Review of Basic R Objects and Tidyverse

Let's Get Started

Objects!!

- What are R objects?
- Why are they important?
- Can you name some object types?
    - vectors
    - data frames
    - lists
    - functions
    - operators
- How do you create an object (e.g., a vector)?
    - The assignment operator (`<-` or `=`) and the `c()` function 😃

```
x <- c(lots, of, cool, stuff)
```

R Objects

Vectors

What type of vector does each produce?

```
## Vectors
x = c(9,4,5,2,6)
y = c("male", "female", "female", "male", "female")
z = factor(y)
```

1. x: numeric
2. y: character
3. z: factor

R Objects

Data Frames and Lists

What is a `data.frame`? What about a `list`?

```
df = data.frame(
  x = c(9,4,5,2,6),
  y = c("male", "female",
        "female", "male",
        "female"),
  z = factor(y)
)
```

```
l1 = list(
  x = c(9,4,5,2,6),
  y = c("male", "female",
        "female", "male",
        "female"),
  z = factor(y)
)
```


A spreadsheet-like table and is the core data object that you will use.

It's much like a bag that you can put any object in (i.e., a flexible version of a `data.frame`).

Note: Data frames are a special list where all the elements are the same size.

Importing Data

Almost any type of data can be imported into R with little effort.

We will show:

- CSV
- tab delimited
- space delimited
- SPSS
- SAS export

Others are possible 😃

Importing Data

Base R

CSV

```
df <- read.csv("File.csv")
```

Tab Delimited

```
df <- read.table("File.txt",
                 header = TRUE,
                 sep = "\t")
```

Space Delimited

```
df <- read.table("File.txt",
                 header = TRUE,
                 sep = " ")
```

Others

CSV

```
library(readr)
df <- read_csv("File.csv")
```

SPSS

```
library(haven)
df <- read_spss("File.sav")
```

SAS

```
library(foreign)
df <- read.xport("File.xpt")
```

R Objects

Functions 💪

It's how we do stuff in R!

What are the pieces of a function (a named function)?

```
myfunction = function(arguments){
  stuff = that(the, function, does)
  stuff
}
```

1. a name
2. `function()`
3. arguments
4. stuff inside the `{}`
5. a return value (usually)

Functions 💪

- How do you create your own function?

  Write your own function that gives you the mean and range of each variable in a data frame.

  Now update that function so it only takes the mean and range of numeric variables.

- One way:

```
mean_range = function(df){
  means = lapply(df, mean)
  mins = lapply(df, min)
  maxs = lapply(df, max)
  final = data.frame("Mean" = unlist(means),
                     "Range" = unlist(maxs) - unlist(mins))
  final
}
```

Functions 💪

```
mean_range = function(df){
  means = lapply(df, mean)
  mins = lapply(df, min)
  maxs = lapply(df, max)
  final = data.frame("Mean" = unlist(means),
                     "Range" = unlist(maxs) - unlist(mins))
  final
}
```

- What does each piece do?
  - `lapply()`
  - `unlist()`
  - `final`
- What does `mean_range()` return?

Functions 💪

```
mean_range = function(df){
  means = lapply(df, mean)
  mins = lapply(df, min)
  maxs = lapply(df, max)
  final = data.frame("Mean" = unlist(means),
                     "Range" = unlist(maxs) - unlist(mins))
  final
}
```

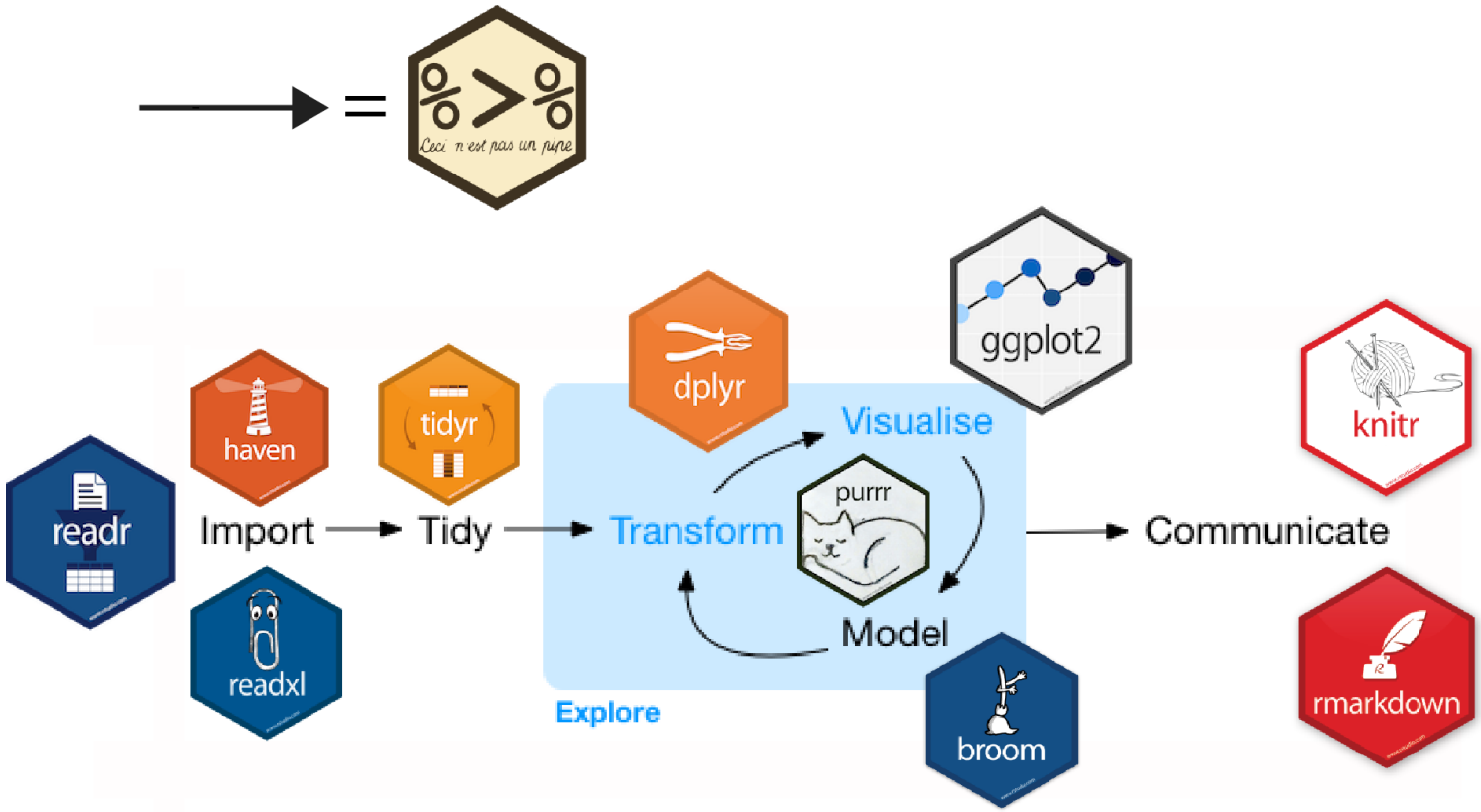
- What does each piece do?
  - `lapply()`
  - `unlist()`
  - `final`
- What does `mean_range()` return?

What is something you'd like R to do for you? (let's create a function for it)

Questions about R Objects?

Questions about R Objects?

Now onto Tidyverse



data.frames

factors

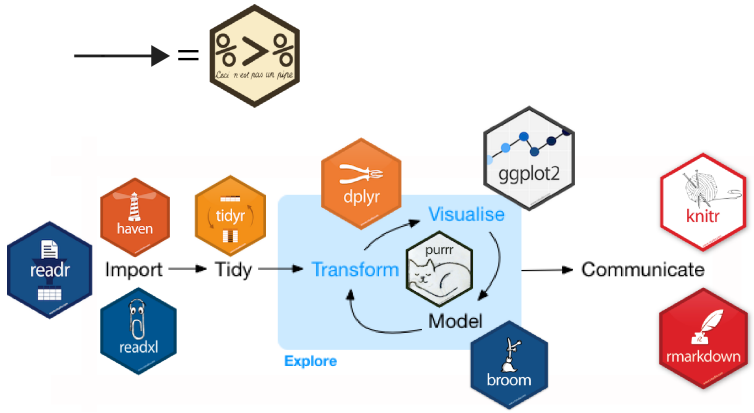
strings

Tidyverse 📦

A group of packages that make working with data intuitive and clean.

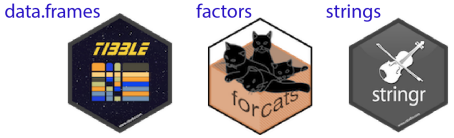
The steps of tidyverse:

1. import
2. tidy
3. transform
4. visualize
5. model
6. communicate



All these have the same overall goal:

- **to produce "tidy" data.**



Tidy Data

Tidy data:

1. column is a variable
2. row is observation

The tidy form depends on many features of the data set.

E.g.

- For longitudinal data, this is generally the long form.

ID	X	Y	Z
1	.	.	.
1	.	.	.
2	.	.	.
2	.	.	.
...	.	.	.

Tidy Data

We often need to reshape the data to get it into this form:

```
library(furniture)
df_long = long(df,
               c("var1", "var2")
               v.names = "var",
               idvar = "id")
```

`long()` takes a wide form to long form (which is many cases is the "tidy" form)

ID	X	Y	Z
1	.	.	.
1	.	.	.
2	.	.	.
2	.	.	.
...	.	.	.

Selecting, Filtering, Grouping, Summarizing

Base R

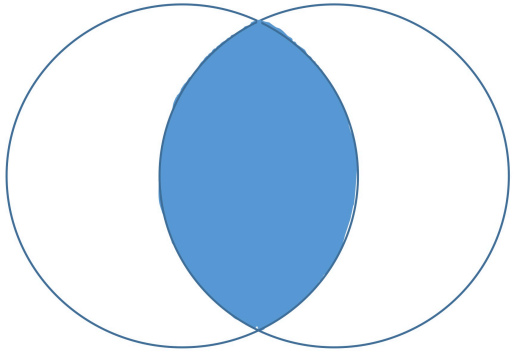
```r
## Selecting
d[, c("var1", "var2")]

## Filtering
d[d$var1 == 1, ]

## Summarizing
lapply(d, mean)

## Summarizing by Groups
tapply(d$var1, d$group, mean)
```

Tidyverse

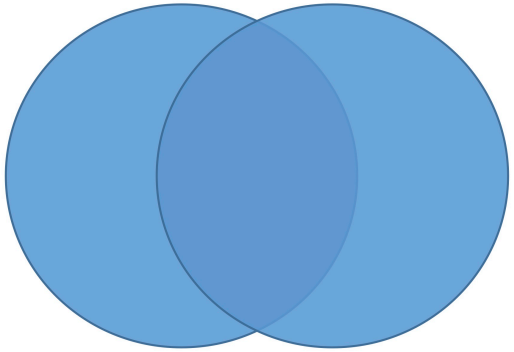
```r
## Selecting
select(d, var1, var2)

## Filtering
filter(d, var1 == 1)

## Summarizing
map(d, mean)

## Summarizing by Groups
d %>%
  group_by(group) %>%
  summarize(mean(var1))
```

Join

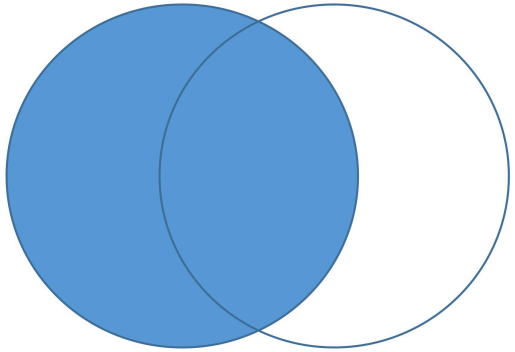
Inner Join



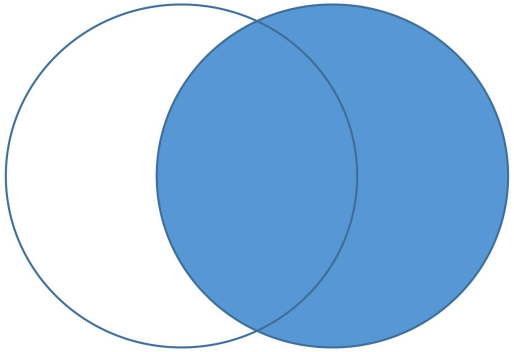
Full Join



Left Join



Right Join



Join

From the tidyverse, you can easily do each type of join.

```
joined = d1 %>%
  inner_join(d2, by = "ID")
```

```
joined = d1 %>%
  full_join(d2, by = "ID")
```

```
joined = d1 %>%
  left_join(d2, by = "ID")
```

```
joined = d1 %>%
  right_join(d2, by = "ID")
```

Can do several in-a-row:

```
joined = d1 %>%
  full_join(d2, by = "ID") %>%
  full_join(d3, by = "ID") %>%
  full_join(d4, by = "ID")
```

Long and Wide

Data Reshaping in R

Using the `furniture` package's `long()` and `wide()` functions

Tyson S. Barrett | t.barrett@aggiemail.usu.edu | tysonstanley.github.io

Changing data from wide to long format is an essential tool in most data analyses. Long ("tidy") form is generally needed for plotting, statistical and other analyses.

Wide Format

ID	x1	x2	z1	z2
a	1	8	.	.
b	2	9	.	.
c	3	10	.	.
d	4	11	.	.

Long Format

ID	Time	x	z
a	1	1	.
a	2	8	.
b	1	2	.
b	2	9	.
c	1	3	.
c	2	10	.

Wide to Long Form

df

ID	x1	x2	z1	z2
a	1	8	.	.
b	2	9	.	.
c	3	10	.	.
d	4	11	.	.

long_df

ID	Time	x	z
a	1	1	.
a	2	8	.
b	1	2	.
b	2	9	.
c	1	3	.
c	2	10	.

Newly created data.frame — Function name — Original data.frame — The time varying variables from original data.frame

```
long_df <- long(df,
                c("x1", "x2"), c("z1", "z2"),
                v.names = c("x", "z"),
                timevar = "Time")
```

`long()` means we are going from wide to long form

The name of the new "time" variable

The new variable names for each group of measured variables

Note: there can be any number of vectors like the c("x1", "x2") and c("z1", "z2") but they must have the same number of elements. For unbalanced data (where some have more "times" than others, see Unbalanced Measures to the right.

Long to Wide Form

df

ID	Time	x	z
a	1	1	.
a	2	8	.
b	1	2	.
b	2	9	.
c	1	3	.
c	2	10	.

wide_df

ID	x1	x2	z1	z2
a	1	8	.	.
b	2	9	.	.
c	3	10	.	.
d	4	11	.	.

Newly created data.frame — Function name — Original data.frame — The variables that will be split into the different times (e.g., "x1" and "x2" stem from "x")

```
wide_df <- wide(df,
                v.names = c("x", "z"),
                timevar = "Time",
                idvar = "ID")
```

The ID of the individual. The function automatically looks for a variable with the letters "ID" within it.

The name of the time variable in the data frame.

Unbalanced Measures

If the various measures have differing number of measurement points ("times"), then use the placeholder "miss". Say, our "x" measure had only one time point but our "z" measure had two, this code would work.

```
long(df,
     c("x1", "miss"), c("z1", "z2"),
     v.names = c("x", "z"),
     timevar = "time")
```

Additional Notes Regarding Data Reshaping:

- Recommendation: Save only one data set (in wide or long format) that is your "main" data set—to avoid confusion about what different data sets contain. Then, use these reshaping functions in R to manipulate the data to get it in the form needed to plot and analyze. Finally, save the code ("syntax") and any output (you can save the reshaped data but it is not necessary).
- Term Definitions: *measure* implies any distinct variable in the data set (e.g., x2), *time* implies anything that distinguishes the observations (e.g., time, cluster, location), *time varying variable* is any variable that has or could have a different value at each observation occasion.

Long

Using the wide data below:

```
df_wide = data.frame(
  "ID" = 1:10,
  "X1" = rnorm(10),
  "X2" = rnorm(10),
  "Y1" = runif(10),
  "Y2" = runif(10)
)
```

We are going to reshape this to long format.

```
library(furniture)
df_long = long(df_wide,
               c("X1", "X2"),
               c("Y1", "Y2"),
               v.names = c("X", "Y"))
```

```
## id = ID
```

Long

Search:

	ID	time	X	Y
1	1	1	0.1217	0.5385
2	1	2	-0.439	0.1527
3	2	1	0.8042	0.313
4	2	2	-0.1661	0.4858
5	3	1	0.4463	0.2052

Showing 1 to 5 of 20 entries

Previous 1 2 3 4 Next

Wide

Using the `df_long` we just created, let's reshape it back to wide.

```
df_wide = wide(df_long,
               v.names = c("X", "Y"),
               timevar = "time")
```

```
## id = ID
```

Search:

	ID	X.1	Y.1	X.2	Y.2
1	1	0.1217	0.5385	-0.439	0.1527
2	2	0.8042	0.313	-0.1661	0.4858
3	3	0.4463	0.2052	-0.7307	0.8848
4	4	0.229	0.008	1.1266	0.852
5	5	1.8668	0.9512	-1.5487	0.7041

Showing 1 to 5 of 10 entries

End of Review of Data Manipulation

Hopefully much of this was familiar.

Next we will review tables and `ggplot2`