UNIVERSITY OF LINCOLN

Structuring your data

PSY9219M - Research Methods and Skills

Dr Matt Craddock

9/10/2018

Writing R Scripts

Scripts are text documents that contain a sequence of commands to be executed sequentially.

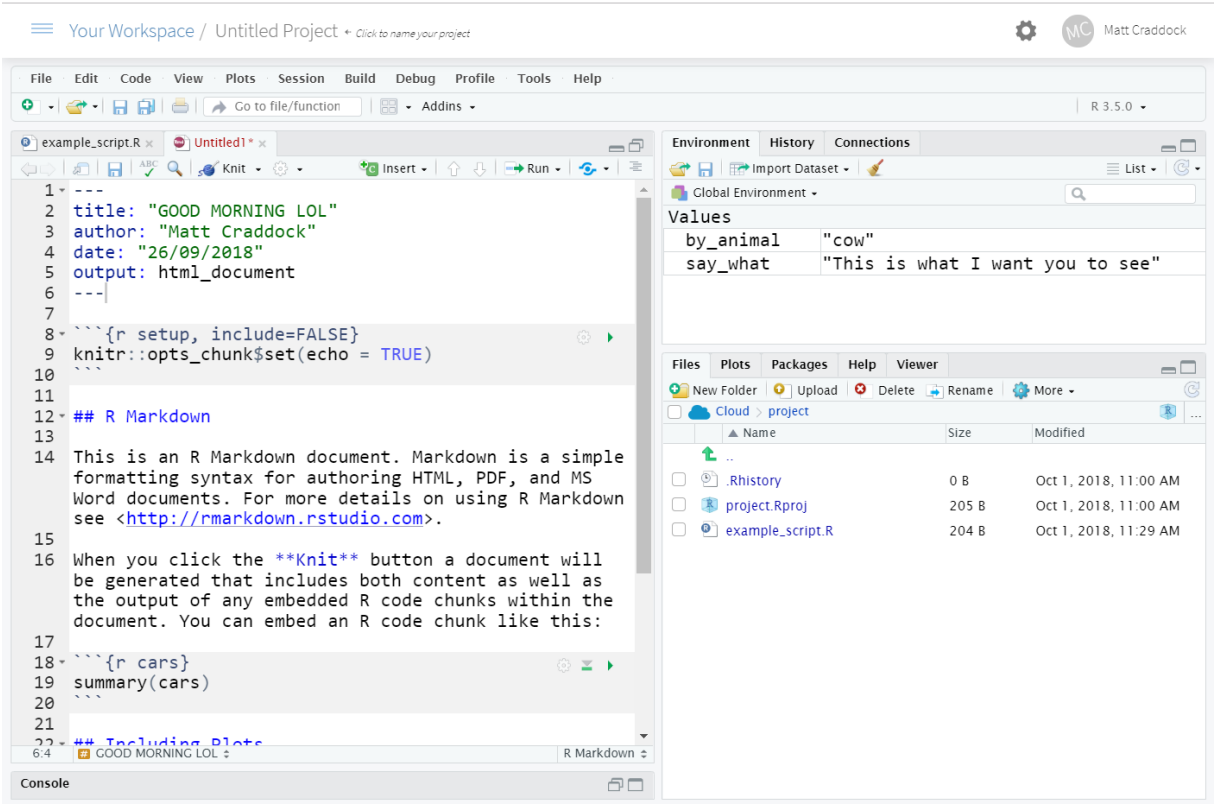
A typical script looks something like this:

```
# Load in required packages using library()
library(tidyverse)

# Define any custom functions here (we haven't covered this!)

# Now load any data you want to work on. (again, we'll cover this later!)
test_data <-
  read_csv("data/a-random-RT-file.csv") %>% # I'll explain what %>% means later
  rename(RT = `reaction times`)

# The rest of the script then runs whatever analyses or plotting you want to do
ggplot(test_data,
       aes(x = RT,
           fill = viewpoint)) +
  geom_density()
```

RMarkdown

RMarkdown documents contain a mixture of code and plain text.

They can be used to produce *reports* and fully formatted documents with whatever structure you choose.



Basic data types

There are five basic data types in R:

Type	Description	Examples
double	Floating point value	3.12
integer	Integer	1, 2, 3
numeric	Any real number	3.4, 2, -2.3
character	Text	"Hi there", "8.5", "ABC123"
logical	Assertion of truth/falsity	TRUE, FALSE

Containers

Vectors are one-dimensional collections of values of the same basic data type.

Matrices are two-dimensional collections of values of the same basic data type.

Lists are collections of objects of varying length and type.

Data frames are tables of data.

Accessing elements from containers

You can use the **[]** operator after the name of an object to extract indvidual elements from that object.

```
one_to_four
```

```
##    Monday   Tuesday Wednesday  Thursday
##         1         2         3         4
```

```
test_matrix
```

```
##              [,1]       [,2]      [,3]
## [1,]  0.45650588  1.0698847 0.8399025
## [2,] -0.47721600 -0.9100745 1.3884486
## [3,] -0.09986935  0.9590143 0.7455983
```

```
one_to_four["Wednesday"]
```

```
## Wednesday
##         3
```

```
test_matrix[2:3, 1:2]
```

```
##              [,1]       [,2]
## [1,] -0.47721600 -0.9100745
## [2,] -0.09986935  0.9590143
```

Relating data to structure

Example data frames

R has a number of example data frames built-in.

```
data()
help("ChickWeight")
head(ChickWeight)
```

You can these to get an idea of the kind of ways data is structured by R.

(Note here I use the **head()** function to see the first 6 rows of the data frame "ChickWeight")

Example data frames

```
head(ChickWeight, n = 10)
```

```
## Grouped Data: weight ~ Time | Chick
##    weight Time Chick Diet
## 1      42    0     1    1
## 2      51    2     1    1
## 3      59    4     1    1
## 4      64    6     1    1
## 5      76    8     1    1
## 6      93   10     1    1
## 7     106   12     1    1
## 8     125   14     1    1
## 9     149   16     1    1
## 10    171   18     1    1
```

ChickWeight has 578 datapoints from an experiment on effects of diet on early growth of chicks.

Each column is a separate *variable*, with different information.

Scenario A

You've just started work in a psychology lab. You're asked to help analyse some old data. There is reaction time data from 50 participants. Each participant's data is stored in a separate text file.

- How do you combine the data from each participant together to be able to analyse the data?
- It turns out some of the participants only completed part of the experiment - which ones, and what should you do with their data?
- What steps should you take to select and perform appropriate statistical analysis?

Let's think about the *experiment*

The experiment is a reaction time experiment with a two-by-two repeated measures design.

Participants see pictures of objects twice. Sometimes they are seen from the *same* viewpoint twice, sometimes from *different* viewpoints each time.

There are two separate blocks of trials. The dependent variable is how long it takes them to name the objects, or *reaction time*.

You're interested in 1) whether they get faster the second time, and 2) whether they faster when the same view is presented both times.

How the design informs the structure

Variables	R Data Type
Participant ID	Numeric or character
Reaction times	Numeric
Block first/second	Character/factor
Viewpoint same/different	Character/factor

The final dataset needs to be able to do several things.

1. It needs to uniquely identify each participant.
2. It needs to tie each value to the right participant.
3. It needs to identify what each value represents in terms of the design.

Some possible data frames

Dependent variable split across columns

```
## # A tibble: 16 x 4
## # Groups:   Participant [8]
##    Participant Viewpoint  B1RT  B2RT
##          <int> <fct>     <dbl> <dbl>
##  1           1 Different  448.  455.
##  2           1 Same       509.  384.
##  3           2 Different  445.  386.
##  4           2 Same       438.  370.
##  5           3 Different  448.  455.
##  6           3 Same       509.  384.
##  7           4 Different  445.  386.
##  8           4 Same       438.  370.
##  9           5 Different  448.  455.
## 10           5 Same       509.  384.
## 11           6 Different  445.  386.
## 12           6 Same       438.  370.
## 13           7 Different  448.  455.
## 14           7 Same       509.  384.
## 15           8 Different  445.  386.
## 16           8 Same       438.  370.
```

One column for condition, one column for RT

```
## # A tibble: 40 x 3
## # Groups:   Participant [10]
##    Participant exp_condition       RT
##          <int> <fct>            <dbl>
##  1           1 Block1_different  395.
##  2           1 Block1_same       435.
##  3           1 Block2_different  417.
##  4           1 Block2_same       431.
##  5           2 Block1_different  368.
##  6           2 Block1_same       416.
##  7           2 Block2_different  394.
##  8           2 Block2_same       433.
##  9           3 Block1_different  370.
## 10           3 Block1_same       437.
## # ... with 30 more rows
```

One column per condition

```
## # A tibble: 10 x 5
##    Participant Block1_same Block2_same Block1_different Block2_different
##          <int>       <dbl>       <dbl>             <dbl>             <dbl>
##  1           1        520.        288.              541.              420.
##  2           2        466.        350.              583.              436.
##  3           3        492.        279.              606.              382.
##  4           4        567.        338.              549.              379.
##  5           5        511.        322.              507.              429.
##  6           6        531.        329.              540.              373.
##  7           7        471.        317.              561.              402.
##  8           8        467.        267.              548.              356.
##  9           9        478.        273.              504.              496.
## 10          10        529.        347.              607.              355.
```

Create a data frame

Your task:

1. Create a data frame with one column per condition.
2. Each row should be for one participant.
3. Each row should have an identifier for the participant, with the identifiers in one column.

Create a data frame

Some helpful functions

- **rnorm()**
  - Use **rnorm()** to generate a normally distributed set of numbers.
  - Type ?rnorm in console for help
  - **rnorm(n, mean = 0, sd = 1)**

```
rnorm(100, 500, 100) # This creates a vector of 100 values
      #with a mean of 500 and a standard deviation of 100
```

- **data.frame()**
  - Create columns in a data frame
  - **data.frame(colname = values)**
  - Separate columns with a comma ","

```
data.frame(col_one = rnorm(...),
           col_two = rnorm(...))
```

Create a data frame

Tips

- Remember to *assign* output to an object using the <- operator.
- You can create a sequential vector using the colon (:) operator (e.g. 1:5) or the **seq()** function (type ?seq)
- Remember that objects can *stand in* for their values

```
some_rts <- rnorm(23, 34, 50)
test_df <-
  data.frame(col_one = some_rts,
             col_two = rnorm(...),
             ...)
```

- Once you've created a data frame, try calculating the **mean()** and **sd()** of the columns

Your target

```
## # A tibble: 10 x 5
##    Participant Block1_same Block2_same Block1_different Block2_different
##          <int>       <dbl>       <dbl>             <dbl>             <dbl>
##  1           1        497.        296.              561.              361.
##  2           2        499.        275.              566.              367.
##  3           3        558.        302.              573.              328.
##  4           4        544.        289.              580.              396.
##  5           5        481.        308.              599.              424.
##  6           6        488.        325.              509.              384.
##  7           7        474.        287.              531.              394.
##  8           8        463.        334.              551.              460.
##  9           9        482.        291.              557.              338.
## 10          10        481.        301.              579.              376.
```

A possible solution

```
example_rt_df <-
  data.frame(Participant = seq(1, 10),
             Block1_same = rnorm(10, 500, 100),
             Block2_same = rnorm(10, 350, 100),
             Block1_different = rnorm(10, 500, 100),
             Block2_different = rnorm(10, 400, 100))
```

```
##   Participant Block1_same Block2_same Block1_different Block2_different
## 1           1    516.5898    305.1622         498.2227         313.3878
## 2           2    382.0560    386.3059         679.8127         339.2907
## 3           3    514.7835    378.0082         355.0890         442.7510
## 4           4    604.1356    450.9009         616.3194         477.3192
## 5           5    468.6312    211.0952         434.6988         340.4552
```


tidyverse
www.rstudio.com

Tidyverse

The **tidyverse** is a collection of packages that expand R's functions in a structured, coherent way.

```
install.packages("tidyverse")
```

There are eight core **tidyverse** packages loaded using **library(tidyverse)**.

- ggplot2
- **tidyr**
- dplyr
- **tibble**
- purrr
- readr
- stringr
- forcats

Tidyverse

You can load all these packages at once.

```
library(tidyverse) # This loads all the tidyverse packages at once
```

You can also load each one individually. We'll be using the **tibble** package next.

```
library(tibble)
```

Many of the *tidyverse* packages convert or output *tibbles*, which are essentially a more user-friendly version of data frames.

Tibbles

Tibbles directly show what *data type* is stored in each column.

You can convert a data frame to a *tibble* using the **as_tibble()** function.

```
example_rt_tibble <- as_tibble(example_rt_df)
head(example_rt_tibble)
```

```
## # A tibble: 6 x 5
##   Participant Block1_same Block2_same Block1_different Block2_different
##         <int>       <dbl>       <dbl>             <dbl>             <dbl>
## 1           1        517.        305.              498.              313.
## 2           2        382.        386.              680.              339.
## 3           3        515.        378.              355.              443.
## 4           4        604.        451.              616.              477.
## 5           5        469.        211.              435.              340.
## 6           6        318.        310.              480.              467.
```

Tibbles

You can create a *tibble* similarly to how you create a data frame, using **tibble()**.

```
age_tibb <-
  tibble(Participant = 1:10,
       cond1 = rnorm(10),
       age_group = rep(c("Old", "Young"), each = 5))
head(age_tibb)
```

```
## # A tibble: 6 x 3
##   Participant   cond1 age_group
##         <int>   <dbl> <chr>
## 1           1 -2.39   Old
## 2           2 -0.709  Old
## 3           3  1.07   Old
## 4           4  0.438  Old
## 5           5  0.0492 Old
## 6           6 -0.963  Young
```

Tibbles

```
age_tibb <-
  tibble(Participant = 1:10,
       cond1 = rnorm(10),
        age_group = rep(c("Old", "Young"), each = 5))
```

Here I used the **rep()** function to generate a character vector with the values "Old" and "Young".

```
rep(c("Old", "Young"), each = 5)
```

```
##  [1] "Old"   "Old"   "Old"   "Old"   "Old"   "Young" "Young" "Young"
##  [9] "Young" "Young"
```

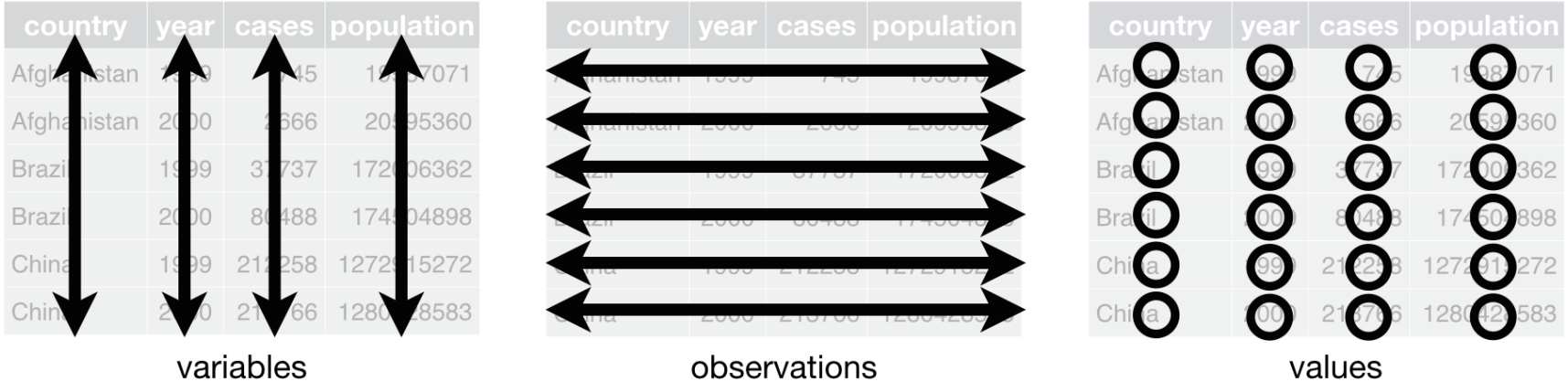
```
rep(c("Old", "Young"), 5)
```

```
##  [1] "Old"   "Young" "Old"   "Young" "Old"   "Young" "Old"   "Young"
##  [9] "Old"   "Young"
```

Tidy data

The three principles of tidy data

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.



How many *variables* are there?

```
##    Participant Block1_same Block2_same Block1_different Block2_different
## 1            1    516.5898    305.1622         498.2227          313.3878
## 2            2    382.0560    386.3059         679.8127          339.2907
## 3            3    514.7835    378.0082         355.0890          442.7510
## 4            4    604.1356    450.9009         616.3194          477.3192
## 5            5    468.6312    211.0952         434.6988          340.4552
## 6            6    317.7252    309.5307         479.6596          467.2746
## 7            7    451.3933    274.3238         546.2049          297.5012
## 8            8    704.9653    317.5622         594.0583          405.3054
## 9            9    383.3551    385.1461         489.2698          368.8661
## 10          10    668.2327    333.7526         448.7217          596.6026
```

How many *variables* are there?

```
##   Participant Block1_same Block2_same Block1_different Block2_different
## 1           1    516.5898    305.1622         498.2227         313.3878
## 2           2    382.0560    386.3059         679.8127         339.2907
## 3           3    514.7835    378.0082         355.0890         442.7510
## 4           4    604.1356    450.9009         616.3194         477.3192
## 5           5    468.6312    211.0952         434.6988         340.4552
## 6           6    317.7252    309.5307         479.6596         467.2746
## 7           7    451.3933    274.3238         546.2049         297.5012
## 8           8    704.9653    317.5622         594.0583         405.3054
```

- Participant
- Block (1 or 2)

- Viewpoint (same or different)
- Reaction time

This data is *untidy*

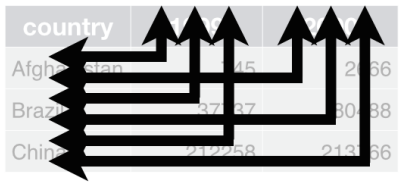
One variable - RT - is split across four columns.

Another variable - Block - is split across two columns.

A third variable - viewpoint - is also split across two columns.

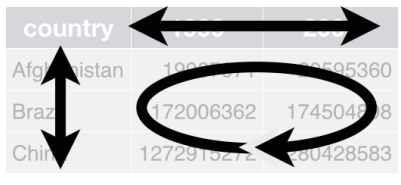
country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

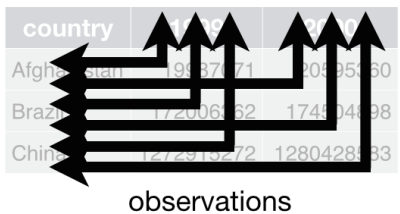


country	1999	2000
Afghanistan	19987071	20595360
Brazil	172006362	174504898
China	1272915272	1280428583

table5



variables



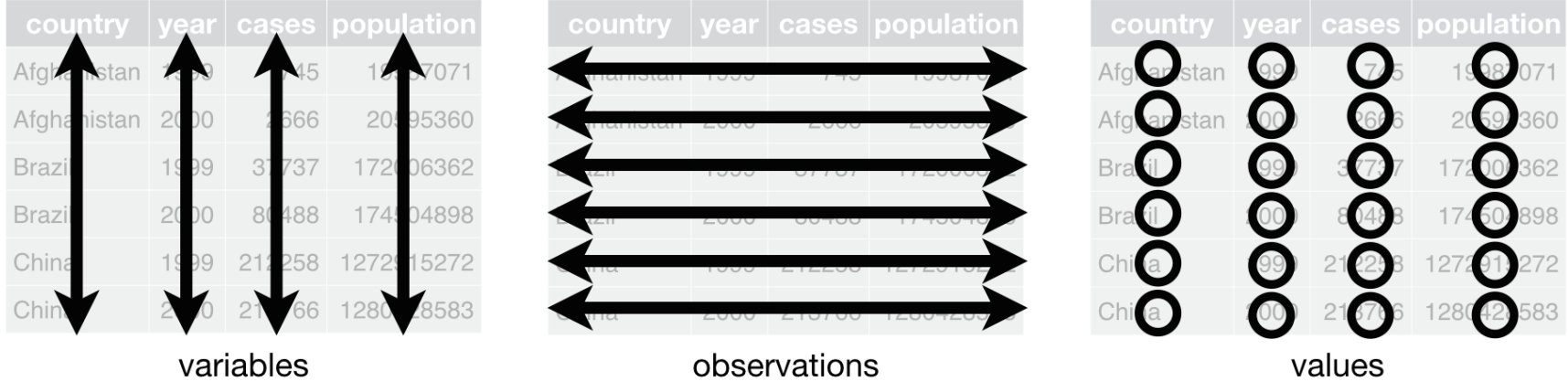
observations

Why Tidy?

1. Many functions in R operate on so-called *long* format data, requiring dependent and independent variables to be in different columns of a data frame.

2. Having a consistent way to store and structure your data makes it more *generic*. This makes it easier to use it with different functions.

3. Being *generic* also makes it easier to understand a new dataset in this format.



A Tidy Structure

```
##    Participant  Block Viewpoint       RT
## 1            1 Block1      same 516.5898
## 2            2 Block1      same 382.0560
## 3            3 Block1      same 514.7835
## 4            4 Block1      same 604.1356
## 5            5 Block1      same 468.6312
## 6            6 Block1      same 317.7252
## 7            7 Block1      same 451.3933
## 8            8 Block1      same 704.9653
## 9            9 Block1      same 383.3551
## 10          10 Block1      same 668.2327
## 11           1 Block2      same 305.1622
## 12           2 Block2      same 386.3059
## 13           3 Block2      same 378.0082
## 14           4 Block2      same 450.9009
## 15           5 Block2      same 211.0952
```

Tidyr

The **tidyr** package contains functions to help tidy up your data.

We'll look now at **gather()** and **separate()**.

To start tidying our data, we need the RTs to be in a single column.

```
head(example_rt_df, n = 4)
```

```
##   Participant Block1_same Block2_same Block1_different Block2_different
## 1           1    516.5898    305.1622         498.2227         313.3878
## 2           2    382.0560    386.3059         679.8127         339.2907
## 3           3    514.7835    378.0082         355.0890         442.7510
## 4           4    604.1356    450.9009         616.3194         477.3192
```

The function **gather()** can be used to combine columns into one.

Look at the help using **?gather**

Step 1 - gather()

The syntax looks like this:

```
gather(data,
       key = "key",
       value = "value",
       ...)
```

The first argument, *data*, is the name of the data frame you want to modify.

key is the name of the new column that will contain the values of a single categorical variable.

value is the name of the new column containing the values for each level of that variable.

Step 1 - gather()

```
gather_rt <-
  gather(example_rt_df,
         key = "exp_cond",
         value = "RT")
head(gather_rt)
```

```
##       exp_cond RT
## 1 Participant  1
## 2 Participant  2
## 3 Participant  3
## 4 Participant  4
## 5 Participant  5
## 6 Participant  6
```

But this isn't right! We need to tell it to leave the *Participant* column alone.

Step 1 - gather()

After we specify the "key" and "value" columns, we need to specify which columns we want to be *gathered*.

```
gather_rt <-
  gather(example_rt_df,
         key = "exp_cond",
         value = "RT",
         2:5)
head(gather_rt)
```

```
##   Participant    exp_cond       RT
## 1           1 Block1_same 516.5898
## 2           2 Block1_same 382.0560
## 3           3 Block1_same 514.7835
## 4           4 Block1_same 604.1356
## 5           5 Block1_same 468.6312
## 6           6 Block1_same 317.7252
```

```
gather_rt <-
  gather(example_rt_df,
         key = "exp_cond",
         value = "RT",
         Block1_same:Block2_different)
head(gather_rt)
```

```
##   Participant    exp_cond       RT
## 1           1 Block1_same 516.5898
## 2           2 Block1_same 382.0560
## 3           3 Block1_same 514.7835
## 4           4 Block1_same 604.1356
## 5           5 Block1_same 468.6312
## 6           6 Block1_same 317.7252
```

Step 2 - separate()

We have the RTs in one column, but we still have another problem:

The "Block" and "Viewpoint" variables are combined into a single column.

```
head(gather_rt)
```

```
##   Participant    exp_cond       RT
## 1           1 Block1_same 516.5898
## 2           2 Block1_same 382.0560
## 3           3 Block1_same 514.7835
## 4           4 Block1_same 604.1356
## 5           5 Block1_same 468.6312
## 6           6 Block1_same 317.7252
```

Step 2 - separate()

Fortunately, the values in the *exp_cond* column can be easily split:

```
unique(gather_rt$exp_cond)
```

```
## [1] "Block1_same"      "Block2_same"      "Block1_different"
## [4] "Block2_different"
```

The value of "Block" comes before the underscore ("_"), while the value of viewpoint comes after it.

We can use the **separate()** command split this into two columns.

Type **?separate()**

Step 2 - separate()

Let's look at the syntax.

```
separate(data,
         col,
         into,
         sep, ...)
```

Data is the data frame you want to modify.

col is the name of column you want to separate.

into is the names of the new columns you want to create.

sep is the character that *separates* the values you want to split.

Step 2 - separate()

```
gather_rt <-
  separate(gather_rt,
           col = "exp_cond",
           into = c("Block", "Viewpoint"),
           sep = "_")
head(gather_rt)
```

```
##   Participant  Block Viewpoint       RT
## 1           1 Block1      same 516.5898
## 2           2 Block1      same 382.0560
## 3           3 Block1      same 514.7835
## 4           4 Block1      same 604.1356
## 5           5 Block1      same 468.6312
## 6           6 Block1      same 317.7252
```

Your target

```
##    Participant  Block Viewpoint       RT
## 1            1 Block1      same 516.5898
## 2            2 Block1      same 382.0560
## 3            3 Block1      same 514.7835
## 4            4 Block1      same 604.1356
## 5            5 Block1      same 468.6312
## 6            6 Block1      same 317.7252
## 7            7 Block1      same 451.3933
## 8            8 Block1      same 704.9653
## 9            9 Block1      same 383.3551
## 10          10 Block1      same 668.2327
## 11           1 Block2      same 305.1622
## 12           2 Block2      same 386.3059
## 13           3 Block2      same 378.0082
## 14           4 Block2      same 450.9009
## 15           5 Block2      same 211.0952
```

Your target

```
##   Participant  Block Viewpoint       RT
## 1           1 Block1      same 516.5898
## 2           2 Block1      same 382.0560
## 3           3 Block1      same 514.7835
## 4           4 Block1      same 604.1356
## 5           5 Block1      same 468.6312
## 6           6 Block1      same 317.7252
## 7           7 Block1      same 451.3933
## 8           8 Block1      same 704.9653
```

Step 1 - **gather()** the RTs

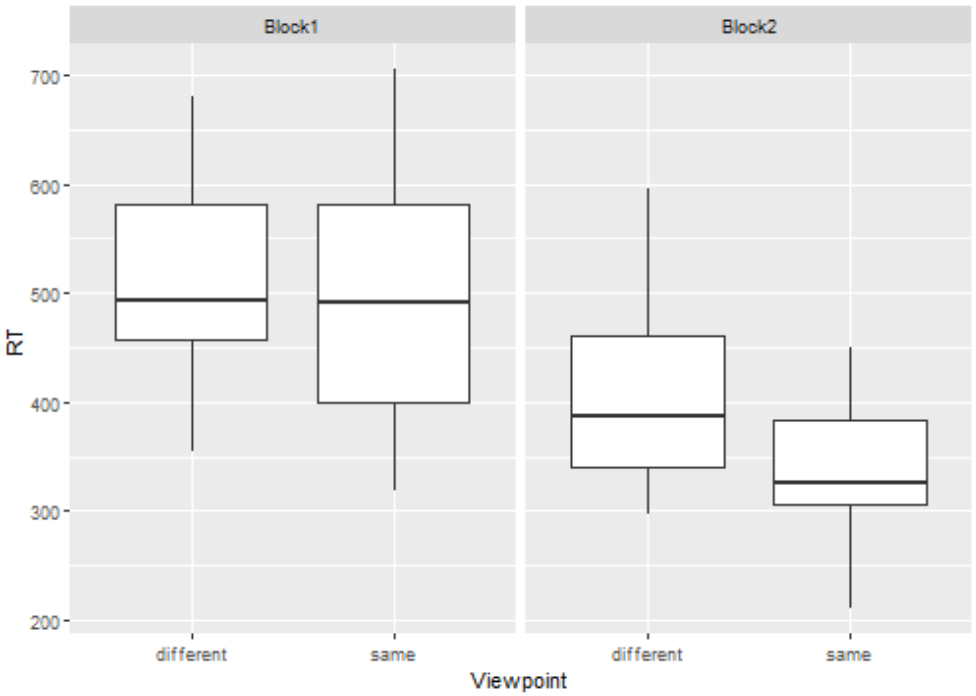
Step 2 - **separate()** the variables

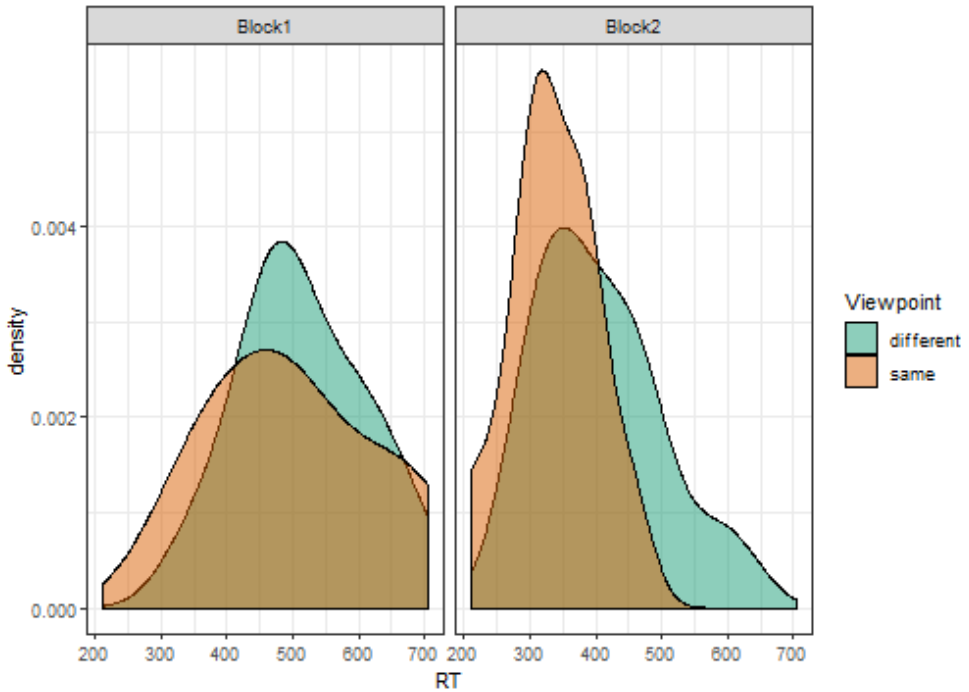
Now what?

Now that it's tidy...

Now that we've got the data in a tidy format, we can begin to use some of the more interesting features of R!

We can produce a boxplot using **ggplot2** (more next week!)





Now that it's tidy...

We can produce some summary statistics using **dplyr** (more soon!)

```
## # A tibble: 4 x 4
## # Groups:   Block [2]
##   Block  Viewpoint mean_RT sd_RT
##   <chr>  <chr>       <dbl> <dbl>
## 1 Block1 different    514.  96.2
## 2 Block1 same         501. 127.
## 3 Block2 different    405.  92.5
## 4 Block2 same         335.  67.7
```

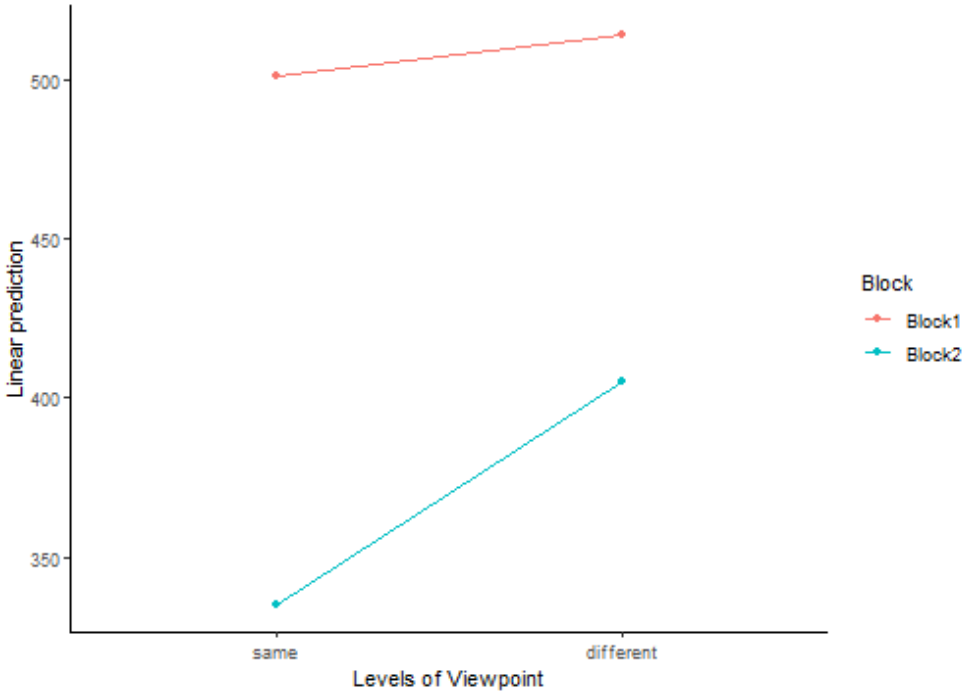
Now that it's tidy...

We can run ANOVA with **afex**.

```
## Anova Table (Type 3 tests)
##
## Response: RT
##             Effect   df      MSE         F ges p.value
## 1            Block 1, 9  7947.32 23.85 *** .35   .0009
## 2        Viewpoint 1, 9  3909.24    4.38 + .05     .07
## 3 Block:Viewpoint 1, 9 12691.81      0.63 .02     .45
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '+' 0.1 ' ' 1
```

Now that it's tidy...

We can create interaction plots using **emmeans**.




```
## Block = Block1:
##  contrast         estimate   SE   df t.ratio p.val
##  same - different    -13.0 40.7 14.1 -0.320  0.754
##
## Block = Block2:
##  contrast         estimate   SE   df t.ratio p.val
##  same - different    -69.7 40.7 14.1 -1.711  0.109
```

Next week

- Chapters 3 and 28 of R for Data Science
  - Data Visualization
  - Graphics for communication
- RStudio.cloud Primer
  - Visualize Data
- Datacamp
  - Data Visualization with ggplot2 (Part 1)

UNIVERSITY OF
LINCOLN

Splitting columns up

Sometimes you want to go in the *opposite* direction.

spread() is the *opposite* of **gather()**.

```
spread_rt <-
  spread(gather_rt,
       Block,
       RT)
head(spread_rt, 10)
```

```
##    Participant Viewpoint   Block1   Block2
## 1            1 different 498.2227 313.3878
## 2            1      same 516.5898 305.1622
## 3            2 different 679.8127 339.2907
## 4            2      same 382.0560 386.3059
## 5            3 different 355.0890 442.7510
## 6            3      same 514.7835 378.0082
## 7            4 different 616.3194 477.3192
## 8            4      same 604.1356 450.9009
## 9            5 different 434.6988 340.4552
## 10           5      same 468.6312 211.0952
```

Combining columns into one

We used **separate()** to split up our "exp_cond" column.

We can use **unite()** to put hem back together.

```
united_rt <-
  unite(gather_rt,
      exp_cond, # the name of the new column
      Block,
      Viewpoint,
      sep = "_")
head(united_rt, 8)
```

```
##   Participant    exp_cond       RT
## 1           1 Block1_same 516.5898
## 2           2 Block1_same 382.0560
## 3           3 Block1_same 514.7835
## 4           4 Block1_same 604.1356
## 5           5 Block1_same 468.6312
## 6           6 Block1_same 317.7252
## 7           7 Block1_same 451.3933
## 8           8 Block1_same 704.9653
```

Recreating the original data frame

Step 1 - **unite()** the columns

```
united_rt <-
  unite(gather_rt,
      exp_cond, # the name of the new column
      Block,
      Viewpoint,
      sep = "_")
```

Step 2 - **spread()** the columns

```
spread_rt <-
  spread(united_rt,
       exp_cond,
       RT)
```

Recreating the original data frame

```
spread_rt
```

```
##    Participant Block1_different Block1_same Block2_different Block2_same
## 1            1         498.2227    516.5898         313.3878    305.1622
## 2            2         679.8127    382.0560         339.2907    386.3059
## 3            3         355.0890    514.7835         442.7510    378.0082
## 4            4         616.3194    604.1356         477.3192    450.9009
## 5            5         434.6988    468.6312         340.4552    211.0952
## 6            6         479.6596    317.7252         467.2746    309.5307
## 7            7         546.2049    451.3933         297.5012    274.3238
## 8            8         594.0583    704.9653         405.3054    317.5622
## 9            9         489.2698    383.3551         368.8661    385.1461
## 10          10         448.7217    668.2327         596.6026    333.7526
```