```
library(dplyr)

rladies_global %>%
  filter(city == 'Austin')
```

R FOR DATA SCIENCE: Exploring Data with ggplot2 and dplyr

Hello!

Welcome to R-Ladies

1. Introduction

R language, RStudio,
R4DS Workshop series

Three things
you'll need to install

1. **Install R** -- this is the open-source programming language we'll use (download via CRAN -- Comprehensive R Archive Network)
2. **Install RStudio** -- this is the most popular IDE for R and will make your life a lot easier (download from rstudio.com/download)
3. **Install the tidyverse** -- this is the group of packages we'll use within R to work with data. Install with one line of code in R:
   `install.packages("tidyverse")`

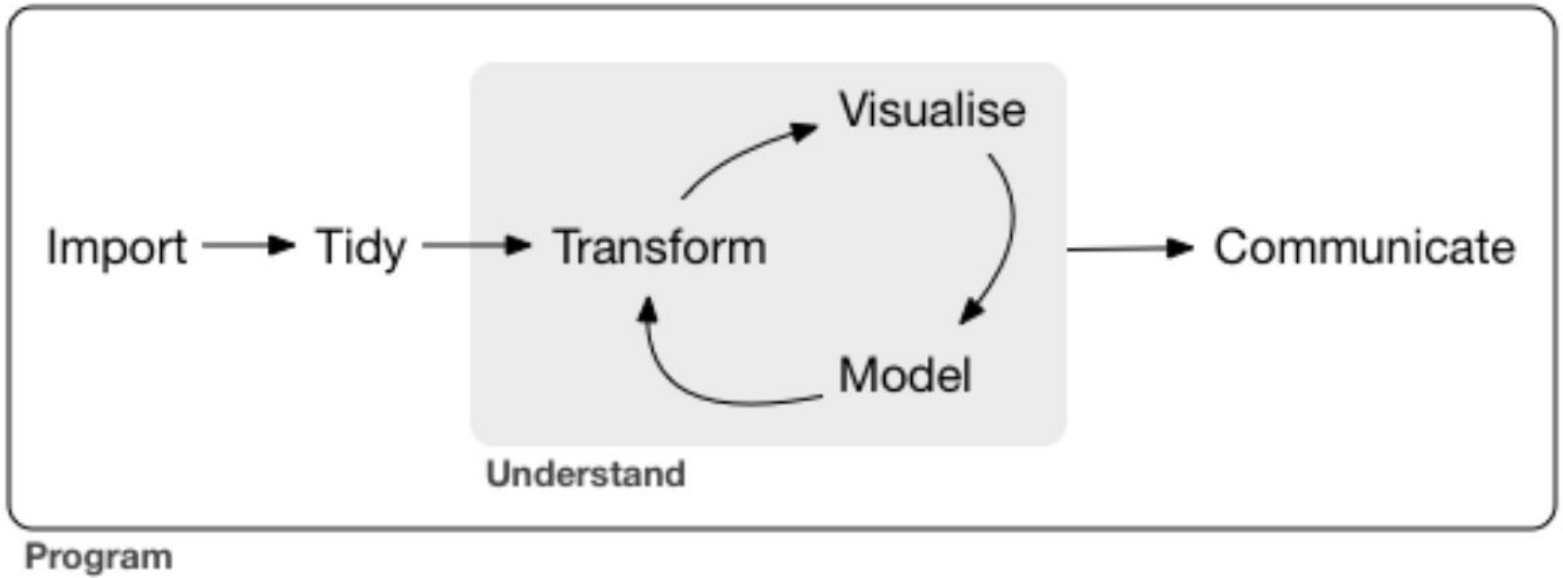
1b. Introduction

R for Data Science Workshop Series

R4DS
Workshop Series

- **Exploring Data with ggplot2 + dplyr** [today]
- **Exploratory Data Analysis and Workflow** [October 25]
- **Data Wrangling in the Tidyverse** [November 29]
- **Programming -- Functions, Vectors, and Iteration** [December 13]
- **Modeling with modelr, purrr, and broom** [January 24]
- **Communicating Results with rmarkdown and ggplot2** [February 21]

The data science process (tidied)



What is
the tidyverse?

- Collection of R packages based on tidy data principles
- Designed to work together
- An easier way to code!
- AKA “Hadleyverse” (most packages written by Hadley Wickham)

What is
the tidyverse?

What is
tidy data?

- Each variable is a column
- Each observation is a row
- Each type of observational unit is a table

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

2.
ggplot2

Let’s start with the first set of slides

ggplot2-how to start

```
library(tidyverse)
library(ggplot2)
```

General form:

```
ggplot() +
    geom_<TYPE> (mapping = aes())
```

aesthetics

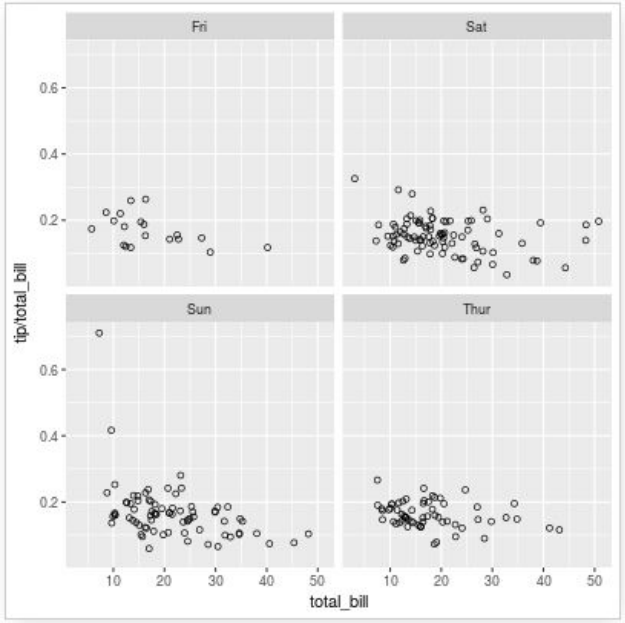
aesthetics = aes = the look of the plot!

- x
- y
- Color (colour)
- Size
- Alpha - opacity
- Shape
- Linetype
- group

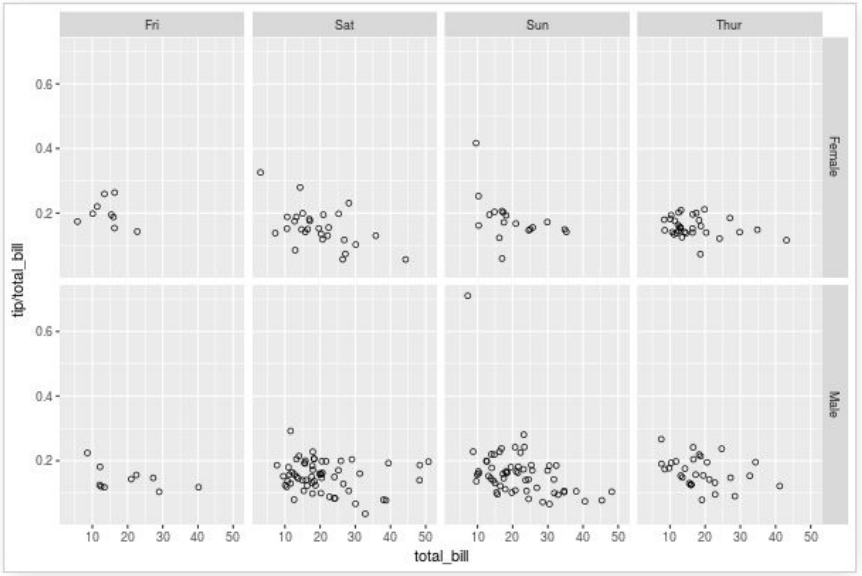
p.12 exercises

facets

`facet_wrap()`



`facet_grid()`



Geometric objects

There are many geoms!

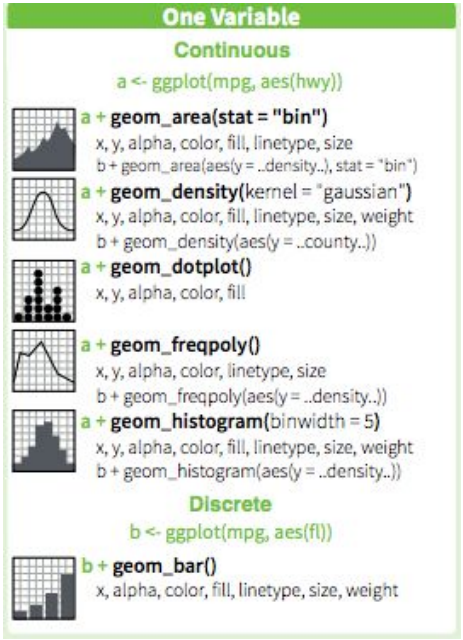
- Can have multiple geoms (different layers)
- Get to know the ggplot2 cheatsheet!

https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Mapping to `ggplot()` vs `geom_<TYPE>()`

- Avoid duplicate code
- If in `geom_<TYPE>()`, applied to that layer only
- If in `ggplot()`, global!

p.21 #6 exercise



Statistical transformations

Create new values for a chart

Can interchange geoms and stats

- Every geom has default stat..
- ...every stat a default geom

`geom_bar() = stat_count()`

Want to change the default?

`stat = “<STAT HERE>”` in mapping

p.26 #2 and #5 exercises

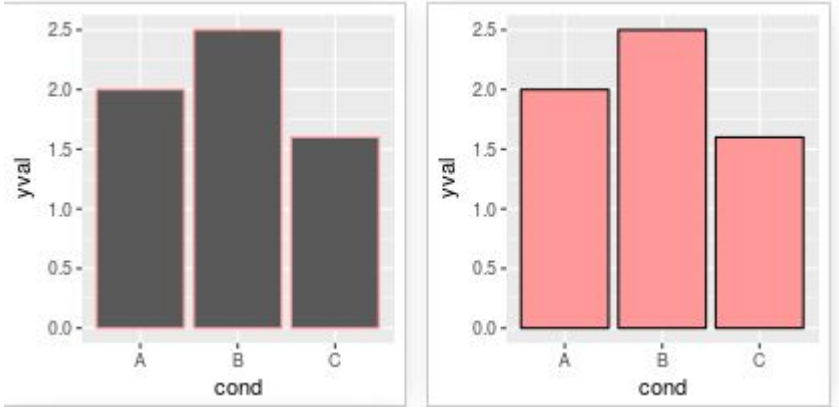
Position adjustments

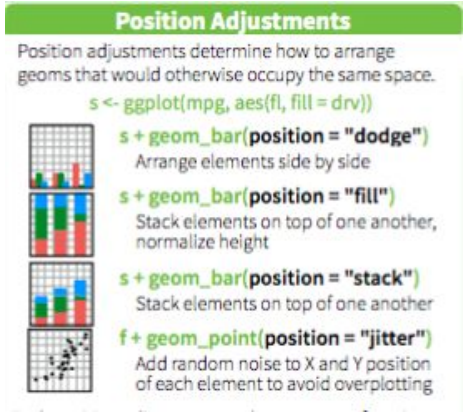
`fill = vs. colour =`

Can fill with a variable = stacked bar!

3 options

1. `position = “identity”`
    a. Not too useful for bar plots
2. `position = “fill”`
    a. Stacked but same height
3. `position = “dodge”`
    a. Grouped!
4. `position = “jitter”`
    a. Adds smudge factor





Coordinate systems

`coord_flip()`

- Swap x and y
- Nice for long labels

`coord_quickmap()`

- Spatial data

`coord_polar()`

- Polar coordinates

p. 33 #2 and #4 exercise

Bring it all together

7 basic parameters for any plot!

```
ggplot(data = <DATA>)+
    geom_<TYPE> (
        mapping=aes(<MAPPINGS>,
                    stat= <STATS>,
                    position= <POSITION>) +
    <COORD FUNCTION> +
    <FACET FUNCTION>
```

lagniappe

`ggtitle()` - give your plot a title!

`xlab()` - label x axis

`ylab()` - label y axis

`theme_bw()` - make your plot look nice easily

COLORS!

http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/

A colorblind-friendly palette

These are color-blind-friendly palettes, one with gray, and one with black.

To use with ggplot2, it is possible to store the palette in a variable, then use it later.

```
# The palette with grey:
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

# The palette with black:
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

# To use for fills, add
  scale_fill_manual(values=cbPalette)

# To use for line and point colors, add
  scale_colour_manual(values=cbPalette)
```

3.
dplyr

Let’s start with the first set of slides

What is dplyr?

Grammar of data manipulation:

+ `mutate()` to create new variables from existing ones
+ `select()` picks variables based on their names
+ `filter()` allows pointed selection based on given criteria
+ `summarise()` reduces multiple values down to a single summary
+ `arrange()` changes the ordering of rows
+ `group_by()` performs any of the above on a group-by-group basis

dplyr
syntax

+ All calls to dplyr verbs follow the same format:
  1. The first argument is a dataframe
  2. The subsequent arguments describe what to do to that dataframe, using unquoted variable names.

+ Each call returns a new dataframe (rather than overwriting the ‘old’ one)

+ Example:
  ```
  filter(babynames, name == “Caitlin”)
  ```

What is magrittr?

Simplifying R code with pipes (%>%)

+ Easy way to pass data through functions without nesting
+ First argument of each function is “piped” in to reduce redundancy

dplyr + magrittr
example

before

```
summarise(
  group_by((filter
            (babynames, name == "Caitlin")
            ),
           year
           ),
  total = sum(n))
```

after

```
babynames %>%
  filter(name == "Caitlin") %>%
  group_by(year) %>%
  summarise(total = sum(n))
```

What is dplyr?

Grammar of data manipulation:

+ `select()` picks variables based on their names
+ `arrange()` changes the ordering of rows
+ `filter()` allows pointed selection based on given criteria
+ `mutate()` to create new variables from existing ones
+ `summarise()` reduces multiple values down to a single summary
+ `group_by()` performs any of the above on a group-by-group basis

Quick aside:
iris dataset

+ Included in R (**iris** to view)
+ 150 observations of 5 variables:
  Iris type, sepal length + width, and petal length + width

select()

+ Picks variables based on their names
+ First argument is dataframe; subsequent arguments represent columns to select

iris %>% select(Species, Petal.Length, Petal.Width)

```
> 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
```

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

`select()` + helper functions

Helper functions you can use within `select()`:

+ `starts_with("a")` matches names that begin with "a"
+ `ends_with("z")` matches names that begin with "z"
+ `contains("lady")` matches names that contain "lady"
+ `matches(<regex>)` allows you to do regex matching on names

```
> iris %>% select(starts_with("p"))
    Petal.Length Petal.Width
1            1.4         0.2
2            1.4         0.2
3            1.3         0.2
4            1.5         0.2
5            1.4         0.2
```

arrange()

- Changes the ordering of rows
- First argument is the dataframe, subsequent arguments are columns and/or expressions used to re-arrange the dataframe
- Note: default is ascending order, and NA's are always at the end

```
iris %>% arrange(Sepal.Length, Sepal.Width)
```

```
> 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
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            4.3         3.0          1.1         0.1     setosa
2            4.4         2.9          1.4         0.2     setosa
3            4.4         3.0          1.3         0.2     setosa
4            4.4         3.2          1.3         0.2     setosa
5            4.5         2.3          1.3         0.3     setosa
```

filter()

+ Allows pointed selection based on given criteria
+ First argument is the dataframe, subsequent arguments are expressions used to filter the dataframe

```
iris %>% filter(Species == “setosa”)
```

```
> 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
```

nrow = 150

```
  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
```

nrow = 50

`filter()` + booleans

Multiple arguments to `filter()` are combined with “and”: every expression must be true in order for a row to be included in the output. For other types of combinations, you’ll need to use Boolean operators yourself: `&` is “and”, `|` is “or”, and `!` is “not”. Figure 5.1 shows the complete set of Boolean operations.

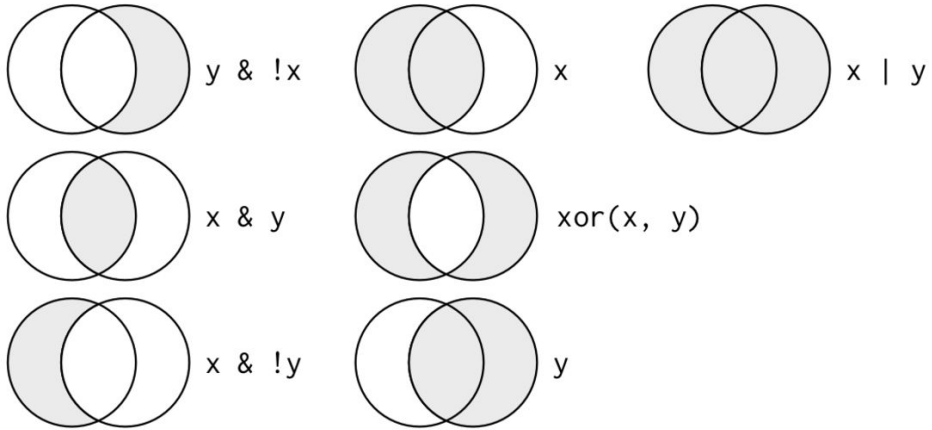


Figure 5.1: Complete set of boolean operations. `x` is the left-hand circle, `y` is the right-hand circle, and the shaded region show which parts each operator selects.

Quick aside:
Missing values

- NA represents a missing (unknown) value
- Comparisons involve unknown values typically result in unknown values
- To see whether a value is missing, use `is.na()`
- `filter()` only includes rows where the condition is true (not false or NA)

```
# Let x be Mary's age. We don't know how old she is.
x <- NA

# Let y be John's age. We don't know how old he is.
y <- NA

# Are John and Mary the same age?
x == y
#> [1] NA
# We don't know!
```

mutate()

+ Creates new variables from existing ones
+ Note: columns created with `mutate()` are always added to end of dataset

```
iris %>% mutate(petal_area = Petal.Length * Petal.Width)
```

```
> 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
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	petal_area
1	5.1	3.5	1.4	0.2	setosa	0.28
2	4.9	3.0	1.4	0.2	setosa	0.28
3	4.7	3.2	1.3	0.2	setosa	0.26
4	4.6	3.1	1.5	0.2	setosa	0.30
5	5.0	3.6	1.4	0.2	setosa	0.28

`mutate()`
Useful functions

+ Arithmetic operators (+, -, *, /, ^)
+ Log functions (like `log10()`)
+ Offsets like `lead()` and `lag()`
+ Logical comparisons (<, <=, >, >=, !=)
+ Ifelse statements (if this, then this, else this)
+ Cumulative and rolling aggregates
+ Ranking (like `ntile()`)

group_by() and summarise()

+ group_by applies dplyr verbs by group
+ summarise reduces multiple values down to a single summary

```
iris %>%
    group_by(Species) %>%
    summarise(avg_petal_width = mean(Petal.Width)
```

```
> 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
```

```
# A tibble: 3 × 2
     Species avg_petal_width
      <fctr>           <dbl>
1     setosa           0.246
2 versicolor           1.326
3  virginica           2.026
```

`summarise()`
Useful functions

+ Counts (`n()`, `n_distinct()`)
+ Measures of location (`mean()`, `median()`)
+ Measures of spread (`sd()`, `IQR()`)
+ Measures of position (`first()`, `last()`)

Tips & Tricks

- If you don’t have the result of a dplyr chain to a dataframe, it will print
- If you want to print and save, wrap assignment in parenthesis
  - Example: `(iris_names <- iris %>% filter(Species == “setosa”))`
- `rename()` is a cool function to clean up messy column names
- After grouping with `group_by()`, you can `ungroup()` to remove groupings
- There is a cheat sheet for data wrangling!

4.
Wrap-up

Announcements, upcoming events, etc.

R-Ladie Austin

Upcoming Events

Boo! Making GitHub Less Scary [October 12]

Exploratory Data Analysis and Workflow [October 25]

Book Club: Dear Data [November 2]

Looking for presenters: Workshop on package development