Tibbles & List Columns & Nested Data Frames Oh My!

samp_id	meas1	meas2	data
<int>	*<dbl>*	*<dbl>*	*<list>*
1	32.2	89.4	<tibble [5x10]>
2	34.9	89.7	<tibble [3x10]>
3	33.7	86.3	<tibble [17x10]>
4	31.0	87.4	<tibble [14x10]>

Ryan Benz • OCRUG • Jan 2019 Meet-up

What's on your list of awesome things about R?

What's on your list of awesome things about R?

Data Visualization

What's on your list of awesome things about R?

Data Visualization

RStudio

What's on your list of awesome things about R?

What's on your list of awesome things about R?

Data Visualization

RStudio

The Community

What's on your list of awesome things about R?

Data Visualization

RStudio

The Community

tidyverse

What's on your list of awesome things about R?

Data Visualization

RStudio

The Community

tidyverse

Amazing Packages

How about the Data Frame?

The Data Frame Is a Truly Awesome Data Structure!

The Data Frame Is a Truly Awesome Data Structure!

Data frames organize the data we work with

The Infamous Iris Data Set

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

...

The Data Frame Is a Truly Awesome Data Structure!

Data frames organize the data we work with

columns ↓

The Infamous Iris Data Set

rows →

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

...

Rows and columns have meaning

The Data Frame Is a Truly Awesome Data Structure!

Data frames organize the data we work with

The Infamous Iris Data Set

column names →

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

...

Columns have names

The Data Frame Is a Truly Awesome Data Structure!

Data frames organize the data we work with

numbers

characters ...actually factors

The Infamous Iris Data Set

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

...

Can contain different types of data

The Data Frame Is a Truly Awesome Data Structure!

Data frames organize the data we work with

The Infamous Iris Data Set

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

...

Are a fundamental part of R

Under the Hood…
What Are Data Frames?

Under the Hood… What Are Data Frames?

A **named list** of **vectors**

Under the Hood…
What Are Data Frames?

A **named list** of **vectors**

the column names

Under the Hood…
What Are Data Frames?

A **named list** of **vectors**

- the column names
- holds the columns

Under the Hood… What Are Data Frames?

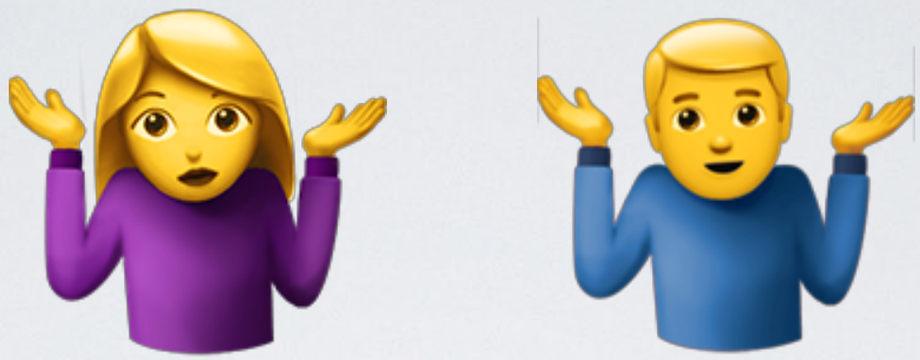
A **named list** of **vectors**

- the column names
- holds the columns
- can be atomic vectors or lists (same length)

Under the Hood…
What Are Data Frames?

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
>
> typeof(iris)
[1] "list"
>
> length(iris)
[1] 5
>
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  “Species”
>
> iris$Sepal.Length
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
 [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
```

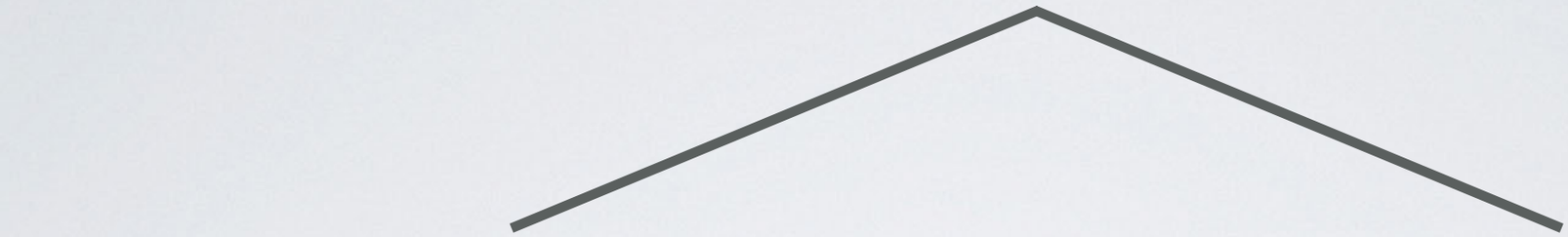
output truncated…



So what are these `tibble` things and why do we need them?

Data Frame

Data Frame



`data.frame`

`tibble`

Data Frame

`data.frame`

named list of vectors

`tibble`

Data Frame

`data.frame`

named list of vectors

`tibble`

named list of vectors

Data Frame

`data.frame`

named list of vectors

… but the data.frame machinery does a bunch of stuff behind the scenes

`tibble`

named list of vectors

Data Frame

`data.frame`

named list of vectors

… but the data.frame machinery does a bunch of stuff behind the scenes

stuff that can sometimes be really annoying

`tibble`

named list of vectors

Data Frame

`data.frame`

named list of vectors

… but the data.frame machinery does a bunch of stuff behind the scenes

stuff that can sometimes be really annoying

`tibble`

named list of vectors

tibbles *don't* do a bunch of stuff behind the scenes

Data Frame

`data.frame`

named list of vectors

… but the data.frame machinery does a bunch of stuff behind the scenes

stuff that can sometimes be really annoying

`tibble`

named list of vectors

tibbles *don't* do a bunch of stuff behind the scenes

makes them more consistent & easier to use

Data Frame

`data.frame`

named list of vectors

… but the data.frame machinery does a bunch of stuff behind the scenes

stuff that can sometimes be really annoying

`tibble`

named list of vectors

tibbles *don't* do a bunch of stuff behind the scenes

makes them more consistent & easier to use

Plus they have additional super powers! 🦸

Differences Between `data.frames` and `tibbles`

`data.frame`	`tibble`
Can change input types (e.g. strings to factors)	Never changes input types
Modifies column names to be valid R names	Leaves column names as is
Variables can be accessed by partially matched names	Variables must be accessed by their complete name
Printing can be overwhelming for big data sets	Uses more sensible printing
`[` sometimes returns a `data.frame`, sometimes a vector	`[` always return a `tibble`

and other things too (see resources at the end)

`tibble`s modify some old behavior of `data.frame`s to make them more useful for modern data analysis

i.e. the tidyverse

Super Power #1: List Columns

Recall: a data frame is a named list of vectors

a list is a vector too

So, we should be able to store lists as columns in data frames

Unfortunately, `data.frame`s make this a bit hard to do

`tibble`s make it easy!

Super Power #1: List Columns

But, why should I care about using lists as columns?

Lots of the data we typically deal with can be represented as *atomic vectors* (numbers, characters, etc)

The Infamous Iris Data Set

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

...

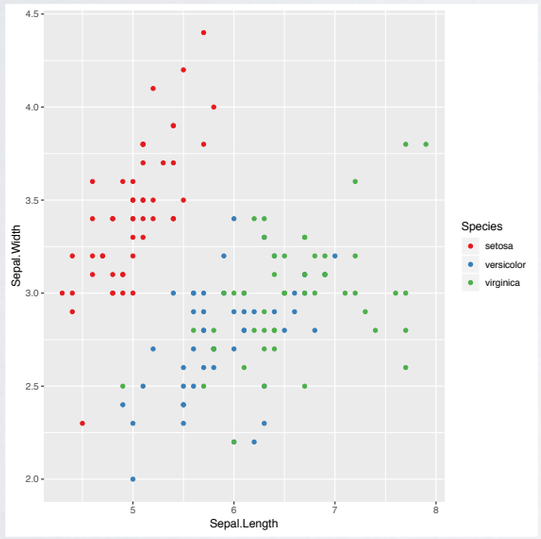
Super Power #1: List Columns

…but lots of (really important) data we deal
with can't be represented as atomic vectors

Super Power #1: List Columns

…but lots of (really important) data we deal
with can't be represented as atomic vectors

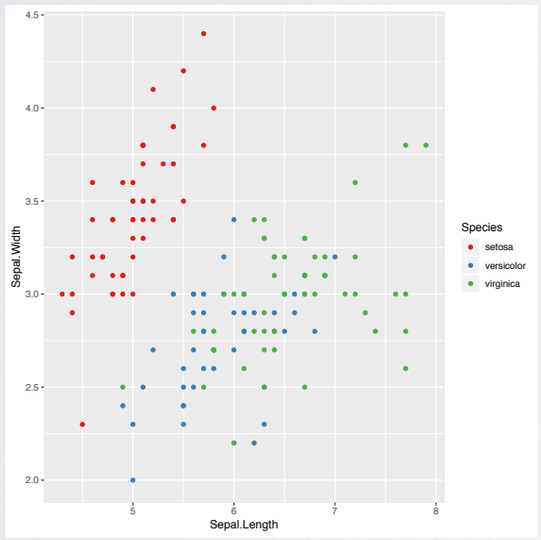
plots



Super Power #1: List Columns

…but lots of (really important) data we deal with can't be represented as atomic vectors

plots



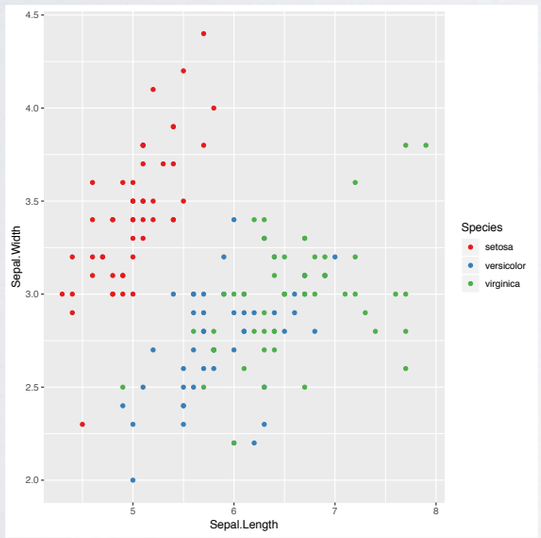
models

```
> lm(Sepal.Length ~ Sepal.Width, data = iris)

Call:
lm(formula = Sepal.Length ~ Sepal.Width, data = iris)

Coefficients:
(Intercept)  Sepal.Width
     6.5262      -0.2234
```

Super Power #1: List Columns

…but lots of (really important) data we deal with can't be represented as atomic vectors

plots



models

```
> lm(Sepal.Length ~ Sepal.Width, data = iris)

Call:
lm(formula = Sepal.Length ~ Sepal.Width, data = iris)

Coefficients:
(Intercept)  Sepal.Width
     6.5262      -0.2234
```

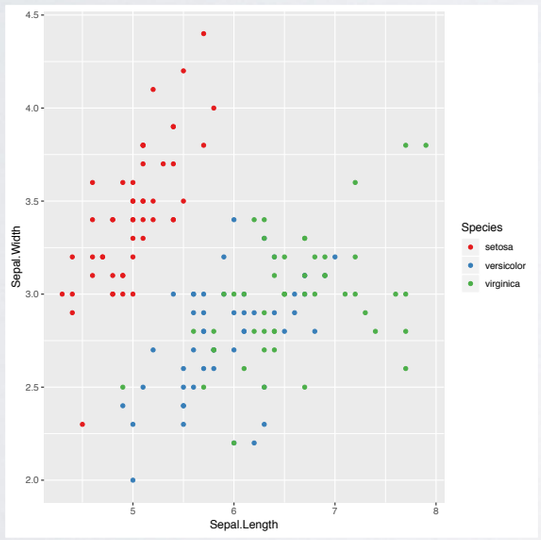
data frames

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

Super Power #1: List Columns

…but lots of (really important) data we deal with can't be represented as atomic vectors

plots



models

```
> lm(Sepal.Length ~ Sepal.Width, data = iris)

Call:
lm(formula = Sepal.Length ~ Sepal.Width, data = iris)

Coefficients:
(Intercept)  Sepal.Width
     6.5262      -0.2234
```

data frames

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

And all sorts of other complex objects from packages you use every day

Super Power #1: List Columns

`tibble`s allow you to easily add columns
that are lists of complex object

```
> my_favorites <- tibble(dataset_name = c("iris", "mtcars", "starwars"),
+                        data = list(iris, mtcars, starwars))
>
> my_favorites
# A tibble: 3 x 2
  dataset_name data
  <chr>        <list>
1 iris         <data.frame [150 × 5]>
2 mtcars       <data.frame [32 × 11]>
3 starwars     <tibble [87 × 13]>
>
> head(my_favorites$data[[2]])
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Super Power #2: Nested Data Frames

Strange as it might sound, sticking data frames inside of data frames can be really useful!

Allows you to use the organizational power of data frames combined with the power of data pipelines & functional programming

`tibble`s + `tidyr::nest` = easy nested data frames

Super Power #2: Nested Data Frames

```
> library(gapminder)
> library(tidyverse)
>
> gapminder
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
 6 Afghanistan Asia       1977    38.4 14880372      786.
 7 Afghanistan Asia       1982    39.9 12881816      978.
 8 Afghanistan Asia       1987    40.8 13867957      852.
 9 Afghanistan Asia       1992    41.7 16317921      649.
10 Afghanistan Asia       1997    41.8 22227415      635.
# ... with 1,694 more rows
>
> gapminder %>%
+   group_by(country, continent) %>%
+   nest()
# A tibble: 142 x 3
   country     continent data
   <fct>       <fct>     <list>
 1 Afghanistan Asia      <tibble [12 × 4]>
 2 Albania     Europe    <tibble [12 × 4]>
 3 Algeria     Africa    <tibble [12 × 4]>
 4 Angola      Africa    <tibble [12 × 4]>
 5 Argentina   Americas  <tibble [12 × 4]>
 6 Australia   Oceania   <tibble [12 × 4]>
 7 Austria     Europe    <tibble [12 × 4]>
 8 Bahrain     Asia      <tibble [12 × 4]>
 9 Bangladesh  Asia      <tibble [12 × 4]>
10 Belgium     Europe    <tibble [12 × 4]>
# ... with 132 more rows
>
```

using the gapminder data set as an example

here's what it looks like: data by country & continent

let's prepare a version where create data partitions by country & continent

presto! we've created nested data frames inside the main data set

Super Power #3: The Power of `purrr`

The `purrr` R package provides the ability to compute on vector elements; unlock super powers when combined with list cols & nested data frames

The set of map functions are fairly simple to use (after you get used to the syntax)

```
map_dbl(my_vector, some_function)
```

returns a vector of doubles

`c(1.2, 4.5, 6.9)`

`sqrt`

Computes the sq. root of each element

Super Power #3: The Power of `purrr`

This is a more interesting example!

using the gapminder data set again

nesting the data again by country & continent

computing a linear model on *the data in each row*

```
> gapminder %>%
+   group_by(country, continent) %>%
+   nest() %>%
+   mutate(model = map(data, function(d) {
+     lm(lifeExp ~ year, data = d)
+   }))
# A tibble: 142 x 4
   country     continent data              model
   <fct>       <fct>     <list>            <list>
 1 Afghanistan Asia      <tibble [12 × 4]> <S3: lm>
 2 Albania     Europe    <tibble [12 × 4]> <S3: lm>
 3 Algeria     Africa    <tibble [12 × 4]> <S3: lm>
 4 Angola      Africa    <tibble [12 × 4]> <S3: lm>
 5 Argentina   Americas  <tibble [12 × 4]> <S3: lm>
 6 Australia   Oceania   <tibble [12 × 4]> <S3: lm>
 7 Austria     Europe    <tibble [12 × 4]> <S3: lm>
 8 Bahrain     Asia      <tibble [12 × 4]> <S3: lm>
 9 Bangladesh  Asia      <tibble [12 × 4]> <S3: lm>
10 Belgium     Europe    <tibble [12 × 4]> <S3: lm>
# ... with 132 more rows
```

Demo Time

Resources

- Managing many models with R (H. Wickham)
  https://www.youtube.com/watch?v=rz3_FDVt9eg

- R for Data Science (G. Grolemund, H. Wickham)
  https://r4ds.had.co.nz
  - Chapter 10: Tibbles
  - Chapter 25: Many models (list columns, nested data frames, examples)

- Purrr Tutorial (J. Bryan)
  https://jennybc.github.io/purrr-tutorial/index.html