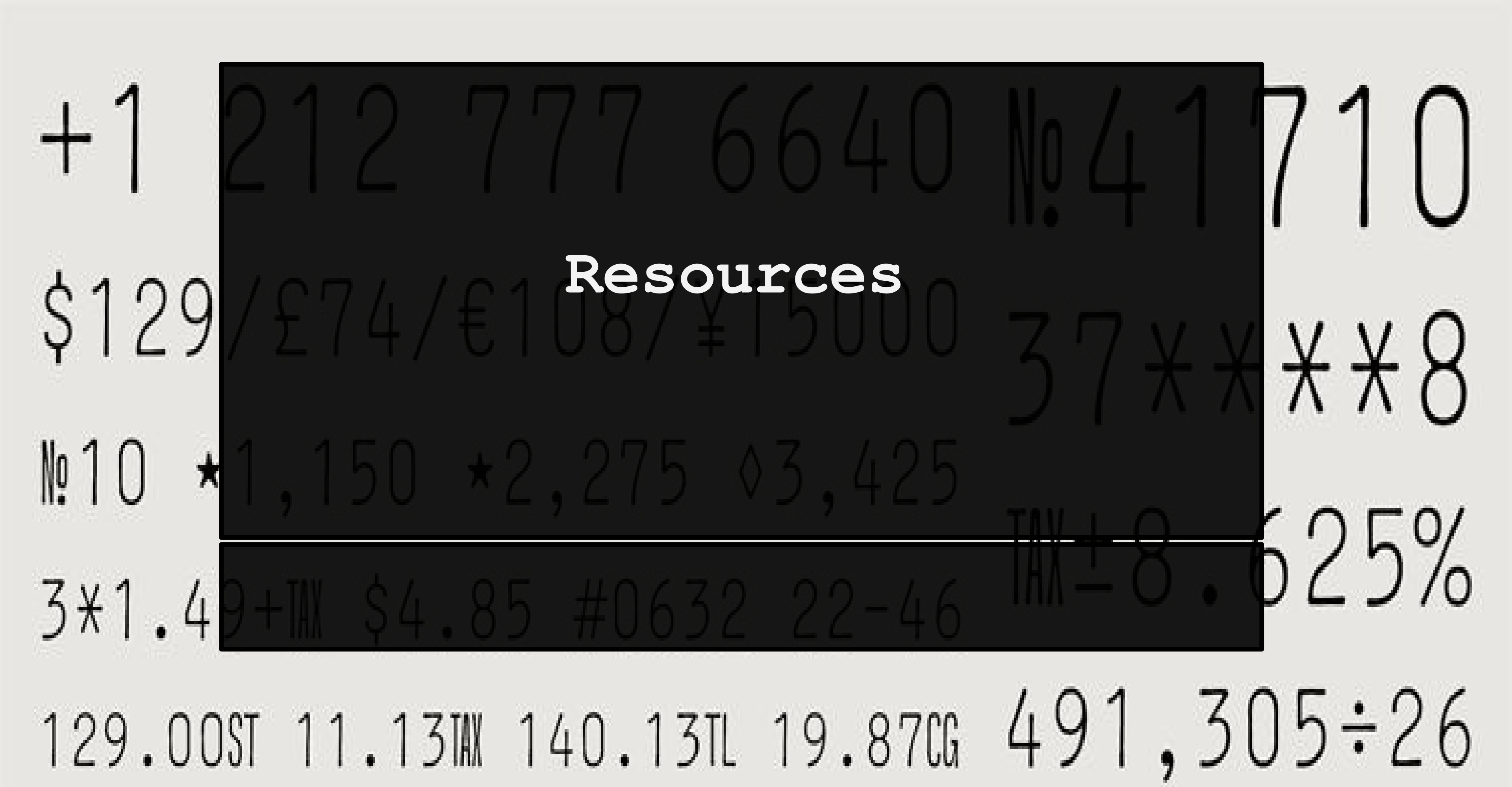
```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3746.8    13737.5   0.273   0.7851
height         371.7      195.7   1.899   0.0578 .
sexfemale    -9479.7     1526.0  -6.212 7.21e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18610 on 1189 degrees of freedom
Multiple R-squared:  0.08809, Adjusted R-squared:  0.08656
F-statistic: 57.43 on 2 and 1189 DF,  p-value: < 2.2e-16
```

Linear Modeling

```
qplot(height, earn, color= sex, data = heights) + geom_smooth(method = "lm", se = FALSE)
```



Resources

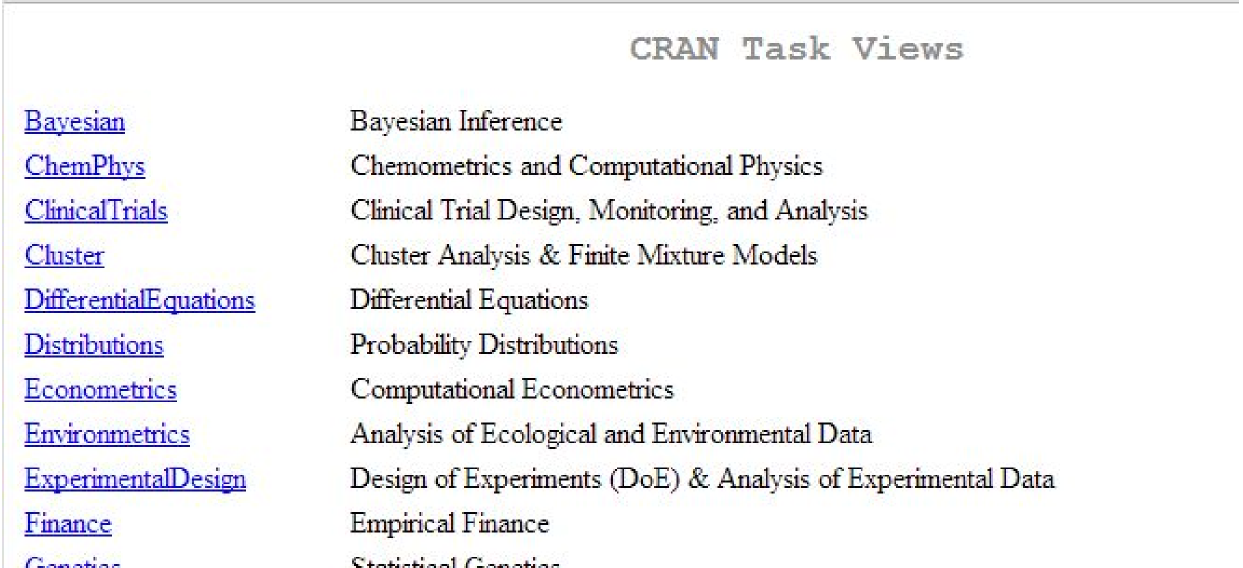

+1 212 777 6640 №417710
$129/£74/€108/¥15000 37****8
№10 ★1,150 ★2,275 ◊3,425 TAX±8.625%
3*1.49+TAX $4.85 #0632 22-46
129.00ST 11.13TAX 140.13TL 19.87CG 491,305÷26

Source: http://www.typography.com/images/whatsInsidePageImages/numbers-revenue.png

Cran task views

http://cran.r-project.org/web/views/

Categorization of R packages for specific applications. For example, the "Finance" task view displays packages which are applicable in Finance.



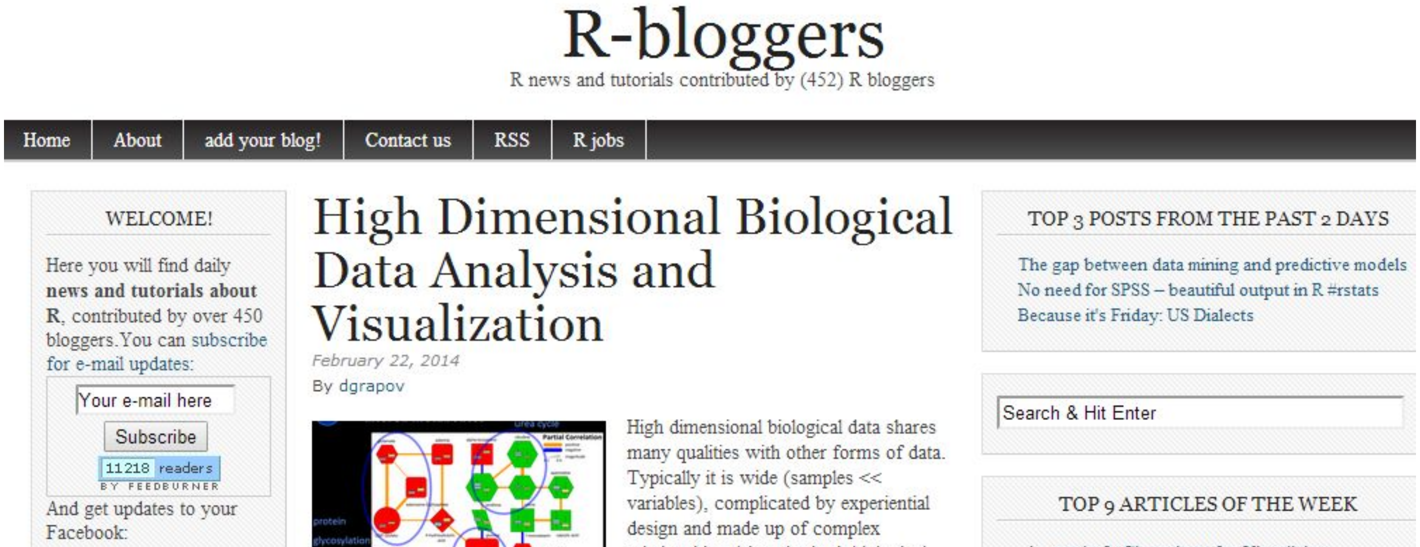
CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics

R Bloggers

http://www.r-bloggers.com/

Aggregation of R blogs. This is a great resource to read about new things in R and what the R community is doing. You'll often find new tips and tricks to enhance your R skills.



R-bloggers

R news and tutorials contributed by (452) R bloggers

WELCOME!

Here you will find daily **news and tutorials about R**, contributed by over 450 bloggers. You can subscribe for e-mail updates:

And get updates to your Facebook:

High Dimensional Biological Data Analysis and Visualization

February 22, 2014

By dgrapov

High dimensional biological data shares many qualities with other forms of data. Typically it is wide (samples << variables), complicated by experiential design and made up of complex

TOP 3 POSTS FROM THE PAST 2 DAYS

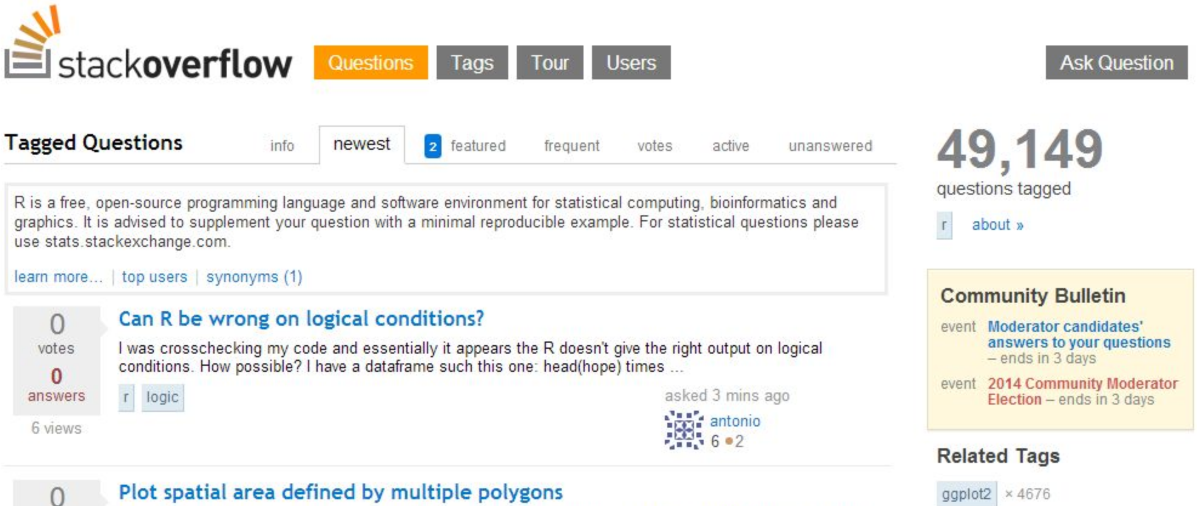
The gap between data mining and predictive models
No need for SPSS – beautiful output in R #rstats
Because it's Friday: US Dialects

TOP 9 ARTICLES OF THE WEEK

Stack Overflow

http://stackoverflow.com/questions/tagged/r

The R tag on Stack Overflow is becoming an increasingly important resource for seeking answers to R related questions. You can search the R tag in general, or refine your search to another tag such as ggplot2 or sweave.



Tagged Questions

49,149 questions tagged

R is a free, open-source programming language and software environment for statistical computing, bioinformatics and graphics. It is advised to supplement your question with a minimal reproducible example. For statistical questions please use stats.stackexchange.com.

learn more… | top users | synonyms (1)

0 votes 0 answers 6 views

Can R be wrong on logical conditions?

I was crosschecking my code and essentially it appears the R doesn't give the right output on logical conditions. How possible? I have a dataframe such this one: head(hope) times ...

r logic

asked 3 mins ago antonio 6 ●2

Community Bulletin

event Moderator candidates' answers to your questions – ends in 3 days

event 2014 Community Moderator Election – ends in 3 days

Related Tags

ggplot2 × 4676

0

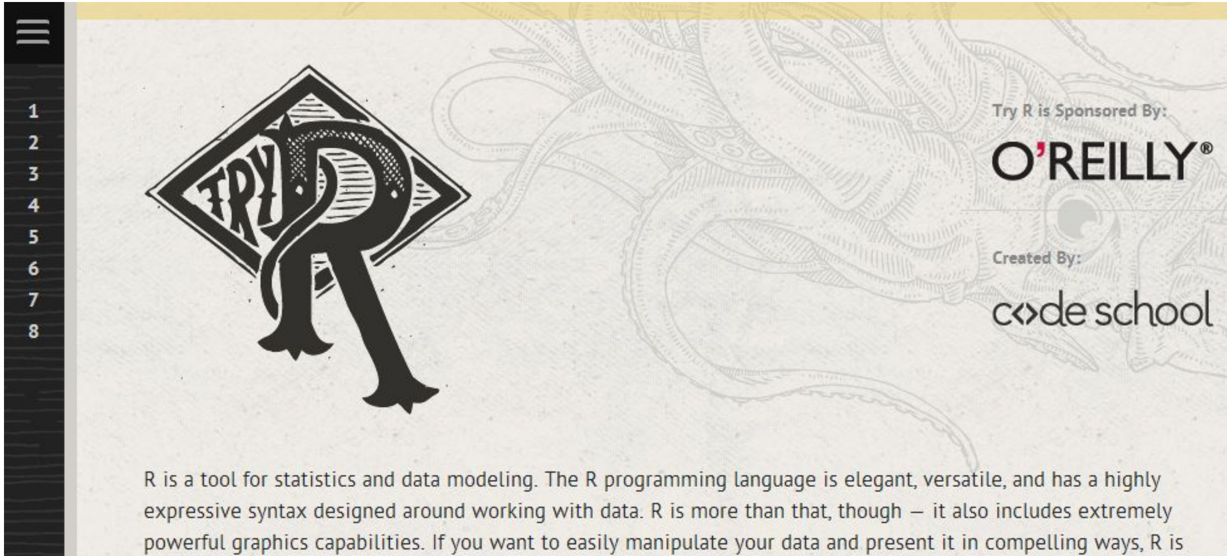
Plot spatial area defined by multiple polygons

Resources

try R

http://tryr.codeschool.com/

A game-ified R lesson that provides an embedded console and quick feedback, a useful way to practice the basics of R.

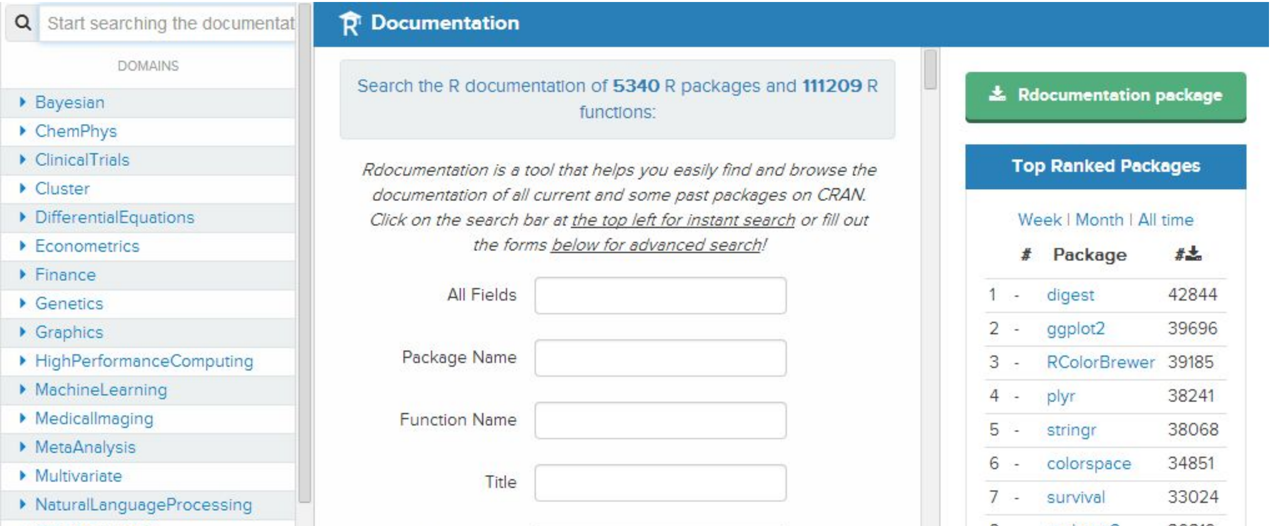


R is a tool for statistics and data modeling. The R programming language is elegant, versatile, and has a highly expressive syntax designed around working with data. R is more than that, though — it also includes extremely powerful graphics capabilities. If you want to easily manipulate your data and present it in compelling ways, R is

R Documentation

http://www.rdocumentation.org/

A searchable, enhanced version of R's documentation pages. Includes package download stats and user comments. You can run and manipulate example code right in the webpage.



Start searching the documentat

DOMAINS

- Bayesian
- ChemPhys
- ClinicalTrials
- Cluster
- DifferentialEquations
- Econometrics
- Finance
- Genetics
- Graphics
- HighPerformanceComputing
- MachineLearning
- MedicalImaging
- MetaAnalysis
- Multivariate
- NaturalLanguageProcessing

Documentation

Search the R documentation of **5340** R packages and **111209** R functions:

Rdocumentation is a tool that helps you easily find and browse the documentation of all current and some past packages on CRAN. Click on the search bar at the top left for instant search or fill out the forms below for advanced search!

All Fields

Package Name

Function Name

Title

Rdocumentation package

Top Ranked Packages

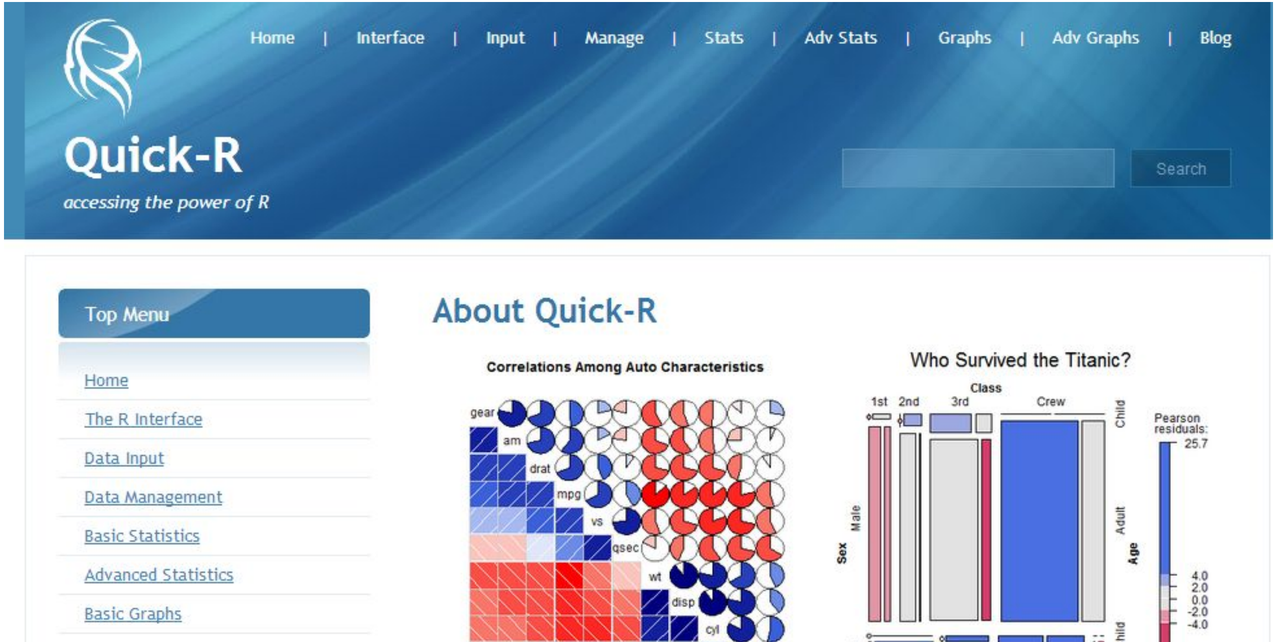
Week | Month | All time

#	Package	#
1 -	digest	42844
2 -	ggplot2	39696
3 -	RColorBrewer	39185
4 -	plyr	38241
5 -	stringr	38068
6 -	colorspace	34851
7 -	survival	33024

Quick-R

http://www.statmethods.net/

Excellent, “quick” reference to common R operations.



Quick-R

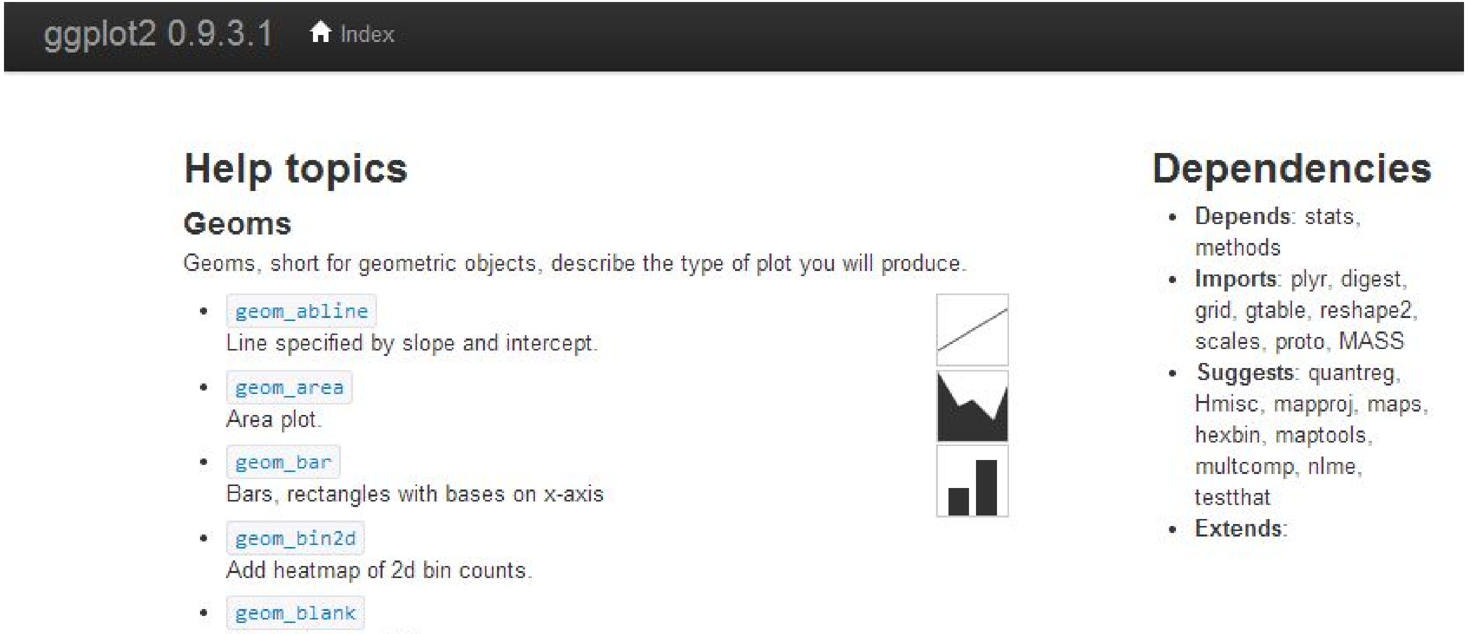
accessing the power of R

About Quick-R

ggplot2 doc

http://docs.ggplot2.org/current/

ggplot2 documentation organized by components of “grammar of graphics”



ggplot2 0.9.3.1 Index

Help topics

Geoms

Geoms, short for geometric objects, describe the type of plot you will produce.

- `geom_abline`
  Line specified by slope and intercept.
- `geom_area`
  Area plot.
- `geom_bar`
  Bars, rectangles with bases on x-axis
- `geom_bin2d`
  Add heatmap of 2d bin counts.
- `geom_blank`

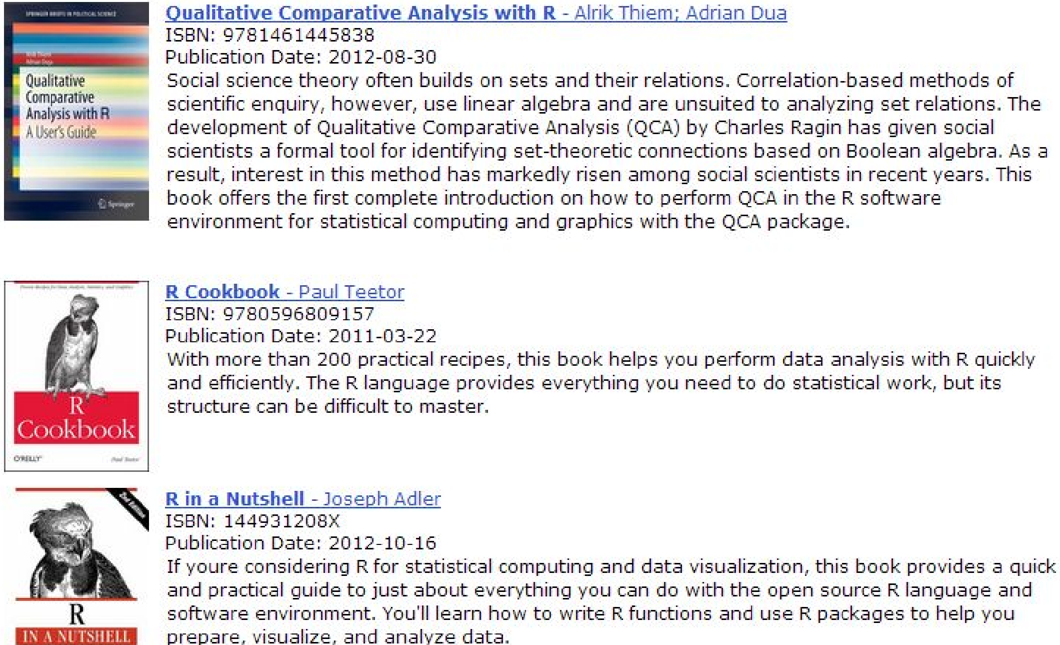
Dependencies

- **Depends**: stats, methods
- **Imports**: plyr, digest, grid, gtable, reshape2, scales, proto, MASS
- **Suggests**: quantreg, Hmisc, mapproj, maps, hexbin, maptools, multcomp, nlme, testthat
- **Extends**:

Library Research Guide

http://researchguides.library.tufts.edu/data/r

Library links to O'Reilly ebooks. Free to the Tufts community!



Qualitative Comparative Analysis with R - Alrik Thiem; Adrian Dua
ISBN: 9781461445838
Publication Date: 2012-08-30

Social science theory often builds on sets and their relations. Correlation-based methods of scientific enquiry, however, use linear algebra and are unsuited to analyzing set relations. The development of Qualitative Comparative Analysis (QCA) by Charles Ragin has given social scientists a formal tool for identifying set-theoretic connections based on Boolean algebra. As a result, interest in this method has markedly risen among social scientists in recent years. This book offers the first complete introduction on how to perform QCA in the R software environment for statistical computing and graphics with the QCA package.

R Cookbook - Paul Teetor
ISBN: 9780596809157
Publication Date: 2011-03-22

With more than 200 practical recipes, this book helps you perform data analysis with R quickly and efficiently. The R language provides everything you need to do statistical work, but its structure can be difficult to master.

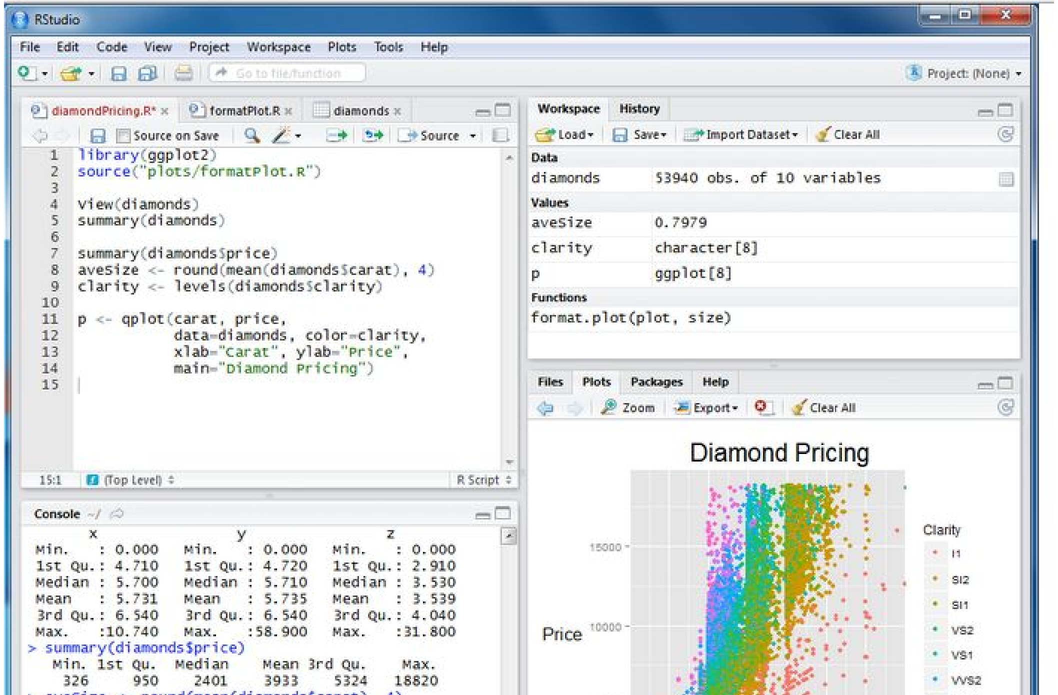
R in a Nutshell - Joseph Adler
ISBN: 144931208X
Publication Date: 2012-10-16

If youre considering R for statistical computing and data visualization, this book provides a quick and practical guide to just about everything you can do with the open source R language and software environment. You'll learn how to write R functions and use R packages to help you prepare, visualize, and analyze data.

RStudio

https://www.rstudio.com/

A powerful and convenient user environment for R. Free and open source. Highly, highly recommended.



```r
library(ggplot2)
source("plots/formatPlot.R")

view(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")

```

Data

diamonds   53940 obs. of 10 variables

Values

aveSize   0.7979

clarity   character[8]

p   ggplot[8]

Functions

format.plot(plot, size)

```
       x               y                z         
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910  
 Median : 5.700   Median : 5.710   Median : 3.530  
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539  
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040  
 Max.   :10.740   Max.   :58.900   Max.   :31.800  
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    326     950    2401    3933    5324   18820 
```

To be successful with R

- Like learning any programming language, take your time and try to run the code in your head before you run it on your machine. Try to predict what will happen.

- Be patient.

- Think of a fun project that you actually would like to do and do it in R.

- Ask your friendly librarian about useful resources.

- Understand that unlike excel, there are many paths to the same solution in R. You need not learn them all but to troubleshoot effectively and ask for help it is worthwhile to understand how others might work with R (ie., subsetting)

- If you master the techniques and concepts in this workshop you've mastered 80% of R. The rest is identifying specific packages/methodologies that are relevant to your domain.

thanks!

contact: datahelp@tufts.edu