R FOR ARTS

SESSION 2

MARCH 17TH 2017

FEEDBACK

RECAP SESSION 1

CORE CONCEPTS

    b. Working directories

2. **Objects and data classes**
    a. Object assignment
    b. Numeric, Character, Logical, Factor

3. **Data structures**
    a. Vectors
    b. Matrices
    c. Lists
    d. Data Frames

4. **Functions**

CODING PRACTICE (CP)

CP1: # EVERYTHING, BECAUSE #YOLO

In R, the # symbol will 'comment out' everything that follows it on that line.

This allows you to make comments about your work and is SUPER IMPORTANT!

CP2: LEAVE YOURSELF SPACE!

Code is hard to read at the best of times.

Make life easy for yourself and put spaces between code elements.

CP3: SENSIBLE NAMES

Object names require the following:

- They MUST begin with a letter
- They MUST NOT contain spaces
- They ARE case sensitive
- They can ONLY contain letters, numbers, _ and .

Be consistent in your naming convention, some example conventions are:

- this_is_snake_case
- ThisIsCamelCase
- this.uses.full.stops

SESSION 2

WHAT WILL BE COVERED

1. Installing and using packages
2. Importing, navigating and tidying data
3. Basic descriptive stats

WHAT WILL NOT BE COVERED

1. Plotting data (next week)
2. Producing data reports (next week)

R FOR DATA SCIENCE

http://r4ds.had.co.nz/index.html

R FOR DATA SCIENCE

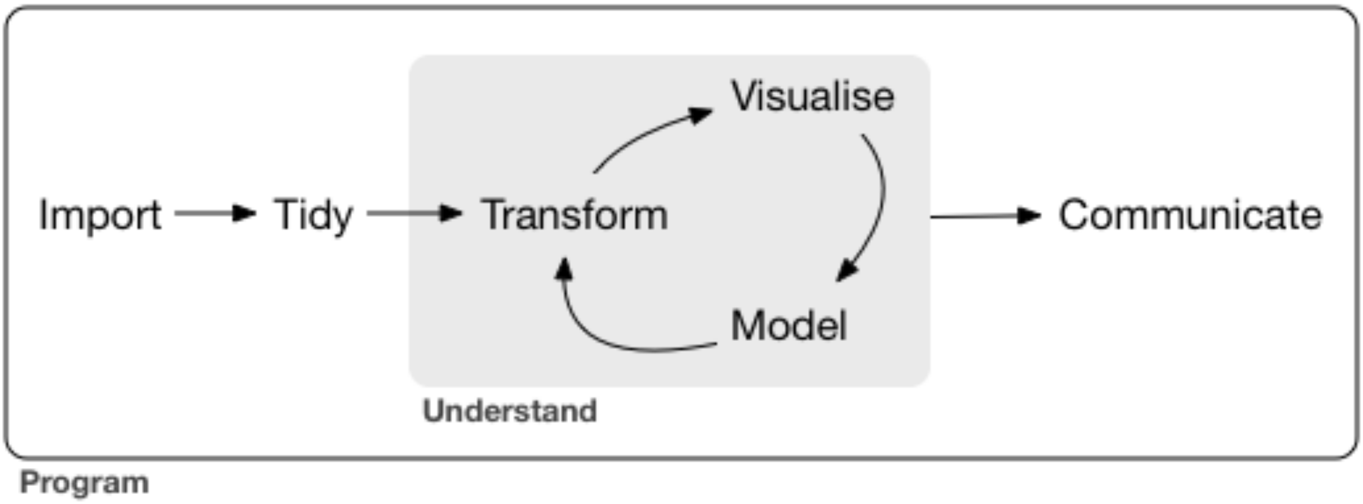


Image drawn from: http://r4ds.had.co.nz/introduction.html

CAT'S SKINS AND FIXING THINGS

R is crazy versatile, this means:

1. You can perform almost any analysis.
2. 'There's more than one way to skin a cat' - There are many ways to do the same thing.

Everything that I present is just one way to approach the problem. There may very well be a more efficient way to do it.

For the time being - 'If it ain't broke, don't fix it'.

PACKAGES

PACKAGES

There is a package for almost everything.

There are currently 10,265 available packages (as of 10:25AM on 15th March 2017).

A full list of all available packages can be found at:

PACKAGES

To install a package, we use the `install.packages()` command.

For instance, if we want to install the psych package, we would use the following code:

```
install.packages('psych')
```

This process only needs to be completed once, just like installing a program.

Note that the package name is encased in quotes. This is because the CRAN server requires character information in order to match the package request against available packages.

PACKAGES: ADMIN PRIVILEGES

Vijay or Robert for help.

USING PACKAGES

Before we can use a package, we have to call it into our R workspace. This is accomplished with the `library()` function, e.g.:

```
library(psych)
```

Notice that here, there are no quotations. This is because the package is now an object in you local R build.

USING PACKAGES

Every package comes with its own selection of functions that perform specific tasks. For example, the psych package has the `headTail()` function which helps with viewing datasets.

```
headTail(mtcars) # mtcars is a built-in dataset. Use data() to
#                 mpg cyl disp  hp drat   wt  qsec  vs  am gear
#Mazda RX4         21   6  160 110  3.9 2.62 16.46   0   1    4
#Mazda RX4 Wag     21   6  160 110  3.9 2.88 17.02   0   1    4
#Datsun 710      22.8   4  108  93 3.85 2.32 18.61   1   1    4
#Hornet 4 Drive  21.4   6  258 110 3.08 3.21 19.44   1   0    3
#...              ... ...  ... ...  ...  ...   ... ... ...  ...
#Ford Pantera L  15.8   8  351 264 4.22 3.17  14.5   0   1    5
#Ferrari Dino    19.7   6  145 175 3.62 2.77  15.5   0   1    5
#Maserati Bora     15   8  301 335 3.54 3.57  14.6   0   1    5
#Volvo 142E      21.4   4  121 109 4.11 2.78  18.6   1   1    4
```

USING PACKAGES: DEALING WITH MASKED FUNCTIONS

Be aware that packages sometimes use the same names for their functions and can 'mask' each other, e.g.:

```
library(dplyr)
#Attaching package: ‘dplyr’
#
#The following objects are masked from ‘package:stats’:
#
```

```
#
```

USING PACKAGES: DEALING WITH MASKED FUNCTIONS

If you have masked a function that you wish to call, you can tell R which function you wish to use.

To do this, we use the `::` notation, e.g.:

```
library(dplyr)

#Attaching package: ‘dplyr’
#
#The following objects are masked from ‘package:stats’:
#
#    filter, lag
#
#The following objects are masked from ‘package:base’:
#
#    intersect, setdiff, setequal, union
#

filter(mtcars, cyl == 8) # Using the dplyr filter function
```

PACKAGES

A full script for installing, loading and using a package might look something like this:

```
install.packages('psych')
```

```
#Mazda RX4 Wag     21   6  160 110  3.9 2.88 17.02   0   1    4
#Datsun 710       22.8   4  108  93 3.85 2.32 18.61   1   1    4
#Hornet 4 Drive 21.4   6  258 110 3.08 3.21 19.44   1   0    3
#...               ...  ...  ...  ...  ...  ...   ...  ...  ...   ...
#Ford Pantera L 15.8   8  351 264 4.22 3.17  14.5   0   1    5
#Ferrari Dino    19.7   6  145 175 3.62 2.77  15.5   0   1    5
#Maserati Bora     15   8  301 335 3.54 3.57  14.6   0   1    5
#Volvo 142E       21.4   4  121 109 4.11 2.78  18.6   1   1    4
```

SOME HANDY PACKAGES:

	What is it?	Install	Load
psych	A package for personality, psychometric, and psychological research	`install.packages('psych`	`library(psych`
dplyr	A fast, consistent tool for working with data frame like objects.	`install.packages('dplyr`	`library(dplyr`
tidyr	Tool designed specifically for data tidying	`install.packages('tidyr`	`library(tidyr`
Hmisc	Contains many functions useful for data analysis	`install.packages('Hmisc`	`library(Hmisc`

IMPORTING DATA

The first step of any data analysis is to load in the data.

R is very versatile and it is possible for it to handle most data types. However, the easiest (and preferred) data format to import into R is a text file.

In this course, we will be working with comma separated value (.csv) files.

If your data is in another format (e.g. .xlsx, .xls, .sav, etc.) it is often best to convert to .csv from within the associated program that created the file.

options are offered here

IMPORTING DATA: .CSV STRUCTURE

A .csv file is simply a text file with a specific structure.

Every row in a .csv file represents a row of data with commas separating the columns on each row.

We have a mock .csv file about cats (available here). It looks like this:

```
#cat,age,weight,colour,loose.in.cat.room
#Pusheen,4,3.00,chocolate,no
#Millie,7,3.75,black,no
#Pumpkin,13,4.25,grey,no
#Prof. Jiggly,9,3.25,tortoiseshell,yes
```

If this was read into R, we would have a data file that looks like this:

```
#           cat age      weight        colour loose.in.cat.roo
#1       Pusheen   4       3.00     chocolate
#2         Millie   7       3.75         black
#3       Pumpkin  13       4.25          grey
#4 Prof. Jiggly   9       3.25 tortoiseshell                y
```

IMPORTING DATA: .CSV STRUCTURE

Although csv stands for comma separated values, the actual character that separates the columns can vary. For instance, tabs are also commonly used, e.g.:

```
#cat    age   weight   colour    loose in cat room?
#Pusheen    4    3.00   chocolate    no
```

This data would be represented in R in the same way as a comma separated file as long as we tell R what the separator is.

IMPORTING DATA

The command that we use to import data into R is `read.table()`, e.g.:

```
read.table( 'cats.csv' , sep = ',')
```

Note that the name of the file and the separator must be encased in quotes.

If the first row of our dataset contains the column headings, we need to let R know.

```
read.table( 'cats.csv' , sep = ',' , header = TRUE)
```

```
cats_demo <- read.table( 'cats.csv' , sep = ',' , header = TRU
```

If the data being imported is not assigned to an object, the whole dataset will be printed to the command window. This is really annoying!

IMPORTING DATA: SPECIAL CASE COMMA IMPORTS

The .csv format is so common in data analysis that R has a dedicated function for its import `read.csv()`.

This function can be used without specifying the separator, provided that the data is comma separated.

```
cats_demo <- read.csv( 'cats.csv' , header = TRUE)
```

NAVIGATING DATA

NAVIGATING DATA

After importing data using `read.table()` or `read.csv()` the data are stored as a `data.frame`

```
demo_cats <- read.csv( 'cats.csv' , header = TRUE )
class(demo_cats)
#[1] "data.frame"
```

NAVIGATING DATA: 1D

R uses square brackets to select elements of data objects.

For example, if we want to access the third element of a vector we would write the following:

```
my_vector <- c( 2 , 4 , 6 , 8 , 10)
my_vector[3]
#[1] 6
```

However, vectors only have 1 dimension and the data the we have imported has 2 dimensions.

NAVIGATING DATA: 2D

Fortunately, the code for navigating 2-dimensional data is the same as that for 1-dimensional data.

e.g.: `[1,3]`

R uses the first number as an index for the row and the second number as an index for the column. So, in this example we would be asking for the data element that is in row 1, column 3 of the dataset.

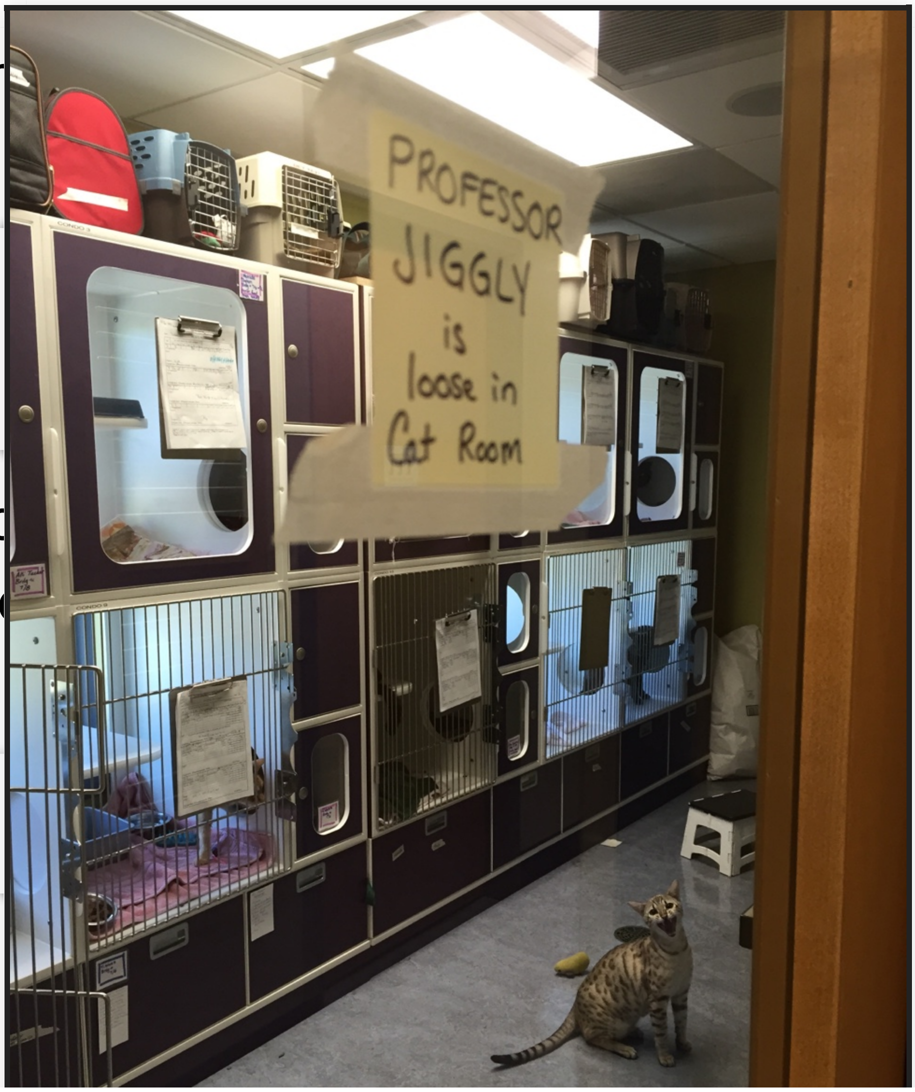
NAVIGATING DATA: 2D

If we now return
selecting data in a

```
#              cat age
#1         Pusheen   4
#2          Millie   7
#3         Pumpkin  13
#4 Prof. Jiggly    9
```

If we wanted to kno
room, we would ne
row 3, colu

```
demo_cats[4,5]
#[1] yes
#Levels: no yes
```



ROWS/COLUMNS

The same code used to access elements of a `data.frame` can also be used to select whole rows or columns. This is acheived by omitting the row or column number.

For example, if we want to know the weights of all our cats we would need column 3 and we would write the following:

```
cats_demo[,3]
#[1] 3.00 3.75 4.25 3.25
```

NAVIGATING DATA: SELECTING ENTIRE ROWS/COLUMNS

If we wanted to get all the information about Prof. Jiggly we would need row 4 and we would write the following:

```
cats_demo[4,]
#            cat age      weight        colour loose.in.cat.roo
#4 Prof. Jiggly   9        3.25 tortoiseshell                y
```

NAVIGATING DATA: $$ BILL Y'ALL

`data.frame`s have a feature which allows us to access a column by using its name.
To do this, we use the `$` operator.
For instance, to get the weight of our cats we could use the method offered 2 slides ago (i.e. `cats_demo[,3]`) or we could use the following:

```
cats_demo$weight
#[1] 3.00 3.75 4.25 3.25
```

Both methods return the same data but by using the `$` operator we can be more targeted in our selections.

TIDYING DATA

TIDY DATA

There exists such a thing as tidy data and it takes the following form:

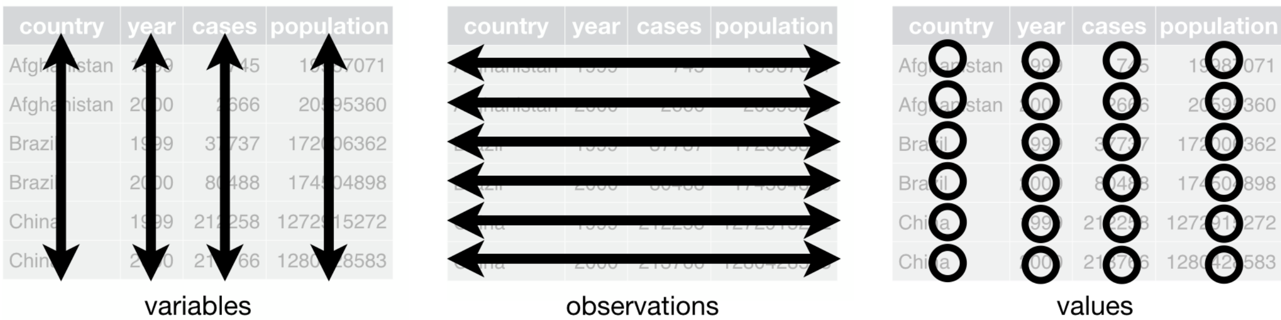


Image drawn from: http://r4ds.had.co.nz/tidy-data.html

This will not be the optimal structure for all data sets but it is certainly a good place to start.

TIDY DATA

Datasets come in a dizzying array of different formats and not all of them will be tidy.

Let's return to our cats to exemplify this. Here we have food consumption data for our cats over a 3-month period, with all values in grams. You can find this data here

```
cat_food
#          Name food_type  Jan  Feb  Mar
#1      Pusheen       dry 2500 3000 2500
#2      Pusheen       wet 3000 3000 2000
#3       Millie       dry 1000 3000 2000
#4       Millie       wet 4000 2000 3000
#5      Pumpkin       dry 4000 5500 3500
#6      Pumpkin       wet 2000 2500 3000
#7 Prof. Jiggly       dry 5000 4500 4000
#8 Prof. Jiggly       wet    0 1000 1500
```

TIDY DATA

First, let's group the months.

Hadley Wickham has a particularly handy package called `tidyr` that is useful here. To do this, we will be using the `gather()` function from this package.

```
library(tidyr)
cat_food <- read.csv( 'https://raw.githubusercontent.com/vj-so
tidy_cats <- gather( data = cat_food, Jan , Feb , Mar , key =
tidy_cats
#           Name food_type Month food_consumed
#1       Pusheen       dry   Jan          2500
#2       Pusheen       wet   Jan          3000
#3        Millie       dry   Jan          1000
#4        Millie       wet   Jan          4000
#5       Pumpkin       dry   Jan          4000
#6       Pumpkin       wet   Jan          2000
#7  Prof. Jiggly       dry   Jan          5000
#8  Prof. Jiggly       wet   Jan             0
#9       Pusheen       dry   Feb          3000
#10      Pusheen       wet   Feb          3000
```

TIDY DATA

Now, let's separate the food types. To do this, we will be using the `spread()` function from this package.

Data is carried over from the previous slide.

```
library(tidyr)
tidy_cats <- spread( data = tidy_cats, key = food_type , value
#            Name Month  dry  wet
#1         Millie   Feb 3000 2000
#2         Millie   Jan 1000 4000
#3         Millie   Mar 2000 3000
#4  Prof. Jiggly   Feb 4000 1000
#5  Prof. Jiggly   Jan 5000    0
#6  Prof. Jiggly   Mar 4000 1500
#7        Pumpkin   Feb 5500 2500
#8        Pumpkin   Jan 4000 2000
#9        Pumpkin   Mar 3500 3000
#10       Pusheen   Feb 3000 3000
#11       Pusheen   Jan 2500 3000
#12       Pusheen   Mar 2500 2000
```

TIDY DATA

The `tidyr` package has a number of useful functions for tidying data. Please do explore them for yourselves.

Function	Purpose
`separate()`	Turns a single character column into multiple columns.

Function	Purpose
	explicit missing values.

CHECKING AND SELECTING DATA

CHECKING DATA: COMPLETE CASES

One should ALWAYS look at their data before

Data can be checked for missing values using the `complete.cases()` function.

```
complete.cases(tidy_cats)
#[1]  TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TR
```

This checks the rows of a dataset for an observation in each column and is very easy to use with tidy data.

CHECKING DATA: COMPLETE CASES

If there was a missing observation in the data, the function would have returned a `FALSE` value for the incomplete row, e.g.:

```
tidy_cats$Month[1] <- NA
tidy_cats
#             Name Month  dry  wet
#1          Millie  <NA> 3000 2000
#2          Millie   Jan 1000 4000
#3          Millie   Mar 2000 3000
#4   Prof. Jiggly   Feb 4000 1000
#5   Prof. Jiggly   Jan 5000    0
#6   Prof. Jiggly   Mar 4000 1500
#7         Pumpkin   Feb 5500 2500
#8         Pumpkin   Jan 4000 2000
#9         Pumpkin   Mar 3500 3000
#10        Pusheen   Feb 3000 3000
#11        Pusheen   Jan 2500 3000
#12        Pusheen   Mar 2500 2000
```

CHECKING DATA: MISSING VALUES

R will default to telling you that something is unknown rather than assuming anything.

```
a <- 2
b <- NA
a + b
#[1] NA
```

Note how the output is not an integer, R is conservative and lets you know that you have missing data.

CHECKING DATA: MISSING VALUES

You can find which data is missing by using the `is.na()` function. This will return a logical vector where a TRUE statement indicates an NA.
To find the position of the NA value(s), combine this with the `which()` function.

```
test <- 1:1000
test[56] <- NA
test
```

SELECTING DATA

There will often be times where you only want a specific subset of the data that you have.

For instance, in our cat data, we may only want the data that pertains to the month of March.

We can do this by employing some logical operators:

Operator	Description
<	Less then
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	equivalent to
!=	not equivalent to

SELECTING DATA

So, let's go ahead and subset our data into only those

```
#12       Pusheen   Mar 2500 2000
```

SELECTING DATA

What if we wanted the data for all months except March?

```
tidy_cats[ tidy_cats$Month != 'Mar' , ]
#            Name Month  dry  wet
#1        Millie   Feb 3000 2000
#2        Millie   Jan 1000 4000
#4  Prof. Jiggly   Feb 4000 1000
#5  Prof. Jiggly   Jan 5000    0
#7       Pumpkin   Feb 5500 2500
#8       Pumpkin   Jan 4000 2000
#10      Pusheen   Feb 3000 3000
#11      Pusheen   Jan 2500 3000
```

Now let's get the data for those cats that eat more than 2000 grams of wet food a month.

```
tidy_cats[ tidy_cats$wet > 2000 , ]
#       Name Month  dry  wet
#2   Millie   Jan 1000 4000
#3   Millie   Mar 2000 3000
#7  Pumpkin   Feb 5500 2500
#9  Pumpkin   Mar 3500 3000
#10 Pusheen   Feb 3000 3000
#11 Pusheen   Jan 2500 3000
```

SELECTING DATA

What about the data for those cats that eat no more than 1000 grams of dry food a month.

```
tidy_cats[ tidy_cats$dry <= 1000 , ]
#       Name Month  dry  wet
#2   Millie   Jan 1000 4000
```

SELECTING DATA: GO FORTH AND SUBSET

There are a whole host of other methods for selecting data and I would encourage you to explore these options but those presented here should suffice for the beginner.

An additional function that you might like to look into is:

```
subset()
```

SELECTING DATA: DEALING WITH TEXT

R handles character (or 'string') data in a similar way to vectors. However, there are some quirks and the

SELECTING DATA: DEALING WITH TEXT

Here we will be dealing with a toy example using the names in out cat data but all examples can be extrapolated to larger datasets.

The most simple task to ask of a character vector is how many characters there are. This can be accomplished by using the `nchar()` function.

```
nchar(as.character(tidy_cats$Name[4]))
[1] 12
```

But the `str_length()` function from the stringr package makes this easier:

```
library(stringr)
str_length(tidy_cats$Name[4])
[1] 12
```

SELECTING DATA: DEALING WITH TEXT

The stringr package also allows us to access the individual characters in a character vector.

```
#[1] "Prof. Jiggly"

str_sub(tidy_cats$Name[4],1,5)
#[1] "Prof."

str_sub(tidy_cats$Name[4],1,-3)
#[1] "Prof. Jigg"
```

stringr also allows us to handle whitespace:

```
"   Professor Jiggly is loose in the cat room!  "
#[1] "   Professor Jiggly is loose in the cat room!  "

str_trim("   Professor Jiggly is loose in the cat room!  ", side = 'b
#[1] "Professor Jiggly is loose in the cat room!"
```

This function can come in very handy when standardising a data set.

SELECTING DATA: DEALING WITH TEXT

Some stringr functions:

Function	Description
`str_length(`	Gets number of characters in a string
`str_sub()`	Allows access to individual characters
`str_trim()`	Removes leading and trailing whitespace
`str_subset(`	returns the elements of a character vector that match a regular expression

DESCRIPTIVES

DESCRIPTIVES

Descriptive statistics do what they say on the tin. They describe your data.

These statistics include things such as the mean, median, mode, range, interquartile range, standard deviation and standard error.

DESCRIPTIVES: MEAN

Let's return to our cat data to calculate a mean.

RQ: What is the mean amount of dry cat food eaten across all cats and all months?

```
mean(tidy_cats$dry)
#[1] 3333.333
```

DESCRIPTIVES: MEDIAN

DESCRIPTIVES: MODE

RQ: What is the mode amount of dry cat food eaten across all cats and all months?

```
cats_dry_table <- table(tidy_cats$dry)
cats_dry_table
#1000 2000 2500 3000 3500 4000 5000 5500
#   1    1    2    2    1    3    1    1
cats_dry_mode <- cats_dry_table[ cats_dry_table == max( cats_d
as.integer( names( cats_dry_mode ) )
#4000
```

RQ: What is the range in the amount of dry cat food eaten across all cats and all months?

```
max(tidy_cats$dry) - min(tidy_cats$dry)
#[1] 4500
```

DESCRIPTIVES: INTERQUARTILE RANGE

RQ: What is the interquartile range for the amount of dry cat food eaten across all cats and all months?

```
IQR(tidy_cats$dry)
#[1] 1500
```

DESCRIPTIVES: STANDARD DEVIATION

RQ: What is the standard deviation for the amount of dry cat food eaten across all cats and all months?

```
sd( tidy_cats$dry )
#[1] 1267.304
```

DESCRIPTIVES: DO IT IN A ONER

RQ: What are the descriptives for the tidy_cats dataset?

```
summary(tidy_cats)

#           Name       Month                    dry           wet
```

```
#                                            Max.    :5500   Max.    :40

library(psych)
describe(tidy_cats)

#         vars  n    mean      sd median trimmed     mad  min  m
#Name*       1 12    2.50    1.17    2.5     2.5    1.48    1
#Month*      2 12     NaN      NA     NA     NaN      NA  Inf -I
#dry         3 12 3333.33 1267.30 3250.0  3350.0 1111.95 1000 55
#wet         4 12 2250.00 1076.61 2250.0  2300.0 1111.95    0 40
```

DESCRIPTIVES: BUT I WANT A GRAPH!

By this point I would be crying out for a visual representation of the data.

I apologise but that is the topic of next week's session.

However, I feel your pain!

PACKAGES

Vist https://cran.r-project.org/web/packages/available_packages_by_date.html and search for a package that you might find useful. Once you have found a package, do the following:

1. Install it
2. Load it into your R workspace
3. Use the help function to learn more about the package
4. Try using one of the package's functions (if you need a test dataset, R has some built in. Use `data()` to see what is available.)

PACKAGES

5. How would you specify that you wanted to specifically use the `alpha()` function from the psych package?
6. Do you need to install a package each time you want to use it?
7. Would it be important to include an `install.packages()` command in a script that you were sharing? Why?

IMPORTING DATA

1. What types of data can R handle?
2. What is the preferred data format and what is its structure?
3. When importing data, what is meant by the term 'separator'?

IMPORTING DATA

5. Using the `read.table()` function, read in the dataset at the following location http://raw.githubusercontent.com/vj-sol/R-for-Arts-2/master/resources/cats.csv and assign it to an object.
6. Using the `read.csv()` function, read in the dataset at the following location http://raw.githubusercontent.com/vj-sol/R-for-Arts-2/master/resources/cat-food.csv and assign it to another object.

NAVIGATING DATA

1. What type of brackets are used to extract data from a 1 dimensional object in R?
2. What type of brackets are used to extract data from a 2 dimensional object in R?
3. What is the difference between the code for extracting data from a 1 dimensional dataset and a 2 dimensional data set?
4. What type of operator can be used to extract

NAVIGATING DATA

5. For the vector `my_vect <- c(23 , 49 , 567 , 32 , 90)` predict what the result of the following code will be without running it:
   a. `my_vect[4]`
   b. `my_vect[-2]`
   c. `my_vect[6]`
   d. `my_Vect[2]`
   e. `my_vect[4,]`

NAVIGATING DATA

6. For the cats data that you imported using `read.table()`, do the following:
    a. Extract the weight column and assign it to an object.
    b. Extract the data value 'grey' and assign it to an object.
    c. Extract all information about Millie the cat and assign it to an object.
    d. Extract Millie's age from the newly created object.

TIDYING DATA

1. Summarise what the structure of tidy data looks like
2. Should all data be stored in the tidy format?
3. Find the two functions from the tidyr package that were used in this session and use the help function to find out more about them. What inputs do they

TIDYING DATA

4. Given the following dataset (found here), perform the following actions:

```
#    whiskers    tail.length    cat1    cat2    cat3
#    5           12             Toby    NA      NA
#    9           15             NA      Cleo    NA
#    7           10             NA      NA      Sir Snuffles
```

a. Collect the cat names into one column
b. Move the newly created name column to the left most data column (do not overwrite any data!)
c. Make sure that the column names match up correctly (you may need to investigate the `colnames()` function).

TIDYING DATA

5. Given the following dataset (found here), perform the following actions:

```
# name          feature        length
#         Toby    whiskers       5
#         Toby tail.length      12
#         Cleo    whiskers       9
#         Cleo tail.length      15
# Sir Snuffles    whiskers       7
# Sir Snuffles tail.length      10
```

a. Arrange the data such that whiskers and tail.length are variables with length as a value

CHECKING AND SELECTING DATA

For this series of exercises, we'll be using some real data.

This data (found here) is taken from the FBI's National Instant Criminal Background Check System and concerns firearm background check data from Nov. 1998 - Feb. 2017.

The data was collated and put into a database form by BuzzFeed News.

CHECKING AND SELECTING DATA

For the data provided, perform the following actions:

1. Check to see if there are any missing cases in the data. Explain your findings.
2. Get all data pertaining to the state of Texas and assign it to an object.
3. Using your new object, find out how many hand guns were sold in January 2004
4. Using your new object, how many `permit_recheck`s were performed in Texas across the whole period of this dataset?
5. Perform steps 2, 3 and 4 for the state of California

DESCRIPTIVES

Here we will continue using the FBI data regarding firearm background checks.

Using that data, find out the following information:

1. The mean number of `private_sale_long_gun`s
2. The median number of `private_sale_long_gun`s in the state of Wyoming
3. The number of `private_sale_long_gun`s in the state of Wyoming in March 2016

DESCRIPTIVES

4. Use a single function to produce descriptives for `private_sale_handgun`s in the state of Alabama
    a. What is the max value?
    b. What is the min value?
    c. What is the mean value?
    d. What is the median value?
    e. Work out the interquartile range without calling the `IQR()` function.
    f. What is the standard deviation of the data? If you can't tell from your descriptives, might there be another function that could help?
    g. On this slide, I offered a way to calculate the mode. Can you think of another way to do this? Apply your method to your current dataset to find the mode.

PROJECT 1: PREP THAT DATA!

This project is designed to mimic the process of wrangling with your own data. It uses actual data from the European Union Open Data Portal and concerns the numbers of accidents at work across the EU from 2008 to 2014 by days lost, sex and age.

found here, save it as a .csv file in a sensible directory.

Reference information for this data can be found here

See the next slide for further details...

PROJECT 1: EU WORK ACCIDENTS

1. Import the data into R
2. Remove the EU27 and EU28 columns
3. Reshape the data so that each country can be considered as an observation
4. Now reshape the data so that the units are variables
5. Use the `describeBy()` function from the psych package to summarise the data by age
   a. What conclusions can be made about the total injuries across the EU for the time period of this data?
   b. Which age group tends to have the most accidents at work (excluding UNK and TOTAL)?

PROJECT 1: EU WORK ACCIDENTS

1. Use the `describeBy()` to summarise the data by country
    a. Do you notice anything odd about the country means? Can you explain this?
    b. Which country appears to have the most accidents at work? Why might comparing this value against some other countries be an issue?
2. Extract the data for Poland (PL)
    a. Subset the data to only represent males
    b. What is the total number of accidents for Polish males aged 25 - 54?

PROJECT 2: LEEDS CAR ACCIDENTS 2015

The data for this project were published by Leeds City Council and can be found here

Have a look at the data and form your own hypotheses.

Before analysing the data, design an anlysis pipeline that will provide you with answers to your hypotheses.