tidyverse overview

Andrew Ghazi - aghazi@broadinstitute.org

March 25th 2021 - Rendered at 2021-04-01 15:49:18

While we wait…

1. Install R from cran.r-project.org
2. Install Rstudio from rstudio.com
3. Install the tidyverse with this R command: `install.packages("tidyverse")`
4. If you'd like to follow along, get the slides (.html) and/or code (.R): github.com/andrewGhazi/tidy_overview

What is the tidyverse?

- A dialect of R
- A collection of R packages with a shared design philosophy

- Install it with this R command:
  `install.packages("tidyverse")`

Why use the tidyverse?

- Shared design philosophy → inter-package consistency
- Focused on data science and statistical analysis
- Easy to learn, write, and read
- It's ubiquitous

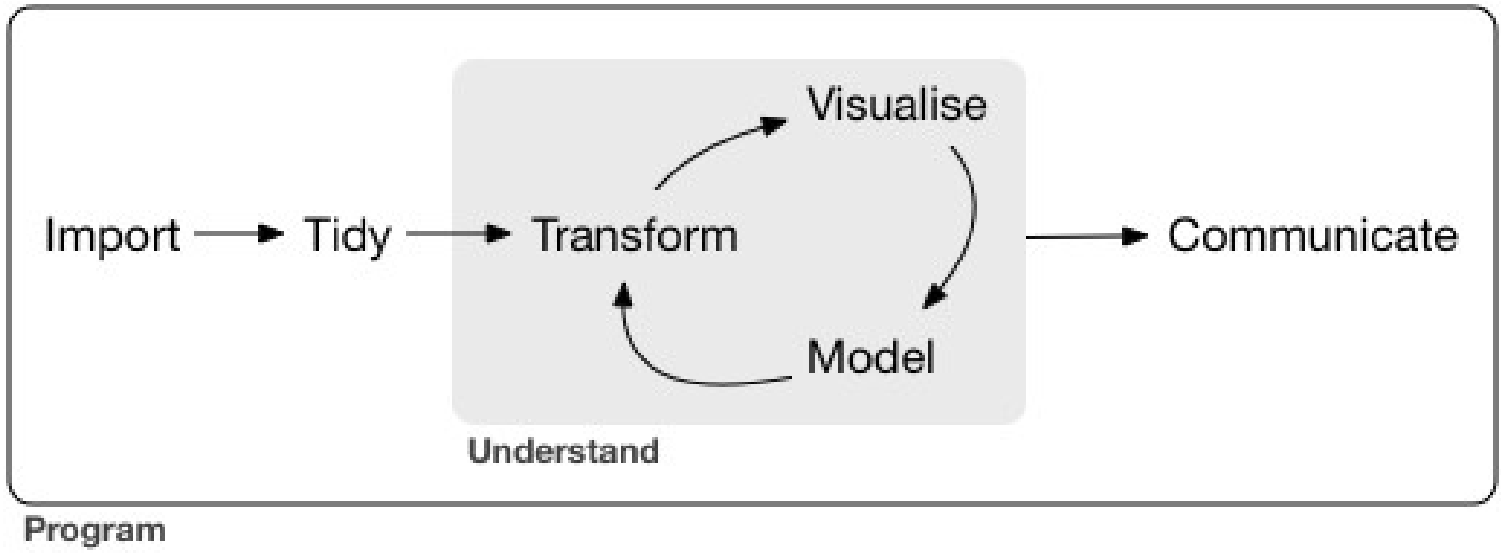


Diagram from R for Data Science by Wickham et al.

Outline

- Core concepts
- Package overview
- End-to-end example
- Questions

How to read these slides

- Lecture notes show up like this.

```
print("Console output has two pound symbols in front.")
```

```
## [1] "Console output has two pound symbols in front."
```

```
x = 5
```

- Assignment prints nothing.
- `package::function()` e.g. `dplyr::filter()`

Load the tidyverse

The startup message lists loaded packages and overwritten functions.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.3
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## Warning: package 'tibble' was built under R version 4.0.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Core concepts

Data frames

- Used to store ordered collections of variables

```
diabetes
```

```
## # A tibble: 532 x 8
##    npreg   glu    bp  skin   bmi   ped   age diabetic
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1     5    86    68    28  30.2 0.364    24 No
##  2     7   195    70    33  25.1 0.163    55 Yes
##  3     5    77    82    41  35.8 0.156    35 No
##  4     0   165    76    43  47.9 0.259    26 No
##  5     0   107    60    25  26.4 0.133    23 No
##  6     5    97    76    27  35.6 0.378    52 Yes
##  7     3    83    58    31  34.3 0.336    25 No
##  8     1   193    50    16  25.9 0.655    24 No
##  9     3   142    80    15  32.4 0.2      63 No
## 10     2   128    78    37  43.3 1.22     31 Yes
## # ... with 522 more rows
```

tidy data

- Columns are variables
- Rows are observations

```
## # A tibble: 532 x 8
##    npreg   glu    bp  skin   bmi   ped   age diabetic
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1     5    86    68    28  30.2 0.364    24 No
##  2     7   195    70    33  25.1 0.163    55 Yes
##  3     5    77    82    41  35.8 0.156    35 No
##  4     0   165    76    43  47.9 0.259    26 No
##  5     0   107    60    25  26.4 0.133    23 No
##  6     5    97    76    27  35.6 0.378    52 Yes
##  7     3    83    58    31  34.3 0.336    25 No
##  8     1   193    50    16  25.9 0.655    24 No
##  9     3   142    80    15  32.4 0.2      63 No
## 10     2   128    78    37  43.3 1.22     31 Yes
## # ... with 522 more rows
```

tidyverse functions

- data frame in, data frame out
- Function names are *verbs*
  - e.g. `filter`, `arrange`, `mutate`

Pipes

- Pipes look like this: `%>%`
- They take the input from the left side, and hand it to the right side:

```
c(1,2,3,4) %>% mean
```

```
## [1] 2.5
```

- Verbalize as: “and then”.
- “Create this vector and then take the mean”.

Pipe example

Why are pipes useful?

Pipe example

Why are pipes useful?

Pipe example

Why are pipes useful?

Pipe example

Compare two ways of doing the same thing:

```
input = "potatoes"

stick(mash(boil(input)), where = 'stew')
```

```
## [1] "stewed, mashed, boiled potatoes"
```

Pipe example

Compare two ways of doing the same thing:

```
input = "potatoes"
```

```
stick(mash(boil(input)), where = 'stew')
```

```
## [1] "stewed, mashed, boiled potatoes"
```

```
input %>% boil() %>% mash() %>% stick(where = 'stew')
```

```
## [1] "stewed, mashed, boiled potatoes"
```

Pipe example

Why are pipes useful?

Because chained verbs look like English sentences.

```
input %>%
  boil() %>%
  mash() %>%
  stick(where = 'stew')
```

- Piped code is easier to write *and easier to read*.
- Keyboard shortcut: `ctrl + shift + M`

package overview

Import - `readr`

- Get the data into R as a data frame.
- Read data from a file on your computer or from a URL
- `read_csv()`, `read_tsv()`
- related package: `readxl`
- Read in the diabetes example dataset:

```
data_path = "data/diabetes.tsv"
diabetes = read_tsv(data_path)
```

Manipulate - `dplyr`

- Package for basic data manipulation
- Most important functions:
  `filter()`, `select()`, `arrange()`, `mutate()`, `group_by()`, `summarise()`
- Focused on data frames

dplyr::filter()

- Subset rows by a condition

```
diabetes %>%
  filter(npreg == 0)
```

```
## # A tibble: 77 x 8
##    npreg   glu    bp  skin   bmi   ped   age diabetic
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1     0   165    76    43  47.9 0.259    26 No
##  2     0   107    60    25  26.4 0.133    23 No
##  3     0   137    40    35  43.1 2.29     33 Yes
##  4     0   139    62    17  22.1 0.207    21 No
##  5     0   101    64    17  21   0.252    21 No
##  6     0   140    65    26  42.6 0.431    24 Yes
##  7     0   121    66    30  34.3 0.203    33 Yes
##  8     0   102    86    17  29.3 0.695    27 No
##  9     0   119    66    27  38.8 0.259    22 No
## 10     0    86    68    32  35.8 0.238    25 No
## # ... with 67 more rows
```

dplyr::filter()

- Subset rows by a condition

```
diabetes %>%
  filter(bmi > 30)
```

```
## # A tibble: 342 x 8
##    npreg   glu    bp  skin   bmi   ped   age diabetic
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1     5    86    68    28  30.2 0.364    24 No
##  2     5    77    82    41  35.8 0.156    35 No
##  3     0   165    76    43  47.9 0.259    26 No
##  4     5    97    76    27  35.6 0.378    52 Yes
##  5     3    83    58    31  34.3 0.336    25 No
##  6     3   142    80    15  32.4 0.2      63 No
##  7     2   128    78    37  43.3 1.22     31 Yes
##  8     0   137    40    35  43.1 2.29     33 Yes
##  9     9   154    78    30  30.9 0.164    45 No
## 10     1   189    60    23  30.1 0.398    59 Yes
## # ... with 332 more rows
```

dplyr::select()

- Subset columns
- Choose columns by name, index, or condition

```
diabetes %>%
  select(diabetic, npreg, age)
```

```
## # A tibble: 532 x 3
##    diabetic npreg   age
##    <chr>    <dbl> <dbl>
##  1 No           5    24
##  2 Yes          7    55
##  3 No           5    35
##  4 No           0    26
##  5 No           0    23
##  6 Yes          5    52
##  7 No           3    25
##  8 No           1    24
##  9 No           3    63
## 10 Yes          2    31
## # ... with 522 more rows
```

dplyr::select()

- Subset columns
- Choose columns by name, index, or condition

```
diabetes %>%
  select(8, 1, 7)
```

```
## # A tibble: 532 x 3
##    diabetic npreg   age
##    <chr>    <dbl> <dbl>
##  1 No           5    24
##  2 Yes          7    55
##  3 No           5    35
##  4 No           0    26
##  5 No           0    23
##  6 Yes          5    52
##  7 No           3    25
##  8 No           1    24
##  9 No           3    63
## 10 Yes          2    31
## # ... with 522 more rows
```

dplyr::select()

- Subset columns
- Choose columns by name, index, or condition

```
diabetes %>%
  select(starts_with('b'), matches('diab'))
```

```
## # A tibble: 532 x 3
##       bp   bmi diabetic
##    <dbl> <dbl> <chr>
##  1    68  30.2 No
##  2    70  25.1 Yes
##  3    82  35.8 No
##  4    76  47.9 No
##  5    60  26.4 No
##  6    76  35.6 Yes
##  7    58  34.3 No
##  8    50  25.9 No
##  9    80  32.4 No
## 10    78  43.3 Yes
## # ... with 522 more rows
```

dplyr::arrange()

- Reorder rows by a variable

```
diabetes %>% arrange(bmi)
```

```
## # A tibble: 532 x 8
##    npreg   glu    bp  skin   bmi   ped   age diabetic
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1     1    97    64    19  18.2 0.299    21 No
##  2     1    97    70    15  18.2 0.147    21 No
##  3     3    99    80    11  19.3 0.284    30 No
##  4     1   103    80    11  19.4 0.491    22 No
##  5     1    92    62    25  19.5 0.482    25 No
##  6     1   100    74    12  19.5 0.149    28 No
##  7     1    95    66    13  19.6 0.334    25 No
##  8     6   129    90     7  19.6 0.582    60 No
##  9     0   105    68    22  20   0.236    22 No
## 10     2    68    62    13  20.1 0.257    23 No
## # ... with 522 more rows
```

dplyr::arrange()

- Reorder rows by a variable

```
diabetes %>% arrange(desc(age))
```

```
## # A tibble: 532 x 8
##    npreg   glu    bp  skin   bmi   ped   age diabetic
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1     9   134    74    33  25.9 0.46     81 No
##  2     4   145    82    18  32.5 0.235    70 Yes
##  3     5   103   108    37  39.2 0.305    65 No
##  4     3   142    80    15  32.4 0.2      63 No
##  5     4   132    86    31  28   0.419    63 No
##  6    10   101    76    48  32.9 0.171    63 No
##  7    12   121    78    17  26.5 0.259    62 No
##  8     2   197    70    99  34.7 0.575    62 Yes
##  9     7   142    60    33  28.8 0.687    61 No
## 10     8   181    68    36  30.1 0.615    60 Yes
## # ... with 522 more rows
```

Quiz: What's the highest blood pressure observed in this data? (hint: desc())

`dplyr::mutate()`

- Add new columns
- Structure:

```
input_df %>%
  mutate(col_name = col_values)
```

dplyr::mutate()

- Add new columns
- Example: add an index

```
diabetes %>%
  mutate(index = 1:532)
```

```
## # A tibble: 532 x 9
##    npreg   glu    bp  skin   bmi   ped   age diabetic index
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>    <int>
##  1     5    86    68    28  30.2 0.364    24 No           1
##  2     7   195    70    33  25.1 0.163    55 Yes          2
##  3     5    77    82    41  35.8 0.156    35 No           3
##  4     0   165    76    43  47.9 0.259    26 No           4
##  5     0   107    60    25  26.4 0.133    23 No           5
##  6     5    97    76    27  35.6 0.378    52 Yes          6
##  7     3    83    58    31  34.3 0.336    25 No           7
##  8     1   193    50    16  25.9 0.655    24 No           8
##  9     3   142    80    15  32.4 0.2      63 No           9
## 10     2   128    78    37  43.3 1.22     31 Yes         10
## # ... with 522 more rows
```

dplyr::mutate()

- Refer to other columns by name
- Example: Calculate birth year as a function of age

```
diabetes %>%
  mutate(birth_year = 2021 - age)
```

```
## # A tibble: 532 x 9
##    npreg   glu    bp  skin   bmi   ped   age diabetic birth_year
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>         <dbl>
##  1     5    86    68    28  30.2 0.364    24 No             1997
##  2     7   195    70    33  25.1 0.163    55 Yes            1966
##  3     5    77    82    41  35.8 0.156    35 No             1986
##  4     0   165    76    43  47.9 0.259    26 No             1995
##  5     0   107    60    25  26.4 0.133    23 No             1998
##  6     5    97    76    27  35.6 0.378    52 Yes            1969
##  7     3    83    58    31  34.3 0.336    25 No             1996
##  8     1   193    50    16  25.9 0.655    24 No             1997
##  9     3   142    80    15  32.4 0.2      63 No             1958
## 10     2   128    78    37  43.3 1.22     31 Yes            1990
## # ... with 522 more rows
```

dplyr::mutate()

- Add new columns
- Example: Calculate birth year as a function of age

```
diabetes %>%
  mutate(birth_year = 2021 - age,
         is_mother = npreg > 0)
```

```
## # A tibble: 532 x 10
##    npreg   glu    bp  skin   bmi   ped   age diabetic birth_year is_mother
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>         <dbl> <lgl>
##  1     5    86    68    28  30.2 0.364    24 No             1997 TRUE
##  2     7   195    70    33  25.1 0.163    55 Yes            1966 TRUE
##  3     5    77    82    41  35.8 0.156    35 No             1986 TRUE
##  4     0   165    76    43  47.9 0.259    26 No             1995 FALSE
##  5     0   107    60    25  26.4 0.133    23 No             1998 FALSE
##  6     5    97    76    27  35.6 0.378    52 Yes            1969 TRUE
##  7     3    83    58    31  34.3 0.336    25 No             1996 TRUE
##  8     1   193    50    16  25.9 0.655    24 No             1997 TRUE
##  9     3   142    80    15  32.4 0.2      63 No             1958 TRUE
## 10     2   128    78    37  43.3 1.22     31 Yes            1990 TRUE
## # ... with 522 more rows
```

dplyr::group_by()

- Group a data frame
- Example: diabetic vs non-diabetic:

```
diabetes %>%
  group_by(diabetic)
```

```
## # A tibble: 532 x 8
## # Groups:   diabetic [2]
##    npreg   glu    bp  skin   bmi   ped   age diabetic
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1     5    86    68    28  30.2 0.364    24 No
##  2     7   195    70    33  25.1 0.163    55 Yes
##  3     5    77    82    41  35.8 0.156    35 No
##  4     0   165    76    43  47.9 0.259    26 No
##  5     0   107    60    25  26.4 0.133    23 No
##  6     5    97    76    27  35.6 0.378    52 Yes
##  7     3    83    58    31  34.3 0.336    25 No
##  8     1   193    50    16  25.9 0.655    24 No
##  9     3   142    80    15  32.4 0.2      63 No
## 10     2   128    78    37  43.3 1.22     31 Yes
## # ... with 522 more rows
```

dplyr::summarise()

- Compute summary values by group

```
diabetes %>%
  group_by(diabetic) %>%
  summarise(mean_age = mean(age))
```

```
## # A tibble: 2 x 2
##   diabetic mean_age
## * <chr>       <dbl>
## 1 No           29.2
## 2 Yes          36.4
```

dplyr::summarise()

- Compute summary values by group

```
diabetes %>%
  group_by(diabetic) %>%
  summarise(mean_age = mean(age),
            group_size = n())
```

```
## # A tibble: 2 x 3
##   diabetic mean_age group_size
## * <chr>       <dbl>      <int>
## 1 No           29.2        355
## 2 Yes          36.4        177
```

dplyr::summarise()

- You can group by multiple variables

```
diabetes %>%
  mutate(is_mother = npreg > 0) %>%
  group_by(diabetic, is_mother)
```

```
## # A tibble: 532 x 9
## # Groups:   diabetic, is_mother [4]
##    npreg   glu    bp  skin   bmi   ped   age diabetic is_mother
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>    <lgl>
##  1     5    86    68    28  30.2 0.364    24 No       TRUE
##  2     7   195    70    33  25.1 0.163    55 Yes      TRUE
##  3     5    77    82    41  35.8 0.156    35 No       TRUE
##  4     0   165    76    43  47.9 0.259    26 No       FALSE
##  5     0   107    60    25  26.4 0.133    23 No       FALSE
##  6     5    97    76    27  35.6 0.378    52 Yes      TRUE
##  7     3    83    58    31  34.3 0.336    25 No       TRUE
##  8     1   193    50    16  25.9 0.655    24 No       TRUE
##  9     3   142    80    15  32.4 0.2      63 No       TRUE
## 10     2   128    78    37  43.3 1.22     31 Yes      TRUE
## # ... with 522 more rows
```

dplyr::summarise()

- You can group by multiple variables

```
diabetes %>%
  mutate(is_mother = npreg > 0) %>%
  group_by(diabetic, is_mother) %>%
  summarise(mean_age = mean(age))
```

```
## `summarise()` has grouped output by 'diabetic'. You can override using the `.groups` argumer
```

```
## # A tibble: 4 x 3
## # Groups:   diabetic [2]
##   diabetic is_mother mean_age
##   <chr>    <lgl>        <dbl>
## 1 No       FALSE         25.1
## 2 No       TRUE          29.9
## 3 Yes      FALSE         26.8
## 4 Yes      TRUE          38.1
```

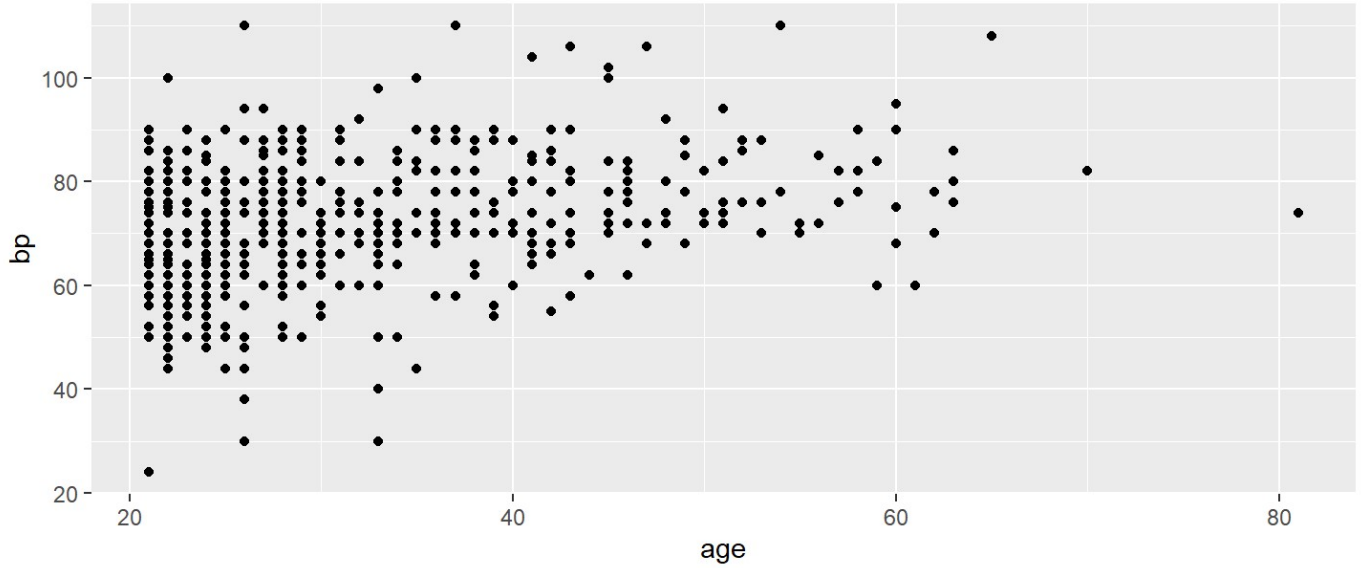
Visualize - `ggplot2`

- "Grammar of graphics"
- Core concepts:
  - Variables from tidy input data
  - Aesthetic mappings between variables and graphical elements: `aes()`
  - Add geometric objects to represent data via `geom_*()` e.g. `geom_point()`

Visualize - `ggplot2` example

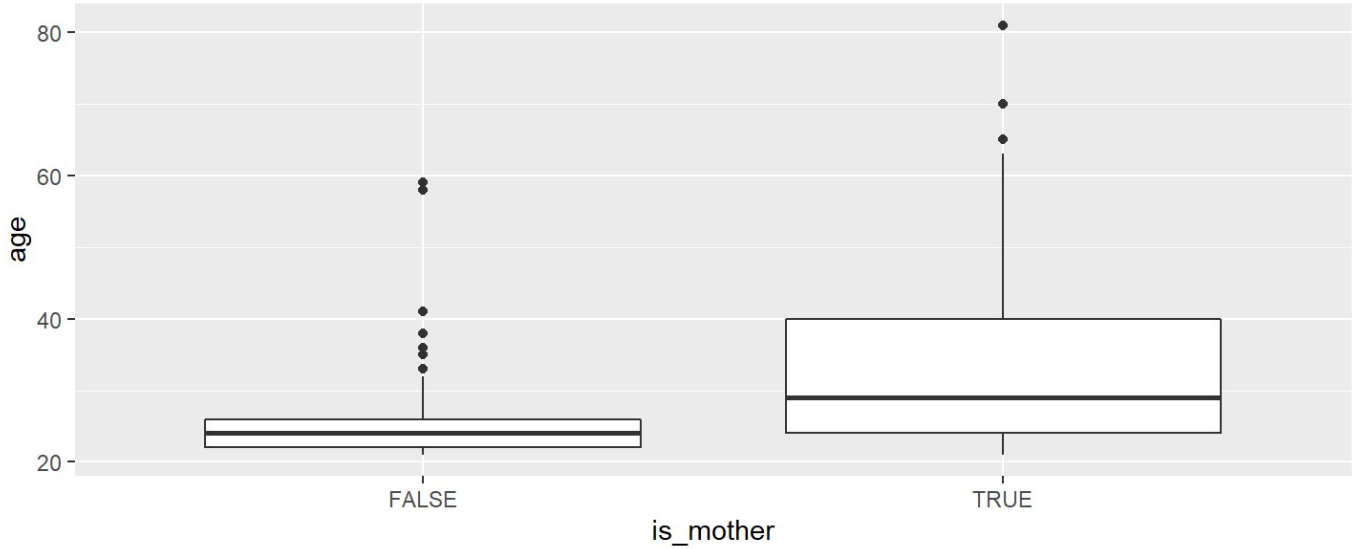
- Tidy input, aesthetic mapping, and a geom

```
diabetes %>%
  ggplot(mapping = aes(x = age, y = bp)) +
  geom_point()
```

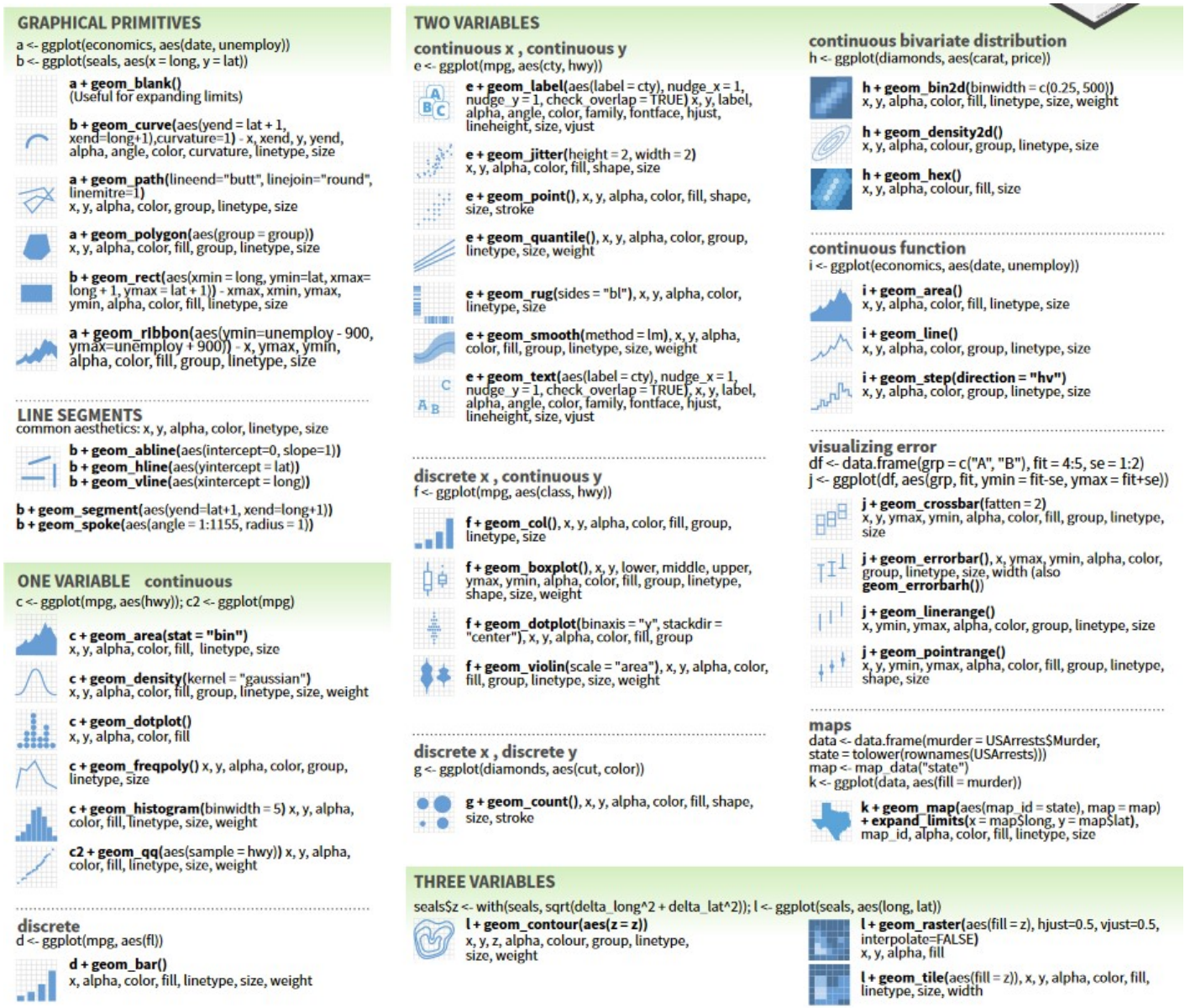


Visualize - ggplot2 example

```
diabetes %>%
  mutate(is_mother = npreg > 0) %>%
  ggplot(aes(is_mother, age)) +
  geom_boxplot()
```



Visualize - ggplot2's suite of geoms



GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1),curvature=1) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS
common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete
d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y
e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y
f <- ggplot(mpg, aes(class, hwy))

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y
g <- ggplot(diamonds, aes(cut, color))

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution
h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function
i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps
data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map) **+ expand_limits**(x = map$long, y = map$lat), map_id, alpha, color, fill, linetype, size

THREE VARIABLES
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

Report - `rmarkdown`

- Communicate analysis results
- Easily interweave notes, code, plots, and $\LaTeX$ in a single file
- Render as reports (.docx, .pdf, .html), slides (.html, .ppt), or web applications (Shiny)
- Rmd + Github = Digital lab notebook
- Demonstration at end of session 1 if there's time

COVID-19 example

COVID-19 data

- New York Times COVID-19 data on Github
- https://github.com/nytimes/covid-19-data
- Provides COVID-19 data by county, state, and nation-wide

Example: US COVID-19 state-level data

```
nyt_url = "https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv"
state_covid = read_csv(nyt_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   date = col_date(format = ""),
##   state = col_character(),
##   fips = col_character(),
##   cases = col_double(),
##   deaths = col_double()
## )
```

Example: Import state-level COVID-19 data

- Look at the data

```
state_covid
```

```
## # A tibble: 21,629 x 5
##    date       state      fips  cases deaths
##    <date>     <chr>      <chr> <dbl>  <dbl>
##  1 2020-01-21 Washington 53        1      0
##  2 2020-01-22 Washington 53        1      0
##  3 2020-01-23 Washington 53        1      0
##  4 2020-01-24 Illinois   17        1      0
##  5 2020-01-24 Washington 53        1      0
##  6 2020-01-25 California 06        1      0
##  7 2020-01-25 Illinois   17        1      0
##  8 2020-01-25 Washington 53        1      0
##  9 2020-01-26 Arizona    04        1      0
## 10 2020-01-26 California 06        2      0
## # ... with 21,619 more rows
```

Example: COVID-19 in California

- Plot one state over time

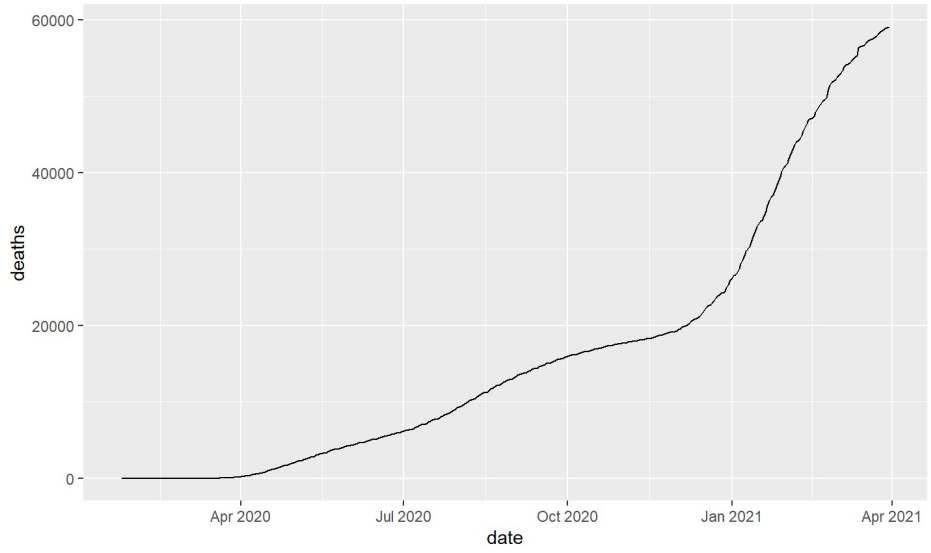
```
state_covid %>%
  filter(state == "California") %>%
  ggplot(aes(date, deaths)) +
  geom_line()
```

- What will the output of this command look like?

Example: COVID-19 in California

- Plot one state over time

```
state_covid %>%
  filter(state == "California") %>%
  ggplot(aes(date, deaths)) +
  geom_line()
```



Example: COVID-19 in California

- Want to define the CHANGE in deaths

```
state_covid %>%
  filter(state == "California", deaths > 0)
```

```
## # A tibble: 392 x 5
##    date       state      fips  cases deaths
##    <date>     <chr>      <chr> <dbl>  <dbl>
##  1 2020-03-04 California 06       55      1
##  2 2020-03-05 California 06       67      1
##  3 2020-03-06 California 06       81      1
##  4 2020-03-07 California 06      100      1
##  5 2020-03-08 California 06      112      1
##  6 2020-03-09 California 06      172      2
##  7 2020-03-10 California 06      179      3
##  8 2020-03-11 California 06      202      4
##  9 2020-03-12 California 06      252      4
## 10 2020-03-13 California 06      320      5
## # ... with 382 more rows
```

Example: COVID-19 in California

- Define daily added deaths using `diff()`

```
state_covid %>%
  filter(state == "California") %>%
  mutate(new_deaths = diff(deaths))
```

That didn't work

- Define daily added deaths using `diff()`

```
state_covid %>%
  filter(state == "California") %>%
  mutate(new_deaths = diff(deaths))
```

```
## Error: Problem with `mutate()` input `new_deaths`.
## x Input `new_deaths` can't be recycled to size 431.
## i Input `new_deaths` is `diff(deaths)`.
## i Input `new_deaths` must be size 431 or 1, not 430.
```

The fix

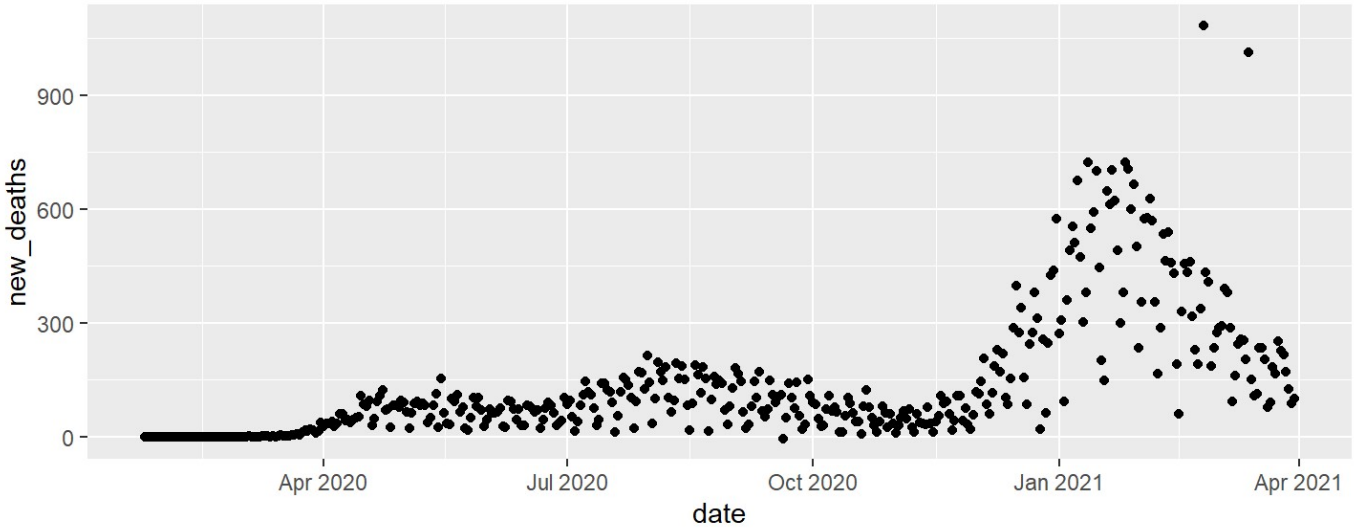
- `diff()` returns a vector that's one element too short – tack on the first observation at the start.

```
state_covid %>%
  filter(state == "California") %>%
  mutate(new_deaths = c(deaths[1], diff(deaths)))
```

```
## # A tibble: 431 x 6
##    date       state      fips  cases deaths new_deaths
##    <date>     <chr>      <chr> <dbl>  <dbl>      <dbl>
##  1 2020-01-25 California 06        1      0          0
##  2 2020-01-26 California 06        2      0          0
##  3 2020-01-27 California 06        2      0          0
##  4 2020-01-28 California 06        2      0          0
##  5 2020-01-29 California 06        2      0          0
##  6 2020-01-30 California 06        2      0          0
##  7 2020-01-31 California 06        3      0          0
##  8 2020-02-01 California 06        3      0          0
##  9 2020-02-02 California 06        6      0          0
## 10 2020-02-03 California 06        6      0          0
## # ... with 421 more rows
```

Example: New deaths over time

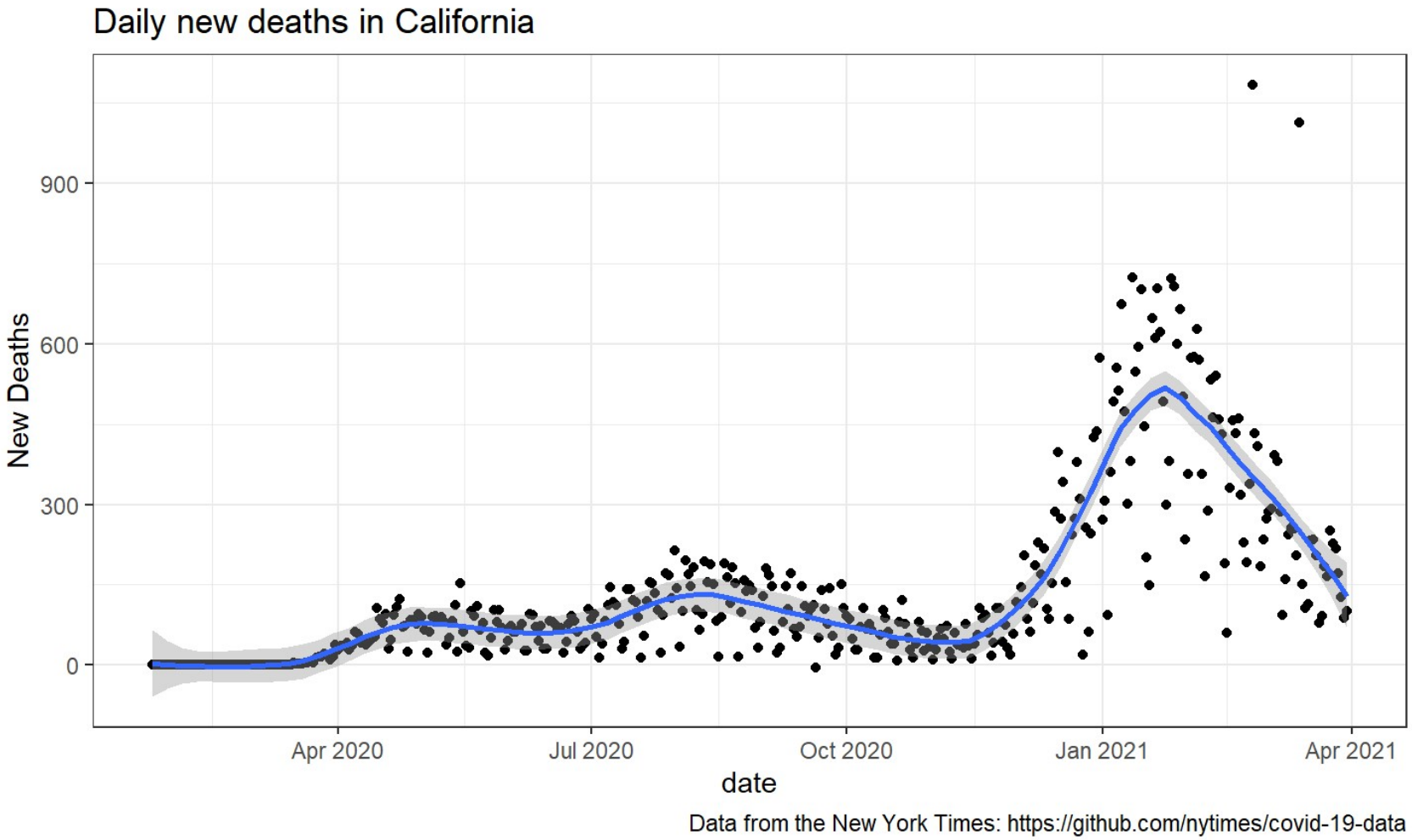
```
state_covid %>%
  filter(state == "California") %>%
  mutate(new_deaths = c(deaths[1], diff(deaths))) %>%
  ggplot(aes(date, new_deaths)) +
  geom_point()
```



Example: Adding bells and whistles to a plot

```
state_covid %>%
  filter(state == "California") %>%
  mutate(new_deaths = c(deaths[1], diff(deaths))) %>%
  ggplot(aes(date, new_deaths)) +
  geom_point() +
  geom_smooth(method = "loess", span = .2) +
  labs(title = "Daily new deaths in California",
       y = "New Deaths",
       caption = "Data from the New York Times: https://github.com/nytimes/covid-19-data") +
  theme_bw()
```

Example: COVID-19 in California



- Further enhancement with a trendline, labels, and theme
- Try plotting several states with different colors (hints: use `%in%` and `aes(color = ...)`)

Example: US COVID-19

- What states had the highest daily new deaths?

Example: US COVID-19

- What states had the highest daily new deaths?
- What do we need to do?

Example: US COVID-19

- What was the highest daily increase in death in any state?

```
state_covid %>%
  group_by(state) %>%
  mutate(new_deaths = c(deaths[1], diff(deaths))) %>%
  summarise(state_max = max(new_deaths)) %>%
  arrange(desc(state_max))
```

```
## # A tibble: 55 x 2
##    state        state_max
##    <chr>            <dbl>
##  1 Ohio              2559
##  2 New Jersey        1877
##  3 Indiana           1546
##  4 Texas             1202
##  5 California        1084
##  6 New York          1036
##  7 Georgia            466
##  8 Pennsylvania       401
##  9 Arizona            390
## 10 Virginia           383
## # ... with 45 more rows
```

- Note: `mutate()` is aware of groups!

Before switching to RStudio…

How to get help

- To look at the documentation for any function, use `?` or `help()`
  - `?summarise`
  - How to read help: https://socviz.co/appendix.html#a-little-more-about-r
- Package-specific walkthroughs: `vignette(package = "dplyr")`
- R tag on Stats Stack Exchange - stats.stackexchange.com

Resources

- R for Data Science - https://r4ds.had.co.nz/
- Cheat Sheets - https://rstudio.com/resources/cheatsheets/
- Tidy Tuesday - Weekly analysis screencasts on youtube

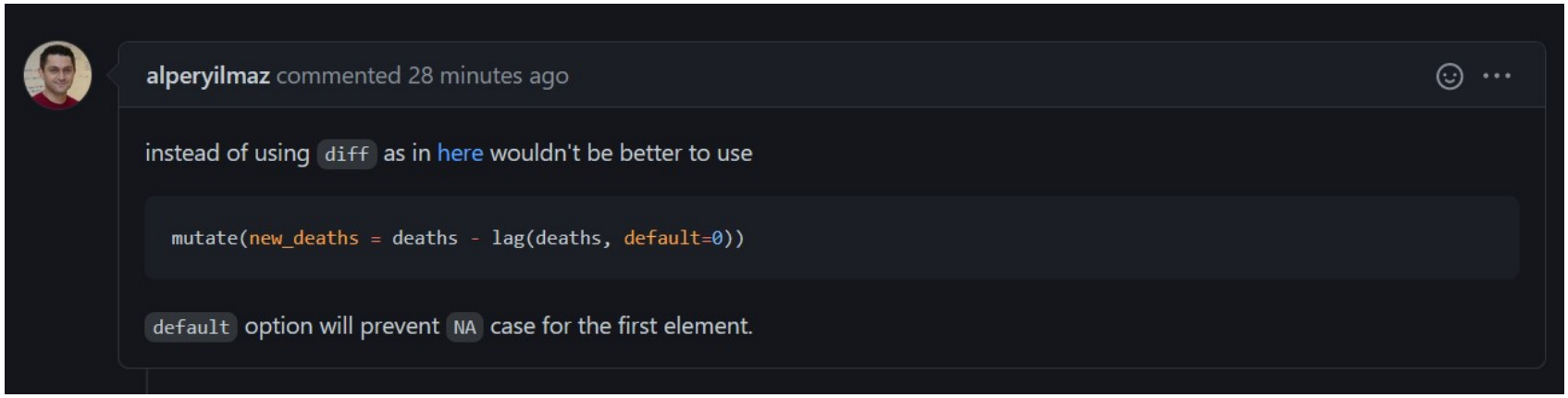
Questions?

While we wait…

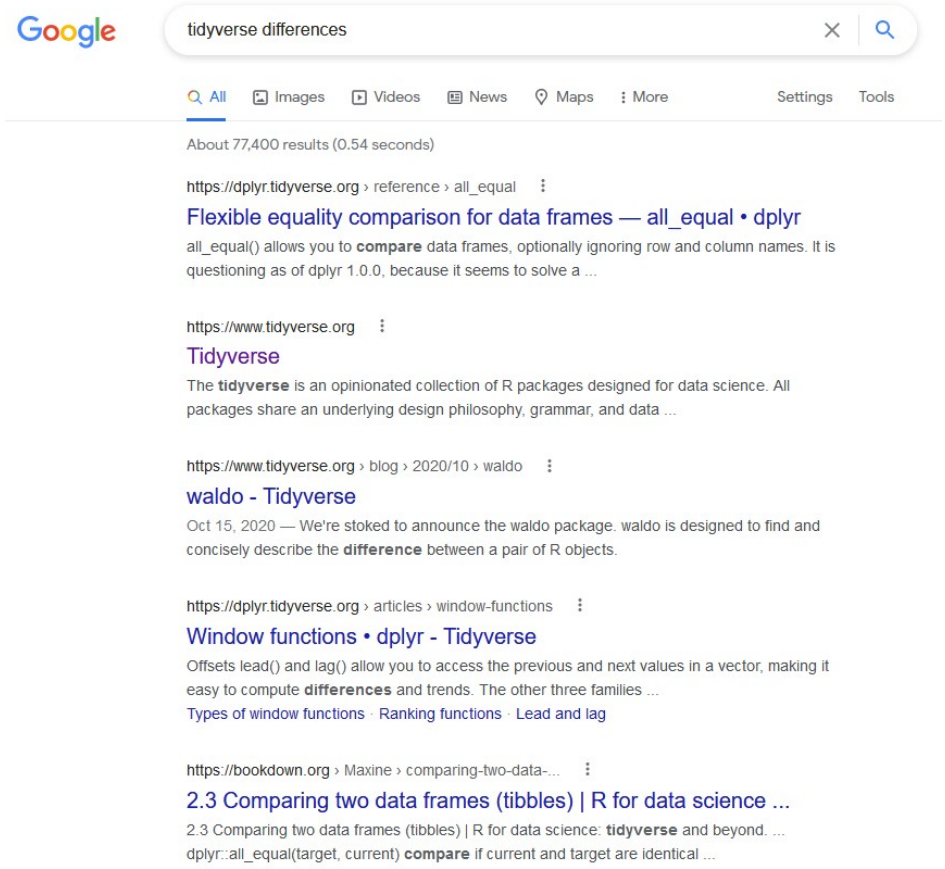
1. Install R from cran.r-project.org
2. Install Rstudio from rstudio.com
3. Install the tidyverse with this R command: `install.packages("tidyverse")`
4. If you'd like to follow along, get the slides (.html) and/or code (.R): github.com/andrewGhazi/tidy_overview

Day 2

Follow up


alperyilmaz commented 28 minutes ago
instead of using diff as in here wouldn't be better to use
mutate(new_deaths = deaths - lag(deaths, default=0))
default option will prevent NA case for the first element.

Follow up


Google
tidyverse differences
All
Images
Videos
News
Maps
More
Settings
Tools
About 77,400 results (0.54 seconds)
https://dplyr.tidyverse.org › reference › all_equal
Flexible equality comparison for data frames — all_equal • dplyr
all_equal() allows you to compare data frames, optionally ignoring row and column names. It is questioning as of dplyr 1.0.0, because it seems to solve a ...
https://www.tidyverse.org
Tidyverse
The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data ...
https://www.tidyverse.org › blog › 2020/10 › waldo
waldo - Tidyverse
Oct 15, 2020 — We're stoked to announce the waldo package. waldo is designed to find and concisely describe the difference between a pair of R objects.
https://dplyr.tidyverse.org › articles › window-functions
Window functions • dplyr - Tidyverse
Offsets lead() and lag() allow you to access the previous and next values in a vector, making it easy to compute differences and trends. The other three families ...
Types of window functions · Ranking functions · Lead and lag
https://bookdown.org › Maxine › comparing-two-data-...
2.3 Comparing two data frames (tibbles) | R for data science ...
2.3 Comparing two data frames (tibbles) | R for data science: tidyverse and beyond. ... dplyr::all_equal(target, current) compare if current and target are identical ...

Outline

- Tidying
  - `tidyr`
  - tidying example
- Iteration
  - Core concepts
  - `purrr`
- Joins

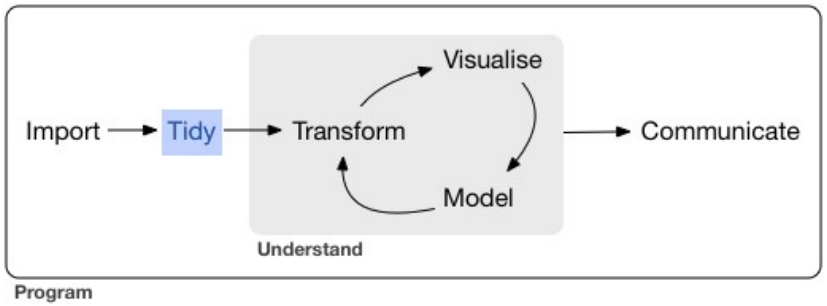


Diagram from R for Data Science by Wickham et al.

Organize - `tidyr`

- Goal: Wrangle your data into tidy format
- `separate()` and `unite()`
- `pivot_longer()` and `pivot_wider()`

`tidyr::separate()`

- Separate one column into multiple

```
tibble(id = c("subj1;ctrl", "subj2;case", "subj3;ctrl"))
```

```
## # A tibble: 3 x 1
##   id
##   <chr>
## 1 subj1;ctrl
## 2 subj2;case
## 3 subj3;ctrl
```

tidyr::separate()

- Separate one column into multiple

```
tibble(id = c("subj1;ctrl", "subj2;case", "subj3;ctrl")) %>%
  separate(id,
           into = c('subject', 'type'),
           sep = ";")
```

```
## # A tibble: 3 x 2
##   subject type
##   <chr>   <chr>
## 1 subj1   ctrl
## 2 subj2   case
## 3 subj3   ctrl
```

tidyr::unite()

- The inverse operation
- Unite multiple columns into one

```
tibble(subject = c("subj1", "subj2", "subj3"),
       type = c("ctrl", "case", "ctrl"))
```

```
## # A tibble: 3 x 2
##   subject type
##   <chr>   <chr>
## 1 subj1   ctrl
## 2 subj2   case
## 3 subj3   ctrl
```

tidyr::unite()

- Unite multiple columns into one

```
tibble(subject = c("subj1", "subj2", "subj3"),
       type = c("ctrl", "case", "ctrl"))  %>%
  unite(col = "id",
        subject, type,
        sep = ";")
```

```
## # A tibble: 3 x 1
##   id
##   <chr>
## 1 subj1;ctrl
## 2 subj2;case
## 3 subj3;ctrl
```

Core concept: wide vs long format

- Two ways of representing the same data:

`wide`

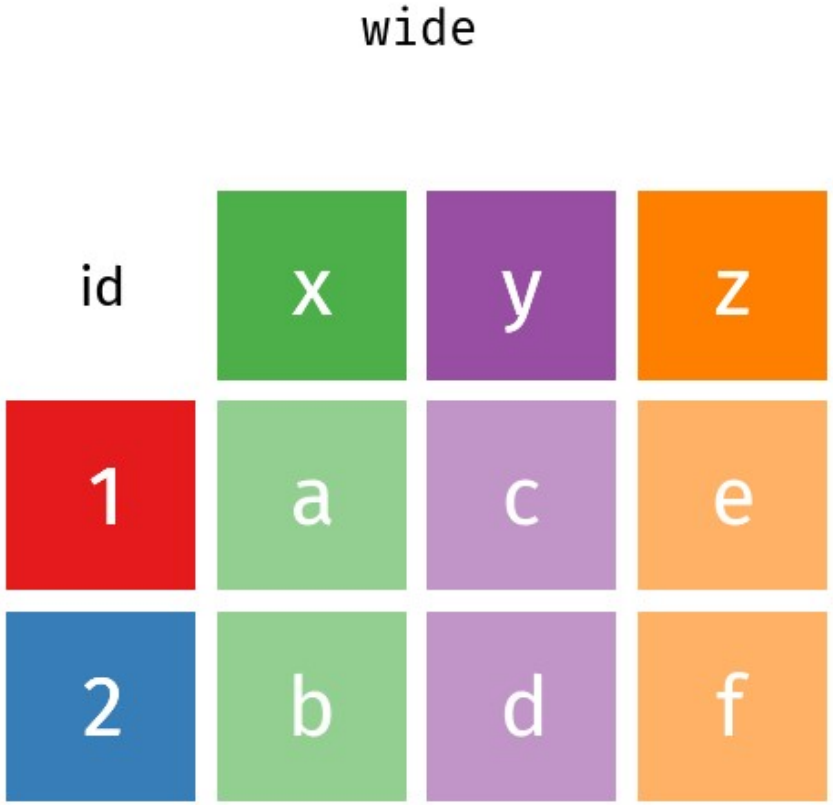
id	x	y	z
1	a	c	e
2	b	d	f

`long`

id	key	val
1	x	a
2	x	b
1	y	c
2	y	d
1	z	e
2	z	f

From https://github.com/gadenbuie/tidyexplain

`tidyr::pivot_longer()` and `pivot_wider()`



Adapted from https://github.com/gadenbuie/tidyexplain

`tidyr::pivot_longer()` and `pivot_wider()`

- Syntax:

```
wide %>%
  pivot_longer(cols = x:z,
               names_to = "key",
               values_to = "val")


long %>%
  pivot_wider(names_from = key,
              values_from = val)
```

Tidying example: WHO tuberculosis data

WHO TB data - `tidyr::who`

- realistically messy!
- We will tidy this data in 5 steps

```
dim(who)
```

```
## [1] 7240   60
```

```
who[1:4, 1:7]
```

```
## # A tibble: 4 x 7
##   country     iso2  iso3   year new_sp_m014 new_sp_m1524 new_sp_m2534
##   <chr>       <chr> <chr> <int>       <int>        <int>        <int>
## 1 Afghanistan AF    AFG    1980          NA           NA           NA
## 2 Afghanistan AF    AFG    1981          NA           NA           NA
## 3 Afghanistan AF    AFG    1982          NA           NA           NA
## 4 Afghanistan AF    AFG    1983          NA           NA           NA
```

WHO TB column names

- First part: “new” or “old”
- Second part: TB type (relapse, extrapulmonary, smear-positive, etc)
- Third part: Sex and age group

```
names(who)
```

```
##  [1] "country"      "iso2"         "iso3"         "year"         "new_sp_m014"
##  [6] "new_sp_m1524" "new_sp_m2534" "new_sp_m3544" "new_sp_m4554" "new_sp_m5564"
## [11] "new_sp_m65"   "new_sp_f014"  "new_sp_f1524" "new_sp_f2534" "new_sp_f3544"
## [16] "new_sp_f4554" "new_sp_f5564" "new_sp_f65"   "new_sn_m014"  "new_sn_m1524"
## [21] "new_sn_m2534" "new_sn_m3544" "new_sn_m4554" "new_sn_m5564" "new_sn_m65"
## [26] "new_sn_f014"  "new_sn_f1524" "new_sn_f2534" "new_sn_f3544" "new_sn_f4554"
## [31] "new_sn_f5564" "new_sn_f65"   "new_ep_m014"  "new_ep_m1524" "new_ep_m2534"
## [36] "new_ep_m3544" "new_ep_m4554" "new_ep_m5564" "new_ep_m65"   "new_ep_f014"
## [41] "new_ep_f1524" "new_ep_f2534" "new_ep_f3544" "new_ep_f4554" "new_ep_f5564"
## [46] "new_ep_f65"   "newrel_m014"  "newrel_m1524" "newrel_m2534" "newrel_m3544"
## [51] "newrel_m4554" "newrel_m5564" "newrel_m65"   "newrel_f014"  "newrel_f1524"
## [56] "newrel_f2534" "newrel_f3544" "newrel_f4554" "newrel_f5564" "newrel_f65"
```

TB tidying 1/5: pivot wide to long

```
who1 = who %>%
  pivot_longer(cols = new_sp_m014:newrel_f65,
               names_to = "key",
               values_to = "tb_cases",
               values_drop_na = TRUE)
who1
```

```
## # A tibble: 76,046 x 6
##    country     iso2  iso3   year key          tb_cases
##    <chr>       <chr> <chr> <int> <chr>           <int>
##  1 Afghanistan AF    AFG    1997 new_sp_m014         0
##  2 Afghanistan AF    AFG    1997 new_sp_m1524       10
##  3 Afghanistan AF    AFG    1997 new_sp_m2534        6
##  4 Afghanistan AF    AFG    1997 new_sp_m3544        3
##  5 Afghanistan AF    AFG    1997 new_sp_m4554        5
##  6 Afghanistan AF    AFG    1997 new_sp_m5564        2
##  7 Afghanistan AF    AFG    1997 new_sp_m65          0
##  8 Afghanistan AF    AFG    1997 new_sp_f014         5
##  9 Afghanistan AF    AFG    1997 new_sp_f1524       38
## 10 Afghanistan AF    AFG    1997 new_sp_f2534       36
## # ... with 76,036 more rows
```

TB tidying 2/5: fix inconsistent names

```
who2 = who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who2
```

```
## # A tibble: 76,046 x 6
##    country     iso2  iso3   year key          tb_cases
##    <chr>       <chr> <chr> <int> <chr>           <int>
##  1 Afghanistan AF    AFG    1997 new_sp_m014         0
##  2 Afghanistan AF    AFG    1997 new_sp_m1524       10
##  3 Afghanistan AF    AFG    1997 new_sp_m2534        6
##  4 Afghanistan AF    AFG    1997 new_sp_m3544        3
##  5 Afghanistan AF    AFG    1997 new_sp_m4554        5
##  6 Afghanistan AF    AFG    1997 new_sp_m5564        2
##  7 Afghanistan AF    AFG    1997 new_sp_m65          0
##  8 Afghanistan AF    AFG    1997 new_sp_f014         5
##  9 Afghanistan AF    AFG    1997 new_sp_f1524       38
## 10 Afghanistan AF    AFG    1997 new_sp_f2534       36
## # ... with 76,036 more rows
```

TB tidying 3/5: separate identifier components

```
who3 = who2 %>%
  separate(key, into = c("case_type", "tb_type", "sex_age"), sep = "_")
who3
```

```
## # A tibble: 76,046 x 8
##    country     iso2  iso3   year case_type tb_type sex_age tb_cases
##    <chr>       <chr> <chr> <int> <chr>     <chr>   <chr>       <int>
##  1 Afghanistan AF    AFG    1997 new       sp      m014            0
##  2 Afghanistan AF    AFG    1997 new       sp      m1524          10
##  3 Afghanistan AF    AFG    1997 new       sp      m2534           6
##  4 Afghanistan AF    AFG    1997 new       sp      m3544           3
##  5 Afghanistan AF    AFG    1997 new       sp      m4554           5
##  6 Afghanistan AF    AFG    1997 new       sp      m5564           2
##  7 Afghanistan AF    AFG    1997 new       sp      m65             0
##  8 Afghanistan AF    AFG    1997 new       sp      f014            5
##  9 Afghanistan AF    AFG    1997 new       sp      f1524          38
## 10 Afghanistan AF    AFG    1997 new       sp      f2534          36
## # ... with 76,036 more rows
```

TB tidying 4/5: separate sex and age group

```
who4 = who3 %>%
  separate(sex_age, into = c("sex", "age_group"), sep = 1)
who4
```

```
## # A tibble: 76,046 x 9
##    country     iso2  iso3   year case_type tb_type sex   age_group tb_cases
##    <chr>       <chr> <chr> <int> <chr>     <chr>   <chr> <chr>         <int>
##  1 Afghanistan AF    AFG    1997 new       sp      m     014               0
##  2 Afghanistan AF    AFG    1997 new       sp      m     1524             10
##  3 Afghanistan AF    AFG    1997 new       sp      m     2534              6
##  4 Afghanistan AF    AFG    1997 new       sp      m     3544              3
##  5 Afghanistan AF    AFG    1997 new       sp      m     4554              5
##  6 Afghanistan AF    AFG    1997 new       sp      m     5564              2
##  7 Afghanistan AF    AFG    1997 new       sp      m     65                0
##  8 Afghanistan AF    AFG    1997 new       sp      f     014               5
##  9 Afghanistan AF    AFG    1997 new       sp      f     1524             38
## 10 Afghanistan AF    AFG    1997 new       sp      f     2534             36
## # ... with 76,036 more rows
```

TB tidying 5/5: remove redundant columns

- Case types are all “new” and ISO codes are redundant

```
who_final = who4 %>%
  select(-case_type, -matches("iso"))
who_final
```

```
## # A tibble: 76,046 x 6
##    country      year tb_type sex   age_group tb_cases
##    <chr>       <int> <chr>   <chr> <chr>         <int>
##  1 Afghanistan  1997 sp      m     014               0
##  2 Afghanistan  1997 sp      m     1524             10
##  3 Afghanistan  1997 sp      m     2534              6
##  4 Afghanistan  1997 sp      m     3544              3
##  5 Afghanistan  1997 sp      m     4554              5
##  6 Afghanistan  1997 sp      m     5564              2
##  7 Afghanistan  1997 sp      m     65                0
##  8 Afghanistan  1997 sp      f     014               5
##  9 Afghanistan  1997 sp      f     1524             38
## 10 Afghanistan  1997 sp      f     2534             36
## # ... with 76,036 more rows
```

TB tidying: Final command

```
tidyr::who %>%
  pivot_longer(new_sp_m014:newrel_f65,
               names_to = "key",
               values_to = "tb_cases",
               values_drop_na = TRUE) %>%
  mutate(key = str_replace(key, "newrel", "new_rel")) %>%
  separate(key, into = c("case_type", "tb_type", "sex_age")) %>%
  separate(sex_age, into = c("sex", "age_group"), sep = 1) %>%
  select(-case_type, -matches("iso"))
```

```
## # A tibble: 76,046 x 6
##    country      year tb_type sex   age_group tb_cases
##    <chr>       <int> <chr>   <chr> <chr>        <int>
##  1 Afghanistan  1997 sp      m     014              0
##  2 Afghanistan  1997 sp      m     1524            10
##  3 Afghanistan  1997 sp      m     2534             6
##  4 Afghanistan  1997 sp      m     3544             3
##  5 Afghanistan  1997 sp      m     4554             5
##  6 Afghanistan  1997 sp      m     5564             2
##  7 Afghanistan  1997 sp      m     65               0
##  8 Afghanistan  1997 sp      f     014              5
##  9 Afghanistan  1997 sp      f     1524            38
## 10 Afghanistan  1997 sp      f     2534            36
## # ... with 76,036 more rows
```

Iteration

Core concept - user defined functions

- Syntax example:

```
sum_of_squares = function(x, y) {
  x^2 + y^2
}

sum_of_squares(3, 4)
```

```
## [1] 25
```

- If you copy and paste code more than 3 times, you probably need a function.
- Don’t repeat yourself!

Core concept - list columns

- So far the columns of our data frames have been “atomic vectors”
  - All the elements of a vector are the same “type”: logicals, character strings, integers, etc
- Lists can contain *any* type of data

```
tibble(random_samples = list(rnorm(20), rnorm(20)))
```

```
## # A tibble: 2 x 1
##   random_samples
##   <list>
## 1 <dbl [20]>
## 2 <dbl [20]>
```

Core concept - list columns

- So far the columns of our data frames have been “atomic vectors”
  - All the elements of a vector are the same “type”: logicals, character strings, integers, etc
- Lists can contain *any* type of data

```
tibble(random_samples = list(rnorm(20), rnorm(20)),
       really_anything = list(diabetes,
                              glm((diabetic == "Yes") ~ age + bmi,
                                  data = diabetes,
                                  family = 'binomial')))
```

```
## # A tibble: 2 x 2
##   random_samples really_anything
##   <list>         <list>
## 1 <dbl [20]>     <spec_tbl_df [532 x 8]>
## 2 <dbl [20]>     <glm>
```

purrr

- “Functional programming” - easier and less error-prone than for loops
- Apply functions to lists of data

purrr::map()

- Apply a function to each element of a list
- Example:

```
map(1:3, sqrt)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 1.414214
##
## [[3]]
## [1] 1.732051
```

`map()`

- Apply a function to each element of a list
- Example:

```
map(1:3, sqrt) # sqrt(1:3) is better in reality
```

- Realistic usage:

```
read_and_tidy = function(file_name) {...}
run_ml_algorithm = function(tidy_df) {...}

tidy_datasets = map(file_names, read_and_tidy)
map(tidy_datasets, run_ml_algorithm)
```

Anonymous functions

- Convenient shorthand: define a single-use function inside the call to `map`
- Use a `~` then `.x` to refer to the argument

```
map(1:3, ~sqrt(.x) + 5)
```

```
## [[1]]
## [1] 6
##
## [[2]]
## [1] 6.414214
##
## [[3]]
## [1] 6.732051
```

`map_*()` variants

- `map()` always returns a list
- Variants return a vector of a specific type

```
map_dbl(1:5, ~sqrt(.x) + 1)
```

```
## [1] 2.000000 2.414214 2.732051 3.000000 3.236068
```

```
map_lgl(1:5, ~.x == 3)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

```
map_chr(letters[1:5], ~paste(rep(.x, 3), collapse=''))
```

```
## [1] "aaa" "bbb" "ccc" "ddd" "eee"
```

purrr example: run many models

purrr example: run many models

- Want to find how to best predict diabetic status from any two biological variables

Steps:

1. Define each unique pair of biological variables
2. Run a logistic regression for each pair
3. Compute a model fit statistic for each pair
4. See which pair did the best

purrr example: run many models

- Define each combination of two explanatory variables

```
colnames(diabetes)[1:7]
```

```
## [1] "npreg" "glu"   "bp"    "skin"  "bmi"   "ped"   "age"
```

```
colnames(diabetes)[1:7] %>%
  combn(m = 2, simplify = FALSE)
```

```
## [[1]]
## [1] "npreg" "glu"
##
## [[2]]
## [1] "npreg" "bp"
##
## [[3]]
## [1] "npreg" "skin"
##
## [[4]]
## [1] "npreg" "bmi"
##
## [[5]]
## [1] "npreg" "ped"
##
```

purrr example: run many models

- Define each combination of two explanatory variables
- The modelling function needs a “formula” e.g. `is_diabetic ~ age + bp`

```
diabetes = diabetes %>% mutate(is_diabetic = diabetic == "Yes")

formula_df = colnames(diabetes)[1:7] %>%
  combn(m = 2, simplify = FALSE) %>%
  map_chr(~paste("is_diabetic ~ ", .x[1], " + ", .x[2], sep = "")) %>%
  tibble(formula = .)
formula_df
```

```
## # A tibble: 21 x 1
##    formula
##    <chr>
##  1 is_diabetic ~ npreg + glu
##  2 is_diabetic ~ npreg + bp
##  3 is_diabetic ~ npreg + skin
##  4 is_diabetic ~ npreg + bmi
##  5 is_diabetic ~ npreg + ped
##  6 is_diabetic ~ npreg + age
##  7 is_diabetic ~ glu + bp
##  8 is_diabetic ~ glu + skin
##  9 is_diabetic ~ glu + bmi
## 10 is_diabetic ~ glu + ped
```

purrr example: run many models

- Run a logistic regression on each combination of explanatory variables

```
formula_df %>%
  mutate(glm_result = map(formula,
                          ~glm(as.formula(.x), data = diabetes, family = 'binomial')))
```

```
## # A tibble: 21 x 2
##    formula                    glm_result
##    <chr>                      <list>
##  1 is_diabetic ~ npreg + glu  <glm>
##  2 is_diabetic ~ npreg + bp   <glm>
##  3 is_diabetic ~ npreg + skin <glm>
##  4 is_diabetic ~ npreg + bmi  <glm>
##  5 is_diabetic ~ npreg + ped  <glm>
##  6 is_diabetic ~ npreg + age  <glm>
##  7 is_diabetic ~ glu + bp     <glm>
##  8 is_diabetic ~ glu + skin   <glm>
##  9 is_diabetic ~ glu + bmi    <glm>
## 10 is_diabetic ~ glu + ped    <glm>
## # ... with 11 more rows
```

purrr example: run many models

- Extract the AIC of each model fit

```
formula_df %>%
  mutate(glm_result = map(formula,
                          ~glm(.x, data = diabetes, family = 'binomial')),
         aic = map_dbl(glm_result,
                       ~.x$aic))
```

```
## # A tibble: 21 x 3
##    formula                    glm_result   aic
##    <chr>                      <list>     <dbl>
##  1 is_diabetic ~ npreg + glu  <glm>       517.
##  2 is_diabetic ~ npreg + bp   <glm>       639.
##  3 is_diabetic ~ npreg + skin <glm>       619.
##  4 is_diabetic ~ npreg + bmi  <glm>       597.
##  5 is_diabetic ~ npreg + ped  <glm>       620.
##  6 is_diabetic ~ npreg + age  <glm>       629.
##  7 is_diabetic ~ glu + bp     <glm>       537.
##  8 is_diabetic ~ glu + skin   <glm>       524.
##  9 is_diabetic ~ glu + bmi    <glm>       517.
## 10 is_diabetic ~ glu + ped    <glm>       523.
## # ... with 11 more rows
```

purrr example: run many models

- See which model had the lowest AIC

```
formula_df %>%
  mutate(glm_result = map(formula,
                          ~glm(.x, data = diabetes, family = 'binomial')),
         aic = map_dbl(glm_result,
                       ~.x$aic)) %>%
  arrange(aic)
```

```
## # A tibble: 21 x 3
##    formula                   glm_result   aic
##    <chr>                     <list>     <dbl>
##  1 is_diabetic ~ npreg + glu <glm>       517.
##  2 is_diabetic ~ glu + bmi   <glm>       517.
##  3 is_diabetic ~ glu + age   <glm>       519.
##  4 is_diabetic ~ glu + ped   <glm>       523.
##  5 is_diabetic ~ glu + skin  <glm>       524.
##  6 is_diabetic ~ glu + bp    <glm>       537.
##  7 is_diabetic ~ bmi + age   <glm>       583.
##  8 is_diabetic ~ npreg + bmi <glm>       597.
##  9 is_diabetic ~ skin + age  <glm>       604.
## 10 is_diabetic ~ ped + age   <glm>       606.
## # ... with 11 more rows
```

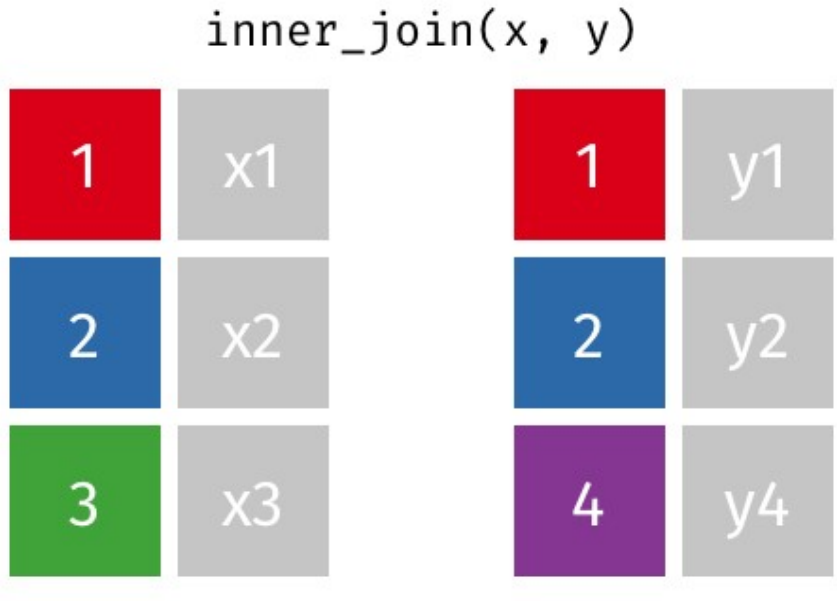
Joins

Core concept - Joins

- Common data structure: multiple data frames of related data.
- Example:
  - patient smoking status, age, weight in DF1
  - patient experimental measurements in DF2
  - both have a `patient_id` column
- Need both information types together in one mega data frame
- There are many types joins. We'll examine two.

Inner joins

- Merge two data frames by a common “key”



From https://github.com/gadenbuie/tidyexplain

dplyr::inner_join()

```
band_members
```

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
band_instruments
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

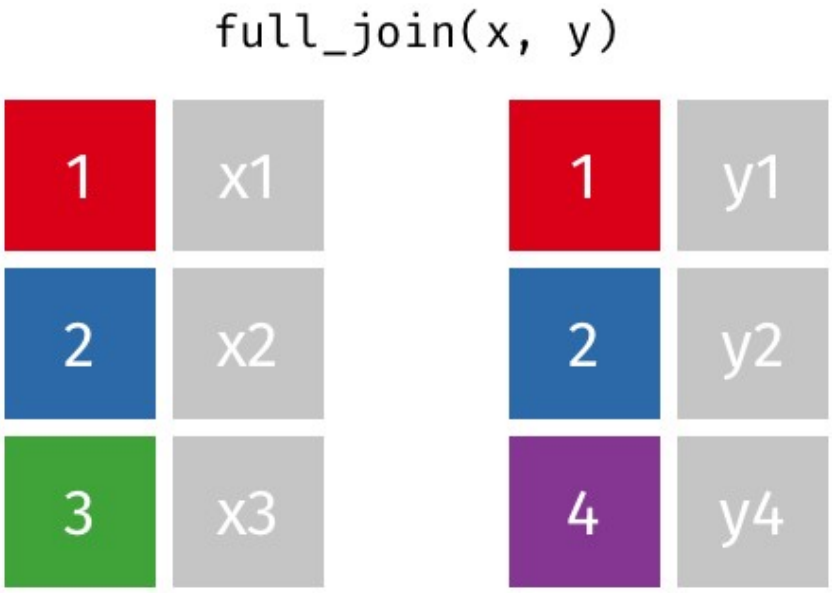
dplyr::inner_join()

```
band_members %>%
  inner_join(band_instruments, by = "name")
```

```
## # A tibble: 2 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

Full joins

- Merge two data frames by a common “key”



From https://github.com/gadenbuie/tidyexplain

dplyr::full_join()

```
band_members %>% full_join(band_instruments, by = 'name')
```

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  NA
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith NA      guitar
```

Wrap up

Galactic core:

- `forcats` - work with factors
- `stringr` - work with strings
- `lubridate` - work with dates
- `rvest` - web scraping

Outer rim:

- `tidymodels` - work with stats/ML
- `reticulate` - work with Python

How to get help

- To look at the documentation for any function, use `?` or `help()`
  - How to read help: https://socviz.co/appendix.html#a-little-more-about-r
- Package-specific walkthroughs:`vignette(package = "dplyr")`
- R tag on Stats Stack Exchange - stats.stackexchange.com
- My email: aghazi@broadinstitute.org

Resources

- R for Data Science - https://r4ds.had.co.nz/
- Cheat Sheets - https://rstudio.com/resources/cheatsheets/
- Tidy Tuesday - Weekly analysis screencasts on youtube