These materials adapted by Amelia McNamara from the RStudio CC BY-SA materials Introduction to R (2014) and Master the Tidyverse (2017).

Introduction to R & RStudio: deck 04: Syntax

Amelia McNamara
Visiting Assistant Professor of Statistical and Data Sciences
Smith College

January 2018

Syntax is the set of rules that govern what code works and doesn't work in a programming language. Most programming languages offer one standardized syntax, but R allows package developers to specify their own syntax. As a result, there is a large variety of (equally valid) R syntaxes.

R Syntax Comparison : : CHEAT SHEET

I've given you a copy of this cheatsheet in the "cheatsheets" folder of your workspace, and I have paper copies for anyone who wants one

Dollar sign syntax

```
goal(data$x, data$y)
```

SUMMARY STATISTICS:

one continuous variable:
```
mean(mtcars$mpg)
```

one categorical variable:
```
table(mtcars$cyl)
```

two categorical variables:
```
table(mtcars$cyl, mtcars$am)
```

one continuous, one categorical:
```
mean(mtcars$mpg[mtcars$cyl==4])
mean(mtcars$mpg[mtcars$cyl==6])
mean(mtcars$mpg[mtcars$cyl==8])
```

PLOTTING:

one continuous variable:
```
hist(mtcars$disp)

boxplot(mtcars$disp)
```

one categorical variable:
```
barplot(table(mtcars$cyl))
```

two continuous variables:
```
plot(mtcars$disp, mtcars$mpg)
```

two categorical variables:
```
mosaicplot(table(mtcars$am, mtcars$cyl))
```

one continuous, one categorical:
```
histogram(mtcars$disp[mtcars$cyl==4])
histogram(mtcars$disp[mtcars$cyl==6])
histogram(mtcars$disp[mtcars$cyl==8])

boxplot(mtcars$disp[mtcars$cyl==4
boxplot(mtcars$disp[mtcars$cyl=
boxplot(mtcars$disp[mtcars$cy
```

WRANGLING:

subsetting:
```
mtcars[mtcars$mpg>30
```

making a new variable:
```
mtcars$efficient
mtcars$effici
```

Formula syntax

```
goal(y~x|z, data=data, group=w)
```

SUMMARY STATISTICS:

one continuous variable:
```
mosaic::mean(~mpg, data=mtcars)
```

one categorical variable:
```
mosaic::tally(~cyl, data=mtcars)
```

two categorical variables:
```
mosaic::tally(cyl~am, data=mtcars)
```

one continuous, one categorical:
```
mosaic::mean(mpg~cyl, data=mtcars)
```

tilde

PLOTTING:

one continuous variable:
```
lattice::histogram(~disp,

lattice::bwplot(~disp,
```

one categorical variable:
```
mosaic::bargraph(
```

two continuous v
```
lattice::xy
```

two cate
```
mosa
```

group=cyl)

data=mtcars)

data=mtcars)

R syntaxes give
ways to "say" the
ing

ead **across** the cheatsheet to see how different syntaxes approach the same problem

Tidyverse syntax

SUMMARY STATISTIC

one continuous v
```
mtcars %>
```

one cate
```
mtc
```

the pipe

```
y(cyl, am) %>%
```

cal:
```
:group_by(cyl) %>%
ize(mean(mpg))
```

ariable:
```
plot(x=mpg, data=mtcars, geom = "histogram")

::qplot(y=disp, x=1, data=mtcars, geom="boxplot")
```

categorical variable:
```
ggplot2::qplot(x=cyl, data=mtcars, geom="bar")
```

two continuous variables:
```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

two categorical variables:
```
ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") +
        facet_grid(.~am)
```

one continuous, one categorical:
```
ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") +
        facet_grid(.~cyl)

ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars,
               geom="boxplot")
```

WRANGLING:

subsetting:
```
mtcars %>% dplyr::filter(mpg>30)
```

making a new variable:
```
mtcars <- mtcars %>%
dplyr::mutate(efficient = if_else(mpg>30,  TRUE, FALSE))
```


RStudio® is a trademark of RStudio, Inc. CC BY Amelia McNamara • amcnamara@smith.edu • @AmeliaMN • science.smith.edu/~amcnamara/ • Updated: 2018-01

within ggplot2

We want to avoid this kind of repetition in programming

```
ggplot(data = bechdel) +
  geom_point(mapping = aes(x = budget, y = domgross)) +
  geom_smooth(mapping = aes(x= budget, y = domgross))
```

Mappings (and data) that appear in ggplot() will apply globally to every layer

```
ggplot(data = bechdel, aes(x= budget, y = domgross)) +
  geom_point() +
  geom_smooth()
```

MANY ways to say the same thing

```
ggplot(bechdel) +
    geom_point(aes(x = budget, y = domgross))
ggplot(bechdel, aes(x = budget, y = domgross)) +
    geom_point()
ggplot(bechdel, aes(x= budget)) +
    geom_point(aes(y = domgross))
ggplot() +
    geom_point(bechdel, aes(x = budget, y = domgross))
```

Mappings (and data) that appear in a geom_ function will add to or override the global mappings for that layer only

```
ggplot(data = bechdel, aes(x= budget, y = domgross)) +
  geom_point(aes(color=clean_test)) +
  geom_smooth()
```

Subsetting

From *R for Data Science* by Hadley Wickham and Garrett Grolemund.

Toy data

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

First— the tidyverse way: dplyr

dplyr methods for isolating data

select() - extract **variables**
filter() - extract **cases**
arrange() - reorder **cases**

select()

Extract columns by name.

```
select(.data, …)
```

data frame to transform

name(s) of columns to extract
(or a select helper function)

select()

Extract columns by name.

```
select(beatles, name, birth)
```

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

→

name	birth
John	1940
Paul	1942
George	1943
Ringo	1940

Your Turn

Alter the code to select just the **instrument** column:

```
select(beatles, name, birth)
```

select() helpers

: - Select range of columns

```
select(storms, storm:pressure)
```

- - Select every column but

```
select(storms, -c(storm, pressure))
```

starts_with() - Select columns that start with…

```
select(storms, starts_with("w"))
```

ends_with() - Select columns that end with…

```
select(storms, ends_with("e"))
```

select() helpers

contains() - Select columns whose names contain…

```
select(storms, contains("d"))
```

matches() - Select columns whose names match regular expression

```
select(storms, matches("^.{4}$"))
```

one_of() - Select columns whose names are one of a set

```
select(storms, one_of(c("storm", "storms", "Storm")))
```

num_range() - Select columns named in prefix, number style

```
select(storms, num_range("x", 1:5))
```

select() helpers

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, ...**)**
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**one_of(**...**)**	**-**, e.g, -Species
matches(match**)**	**starts_with(**match**)**	

Now, the base R way: brackets and dollar signs

Base R bracket subset notation

in base R, you use the same syntax to

extract **variables**

extract **cases**

name of object to subset

brackets
(brackets always mean subset)

vec[ ]

base R

Subset notation

name of object
to subset

vec

base
R

Subset notation

name of object to subset

brackets
(brackets always mean subset)

vec[?]

an index
(that tells R which elements to include)

base R

Each dimension needs its own index!

vec[?]

6	1	3	6	10	5

base R

Each dimension needs its own index!

vec[?]

6	1	3	6	10	5

base R

Each dimension needs its own index!

vec[?]
beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Each dimension needs its own index!

vec[?]

beatles[?,?]

which **rows** to include

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Each dimension needs its own index!

vec[?]

beatles[?, ?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

which **rows** to include

which **columns** to include

base R

Each dimension needs its own index!

vec[?]

beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

which **rows** to include

separate dimensions with a **comma**

which **columns** to include

base R

Each dimension needs its own index!

vec[?]
beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

What goes in the indexes?

Four ways to subset

1. Integers
2. Blank spaces
3. Names
4. Logical vectors (TRUE and FALSE)

Integers (positive)

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

beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

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

beatles[2, ?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Integers (positive)

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

beatles[2,3]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Integers (positive)

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

beatles[2,3]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

c("John","Paul", "George","Ringo")

c(1940, 1942, 1943, 1940)

c("gu
"c

John

Paul

Georg
e

Ri

1940

19

ar

drums

eatles[2]

Numeric indexing is a sign of old R code, and there are few good use cases
We won't focus on it

base
R

Names

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

beatles[,"birth"]

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Names

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

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

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Your Turn

Modify the code below to select just the `instrument` column

```
beatles[ ,"birth"]
```

$

The most common syntax for subsetting lists and data frames

base
R

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

```
beatles$birth
```

base R

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

beatles$birth

name of
data frame

base
R

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

beatles$birth

name of data frame

$

base R

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

beatles$birth

name of data frame

$

name of column
(no quotes)

base R

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

c(1940, 1941, 1943, 1940)

beatles$birth

name of data frame

$

name of column
(no quotes)

base R

Your Turn

Modify the code below to select just the `instrument` column

```
beatles$birth
```

Back to the tidyverse

dplyr methods for isolating data

select() - extract **variables**
filter() - extract **cases**
arrange() - reorder **cases**

filter()

Extract rows that meet logical criteria.

```
filter(.data, … )
```

data frame to transform

one or more logical tests (filter returns each row for which the test is TRUE)

common syntax

Each function takes a data frame / tibble as its first argument and returns a data frame / tibble.

```
filter(.data, … )
```

dplyr function

data frame to transform

function specific arguments

filter()

Extract rows that meet logical criteria.

```
filter(beatles, name == "George")
```

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

→

George	1943	guitar

filter()

Extract rows that meet logical criteria.

```
filter(beatles, name == "George")
```

= sets
(returns nothing)

== tests if equal
(returns TRUE or FALSE)

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

Logical comparisons

Logical comparisons

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Logical comparisons

What will these return?

```
1 < 3
1 > 3
c(1, 2, 3, 4, 5) > 3
```

%in%

```
# What does this do?
1 %in% c(1, 2, 3, 4)
1 %in% c(2, 3, 4)
c(3,4,5,6) %in% c(2, 3, 4)
```

%in%

%in% tests whether the object on the left is a member of the group on the right.

```
1 %in% c(1, 2, 3, 4)
# TRUE
1 %in% c(2, 3, 4)
# FALSE
c(3,4,5,6) %in% c(2, 3, 4)
# TRUE TRUE FALSE FALSE
```

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3		
x < 3		
x <= 3		
x == 3		
x != 3		
x = 3		

01:00

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3		
x <= 3		
x == 3		
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3		
x == 3		
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3		
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3	`c(F, F, T, F, F)`	equal to
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3	`c(F, F, T, F, F)`	equal to
x != 3	`c(T, T, F, T, T)`	not equal to
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3	`c(F, F, T, F, F)`	equal to
x != 3	`c(T, T, F, T, T)`	not equal to
x = 3		same as <-

Your Turn

Alter the code to filter out just rows where birth is 1940:

```
filter(beatles, name == "George")
```

filter()

Extract rows that meet *every* logical criteria.

```
filter(beatles, birth==1940, instrument == "guitar")
```

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

→

John	1940	guitar

Boolean operators

Boolean operators

?base::Logic

`a & b`	and
`a | b`	or
`xor(a,b)`	exactly or
`!a`	not

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$$x > 2 \ \& \ x < 9$$

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

x > 2 & x < 9

TRUE &

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

x > 2 & x < 9

TRUE & TRUE

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

x > 2 & x < 9

TRUE & TRUE

TRUE

&

Are both condition 1 **and** condition 2 true?

expression	outcome
TRUE & TRUE	TRUE
TRUE & FALSE	FALSE
FALSE & TRUE	FALSE
FALSE & FALSE	FALSE

|

Is either condition 1 **or** condition 2 true?

expression	outcome
TRUE | TRUE	TRUE
TRUE | FALSE	TRUE
FALSE | TRUE	TRUE
FALSE | FALSE	FALSE

xor

Is either condition 1 **or** condition 2 true, **but not both**?

expression	outcome
xor(TRUE, TRUE)	FALSE
xor(TRUE, FALSE)	TRUE
xor(FALSE, TRUE)	TRUE
xor(FALSE, FALSE)	FALSE

base R

dplyr

!

Negation

expression	outcome
!(TRUE)	FALSE
!(FALSE)	TRUE

filter()

Extract rows that meet *every* logical criteria.

```
filter(beatles, birth==1940 & instrument == "guitar")
```

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

→

John	1940	guitar

Your Turn

Modify the code below to filter out the rows for which birth is 1943 or instrument is drums

```
filter(beatles, birth==1940 & instrument == "guitar")
```

Base R

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.

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

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

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.

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

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Logical

You can provide a statement that **evaluates** to a logical to get something similar to a `dplyr` `filter()` statement.

```
beatles[beatles$birth == 1940, ]
beatles[c(TRUE, FALSE, FALSE, TRUE), ]
```

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

base R

Logical tests

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Boolean operators

?base::Logic

`a & b`	and
`a | b`	or
`xor(a,b)`	exactly or
`!a`	not

Your Turn

Modify the code below to filter out rows where birth is 1943 or instrument is drums

```
beatles[beatles$birth == 1940, ]
```

Bigger example

Your Turn

We'll play with the `babynames` dataset in R. It comes in its own package, so you will need to

- load the package (what command does that?) and then
- View the data

Filter

	year	sex	name	n	prop
1	1880	F	Mary	7065	0.072384329
2	1880	F	Anna	2604	0.026679234
3	1880	F	Emma	2003	0.020521700
4	1880	F	Elizabeth	1939	0.019865989
5	1880	F	Minnie	1746	0.017888611
6	1880	F	Margaret	1578	0.016167370
7	1880	F	Ida	1472	0.015081349
8	1880	F	Alice	1414	0.014487111
9	1880	F	Bertha	1320	0.013524036
10	1880	F	Sarah	1288	0.013196180
11	1880	F	Annie	1258	0.012888816
12	1880	F	Clara	1226	0.012560961
13	1880	F	Ella	1156	0.011843777

Your Turn

See if you can use filter and/or select to get just the babies with your first name. (If you have a name with less than 5 occurrences in the US in any year, pick a neighbor's name.)

Your Turn

See if you can use the logical operators to show:

- All of the names where **prop** is greater than or equal to 0.08
- All of the children named “Sea”
- All of the names that have a missing value for **n**

(Hint: this should return an empty data set).

04:00

```r
filter(babynames, prop >= 0.08)
#    year   sex    name     n       prop
# 1  1880     M    John  9655 0.08154630
# 2  1880     M William  9531 0.08049899
# 3  1881     M    John  8769 0.08098299
```

```r
filter(babynames, name == "Sea")
#    year   sex  name     n         prop
# 1  1982     F   Sea     5 2.756771e-06
# 2  1985     M   Sea     6 3.119547e-06
# 3  1986     M   Sea     5 2.603512e-06
# 4  1998     F   Sea     5 2.580377e-06
```

```r
filter(babynames, is.na(n))
#  0 rows
```

Two common mistakes

1. Using `=` instead of `==`

```
filter(babynames, name = "Sea")
filter(babynames, name == "Sea")
```

2. Forgetting quotes

```
filter(babynames, name == Sea)
filter(babynames, name == "Sea")
```

Your Turn

Use Boolean operators to alter the code below to return only the rows that contain:

- Girls named Sea
- Names that were used by exactly 5 or 6 children in 1880
- Names that are one of Acura, Lexus, or Yugo

```
filter(babynames, name == "Sea" | name == "Anemone")
```

```
filter(babynames, name == "Sea", sex == "F")
#    year   sex  name     n         prop
# 1  1982     F   Sea     5 2.756771e-06
# 2  1998     F   Sea     5 2.580377e-06
```

```
filter(babynames, n == 5 | n == 6, year == 1880)
#    year   sex    name     n         prop
# 1  1880     F    Abby     6 6.147289e-05
# 2  1880     F  Aileen     6 6.147289e-05
# …     …     …       …     …            …
```

```
filter(babynames, name %in% c("Acura", "Lexus", "Yugo"))
#    year   sex  name     n         prop
# 1  1990     F Lexus    36 1.752932e-05
# 2  1990     M Lexus    12 5.579156e-06
# …     …     …     …     …            …
```

Two more common mistakes

3. Collapsing multiple tests into one

```
filter(babynames, 10 < n < 20)
filter(babynames, 10 < n, n < 20)
```

4. Stringing together many tests (when you could use %in%)

```
filter(babynames, n == 5 | n == 6 | n == 7 | n == 8)
filter(babynames, n %in% c(5, 6, 7, 8))
```

Saving results

Saving results

Print to screen

```
filter(babynames, name %in% c("Acura", "Lexus", "Yugo"))
#    year   sex  name     n         prop
# 1  1990     F Lexus    36 1.752932e-05
# 2  1990     M Lexus    12 5.579156e-06
# …     …     …     …     …            …
```

Save to new data frame (where does this appear?)

```
carnames <- filter(babynames, name %in% c("Acura", "Lexus", "Yugo"))
```

Save over existing data frame (dangerous!)

```
babynames <- filter(babynames, name %in% c("Acura", "Lexus", "Yugo"))
```

```
babynames <- filter(babynames, name %in% c("Acura", "Lexus", "Yugo"))
```

Uh oh...

```
rm(babynames)
str(babynames)
```

Phew!

Your Turn

Try to:

- Filter out the `babynames` data for your name. (You may want to **also** filter for your gender)
- Assign that data its own name
- Use your subsetted data to create a `ggplot` graphic of the popularity of your name over time

```
amelia <- filter(babynames, name=="Amelia" & sex == "F")

ggplot(amelia) + geom_point(aes(x=year, y=n))

ggplot(amelia) + geom_line(aes(x=year, y=n))
```

%>%

Steps

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

1. Filter babynames to just boys born in 2015
2. Select the name and n columns from the result
3. Arrange those columns so that the most popular names appear near the top.

Steps

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

Steps

```
arrange(select(filter(babynames, year == 2015,
  sex == "M"), name, n), desc(n))
```

The pipe operator %>%

```
babynames                filter(_______, n == 99680)
```

Passes result on left into first argument of function on right.
So, for example, these do the same thing. Try it.

```
filter(babynames, n == 99680)
babynames %>% filter(n == 99680)
```

Pipes

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

```
babynames %>%
  filter(year == 2015, sex == "M") %>%
  select(name, n) %>%
  arrange(desc(n))
```

```
foo_foo <- little_bunny()
```

```
foo_foo %>%
  hop_through(forest) %>%
  scoop_up(field_mouse) %>%
  bop_on(head)
```

vs.

```
foo_foo2 <- hop_through(foo_foo, forest)
foo_foo3 <- scoop_up(foo_foo2, field_mouse)
bop_on(foo_foo3,  head)
```

Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)

Your Turn 6

Use %>% to write a sequence of functions that:

1. Filter babynames to just the girls that were born in 2015
2. Select the **name** and **n** columns
3. Arrange the results so that the most popular names are near the top.

```r
babynames %>%
  filter(year == 2015, sex == "F") %>%
  select(name, n) %>%
  arrange(desc(n))
#         name     n
#  1      Emma 20355
#  2    Olivia 19553
#  3    Sophia 17327
#  4       Ava 16286
#  5  Isabella 15504
#  6       Mia 14820
#  7   Abigail 12311
#  8     Emily 11727
#  9 Charlotte 11332
# 10    Harper 10241
# ... with 18,983 more rows
```

Tidy data

Tidy data

Tidy functions all expect and return the same data structure, known as **tidy data**:

1. A **data frame** that contains
2. **variables** in the **columns** and
3. **cases** in the **rows**.

Tidy data

A data set is **tidy** iff:

1. Each **variable** is in its own **column**
2. Each **case** is in its own **row**
3. Each **value** is in its own **cell**

country <chr>	year <int>	cases <int>	population <int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

```
table1$country
table1$year
table1$cases
table1$population
```

country <chr>	year <int>	type <chr>	count <int>
Afghanistan	1999	cases	745
Afghanistan	1999	[illegible]	19987071
Afghanistan	200[illegible]	[illegible]	2666
Afghanistan	[illegible]	[illegible]	[illegible]95360
Brazil	[illegible]	[illegible]	[illegible]737
Brazil	[illegible]	[illegible]	[illegible]62
Brazil	[illegible]	[illegible]	[illegible]8
Brazil	[illegible]	[illegible]	[illegible]8
China	[illegible]	[illegible]	[illegible]8
China	[illegible]	[illegible]	[illegible]72

1–10 of 12 rows [illegible]vious 1 2 Next

```
table2$[illegible]
table2$year
table2$count[c(1,3,5,7,9,11)]
table2$count[c(2,4,6,8,10,12)]
```

country <chr>	year <int>	cases <int>	population <int>	rate <dbl>
Afghanistan	1999	745	19987071	0.0000372741
Afghanistan	2000	2666	20595360	0.0001294466
Brazil	1999	37737	172006362	0.0002193930
Brazil	2000	80488	174504898	0.0004612363
China	1999	212258	1272915272	0.0001667495
China	2000	213766	1280428583	0.0001669488

6 rows

```
table1$cases / table1$population -> table1$rate
```

Tidy tools

Tidy tools

Functions are easiest to use when they are:

1. **Simple** - They do one thing, and they do it well
2. **Composable** - They can be combined with other functions for multi-step operations
3. **Smart** - They can use R objects as input.

Tidy functions do these things in a specific way.

1. Simple - They do one thing, and they do it well

filter() - extract **cases**
arrange() - reorder **cases**
group_by() - group **cases**
select() - extract **variables**
mutate() - create new **variables**
summarise() - summarise **variables** / create **cases**

2. **Composable** - They can be combined with other functions for multi-step operations

```
babynames          mutate(______, percent = prop * 100)
```

Each dplyr function takes a data frame as its first argument and returns a data frame. As a result, you can directly pipe the output of one function into the next.

It's not either/or

Tidyverse

v.

?

No! It's all R, it all works together. More like different philosophy

Diamonds

Diamonds data

- ~**54,000** round diamonds from http://www.diamondse.info/
- comes in the ggplot2 package
- Carat, colour, clarity, cut
- Total depth, table, depth, width, height
- Price

depth = z / diameter
table = table width / x * 100
x, y, z in mm

```
ggplot(diamonds) + geom_point(aes(x = x, y= y))
```

What is weird about these values?

ggplot(diamonds) + geom_point(aes(x = x, y= y))

Your Turn

Can you remove them with subsetting and make the plot of the non-outliers?

05:00

```
ggplot(diamonds) + geom_point(aes(x = x, y= y))
```

```
diamonds %>%
     filter(y > 30 | x < 2)
# A tibble: 10 x 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 1.07  Ideal     F     SI2      61.6  56.0  4954  0     6.62  0
 2 1.00  Very Good H     VS2      63.3  53.0  5139  0     0     0
 3 1.14  Fair      G     VS1      57.5  67.0  6381  0     0     0
 4 2.00  Premium   H     SI2      58.9  57.0 12210  8.09 58.9   8.06
 5 1.56  Ideal     G     VS2      62.2  54.0 12800  0     0     0
 6 1.20  Premium   D     VVS1     62.1  59.0 15686  0     0     0
 7 2.25  Premium   H     SI2      62.8  59.0 18034  0     0     0
 8 0.510 Ideal     E     VS1      61.8  55.0  2075  5.15 31.8   5.12
 9 0.710 Good      F     SI2      64.1  60.0  2130  0     0     0
10 0.710 Good      F     SI2      64.1  60.0  2130  0     0     0
```

Lots of zeroes. Must be missing data (can't have a 0-mm length)

```
diamonds %>%
  filter(y < 30 | x > 2) %>%
  ggplot() +
  geom_point(aes(x = x, y = y))
```

```
diamonds %>%
    filter(y < 30 & x > 2) %>%
    ggplot() +
    geom_point(aes(x = x, y = y))
```

http://bit.ly/day1Feedback