

# Tidy-Tou[R]

Silas Tittes

03 May, 2017

```
library(tidyverse)
```

# Outline

- ▶ General motivation

# Outline

- ▶ General motivation
- ▶ tidyr

# Outline

- ▶ General motivation
- ▶ tidyr
- ▶ dplyr

# Outline

- ▶ General motivation
- ▶ tidyr
- ▶ dplyr
- ▶ ggplot2

# Outline

- ▶ General motivation
- ▶ tidyr
- ▶ dplyr
- ▶ ggplot2
- ▶ purrr

## General motivation: Make reproducibility easy

- ▶ Transparency is an obligation



# General motivation: Make reproducibility easy

- ▶ Transparency is an obligation
- ▶ Be lazy as possible

# General motivation: Make reproducibility easy

- ▶ Transparency is an obligation
- ▶ Be lazy as possible
- ▶ Do as little “by hand” manipulation of data as possible

# General motivation: Make reproducibility easy

- ▶ Transparency is an obligation
- ▶ Be lazy as possible
- ▶ Do as little “by hand” manipulation of data as possible
- ▶ Make your code fast and readable

## Quick note to the experienced R users new to the tidyverse

- ▶ You might find yourself saying, “That’s cool and all, but I prefer . . .”

## Quick note to the experienced R users new to the tidyverse

- ▶ You might find yourself saying, “That’s cool and all, but I prefer . . . ”
- ▶ Incorporate into your repertoire when you desire

## Quick note to the experienced R users new to the tidyverse

- ▶ You might find yourself saying, “That’s cool and all, but I prefer . . . ”
- ▶ Incorporate into your repertoire when you desire
- ▶ 10 ways to do everything (Isn’t R awesome?)

# Quick note to the experienced R users new to the tidyverse

- ▶ You might find yourself saying, “That’s cool and all, but I prefer . . . .”
- ▶ Incorporate into your repertoire when you desire
- ▶ 10 ways to do everything (Isn’t R awesome?)
- ▶ There’s strength in continuity and consistency

## Quick note to the experienced R users new to the tidyverse

- ▶ You might find yourself saying, “That’s cool and all, but I prefer . . . .”
- ▶ Incorporate into your repertoire when you desire
- ▶ 10 ways to do everything (Isn’t R awesome?)
- ▶ There’s strength in continuity and consistency
- ▶ Hadley (usually) knows best



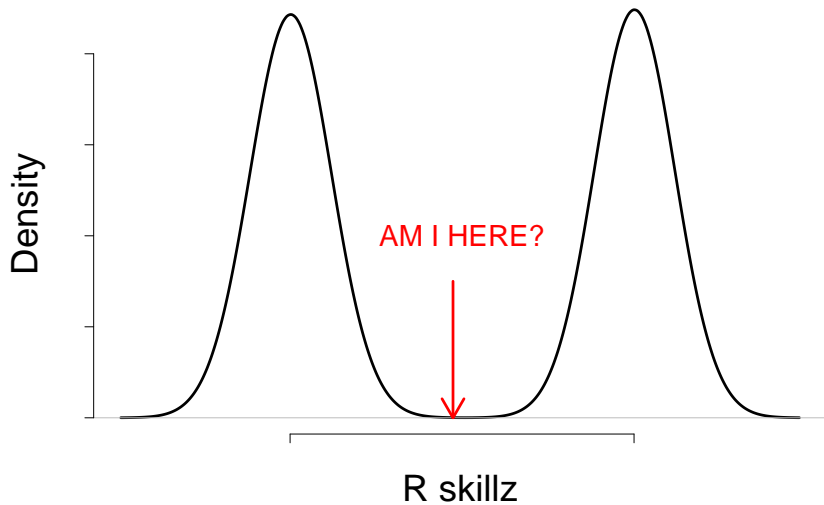
# Quick note to the experienced R users new to the tidyverse

- ▶ You might find yourself saying, “That’s cool and all, but I prefer . . . .”
- ▶ Incorporate into your repertoire when you desire
- ▶ 10 ways to do everything (Isn’t R awesome?)
- ▶ There’s strength in continuity and consistency
- ▶ Hadley (usually) knows best
- ▶ Who?

## Free book: Google “R for data science”



# Who is my audience today?



# The curse of data, data1, data2 ...

```
data <- read.csv("datafile")  
data1 <- data[,c(1,2,3)]  
data2 <- subset(data1, column1 == "test")  
data3 <- ...
```



**Life is better with %>% (pipe)**

## Simple example with %>%

```
iris[,3:5] %>%  
  head(5)
```

##	Petal.Length	Petal.Width	Species
## 1	1.4	0.2	setosa
## 2	1.4	0.2	setosa
## 3	1.3	0.2	setosa
## 4	1.5	0.2	setosa
## 5	1.4	0.2	setosa

# Emphasis on verbs instead of nouns

```
subject %>% (then)
  action1 %>% (then)
  action2 %>% (then)
  action3
```

## Exercise

Get the 5th to last row of the `iris` dataframe using `%>%` twice (there are much better ways to achieve this – just for practice)



## My solution

```
iris[,3:5] %>%  
  tail(5) %>%  
  head(1)
```

```
##      Petal.Length Petal.Width  Species  
## 146           5.2           2.3 virginica
```

# Using %>%

Benefits of approach include:

*Readability*

*Scalability*

*Consistency*

*Ease of use*

**tidyr**

# Make some example data

```
samps <- 5  
ducks <- data.frame(duck = rnorm(samps),  
                     goose = c(rnorm(samps - 1), NA),  
                     idx = 1:samps)
```

## make data loooong

	duck	goose	idx
1	1.33	-1.26	1
2	-0.17	-0.62	2
3	-0.14	0.14	3
4	0.31	1.41	4
5	-0.60		5

## make data loooong

- ▶ What's the problem?

	duck	goose	idx
1	1.33	-1.26	1
2	-0.17	-0.62	2
3	-0.14	0.14	3
4	0.31	1.41	4
5	-0.60		5

## make data loooong

- ▶ How do we fix it?

	duck	goose	idx
1	1.33	-1.26	1
2	-0.17	-0.62	2
3	-0.14	0.14	3
4	0.31	1.41	4
5	-0.60		5

## tidyr::gather() makes data frames long

```
ducks_td <- ducks %>%  
  gather(key = "bird_type", value = "temp", -idx) %>%  
  drop_na()
```

	idx	bird_type	temp
1	1	duck	1.33
2	2	duck	-0.17
3	3	duck	-0.14
4	4	duck	0.31
5	5	duck	-0.60
6	1	goose	-1.26
7	2	goose	-0.62
8	3	goose	0.14
9	4	goose	1.41



## tidyr::spread() makes data wide (sometimes useful)

```
ducks_wd <- ducks_td %>%  
  spread(bird_type, temp)
```

	idx	duck	goose
1	1	1.33	-1.26
2	2	-0.17	-0.62
3	3	-0.14	0.14
4	4	0.31	1.41
5	5	-0.60	

# Exercise

- ▶ Check out the `VADeaths` data matrix (comes with base R)

# Exercise

- ▶ Check out the `VADeaths` data matrix (comes with base R)
- ▶ Make `VADeaths` a data frame, *then* make it long

# One solution

```
data.frame(VADeaths) %>%  
  gather(key = "municipal.sex", value = "deaths_K") %>%  
  head(5)
```

```
##   municipal.sex deaths_K  
## 1   Rural.Male      11.7  
## 2   Rural.Male      18.1  
## 3   Rural.Male      26.9  
## 4   Rural.Male      41.0  
## 5   Rural.Male      66.0
```

## tidyr::separate() to split a single variable into multiple

```
data.frame(VADeaths) %>%  
  gather(key = "municipal.sex", value = "deaths_K") %>%  
  separate(col = municipal.sex,  
           into = c("munic", "sx"),  
           sep = "\\\\.") %>%  
  head(5)
```

```
##   munic   sx deaths_K  
## 1 Rural Male    11.7  
## 2 Rural Male    18.1  
## 3 Rural Male    26.9  
## 4 Rural Male    41.0  
## 5 Rural Male    66.0
```

**dplyr: a glorious catch-all tool**

- ▶ dplyr can do more than what I could show in 50 minutes

- ▶ dplyr can do more than what I could show in 50 minutes
- ▶ I still have a lot to learn about the functions



- ▶ dplyr can do more than what I could show in 50 minutes
- ▶ I still have a lot to learn about the functions
- ▶ Let's talk about a few

## dplyr::select() for accessing columns

```
#by index, name, or mixture of two  
iris %>%  
  select(1, Petal.Length) %>%  
  head(5)
```

```
##   Sepal.Length Petal.Length  
## 1           5.1           1.4  
## 2           4.9           1.4  
## 3           4.7           1.3  
## 4           4.6           1.5  
## 5           5.0           1.4
```

## dply::arrange() for ordering rows

```
#desc() for descending order  
iris %>%  
  arrange(Species, Sepal.Width, desc(Sepal.Length)) %>%  
  select(Sepal.Width, Sepal.Length, Species) %>%  
  head(5)
```

##	Sepal.Width	Sepal.Length	Species
## 1	2.3	4.5	setosa
## 2	2.9	4.4	setosa
## 3	3.0	5.0	setosa
## 4	3.0	4.9	setosa
## 5	3.0	4.8	setosa

## dply::filter() is like base r's subset()

```
#notice all columns would be returned w/o pipe to select()  
iris %>%  
  filter(Species == "setosa", Sepal.Length < 6) %>%  
  select(Sepal.Length, Species) %>%  
  head(5)
```

```
##   Sepal.Length Species  
## 1           5.1  setosa  
## 2           4.9  setosa  
## 3           4.7  setosa  
## 4           4.6  setosa  
## 5           5.0  setosa
```

## dply::mutate() add new columns

```
iris %>%  
  mutate(Sepal.Area = Sepal.Length * Sepal.Width) %>%  
  select(Sepal.Area, Species) %>%  
  head(5)
```

##	Sepal.Area	Species
## 1	17.85	setosa
## 2	14.70	setosa
## 3	15.04	setosa
## 4	14.26	setosa
## 5	18.00	setosa

## `dplyr::group_by()` and `dplyr::summarise()`

*#note: group\_by() can be used with multiple variables*  
`iris %>%`

```
  group_by(Species) %>%  
  summarise(mean_SL = mean(Sepal.Length),  
            sd_SL = sd(Sepal.Length),  
            q25 = quantile(Sepal.Length, 0.25),  
            q75 = quantile(Sepal.Length, 0.75))
```

## # A tibble: 3 × 5

##	Species	mean_SL	sd_SL	q25	q75
##	<fctr>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	setosa	5.006	0.3524897	4.800	5.2
## 2	versicolor	5.936	0.5161711	5.600	6.3
## 3	virginica	6.588	0.6358796	6.225	6.9

## dplyr::sample\_n()

*#also see new modelr package for more like this*

```
iris %>%  
  sample_n(2) %>%  
  select(Petal.Length, Species)
```

```
##      Petal.Length      Species  
## 38             1.4      setosa  
## 65             3.6 versicolor
```

## dplyr::sample\_frac()

```
iris %>%  
  sample_frac(0.02) %>%  
  select(Petal.Length, Species)
```

```
##      Petal.Length  Species  
## 40             1.5    setosa  
## 106             6.6 virginica  
## 103             5.9 virginica
```



## group\_by() and n()

*#surprisingly hard in base R*

```
iris %>%  
  group_by(Species) %>%  
  mutate(counts = n()) %>%  
  select(Species, counts) %>%  
  sample_n(2)
```

```
## Source: local data frame [6 x 2]
```

```
## Groups: Species [3]
```

```
##
```

```
##      Species counts
```

```
##      <fctr>  <int>
```

```
## 1      setosa      50
```

```
## 2      setosa      50
```

```
## 3 versicolor      50
```

```
## 4 versicolor      50
```

```
## 5 virginica       50
```

## Exercise

Using the tidy version of VADeaths, generate a mean and variance for the number of deaths for each municipal-sex combination (group by municipality and sex, *then* computer the mean and standard deviation of deaths).

# One solution

```
death_sum <- data.frame(VADeaths) %>%  
  gather(key = "municipal.sex", value = "deaths_K") %>%  
  separate(col = municipal.sex,  
           into = c("munic", "sx"), sep = "\\\\.") %>%  
  group_by(munic, sx) %>%  
  summarise(mean_death = mean(deaths_K),  
            sd_death = sd(deaths_K))
```

# Result

	munic	sx	mean_death	sd_death
1	Rural	Female	25.18	18.42
2	Rural	Male	32.74	21.60
3	Urban	Female	25.28	17.06
4	Urban	Male	40.48	22.58

**ggplot2**

# ggplot2 template

*#default:*

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

*#alternative 1:*

```
<DATA> %>%  
  ggplot( ) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

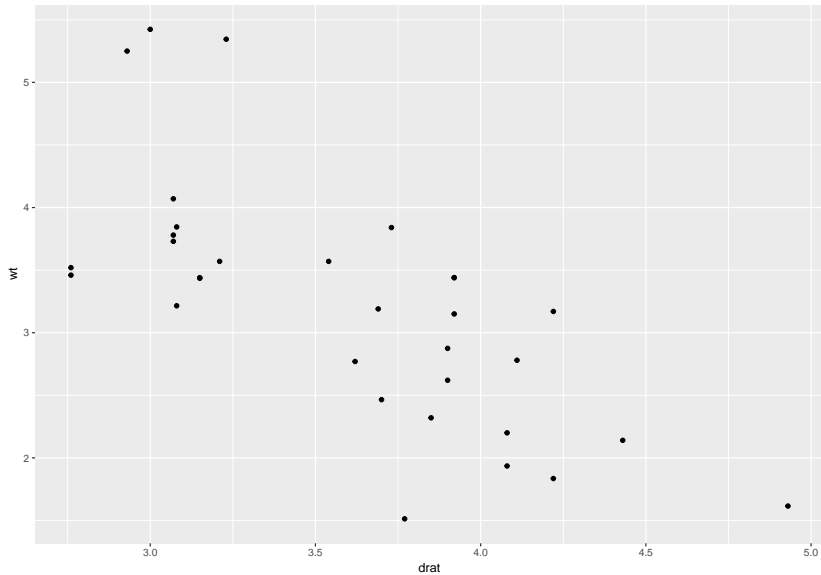
*#alternative 2 (my usual preference):*

```
<DATA> %>%  
  ggplot(mapping = aes(<MAPPINGS>)) +  
  <GEOM_FUNCTION>()
```

## simple example

```
p <- mtcars %>%  
  ggplot(aes(x = drat, y = wt)) +  
  geom_point()
```

# simple example 1

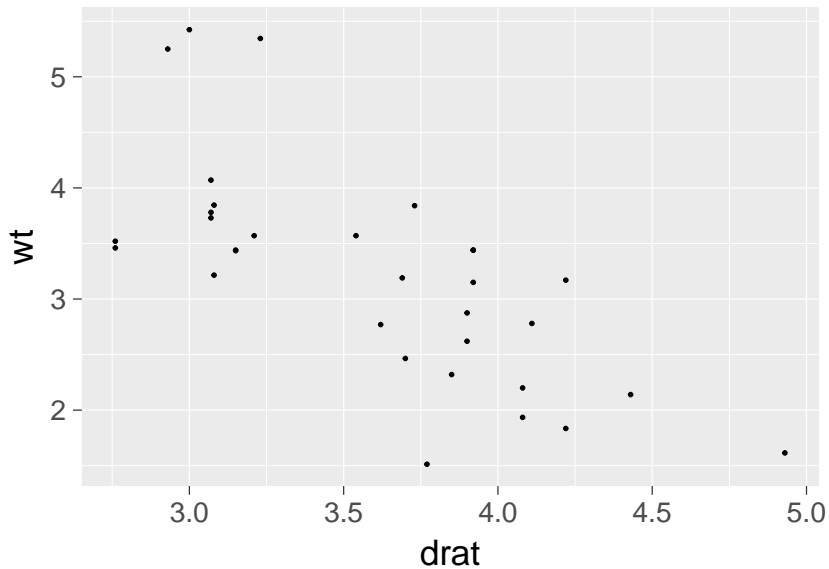




## Looks bad on my screen, but easy to fix

```
p <- mtcars %>%  
  ggplot(aes(x = drat, y = wt)) +  
  geom_point() +  
  theme_gray(base_size = 30)
```

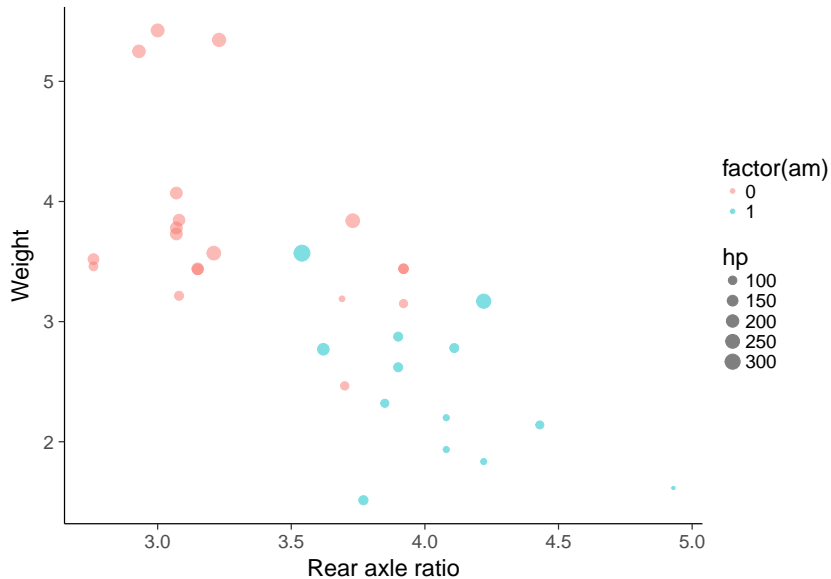
## simple example 1



## simple example 2

```
p <- mtcars %>%  
  ggplot(aes(x = drat, y = wt, size = hp)) +  
  geom_point(aes(colour = factor(am)), alpha = 0.5) +  
  xlab("Rear axle ratio") +  
  ylab("Weight") +  
  theme_classic(base_size = 20)
```

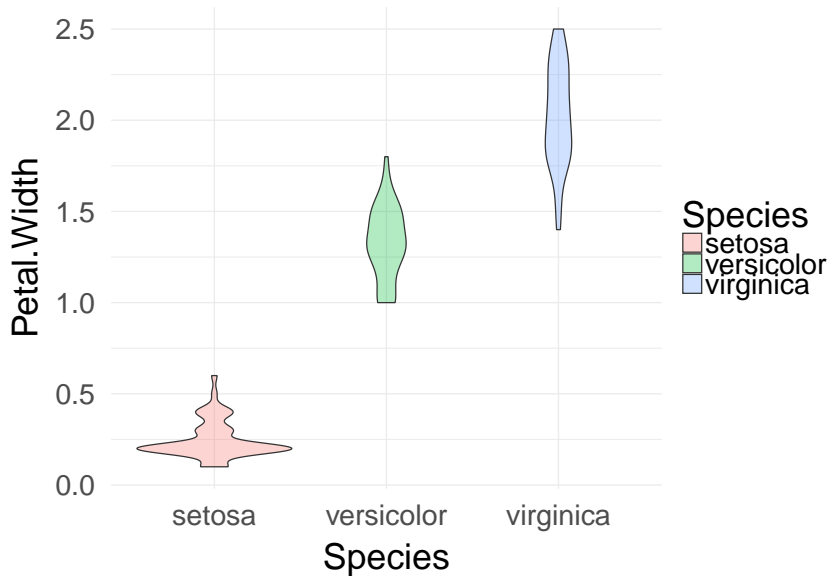
## simple example 2



## violin plots are cool!

```
p <- iris %>%  
  ggplot(aes(x = Species,  
             y = Petal.Width,  
             fill = Species)) +  
  geom_violin(alpha = 0.3) +  
  theme_minimal(base_size = 30)
```

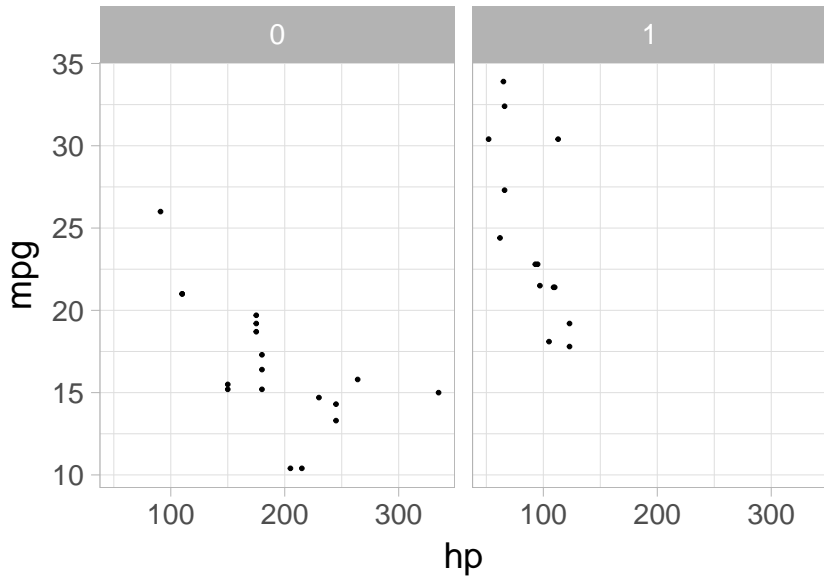
# Result



# Plots with multiple panels

```
p <- mtcars %>%  
  ggplot(aes(x = hp, y = mpg)) +  
  geom_point() +  
  facet_wrap(~ vs) +  
  theme_light(base_size = 30)
```

# Result

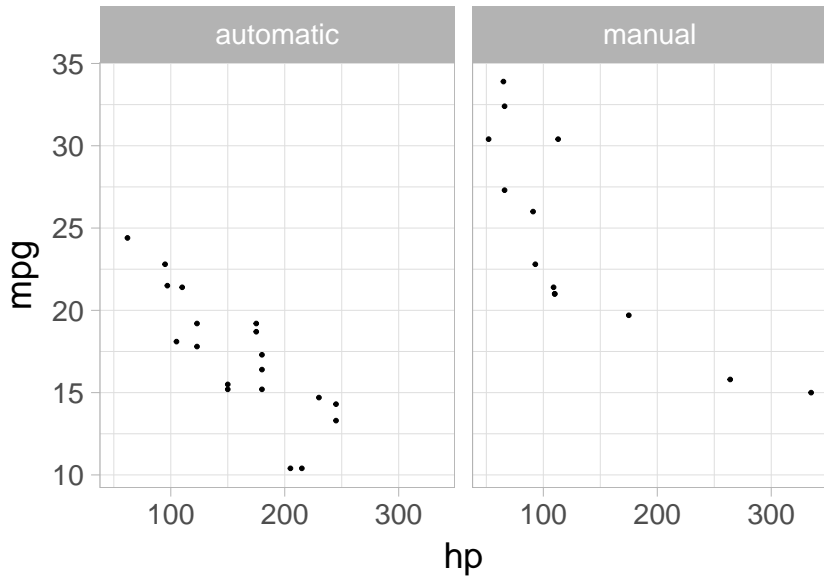




# Plots with multiple panels

```
p <- mtcars %>%  
  mutate(tran = ifelse(am == 0,  
                        "automatic",  
                        "manual")) %>%  
  ggplot(aes(x = hp, y = mpg)) +  
  geom_point() +  
  facet_wrap(~ tran) +  
  theme_light(base_size = 30)
```

# Result



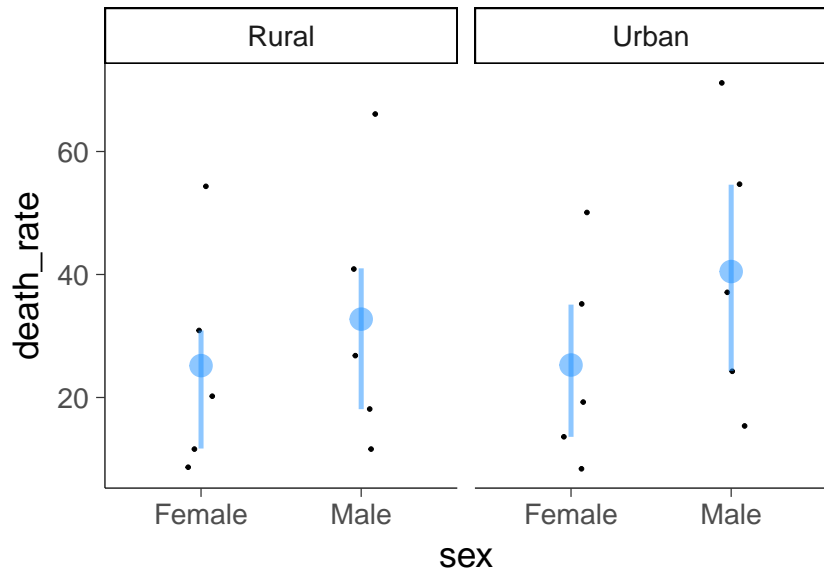
## Exercise

Using the tidiest version of VADeaths, make a plot of death rate by sex, with panels for municipal

# My solution

```
p <- data.frame(VADeaths) %>%
  mutate(age_group = rownames(VADeaths)) %>%
  gather("pop_group", "death_rate", -age_group) %>%
  separate(pop_group, c("municipal", "sex"),
           sep = "\\.") %>%
  ggplot(aes(x = sex, y = death_rate)) + #new stuff below
  geom_jitter(width = 0.1) +
  facet_grid(~municipal) +
  geom_pointrange(stat = "summary",
                 fun.ymin = function(z) quantile(z,0.25),
                 fun.ymax = function(z) quantile(z,0.75),
                 fun.y = mean, colour = "dodgerblue",
                 alpha = 0.5, lwd = 2) +
  theme_classic(base_size = 30)
```

# Result



**purrr: functional programming (like  
lapply/sapply, but safer)**

## Explicit loops are slow (in R) and tedious to write

```
#for example  
vec <- 1:4  
  
store <- rep(NA, length(vec))  
for(i in vec){  
  store[i] <- i*2  
}  
store
```

```
## [1] 2 4 6 8
```

## functional “loop” version

```
vec <- 1:4
```

```
vec %>%  
  map_dbl(function(x) x * 2)
```

```
## [1] 2 4 6 8
```

```
# or
```

```
vec %>%  
  map_dbl(~ .x * 2)
```

```
## [1] 2 4 6 8
```



**But don't do either of the above for this  
("vec\*2")**

# purrr::map functions: also a lot more than I can discuss

- ▶ `map()` – returns a list

## **purrr::map functions: also a lot more than I can discuss**

- ▶ `map()` – returns a list
- ▶ `map_dbl()` – returns a vector of doubles

## **purrr::map functions: also a lot more than I can discuss**

- ▶ `map()` – returns a list
- ▶ `map_dbl()` – returns a vector of doubles
- ▶ `map_lgl()` – returns a vector of integers

## **purrr::map functions: also a lot more than I can discuss**

- ▶ `map()` – returns a list
- ▶ `map_dbl()` – returns a vector of doubles
- ▶ `map_lgl()` – returns a vector of integers
- ▶ `map_int()` – returns a vector of integers

# **purrr::map functions: also a lot more than I can discuss**

- ▶ `map()` – returns a list
- ▶ `map_dbl()` – returns a vector of doubles
- ▶ `map_lgl()` – returns a vector of integers
- ▶ `map_int()` – returns a vector of integers
- ▶ `map_df()` – returns a dataframe

# Important to understand lists and other data types

```
my_list <- list(1, TRUE, rnorm(2), "lists!!")  
#access with double brackets [[]]  
my_list[[3]]
```

```
## [1] 0.7215674 -2.1868046
```

## purrr::map()

```
#I() is the identity function  
#same as function(x) x  
my_list %>%  
  map(I)
```

```
## [[1]]  
## [1] 1  
##  
## [[2]]  
## [1] TRUE  
##  
## [[3]]  
## [1] 0.7215674 -2.1868046  
##  
## [[4]]  
## [1] "lists!!"
```



## purrr::map\_dbl()

```
mtcars %>%  
  map_dbl(mean) %>%  
  head(5)
```

##	mpg	cyl	disp	hp	drat
##	20.090625	6.187500	230.721875	146.687500	3.596563

## purrr::map\_lgl()

```
mtcars %>%  
  map_lgl(is.numeric) %>%  
  head(5)
```

```
##  mpg  cyl disp   hp drat  
## TRUE TRUE TRUE TRUE TRUE
```

## purrr::map\_int()

```
mtcars %>%  
  map_int(length) %>%  
  head(5)
```

```
##   mpg   cyl  disp    hp  drat  
##    32    32   32     32   32
```

## purrr::map\_df()

```
1:4 %>%  
  map_df( ~ data.frame(rbind(rnorm(3))))
```

##		X1	X2	X3
## 1		0.4129384	2.2179540	-1.2387874
## 2		0.1497043	2.3434767	0.6861241
## 3		-0.3053288	-0.5054636	-0.4245817
## 4		1.1284854	-0.1329378	-1.4221895

## Here's a weird one

```
df_list <- list(mtcars = mtcars,  
               iris = iris,  
               VADeaths = VADeaths)
```

*#the nrow function isn't important,  
#the ability to utilize any function here is.*