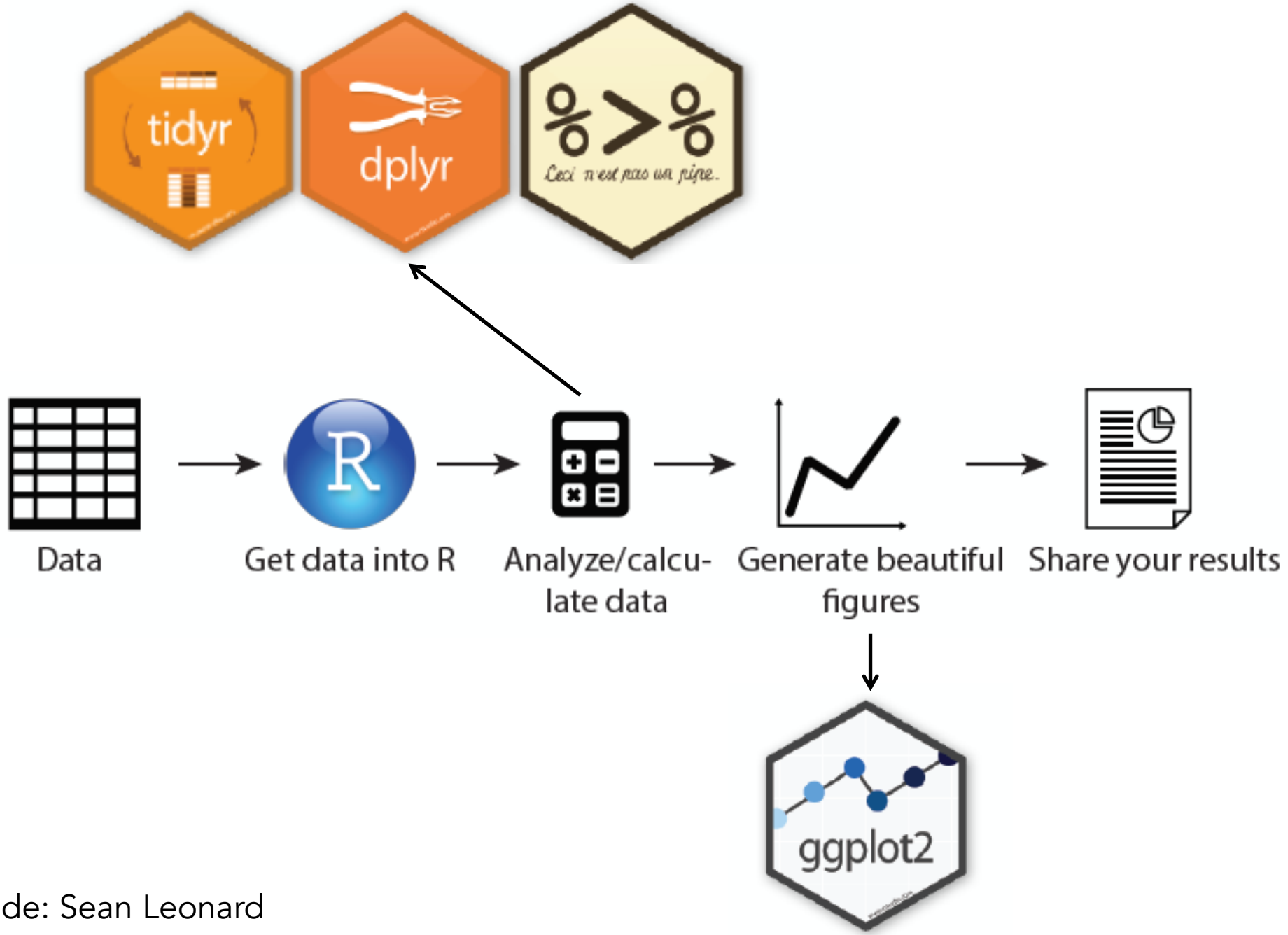
Working with data in R

E. Anne Chambers

September 22, 2017



Slide: Sean Leonard

Tidy data: contingency table

	survived	**died**
drug	15	3
placebo	4	12

treatment	**outcome**	**count**
drug	survived	15
drug	died	3
placebo	survived	4
placebo	died	12

http://wilkelab.org/classes/SDS348/2015_spring_slides/class5.pdf

1. Each variable forms a column

	survived	died
drug	15	3
placebo	4	12

treatment	outcome	count
drug	survived	15
drug	died	3
placebo	survived	4
placebo	died	12

http://wilkelab.org/classes/SDS348/2015_spring_slides/class5.pdf

2. Each observation forms a row

	survived	died
drug	15	3
placebo	4	12

treatment	outcome	count
drug	survived	15
drug	died	3
placebo	survived	4
placebo	died	12

http://wilkelab.org/classes/SDS348/2015_spring_slides/class5.pdf

3. Each value has a cell

	survived	**died**
drug	15	3
placebo	4	12

treatment	**outcome**	**count**
drug	survived	15
drug	died	3
placebo	survived	4
placebo	died	12

http://wilkelab.org/classes/SDS348/2015_spring_slides/class5.pdf

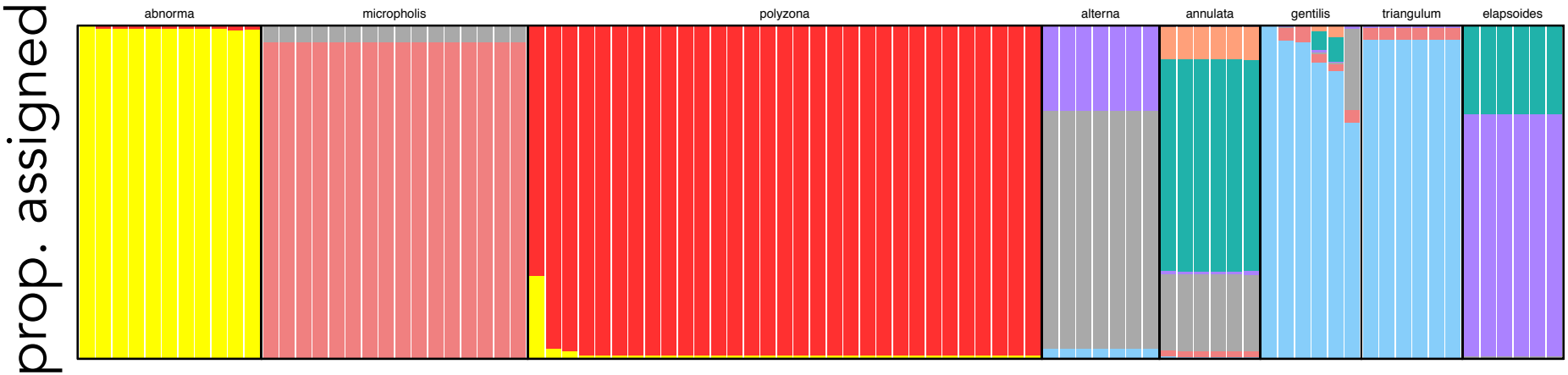
Tidy data

- Rules of tidy data:
  - Each variable must have its own column
  - Each observation must have its own row
  - Each value must have its own cell

Why are tidy data ideal?

- Let me give you a practical example...

Structurama



What do the raw data look like?

```
> View(output)
```

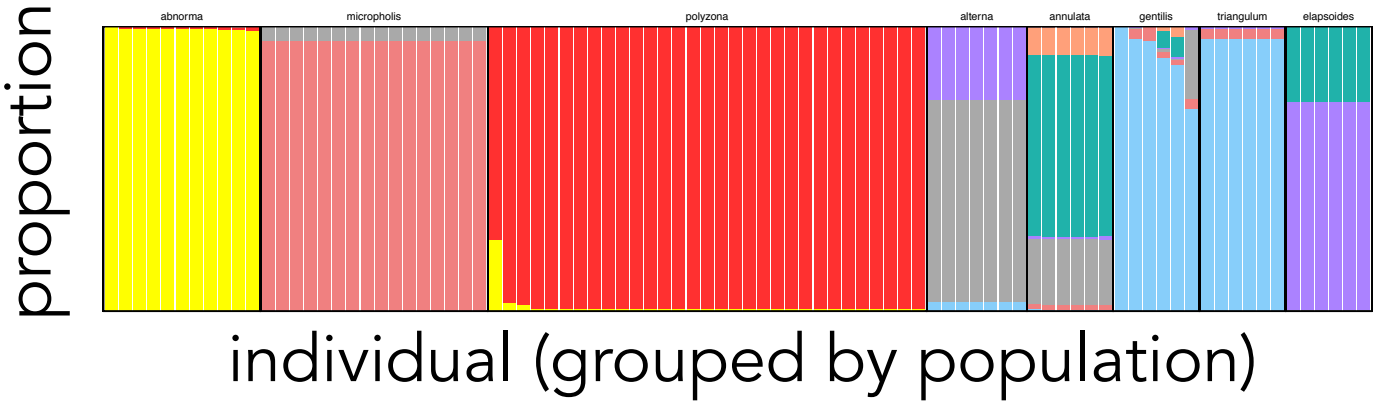
10 populations

97 individuals

100,000 generations ("Cycle")

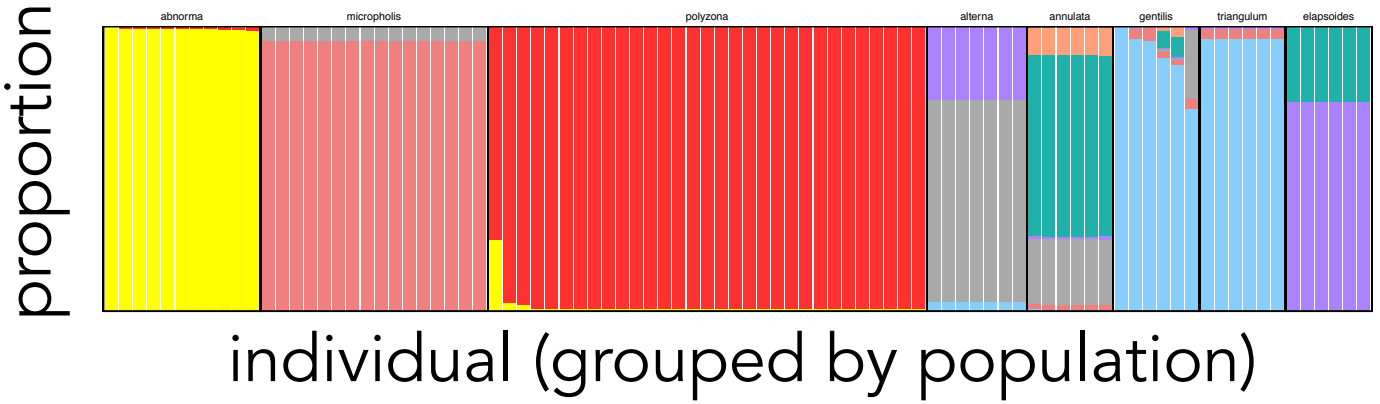
	Cycle	individual	1	2	3	4	5	6	7	8	9	10
1	100	I(1)	1	NA	NA	NA	NA	NA	NA	NA	NA	NA
2	100	I(10)	NA	NA	3	NA	NA	NA	NA	NA	NA	NA
3	100	I(100)	NA	NA	NA	4	NA	NA	NA	NA	NA	NA
4	100	I(101)	1	NA	NA	NA	NA	NA	NA	NA	NA	NA
5	100	I(102)	1	NA	NA	NA	NA	NA	NA	NA	NA	NA
6	100	I(103)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
7	100	I(104)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
8	100	I(105)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
9	100	I(106)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
10	100	I(107)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA

Recall ggplot from last week...




```
> ggplot(data,
```

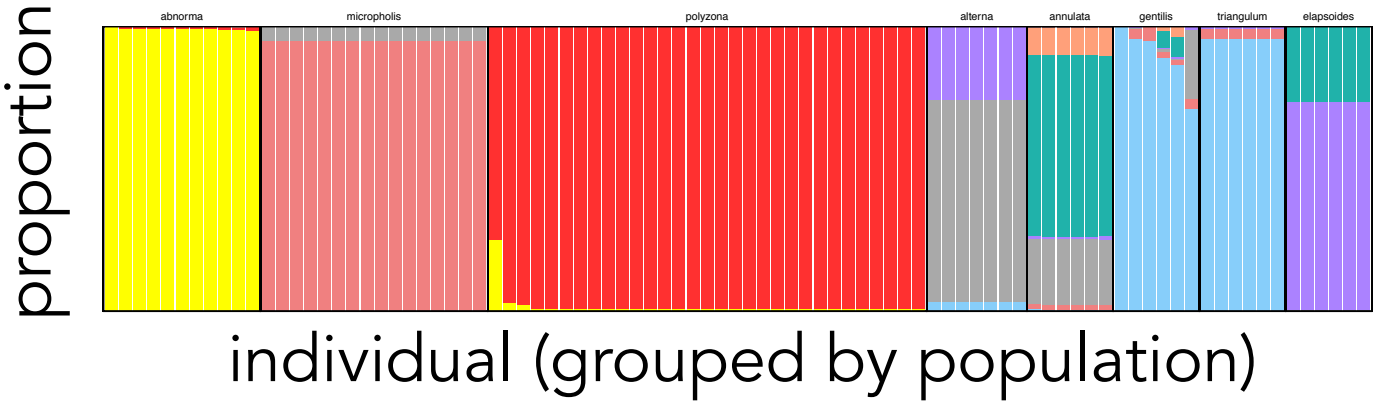
Recall ggplot from last week...




```
> ggplot(data, aes(x=individual,
```

individual
I(1)
I(1)
I(1)
I(1)
I(1)
I(1)
I(1)
I(1)
I(1)
I(1)

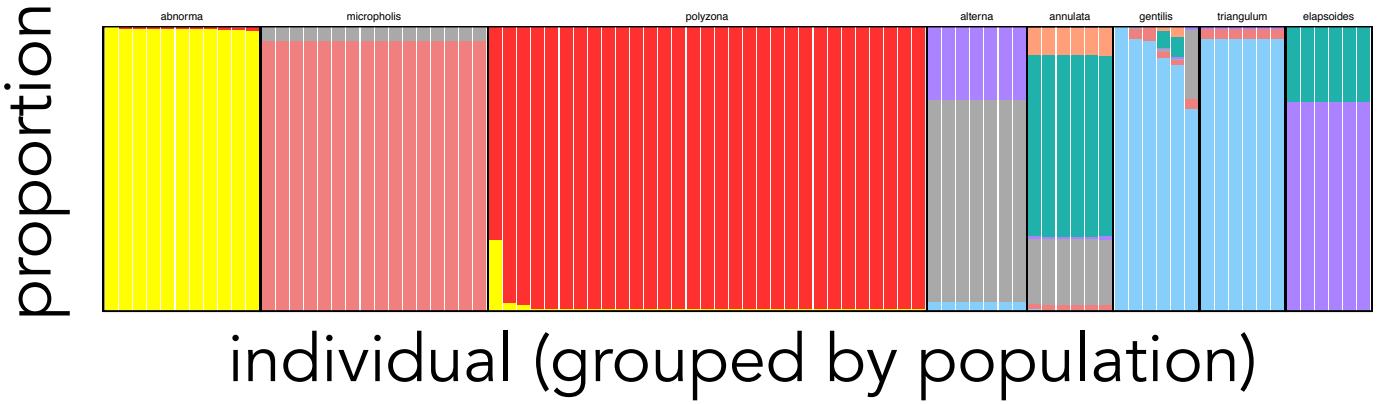
Recall ggplot from last week...




```
> ggplot(data, aes(x=individual, y=proportion,
```

	Cycle	individual
1	100	I(1)
2	200	I(1)
3	300	I(1)
4	400	I(1)
5	500	I(1)
6	600	I(1)
7	700	I(1)
8	800	I(1)
9	900	I(1)
10	1000	I(1)

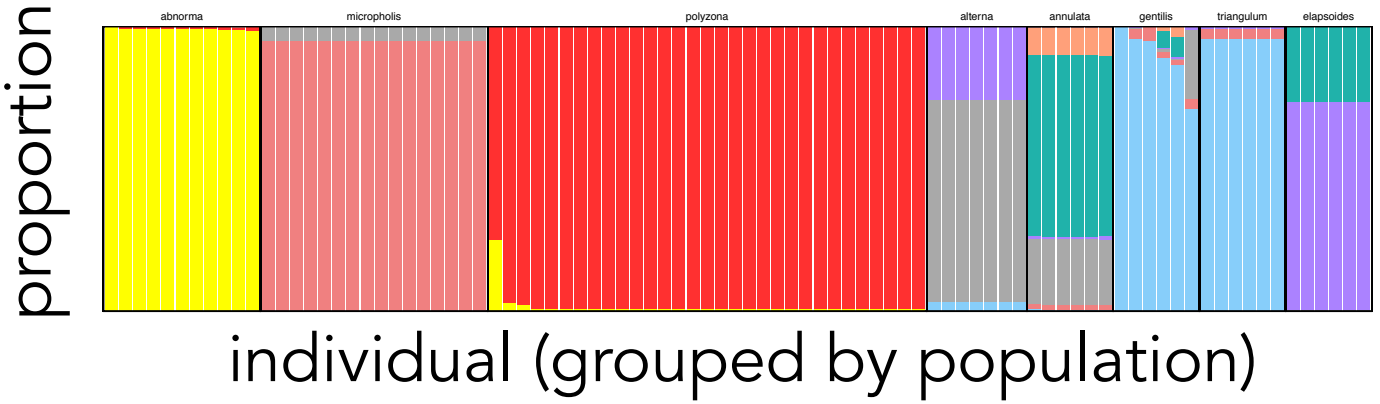
Recall ggplot from last week...




```
> ggplot(data, aes(x=individual, y=proportion, fill=population))
```

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

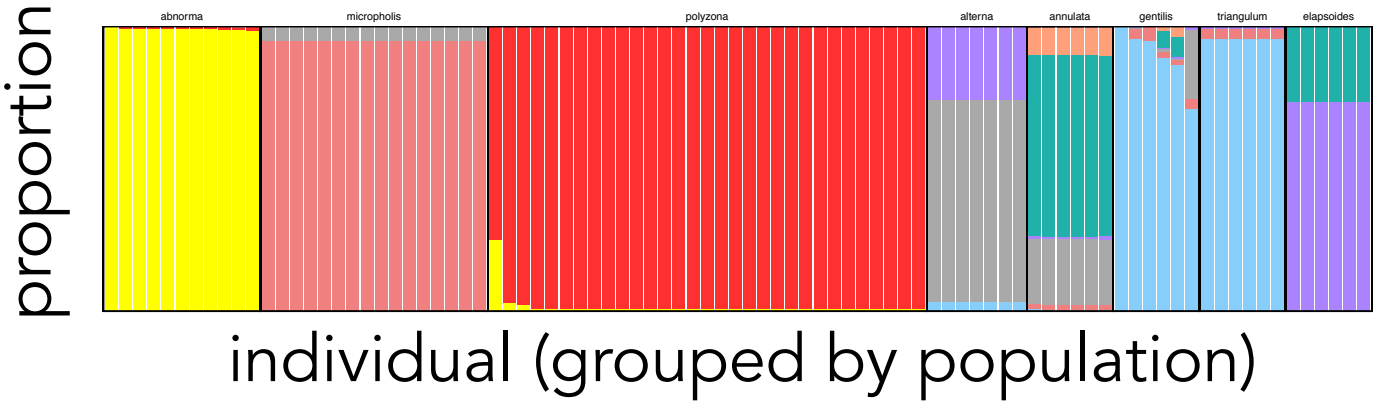
Recall ggplot from last week...




```
> ggplot(data, aes(x=individual, y=proportion, fill=population))
+   geom_bar()
```

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

Recall ggplot from last week...




```
> ggplot(data, aes(x=individual, y=proportion, fill=population))
+   geom_bar()
```

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

→

	individual	population	count	proportion
1	I(1)	1	7500	1.0000000000
2	I(10)	1	385	0.0513333333
3	I(10)	2	370	0.0493333333
4	I(10)	3	3531	0.4708000000
5	I(10)	4	3214	0.4285333333
6	I(100)	2	944	0.1258666667
7	I(100)	3	1012	0.1349333333
8	I(100)	4	3056	0.4074666667
9	I(100)	5	2448	0.3264000000
10	I(100)	6	40	0.0053333333

Before we go any further...

- Standard R:

```
> mean(dataframe$column)
> mean(iris$Sepal.Length)
```

The pipe operator!

%>%

"then"

- With pipe:

```
> iris$Sepal.Length %>% mean()
```

http://wilkelab.org/classes/SDS348/2015_spring_slides/class5.pdf

Before we go any further...

- Left and right assignment

->

<-

- Left assignment:

```
> x <- 5
> x
[1] 5
```

- Right assignment:

```
> 6 -> x
> x
[1] 6
```

http://wilkelab.org/classes/SDS348/2015_spring_slides/class5.pdf

Combining pipe and assignment

- These two lines do the same thing:

```
> iris$Sepal.Length %>% mean() -> mean.length
> mean.length <- iris$Sepal.Length %>% mean()

> mean.length
[1] 5.843333
```

http://wilkelab.org/classes/SDS348/2015_spring_slides/class5.pdf

Spreading and gathering

- `gather()`
  - some columns are values of a variable

	2008	2013	2016
Reptiles	8541	9556	10228

```
> gather(variable value, key, value)
> data %>%
gather(`2008`, `2013`, `2016`,
key=Year, value=Species)
```

Spreading and gathering

- `gather()`
  - some columns are values of a variable

	2008	2013	2016
Reptiles	8541	9556	10228

```
> gather(variable value, key, value)
> data %>%
gather(`2008`:`2016`,
key=Year, value=Species)
```

Spreading and gathering

- `gather()`
  - some columns are values of a variable

	2008	2013	2016
Reptiles	8541	9556	10228

```
> gather(variable value, key, value)
> data %>%
gather(2:4,
key=Year, value=Species)
```

```
> gather(variable value, key, value)
> data %>%
gather(`2008`, `2013`, `2016`,
key=Year, value=Species) -> newdata
> newdata
```

	Year	Species
Reptiles	2008	8541
Reptiles	2013	9556
Reptiles	2016	10228

Spreading and gathering

- `spread()`
  - observation scattered across multiple rows

Group	Type	Year	Species
Reptiles	Lizards	2008	5079
Reptiles	Snakes	2008	3149
Reptiles	Turtles	2008	313
Reptiles	Lizards	2016	6263
Reptiles	Snakes	2016	3619
Reptiles	Turtles	2016	346

```
> spread(key, value)
> data %>%
   spread(key=Type, value=Species)
```

```
> spread(key, value)
> data %>%
   spread(key=Type, value=Species) -> newdata
> newdata
```

Group	Year	Lizards	Snakes	Turtles
Reptiles	2008	5079	3149	313
Reptiles	2016	6263	3619	346

Exercise 1

	Cycle	individual	1	2	3	4	5	6	7	8	9	10
1	100	I(1)	1	NA	NA	NA	NA	NA	NA	NA	NA	NA
2	100	I(10)	NA	NA	3	NA	NA	NA	NA	NA	NA	NA
3	100	I(100)	NA	NA	NA	4	NA	NA	NA	NA	NA	NA
4	100	I(101)	1	NA	NA	NA	NA	NA	NA	NA	NA	NA
5	100	I(102)	1	NA	NA	NA	NA	NA	NA	NA	NA	NA
6	100	I(103)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
7	100	I(104)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
8	100	I(105)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
9	100	I(106)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA
10	100	I(107)	NA	NA	NA	NA	NA	6	NA	NA	NA	NA

output

```
> tidyout <- output %>%
  gather(individual, population, -Cycle)
```

tidyout

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

Working with dplyr

Fundamental actions on data frames:

- select rows – `filter()`
- select columns – `select()`
- make new columns – `mutate()`
- arrange rows – `arrange()`
- calculate summary stats – `summarize()`
- work on groups of data – `group_by()`

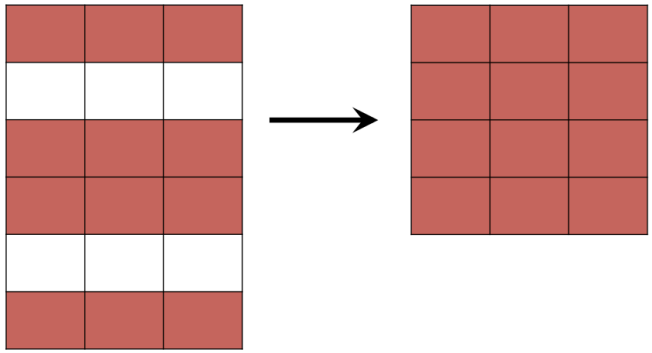
Comparison and logical operators

<	less than
<=	less or equal to
>	greater than
>=	greater or equal to
!=	does not equal
==	equal

&	and
|	or
!	not

- `filter()` rows

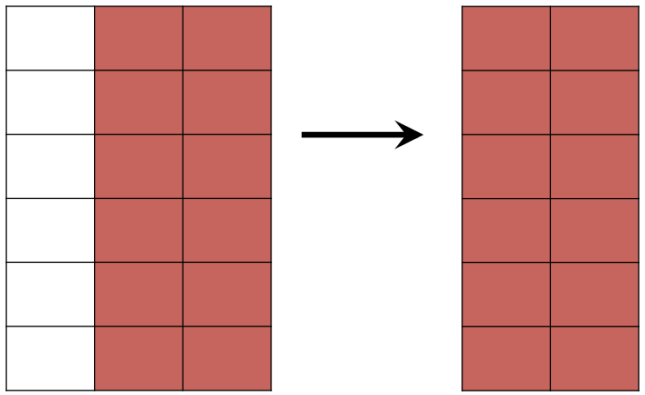
```
> tidyout %>% filter(Cycle==100 | Cycle==200)
```



	Cycle	individual	population
	<int>	<chr>	<int>
1	100	I(1)	1
2	200	I(1)	1
3	100	I(2)	1
4	200	I(2)	2
5	100	I(3)	2
6	200	I(3)	3
7	100	I(4)	2
8	200	I(4)	3
9	100	I(5)	2
10	200	I(5)	3

- `select()` columns

```
> tidyout %>% select(individual:population)
```



	individual	population
	<chr>	<int>
1	I(1)	1
2	I(1)	1
3	I(1)	1
4	I(1)	1
5	I(1)	1
6	I(1)	1
7	I(1)	1
8	I(1)	1
9	I(1)	1
10	I(1)	1

... with 1,239,990 more rows

`mutate()` to add columns

```
> dataframe %>% mutate(newcolname=fxn)
```

- Say I want to add a column with fraction of run that's finished...

```
> tidyout %>% mutate(prop=Cycle/100000)
```

```
   Cycle individual population  prop
   <int>      <chr>      <int> <dbl>
1    100       I(1)          1 0.001
2    200       I(1)          1 0.002
3    300       I(1)          1 0.003
4    400       I(1)          1 0.004
5    500       I(1)          1 0.005
6    600       I(1)          1 0.006
7    700       I(1)          1 0.007
8    800       I(1)          1 0.008
9    900       I(1)          1 0.009
10  1000       I(1)          1 0.010
# ... with 1,239,990 more rows
```

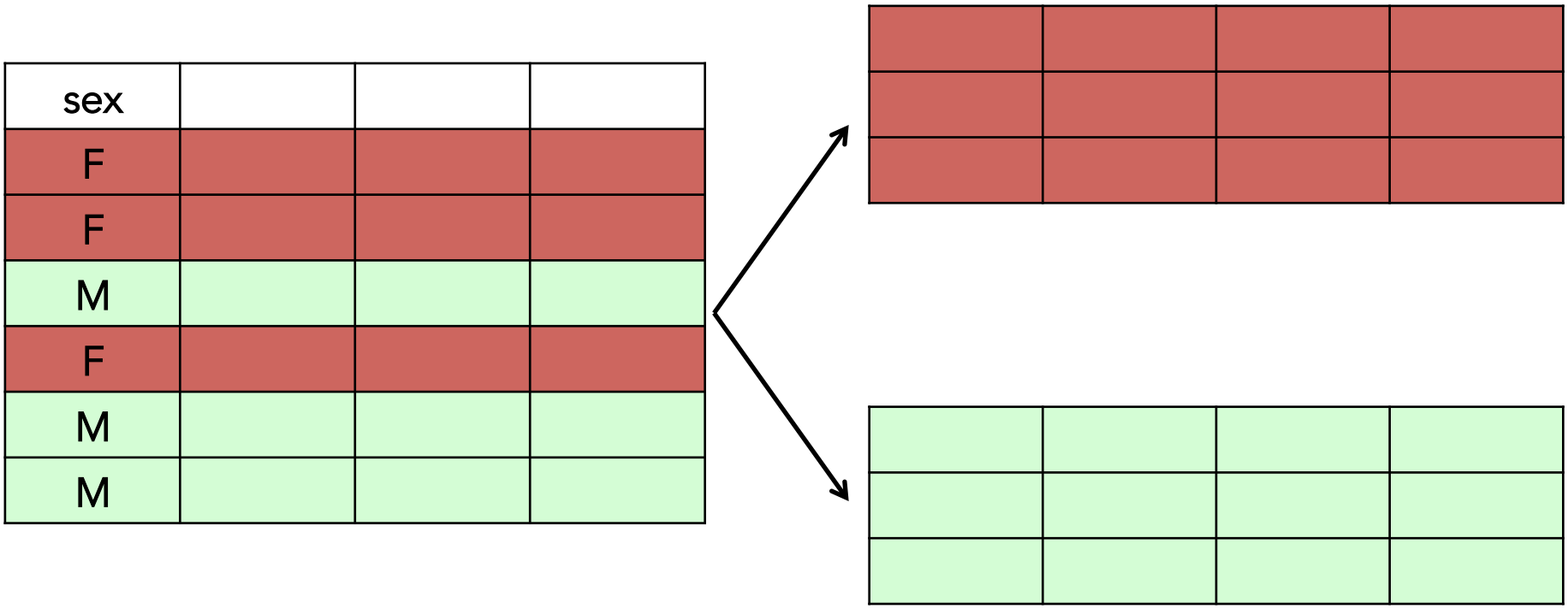
`group_by()` working on groups

```
> dataframe %>% group_by(varyouwantgrouped)
```

sex			
F			
F			
M			
F			
M			
M			

`group_by()` working on groups

```
> dataframe %>% group_by(varyouwantgrouped)
```



`group_by()` working on groups

- Say I want to only consider results from each individual at a time (not meaningful otherwise)

```
> tidyout %>% group_by(individual) %>% ...
```

Proportion of # generations (Cycle) placed each individual into each population

tidyout

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

Proportion of # generations (Cycle) placed each individual into each population

```
> tidyout %>%
   group_by(individual, population)
```

tidyout

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

Proportion of # generations (Cycle) placed each individual into each population

```
> tidyout %>%
   group_by(individual, population) %>%
   summarize(count=n())
```

tidyout

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

Proportion of # generations (Cycle) placed each individual into each population

```
> tidyout %>%
   group_by(individual, population) %>%
   summarize(count=n()) %>%
   mutate(proportion = count/100000)
   -> finalout
```

tidyout

	Cycle	individual	population
1	100	I(1)	1
2	200	I(1)	1
3	300	I(1)	1
4	400	I(1)	1
5	500	I(1)	1
6	600	I(1)	1
7	700	I(1)	1
8	800	I(1)	1
9	900	I(1)	1
10	1000	I(1)	1

Proportion of # generations (Cycle) placed each individual into each population

```
> tidyout %>%
   group_by(individual, population) %>%
   summarize(count=n()) %>%
   mutate(proportion = count/100000)
   -> finalout
```

```
> finalout
```

	individual	population	count	proportion
	<chr>	<int>	<int>	<dbl>
1	I(1)	1	10000	1.33333333
2	I(10)	1	492	0.06560000
3	I(10)	2	524	0.06986667
4	I(10)	3	4678	0.62373333
5	I(10)	4	4306	0.57413333
6	I(100)	2	1239	0.16520000
7	I(100)	3	1330	0.17733333
8	I(100)	4	4083	0.54440000
9	I(100)	5	3297	0.43960000
10	I(100)	6	51	0.00680000

... with 582 more rows

Useful summary functions

- `mean()` – mean of values
- `sum()` – sum values
- `median()` – median
- `sd()` – standard deviation
- `var()` – variance
- `cor()` – correlation

Exercise 2