# Data types and objects

John Tipton

# Readings

- R for data science
  - Introduction
  - Chapters 10 (Relational data with `dplyr`), 11 (Strings with `stringr`), 12 (Factors with `forcats`), and 13 (Dates and times with `lubridate`)

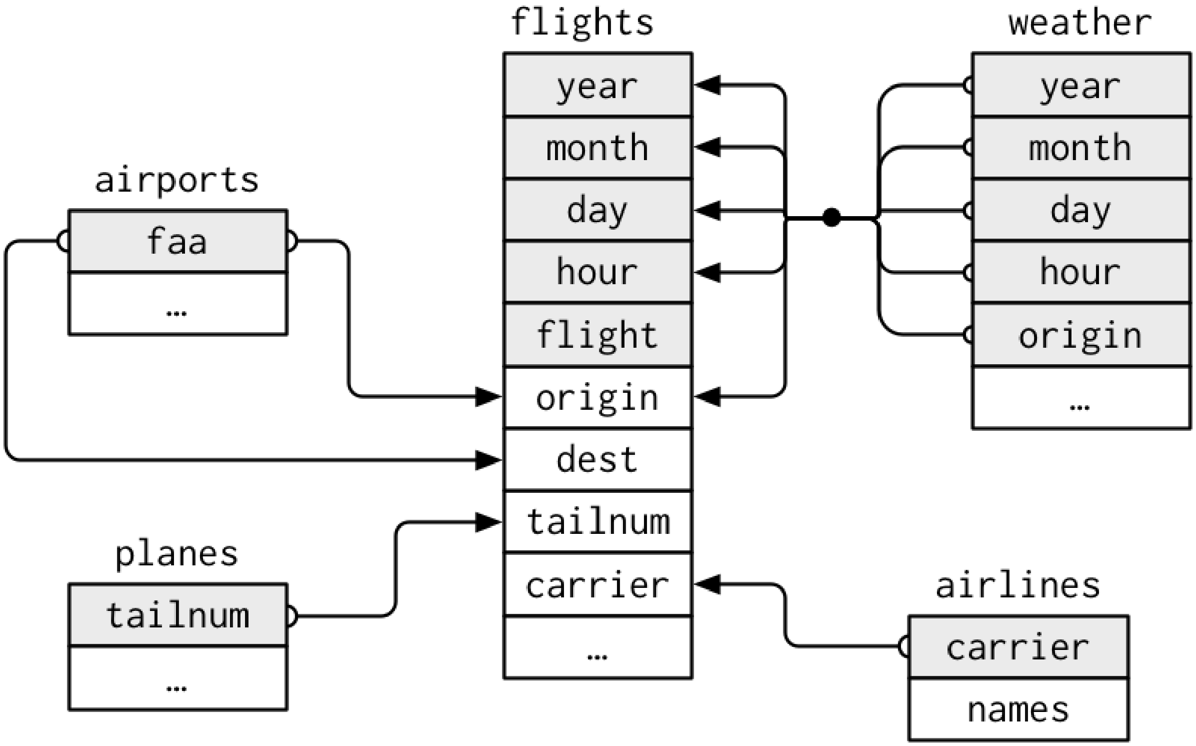
# Relational data

- Multiple tables of data where the relationships between the data matter
- Common in SQL and other database software
- Common operations include mutating joins, filtering joins, and set operations
- Example: `nycflights13` relational data

```
library(tidyverse)
library(nycflights13)
```

# Examine the data

- The data sets include `airlines`, `airports`, `planes` and `weather`



# Relational data

- The `weather` data is connected to `flights` data by the variables `year`, `month`, `day`, `hour`, and `origin`
- The `planes` data is connected to the `flights` data by the `tailnum` variable
- The `airports` data is connected to the `flights` data by the `faa` variable
- The `airlines` data is connected to the `flights` data by the `carrier` variable

# Relational data

- The variables that connect two or more data tables are called **keys**
    - **primary keys** uniquely identifies an observation in its own table
    - **foreign keys** uniquely identifies an observation in another table
- A quick check if a variable is a key is to count it

```
planes %>%
  count(tailnum) %>%
  filter(n > 1)
```

```
## # A tibble: 0 × 2
## # … with 2 variables: tailnum <chr>, n <int>
```

- The `tailnum` variable never shows up more than once in the `planes` data.

```
airlines %>%
  count(carrier) %>%
  filter(n > 1)
```

```
## # A tibble: 0 × 2
## # … with 2 variables: carrier <chr>, n <int>
```

- The `carrier` variable never shows up more than once in the `airlines` data

# Question: What is the key for `flights`?

# The `flights` key

- There isn't a key

```
flights %>%
  count(flight) %>% ## the flight number gets reused
  filter(n > 1)
```

```
## # A tibble: 3,493 × 2
##    flight     n
##     <int> <int>
##  1      1   701
##  2      2    51
##  3      3   631
##  4      4   393
##  5      5   324
##  6      6   210
##  7      7   237
##  8      8   236
##  9      9   153
## 10     10    61
## # … with 3,483 more rows
```

# The `flights` key

- There isn't a key

```
flights %>%
  count(year, month, day, flight) %>% ## the flight number gets reused
  filter(n > 1)
```

```
## # A tibble: 29,768 × 5
##     year month   day flight     n
##    <int> <int> <int>  <int> <int>
##  1  2013     1     1      1     2
##  2  2013     1     1      3     2
##  3  2013     1     1      4     2
##  4  2013     1     1     11     3
##  5  2013     1     1     15     2
##  6  2013     1     1     21     2
##  7  2013     1     1     27     4
##  8  2013     1     1     31     2
##  9  2013     1     1     32     2
## 10  2013     1     1     35     2
## # … with 29,758 more rows
```

# The `flights` key

- There isn't a key

```
flights %>%
  count(year, month, day, flight, tailnum) %>% ## the flight number gets reused
  filter(n > 1)
```

```
## # A tibble: 11 × 6
##     year month   day flight tailnum     n
##    <int> <int> <int>  <int> <chr>   <int>
##  1  2013     2     9    303 <NA>        2
##  2  2013     2     9    655 <NA>        2
##  3  2013     2     9   1623 <NA>        2
##  4  2013     6     8   2269 N487WN      2
##  5  2013     6    15   2269 N230WN      2
##  6  2013     6    22   2269 N440LV      2
##  7  2013     6    29   2269 N707SA      2
##  8  2013     7     6   2269 N259WN      2
##  9  2013     8     3   2269 N446WN      2
## 10  2013     8    10   2269 N478WN      2
## 11  2013    12    15    398 <NA>        2
```

# The `flights` key

- If there isn't a key, you can make one (called a **surrogate key**)

```
flights %>%
  mutate(row_number = row_number())
```

```
## # A tibble: 336,776 × 20
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # … with 336,766 more rows, and 13 more variables:
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>, row_number <int>
```

```
flights %>%
  mutate(row_number = row_number()) %>%
  count(row_number) %>%
  filter(n > 1)
```

# Mutating joins

- Combines variable from two data tables

```
# create a smaller dataset from flights
flights_smaller <- flights %>%
  select(year:day, hour, origin)
```

```
flights_smaller
```

```
## # A tibble: 336,776 × 5
##     year month   day  hour origin
##    <int> <int> <int> <dbl> <chr>
##  1  2013     1     1     5 EWR
##  2  2013     1     1     5 LGA
##  3  2013     1     1     5 JFK
##  4  2013     1     1     5 JFK
##  5  2013     1     1     6 LGA
##  6  2013     1     1     5 EWR
##  7  2013     1     1     6 EWR
##  8  2013     1     1     6 LGA
##  9  2013     1     1     6 JFK
## 10  2013     1     1     6 LGA
## # … with 336,766 more rows
```

# Mutating joins

- Combines variables from two data tables
- Add variable columns to a dataset

```
flights_smaller %>%
  left_join(weather, by = c("year", "month", "day", "hour", "origin"))
```

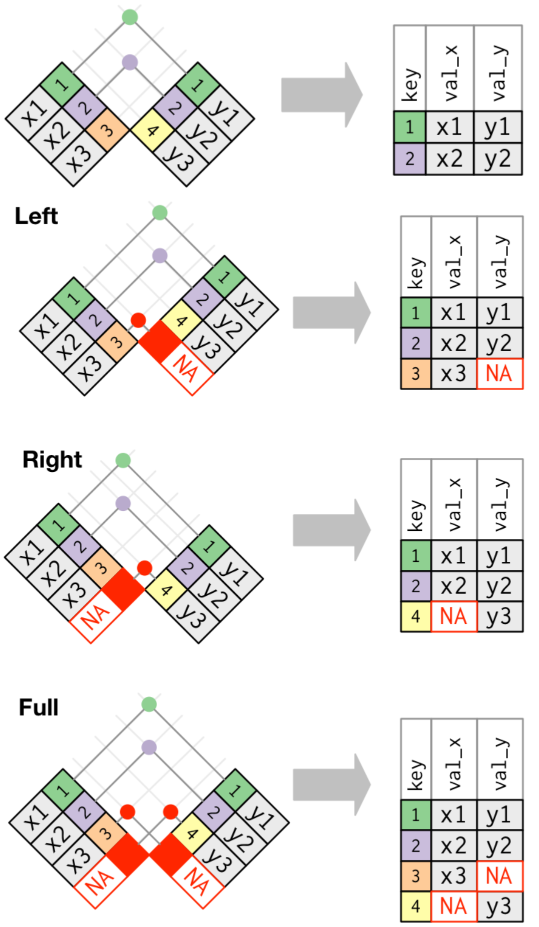
```
## # A tibble: 336,776 × 15
##     year month   day  hour origin  temp  dewp humid wind_dir
##    <int> <int> <int> <dbl> <chr>  <dbl> <dbl> <dbl>    <dbl>
##  1  2013     1     1     5 EWR     39.0  28.0  64.4      260
##  2  2013     1     1     5 LGA     39.9  25.0  54.8      250
##  3  2013     1     1     5 JFK     39.0  27.0  61.6      260
##  4  2013     1     1     5 JFK     39.0  27.0  61.6      260
##  5  2013     1     1     6 LGA     39.9  25.0  54.8      260
##  6  2013     1     1     5 EWR     39.0  28.0  64.4      260
##  7  2013     1     1     6 EWR     37.9  28.0  67.2      240
##  8  2013     1     1     6 LGA     39.9  25.0  54.8      260
##  9  2013     1     1     6 JFK     37.9  27.0  64.3      260
## 10  2013     1     1     6 LGA     39.9  25.0  54.8      260
## # … with 336,766 more rows, and 6 more variables: wind_speed <dbl>,
## #   wind_gust <dbl>, precip <dbl>, pressure <dbl>, visib <dbl>,
## #   time_hour <dttm>
```

- Weather data appended to the `flights_smaller` data

# Types of joins

- Left join (join the two data sets keeping all variables in the left data set)
- Right join (join the two data sets keeping all variables in the right data set)
- Inner join (join the two data sets keeping all variables that are in both data sets)
    - Typically this is poor as it drops missing values silently
    - Missing values are important to keep track of -- only drop missing values intentionally
- Outer join (join the two data sets keeping all variables that are in either data set)

# Joins



| key | val_x | val_y |
|---|---|---|
| 1 | x1 | y1 |
| 2 | x2 | y2 |

Left

| key | val_x | val_y |
|---|---|---|
| 1 | x1 | y1 |
| 2 | x2 | y2 |
| 3 | x3 | NA |

Right

| key | val_x | val_y |
|---|---|---|
| 1 | x1 | y1 |
| 2 | x2 | y2 |
| 4 | NA | y3 |

Full

| key | val_x | val_y |
|---|---|---|
| 1 | x1 | y1 |
| 2 | x2 | y2 |
| 3 | x3 | NA |
| 4 | NA | y3 |

# Example data

```r
dat_x <- tribble(
  ~key, ~x,
  1, "x1",
  2, "x2",
  3, "x3",
  4, "x1",
  5, "x2",
  6, "x3",
)
```

```r
dat_y <- tribble(
  ~key, ~y,
  1, "y1",
  2, "y2",
  3, "y3",
  4, "y1",
  7, "y2",
  8, "y3",
)
```

# Inner Join

```
dat_x %>%
  inner_join(dat_y, by = "key")
```

```
## # A tibble: 4 × 3
##     key x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    y1   
## 2     2 x2    y2   
## 3     3 x3    y3   
## 4     4 x1    y1   
```

- Notice that the variables with keys 5-8 have been dropped
- Half of the data is now missing without any notice!

# Outer joins

- Left joins
  - Keeps all of the observations in the "left" dataset

```
dat_x %>%
  left_join(dat_y, by = "key")
```

```
## # A tibble: 6 × 3
##     key x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    y1   
## 2     2 x2    y2   
## 3     3 x3    y3   
## 4     4 x1    y1   
## 5     5 x2    <NA> 
## 6     6 x3    <NA> 
```

- Notice how all the observations in `dat_x` are kept

# Outer joins

- Right joins
  - Keeps all of the observations in the "right" dataset

```
dat_x %>%
  right_join(dat_y, by = "key")
```

```
## # A tibble: 6 × 3
##     key x     y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    y3
## 4     4 x1    y1
## 5     7 <NA>  y2
## 6     8 <NA>  y3
```

- Notice how all the observations in `dat_y` are kept

# Outer joins

- Full joins
  - Keeps all of the observations in both datasets

```
dat_x %>%
  full_join(dat_y, by = "key")
```

```
## # A tibble: 8 × 3
##     key x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    y1   
## 2     2 x2    y2   
## 3     3 x3    y3   
## 4     4 x1    y1   
## 5     5 x2    <NA> 
## 6     6 x3    <NA> 
## 7     7 <NA>  y2   
## 8     8 <NA>  y3   
```

- Notice how all the observations in `dat_x` and `dat_y` are kept

# Duplicate keys

- Typically, there should not be duplicate keys, but this isn't always true

```r
dat_x <- tribble(
  ~key, ~x,
  1, "x1",
  2, "x2",
  3, "x3",
  4, "x1",
  1, "x2",
  2, "x3",
)
```

```r
dat_y <- tribble(
  ~key, ~y,
  1, "y1",
  2, "y2",
  3, "y3",
  5, "y5",
)
```

# Left Joins with duplicate keys

```
dat_x %>%
  left_join(dat_y, by = "key")
```

```
## # A tibble: 6 × 3
##     key x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    y1   
## 2     2 x2    y2   
## 3     3 x3    y3   
## 4     4 x1    <NA> 
## 5     1 x2    y1   
## 6     2 x3    y2   
```

- Notice that we get all combinations (Cartesian product) of the variables keeping all observations in `dat_x`

# Outer joins

- Right joins
    - Keeps all of the observations in the "right" dataset

```
dat_x %>%
  right_join(dat_y, by = "key")
```

```
## # A tibble: 6 × 3
##     key x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    y1   
## 2     2 x2    y2   
## 3     3 x3    y3   
## 4     1 x2    y1   
## 5     2 x3    y2   
## 6     5 <NA>  y5   
```

- Notice how all the observations in `dat_y` are kept

# Outer joins

- Full joins
  - Keeps all of the observations in both datasets

```
dat_x %>%
  full_join(dat_y, by = "key")
```

```
## # A tibble: 7 × 3
##     key x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    y1   
## 2     2 x2    y2   
## 3     3 x3    y3   
## 4     4 x1    <NA> 
## 5     1 x2    y1   
## 6     2 x3    y2   
## 7     5 <NA>  y5   
```

- Notice how all the observations in `dat_x` and `dat_y` are kept

# Defining the key

- Most of the time, the key variable (or variables) are not given the name `key`
- A natural join uses all of the variables in the two dataframes that are in common

```
flights_reduced <- flights %>%
  select(year:day, hour, origin, dest, tailnum, carrier)
flights_reduced
```

```
## # A tibble: 336,776 × 8
##     year month   day  hour origin dest  tailnum carrier
##    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>  
##  1  2013     1     1     5 EWR    IAH   N14228  UA     
##  2  2013     1     1     5 LGA    IAH   N24211  UA     
##  3  2013     1     1     5 JFK    MIA   N619AA  AA     
##  4  2013     1     1     5 JFK    BQN   N804JB  B6     
##  5  2013     1     1     6 LGA    ATL   N668DN  DL     
##  6  2013     1     1     5 EWR    ORD   N39463  UA     
##  7  2013     1     1     6 EWR    FLL   N516JB  B6     
##  8  2013     1     1     6 LGA    IAD   N829AS  EV     
##  9  2013     1     1     6 JFK    MCO   N593JB  B6     
## 10  2013     1     1     6 LGA    ORD   N3ALAA  AA     
## # … with 336,766 more rows
```

# Natural join

- What are the key variables used here?

```
flights_reduced %>%
  left_join(weather)
```

```
## Joining, by = c("year", "month", "day", "hour", "origin")

## # A tibble: 336,776 × 18
##     year month   day  hour origin dest  tailnum carrier  temp  dewp
##    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl> <dbl>
##  1  2013     1     1     5 EWR    IAH   N14228  UA       39.0  28.0
##  2  2013     1     1     5 LGA    IAH   N24211  UA       39.9  25.0
##  3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0  27.0
##  4  2013     1     1     5 JFK    BQN   N804JB  B6       39.0  27.0
##  5  2013     1     1     6 LGA    ATL   N668DN  DL       39.9  25.0
##  6  2013     1     1     5 EWR    ORD   N39463  UA       39.0  28.0
##  7  2013     1     1     6 EWR    FLL   N516JB  B6       37.9  28.0
##  8  2013     1     1     6 LGA    IAD   N829AS  EV       39.9  25.0
##  9  2013     1     1     6 JFK    MCO   N593JB  B6       37.9  27.0
## 10  2013     1     1     6 LGA    ORD   N3ALAA  AA       39.9  25.0
## # … with 336,766 more rows, and 8 more variables: humid <dbl>,
## #   wind_dir <dbl>, wind_speed <dbl>, wind_gust <dbl>, precip <dbl>,
## #   pressure <dbl>, visib <dbl>, time_hour <dttm>
```

- Uses the variables in common between the datasets

# Natural join

```
colnames(flights_reduced)
```

```
## [1] "year"    "month"   "day"     "hour"    "origin"  "dest"   
## [7] "tailnum" "carrier"
```

```
colnames(weather)
```

```
##  [1] "origin"     "year"       "month"      "day"        "hour"      
##  [6] "temp"       "dewp"       "humid"      "wind_dir"   "wind_speed"
## [11] "wind_gust"  "precip"     "pressure"   "visib"      "time_hour"
```

- What variable names are in common

```
intersect(colnames(flights_reduced), colnames(weather))
```

```
## [1] "year"   "month"  "day"    "hour"   "origin"
```

# Joining with keys

- left join `flights_reduced` and `planes`
- Notice the output of the `years` variables

```
flights_reduced %>%
  left_join(planes, by = "tailnum")
```

```
## # A tibble: 336,776 × 16
##    year.x month   day  hour origin dest  tailnum carrier year.y type 
##     <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>    <int> <chr>
##  1   2013     1     1     5 EWR    IAH   N14228  UA        1999 Fixe…
##  2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixe…
##  3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixe…
##  4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixe…
##  5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixe…
##  6   2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixe…
##  7   2013     1     1     6 EWR    FLL   N516JB  B6        2000 Fixe…
##  8   2013     1     1     6 LGA    IAD   N829AS  EV        1998 Fixe…
##  9   2013     1     1     6 JFK    MCO   N593JB  B6        2004 Fixe…
## 10   2013     1     1     6 LGA    ORD   N3ALAA  AA          NA <NA>
## # … with 336,766 more rows, and 6 more variables:
## #   manufacturer <chr>, model <chr>, engines <int>, seats <int>,
## #   speed <int>, engine <chr>
```

# Joining with keys

- right join `flights_reduced` and `planes`
- Notice the output of the `years` variables

```r
flights_reduced %>%
  right_join(planes, by = "tailnum")
```

```
## # A tibble: 284,170 × 16
##    year.x month   day  hour origin dest  tailnum carrier year.y type 
##     <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>    <int> <chr>
##  1   2013     1     1     5 EWR    IAH   N14228  UA        1999 Fixe…
##  2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixe…
##  3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixe…
##  4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixe…
##  5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixe…
##  6   2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixe…
##  7   2013     1     1     6 EWR    FLL   N516JB  B6        2000 Fixe…
##  8   2013     1     1     6 LGA    IAD   N829AS  EV        1998 Fixe…
##  9   2013     1     1     6 JFK    MCO   N593JB  B6        2004 Fixe…
## 10   2013     1     1     6 JFK    PBI   N793JB  B6        2011 Fixe…
## # … with 284,160 more rows, and 6 more variables:
## #   manufacturer <chr>, model <chr>, engines <int>, seats <int>,
## #   speed <int>, engine <chr>
```

# Joining with keys with different names

- `flights_reduced` has variable `dest`, `airports` has the variable `faa`
- left join `flights_reduced` (`dest` variable) and `airports` (`faa` variable)

```
flights_reduced %>%
  left_join(airports, by = c("dest" = "faa"))
```

```
## # A tibble: 336,776 × 15
##     year month   day  hour origin dest  tailnum carrier name      lat
##    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>   <dbl>
##  1  2013     1     1     5 EWR    IAH   N14228  UA      George…  30.0
##  2  2013     1     1     5 LGA    IAH   N24211  UA      George…  30.0
##  3  2013     1     1     5 JFK    MIA   N619AA  AA      Miami …  25.8
##  4  2013     1     1     5 JFK    BQN   N804JB  B6      <NA>     NA  
##  5  2013     1     1     6 LGA    ATL   N668DN  DL      Hartsf…  33.6
##  6  2013     1     1     5 EWR    ORD   N39463  UA      Chicag…  42.0
##  7  2013     1     1     6 EWR    FLL   N516JB  B6      Fort L…  26.1
##  8  2013     1     1     6 LGA    IAD   N829AS  EV      Washin…  38.9
##  9  2013     1     1     6 JFK    MCO   N593JB  B6      Orland…  28.4
## 10  2013     1     1     6 LGA    ORD   N3ALAA  AA      Chicag…  42.0
## # … with 336,766 more rows, and 5 more variables: lon <dbl>,
## #   alt <dbl>, tz <dbl>, dst <chr>, tzone <chr>
```

# Filtering joins

- Filter joins drop observations that are missing (typically not used much)
- `semi_join(x, y)` keeps all observations in `x` that have a match in `y`
  - Drops all observations in `x` that don't have a match in `y`
  - Does not duplicate observations (no Cartesian products)
- `anti_join(x, y)` drops all observations in `x` that have a match in `y`
  - useful for diagnosing join errors

# Working with joins

- Identifying keys
    - Use your knowledge of the data
    - Trying to identify keys based on the data values can lead to red herrings

# Strings

```
string1 <- "This is a string"
string2 <- 'This is also a string'
string3 <- 'This is a "quoted" string'
```

```
string1
```

```
## [1] "This is a string"
```

```
string2
```

```
## [1] "This is also a string"
```

```
string3
```

```
## [1] "This is a \"quoted\" string"
```

# Strings - Escape characters

- The escape character for strings is `\`
- Note: printed strings show the escape characters, not the string

```
print(string3)
```

```
## [1] "This is a \"quoted\" string"
```

- To show the actual string, use `writelines`

```
writeLines(string3)
```

```
## This is a "quoted" string
```

- Other special characters include `\n`, `\t`

# Strings

- Join strings into a vector using `c()`

```
c(string1, string2, string3)
```

```
## [1] "This is a string"            "This is also a string"
## [3] "This is a \"quoted\" string"
```

- String length (number of characters) using `str_length()`

```
str_length(c(string1, string2, string3, NA))
```

```
## [1] 16 21 25 NA
```

- Join strings together using `str_c()`

```
str_c(string1, string2, string3)
```

```
## [1] "This is a stringThis is also a stringThis is a \"quoted\" string"
```

- Join strings with a separator character

```
str_c(string1, string2, sep = "\n")
```

```
## [1] "This is a string\nThis is also a string"
```

# Strings

- Missing values (`NA`) are contagious
- The `str_c()` function (like most R functions) are vectorized

```
str_c("abc", c("var", NA), "def", sep = "-")
```

```
## [1] "abc-var-def" NA
```

- Collapse a vector of strings using `collapse`

```
str_c(c("abc", "def", "ghi"), collapse = ", ")
```

```
## [1] "abc, def, ghi"
```

# Subsetting strings

- Subset strings using `str_sub()`

```
beatles <- c("John", "Paul", "Ringo", "George")
# select the 1st through 3rd characters, inclusive
str_sub(beatles, 1, 3)
```

```
## [1] "Joh" "Pau" "Rin" "Geo"
```

```
# select the last through 3rd from last characters, inclusive
str_sub(beatles, -3, -1)
```

```
## [1] "ohn" "aul" "ngo" "rge"
```

- Convert string to lower case

```
str_to_lower(beatles)
```

```
## [1] "john"   "paul"   "ringo"  "george"
```

- Can use subsetting in assignment

```
str_sub(beatles, 1, 1) <- str_to_lower(str_sub(beatles, 1, 1))
beatles
```

```
## [1] "john"   "paul"   "ringo"  "george"
```

# Sting manipulation

- `str_to_lower()`

```
str_to_lower(beatles)
```

```
## [1] "john"   "paul"   "ringo"  "george"
```

- `str_to_upper()`

```
str_to_upper(beatles)
```

```
## [1] "JOHN"   "PAUL"   "RINGO"  "GEORGE"
```

- `str_to_title()`

```
str_to_title(beatles)
```

```
## [1] "John"   "Paul"   "Ringo"  "George"
```

- `str_sort()`

```
str_sort(beatles)
```

```
## [1] "george" "john"   "paul"   "ringo"
```

- `str_order()`

```
str_order(beatles)
```

```
## [1] 4 1 2 3
```

# Strings and searching

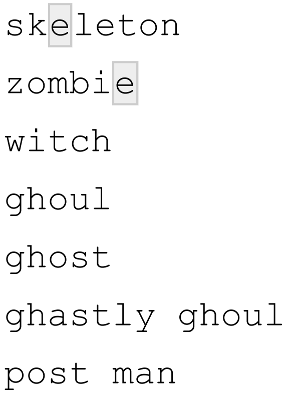
- To help troubleshoot string searches, the functions `str_view()` and `str_view_all()` are useful

```
costumes <- c("skeleton", "zombie", "witch", "ghoul", "ghost", "ghastly ghoul", "post ma
str_view(costumes, "gh")
```

```
skeleton
zombie
witch
ghoul
ghost
ghastly ghoul
post man
```

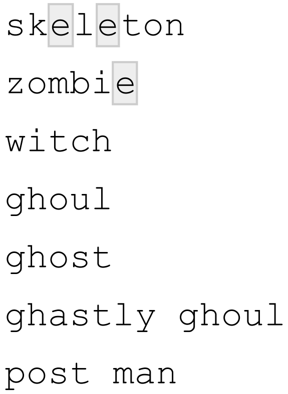
# Strings and searching

```
str_view(costumes, "e")
```



```
skeleton
zombie
witch
ghoul
ghost
ghastly ghoul
post man
```

```
str_view_all(costumes, "e")
```



```
skeleton
zombie
witch
ghoul
ghost
ghastly ghoul
post man
```

# Regular expressions

- The `.` character matches any character (except `\n`)

```
str_view(costumes, ".h.")
```

```
skeleton
zombie
witch
ghoul
ghost
ghastly ghoul
post man
```

# Regular expressions

- How do you search for a `.`?
  - Remember `\` is an escape character
  - To create the regular expression `\.`, you need to escape the escape `\`
  - To search for a `.`, you need the expression `\\.`

```
dot <- "\\."
writeLines(dot)
```

```
## \.
```

```
str_view(c("a.c", "d.f", "ghi"), "\\.")
```

```
a.c
d.f
ghi
```

# Regular expressions

- How do you search for a `\`?
  - `\\\\`
- Search at the beginning of a string with `^`
- Search at the end of a string with `$`

```
str_view(costumes, "^gho")
```

```
skeleton
zombie
witch
ghoul
ghost
ghastly ghoul
post man
```

```
str_view(costumes, "ost$")
```

```
skeleton
zombie
witch
ghoul
ghost
ghastly ghoul
post man
```

# Regular expressions

- Character classes
  - `\d` - digit
  - `\s` - whitespace
  - `\S` - non-whitespace
  - `[abc]` - matches a, b, or c
  - `[^abc]` - matches everything except a, b, or c
- Recall: `\` is an escape so you need to match `\\d` or `\\s` or `\\S`
- Repetition
  - `?` - zero or one
  - `+` - one or more
  - `*` - zero or more
  - `{n}` - exactly n matches
  - `{n, }` - n or more matches
  - `{, n}` - no more than n matches
  - `{n, m}` - between n and m matches
- Groupings
  - `(abc)` searches for the string "abc

# String detection

- Detect whether a string matches a pattern with `str_detect()`

```
str_detect(costumes, "gh")
```

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

- How many words in the `stringr` library dataset `words` contain `ie` after a `c`?

```
sum(str_detect(words, "cie"))
```

```
## [1] 2
```

- What proportion of words in `words` begin with a `ch` or a `th`?

```
mean(str_detect(words, "(^ch)|(^th)"))
```

```
## [1] 0.03265306122448979886386
```

# String detection

- What are some of the words that begin with a `ch` or a `th`?

```r
# display 5 of these words at random
sample(words[str_detect(words, "(^ch)|(^th)")], 5)
```

```
## [1] "than"      "though"    "they"      "therefore" "thank"
```

- Using `str_subset()`

```r
# display 5 of these words at random
sample(str_subset(words, "(^ch)|(^th)"), 5)
```

```
## [1] "this"     "though"   "chap"     "chance"   "thousand"
```

# String detection

- Using `str_detect()` within a data.frame to select words starting with "ch" or "th"

```
df <- tibble(word = words, i = seq_along(word))
df %>%
  filter(str_detect(word, "(^ch)|(^th)"))
```

```
## # A tibble: 32 × 2
##    word          i
##    <chr>     <int>
##  1 chair       138
##  2 chairman    139
##  3 chance      140
##  4 change      141
##  5 chap        142
##  6 character   143
##  7 charge      144
##  8 cheap       145
##  9 check       146
## 10 child       147
## # … with 22 more rows
```

# String counting

- Count the number of matches using `str_count()`
- Count the number of vowels and consonants in each word

```
df %>%
  mutate(
    vowels = str_count(word, "[aeiou]"),
    consonants = str_count(word, "[^aeiou]")
  )
```

```
## # A tibble: 980 × 4
##    word         i vowels consonants
##    <chr>    <int>  <int>      <int>
##  1 a            1      1          0
##  2 able         2      2          2
##  3 about        3      3          2
##  4 absolute     4      4          4
##  5 accept       5      2          4
##  6 account      6      3          4
##  7 achieve      7      4          3
##  8 across       8      2          4
##  9 act          9      1          2
## 10 active      10      3          3
## # … with 970 more rows
```

# Strings examples

- Use the Trump speech data
- A subset of speeches at Trump rallies

```r
file_path <- here::here("data", "Trump_rallies")
all_files <- list.files(file_path, pattern = ".txt")

dat <- list()

for (i in 1:length(all_files)) {
  dat[[i]] <- read_file(paste(file_path, all_files[i], sep = "/"))
}

str_sub(dat[[1]], 1, 200)
```

```
## [1] "Thank you. Thank you. Thank you to Vice President Pence. He's a good guy. We've done a grea
```

# String splitting

- Use `str_split()` to split the speech into sentences

```
trump_sentences <- unlist(str_split(dat[[1]], "(\\.\\s)|(\\?\\s)"))
head(trump_sentences)
```

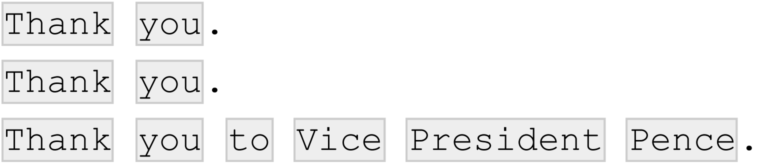
```
## [1] "Thank you"
## [2] "Thank you"
## [3] "Thank you to Vice President Pence"
## [4] "He's a good guy"
## [5] "We've done a great job together"
## [6] "And Merry Christmas, Michigan"
```

- Use the `boundary()` function to split on line breaks, sentences, words, or characters

```
trump_sentences <- unlist(str_split(dat[[1]], boundary("sentence")))
```

- View the words in the first three sentences

```
str_view_all(trump_sentences[1:3], boundary("word"))
```



```
Thank you.
Thank you.
Thank you to Vice President Pence.
```

# Extracting matches

- How many sentences contain "Michigan"?

```
has_michigan <- str_subset(trump_sentences, "Michigan")
michigan_mentions <- str_match(has_michigan, "Michigan")
head(michigan_mentions)
```

```
##      [,1]
## [1,] "Michigan"
## [2,] "Michigan"
## [3,] "Michigan"
## [4,] "Michigan"
## [5,] "Michigan"
## [6,] "Michigan"
```

# Extracting matches

- How many sentences contain "because", "Thank", or "you"?

```r
words_to_find <- c("because", "Thank", "thank", "you")
words_match <- str_c(words_to_find, collapse = "|")
has_words <- str_subset(trump_sentences, words_match)
mentions <- str_match(has_words, words_match)
head(mentions)
```

```
##      [,1]
## [1,] "Thank"
## [2,] "Thank"
## [3,] "Thank"
## [4,] "Thank"
## [5,] "Thank"
## [6,] "you"
```

- Note: this only returns the first match

# Extracting matches

- How to view all the matches?

```
has_words <- str_subset(trump_sentences, words_match)
str_view_all(head(has_words), words_match)
```

```
Thank you.
Thank you.
Thank you to Vice President Pence.
Thank you, Michigan.
Thank you very much.
And did you notice that everybody is saying Merry Christmas again?
```

# Extracting matches

- Extract all matches with `str_match_all()`

```
head(str_match(has_words, words_match))
```

```
##      [,1]   
## [1,] "Thank"
## [2,] "Thank"
## [3,] "Thank"
## [4,] "Thank"
## [5,] "Thank"
## [6,] "you"
```

```
head(str_match_all(has_words, words_match))
```

```
## [[1]]
##      [,1]   
## [1,] "Thank"
## [2,] "you"  
## 
## [[2]]
##      [,1]   
## [1,] "Thank"
## [2,] "you"  
## 
## [[3]]
##      [,1]   
## [1,] "Thank"
## [2,] "you"  
## 
```

# Extracting matches

- Format results in a matrix rather than a list

```
head(str_extract_all(has_words, words_match, simplify = TRUE))
```

```
##      [,1]    [,2]  [,3] [,4] [,5] [,6]
## [1,] "Thank" "you" ""   ""   ""   ""  
## [2,] "Thank" "you" ""   ""   ""   ""  
## [3,] "Thank" "you" ""   ""   ""   ""  
## [4,] "Thank" "you" ""   ""   ""   ""  
## [5,] "Thank" "you" ""   ""   ""   ""  
## [6,] "you"   ""    ""   ""   ""   ""  
```

# Grouped matches

- Define a noun as a word that comes after an "a" or a "the"

```
noun <- "(a|the) ([^ ]+)"
has_noun <- str_subset(trump_sentences, noun)
head(has_noun)
```

```
## [1] "He's a good guy. "
## [2] "We've done a great job together. "
## [3] "What a victory we had in Michigan. "
## [4] "What a victory was that. "
## [5] "One of the greats. "
## [6] "Was that the greatest evening? "
```

- Doesn't do too great for nouns but seems ok

```
has_noun %>%
str_extract(noun) %>%
  head()
```

```
## [1] "a good"       "a great"      "a victory"    "a victory"
## [5] "the greats."  "the greatest"
```

# Grouped matches

- `str_extract()` gives complete match, `str_match()` gives the individual groups

```
has_noun %>%
str_extract(noun) %>%
  head()
```

```
## [1] "a good"       "a great"      "a victory"    "a victory"
## [5] "the greats."  "the greatest"
```

```
has_noun %>%
str_match(noun) %>%
  head()
```

```
##      [,1]           [,2]  [,3]
## [1,] "a good"       "a"   "good"
## [2,] "a great"      "a"   "great"
## [3,] "a victory"    "a"   "victory"
## [4,] "a victory"    "a"   "victory"
## [5,] "the greats."  "the" "greats."
## [6,] "the greatest" "the" "greatest"
```

# Replacing matches

- Replace matches with `str_replace()` (only replace first instance) and `str_replace_all()`
- Replace Trump speeches vowels with "%"

```
head(str_replace(trump_sentences, "[aeiou]", "%"))
```

```
## [1] "Th%nk you. "
## [2] "Th%nk you. "
## [3] "Th%nk you to Vice President Pence. "
## [4] "H%'s a good guy. "
## [5] "W%'ve done a great job together. "
## [6] "And M%rry Christmas, Michigan. "
```

```
trump_sentences %>%
  str_replace_all("[aeiou]", "%") %>%
  head()
```

```
## [1] "Th%nk y%%. "
## [2] "Th%nk y%%. "
## [3] "Th%nk y%% t% V%c% Pr%s%d%nt P%nc%. "
## [4] "H%'s % g%%d g%y. "
## [5] "W%'v% d%n% % gr%%t j%b t%g%th%r. "
## [6] "And M%rry Chr%stm%s, M%ch%g%n. "
```

# Replacing matches

- Replace multiple matches with `str_replace_all()`

```
trump_sentences %>%
  str_replace_all(c("you" = "me", "Pence" = "%%%")) %>%
  head()
```

```
## [1] "Thank me. "
## [2] "Thank me. "
## [3] "Thank me to Vice President %%%. "
## [4] "He's a good guy. "
## [5] "We've done a great job together. "
## [6] "And Merry Christmas, Michigan. "
```

# Locate strings

- Locate where the strings are with `str_locate()` and `str_locate_all()`
-

```
head(str_locate(trump_sentences, "Thank"))
```

```
##      start end
## [1,]     1   5
## [2,]     1   5
## [3,]     1   5
## [4,]    NA  NA
## [5,]    NA  NA
## [6,]    NA  NA
```

# Factors

- A factor is a "qualitative" variable that is encoded with a "numeric" value
- Types of factors
  - nominal (order doesn't matter) -- colors, religion, etc.
  - ordinal (order matters) -- low/medium/high, young/middle aged/old

```r
names <- c("Joe", "Frank", "Prudence", "Cora")
ages <- c("young", "middle aged", "old", "young")
sort(ages)
```

```
## [1] "middle aged" "old"         "young"       "young"
```

- Sorted alphabetically, not by age

```r
age_levels <- c("young", "middle aged", "old")
ages_factor <- factor(ages, levels = age_levels)
sort(ages_factor)
```

```
## [1] young       young       middle aged old
## Levels: young middle aged old
```

- much better as a factor

# Factors

- Be careful of typos

```
names <- c("Joe", "Frank", "Prudence", "Cora")
ages <- c("young", "middle aged", "old", "youngs")
sort(ages)
```

```
## [1] "middle aged" "old"         "young"       "youngs"
```

- Sorted alphabetically, not by age

```
age_levels <- c("young", "middle aged", "old")
ages_factor <- factor(ages, levels = age_levels)
ages_factor
```

```
## [1] young       middle aged old         <NA>
## Levels: young middle aged old
```

- Notice the NA value was created without a warning

# Working with characters

```
library(openintro)
```

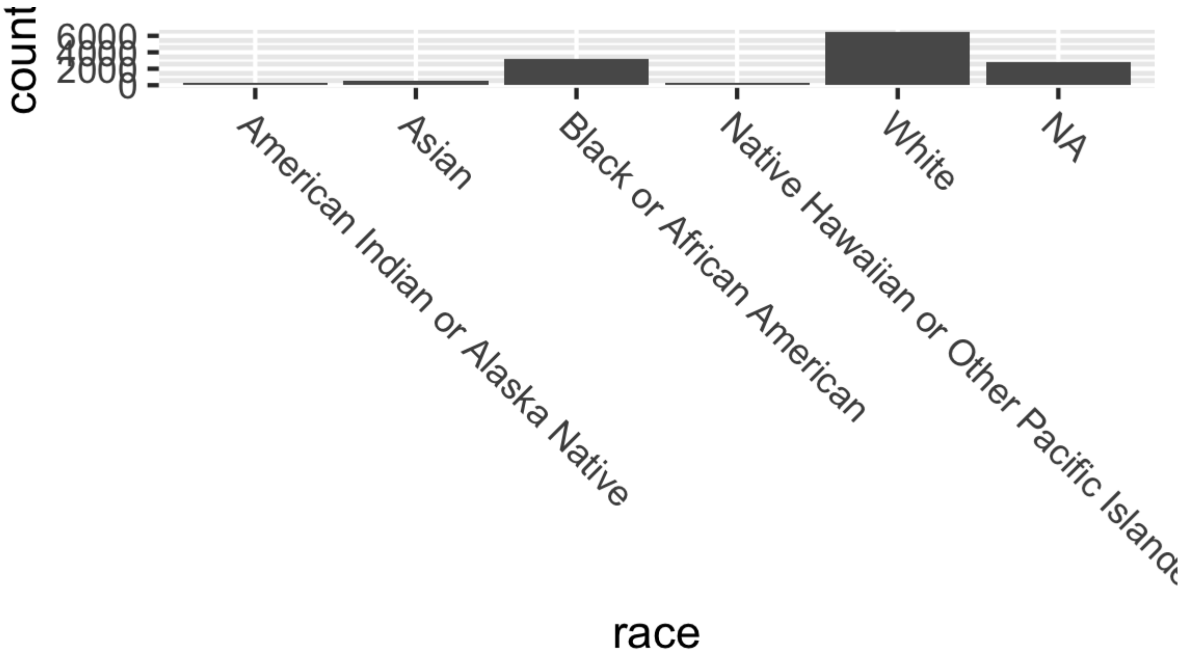
```
## Warning: package 'openintro' was built under R version 4.1.2

## Loading required package: airports

## Loading required package: cherryblossom

## Loading required package: usdata
```

```
data("yrbss")
# ?yrbss
yrbss
```

```
## # A tibble: 13,583 × 13
##     age gender grade hispanic race          height weight helmet_12m
##    <int> <chr>  <chr> <chr>    <chr>          <dbl>  <dbl> <chr>
##  1    14 female 9     not      Black or Afr…  NA       NA   never
##  2    14 female 9     not      Black or Afr…  NA       NA   never
##  3    15 female 9     hispanic Native Hawai…   1.73   84.4 never
##  4    15 female 9     not      Black or Afr…   1.6    55.8 never
##  5    15 female 9     not      Black or Afr…   1.5    46.7 did not r…
##  6    15 female 9     not      Black or Afr…   1.57   67.1 did not r…
##  7    15 female 9     not      Black or Afr…   1.65  132.  did not r…
##  8    14 male   9     not      Black or Afr…   1.88   71.2 never
##  9    15 male   9     not      Black or Afr…   1.75   63.5 never
## 10    15 male   10    not      Black or Afr…   1.37   97.1 did not r…
## # … with 13,573 more rows, and 5 more variables:
```

# Working with characters

```
yrbss %>%
  ggplot(aes(race)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle =- 45, hjust = 0))
```



# Working with Factors

```
## change race to a factor
yrbss$race <- factor(yrbss$race)
```

- count the different `race` variables

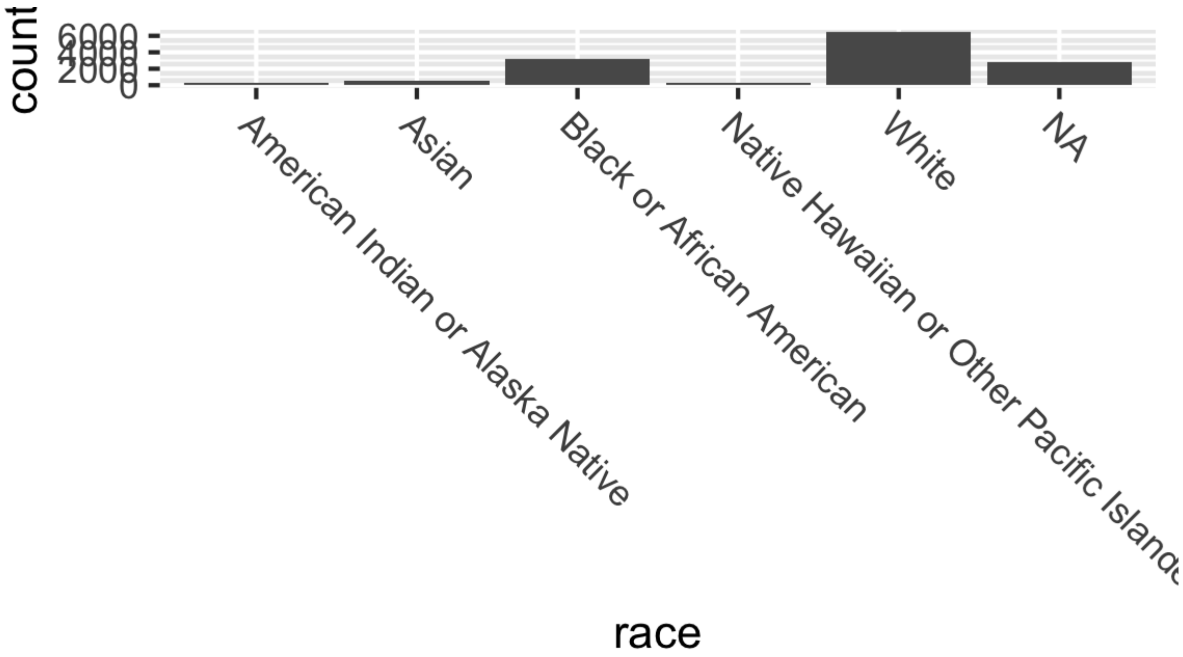
```
yrbss %>%
  count(race)
```

```
## # A tibble: 6 × 2
##   race                                          n
##   <fct>                                     <int>
## 1 American Indian or Alaska Native            323
## 2 Asian                                       552
## 3 Black or African American                  3229
## 4 Native Hawaiian or Other Pacific Islander   258
## 5 White                                      6416
## 6 <NA>                                       2805
```

# Working with Factors

- Names are too long

```
yrbss %>%
  ggplot(aes(race)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle =- 45, hjust = 0))
```



# Working with Factors

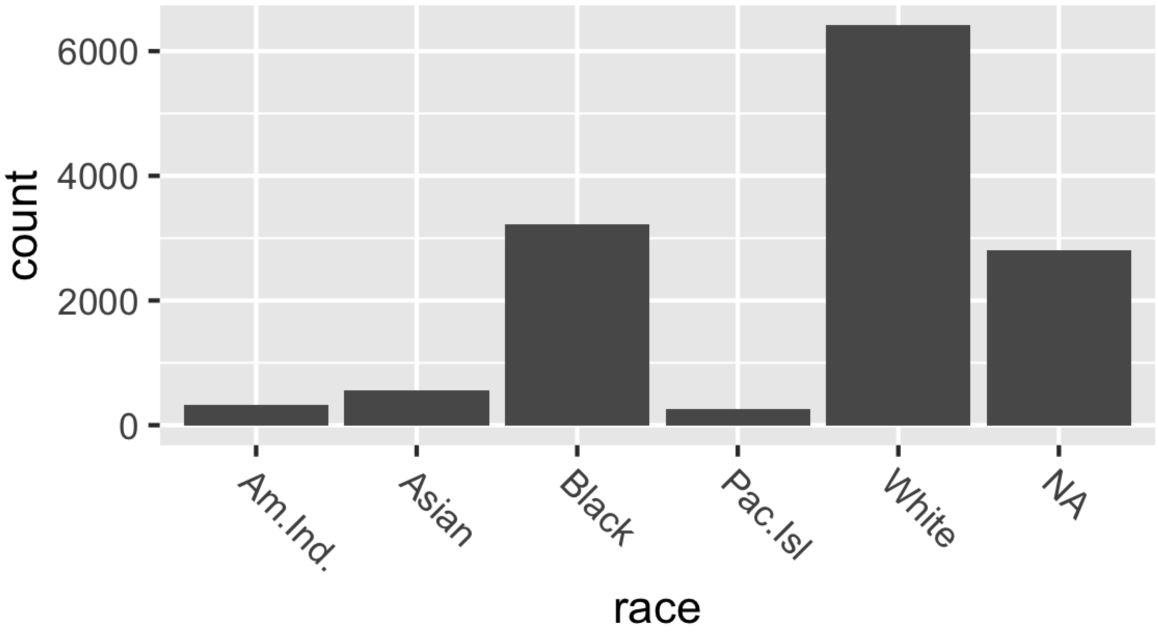
- rename factors with `fct_recode()`

```
yrbss$race <- fct_recode(
  yrbss$race,
  Am.Ind. = "American Indian or Alaska Native",
  Asian   = "Asian",
  Black   = "Black or African American",
  Pac.Isl = "Native Hawaiian or Other Pacific Islander",
  White   = "White"
)
```

# Working with Factors

- Better figure

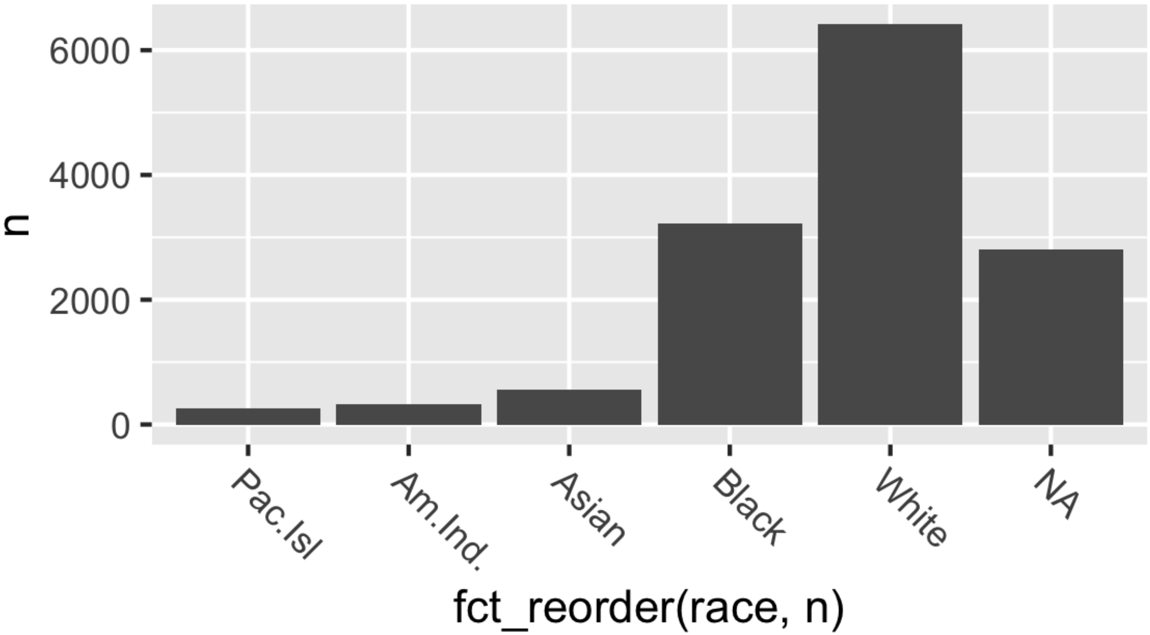
```
yrbss %>%
  ggplot(aes(race)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle =- 45, hjust = 0))
```



# Working with Factors

- reorder based on count using `fct_reorder`

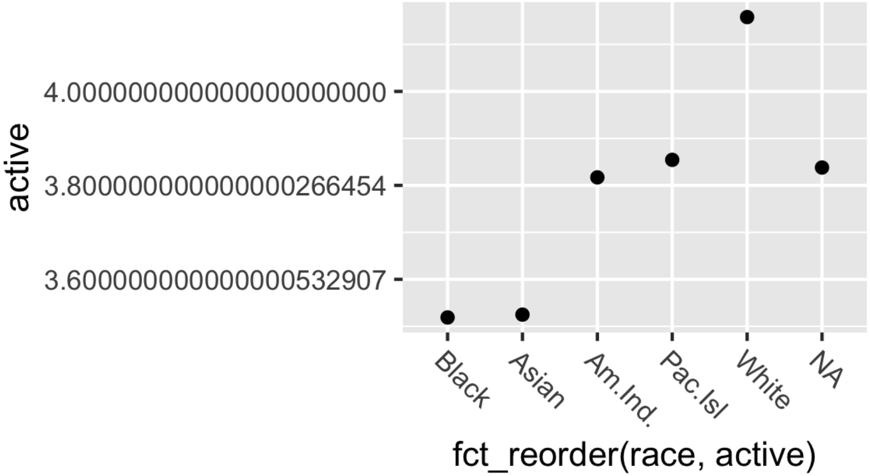
```
yrbss %>%
  count(race) %>%
  ggplot(aes(x = fct_reorder(race, n), y = n)) +
  geom_col() +
  theme(axis.text.x = element_text(angle =- 45, hjust = 0))
```



# Working with Factors

- reorder based on count using `fct_reorder`

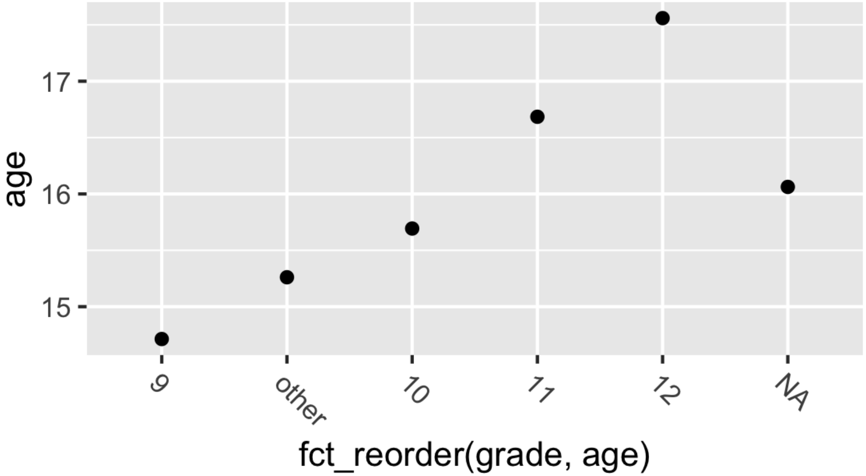
```
yrbss %>%
  group_by(race) %>%
  summarize(
    active = mean(physically_active_7d, na.rm = TRUE),
    age    =  mean(age, na.rm = TRUE),
    n      = n()
  ) %>%
  ggplot(aes(x = fct_reorder(race, active), y = active)) +
  geom_point() +
  theme(axis.text.x = element_text(angle =- 45, hjust = 0))
```



# Working with Factors

- reorder based on count using `fct_reorder` -- doesn't make sense here

```
yrbss %>%
  group_by(grade) %>%
  summarize(
    active = mean(physically_active_7d, na.rm = TRUE),
    age    =  mean(age, na.rm = TRUE),
    n      = n()
  ) %>%
  ggplot(aes(x = fct_reorder(grade, age), y = age)) +
  geom_point() +
  theme(axis.text.x = element_text(angle =- 45, hjust = 0))
```



# Working with Factors

- `gss_cat` data in `forcats` package

```
gss_cat
```

```
## # A tibble: 21,483 × 9
##     year marital        age race  rincome partyid relig denom tvhours
##    <int> <fct>        <int> <fct> <fct>   <fct>   <fct> <fct>   <int>
##  1  2000 Never marri…    26 White $8000 … Ind,ne… Prot… Sout…      12
##  2  2000 Divorced        48 White $8000 … Not st… Prot… Bapt…      NA
##  3  2000 Widowed         67 White Not ap… Indepe… Prot… No d…       2
##  4  2000 Never marri…    39 White Not ap… Ind,ne… Orth… Not …       4
##  5  2000 Divorced        25 White Not ap… Not st… None  Not …       1
##  6  2000 Married         25 White $20000… Strong… Prot… Sout…      NA
##  7  2000 Never marri…    36 White $25000… Not st… Chri… Not …       3
##  8  2000 Divorced        44 White $7000 … Ind,ne… Prot… Luth…      NA
##  9  2000 Married         44 White $25000… Not st… Prot… Other       0
## 10  2000 Married         47 White $25000… Strong… Prot… Sout…       3
## # … with 21,473 more rows
```

# Working with Factors

- Recode multiple variables

```
gss_cat %>%
  count(partyid)
```

```
## # A tibble: 10 × 2
##    partyid                n
##    <fct>              <int>
##  1 No answer            154
##  2 Don't know             1
##  3 Other party          393
##  4 Strong republican   2314
##  5 Not str republican  3032
##  6 Ind,near rep        1791
##  7 Independent         4119
##  8 Ind,near dem        2499
##  9 Not str democrat    3690
## 10 Strong democrat     3490
```

# Working with Factors

- Recode multiple variables at one time with `fct_collapse()`

```
gss_cat %>%
  mutate(partyid = fct_collapse(
    partyid,
    other = c("No answer", "Don't know", "Other party"),
    rep = c("Strong republican", "Not str republican"),
    ind = c("Ind,near rep", "Independent", "Ind,near dem"),
    dem = c("Not str democrat", "Strong democrat")
  )) %>%
  count(partyid)
```

```
## # A tibble: 4 × 2
##   partyid     n
##   <fct>   <int>
## 1 other     548
## 2 rep      5346
## 3 ind      8409
## 4 dem      7180
```

# Dates and times

- dates and times can be really complicated (`lubridate` package)
- many different computer formats (POSIXct is really common)

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
today()
```

```
## [1] "2022-06-02"
```

```r
now()
```

```
## [1] "2022-06-02 08:50:31 CDT"
```

- How many seconds in a day? In a year?
- What about leap years? Time zones?

# Dates and times

# Dates from strings

- Helpful functions: `ymd()`, `ymd()`, `myd()`, `mdy()`, `dym()`, and `dmy()` for year, month, and day

```
ymd("2020-10-31")
```

```
## [1] "2020-10-31"
```

```
mdy("October 31st, 2020")
```

```
## [1] "2020-10-31"
```

- Can add in time with `_hms()` functions

```
# 6 PM on halloween
ymd_h("2020-10-31 18")
```

```
## [1] "2020-10-31 18:00:00 UTC"
```

```
# Right before midnight
ymd_hms("2020-10-31 11:59:59")
```

```
## [1] "2020-10-31 11:59:59 UTC"
```

# Dates

- Process dates with `make_date()`

```
dat <- data.frame(
  year = c(2012, 2012, 2013, 2013, 2013, 2014, 2016),
  month = c(5, 6, 11, 4, 1, 10, 9),
  day = c(11, 22, 13, 30, 9, 5, 16),
  hour = c(2, 16, 9, 22, 4, 15, 17)
)
```

```
dat %>%
  mutate(date = make_date(year, month, day))
```

```
##   year month day hour       date
## 1 2012     5  11    2 2012-05-11
## 2 2012     6  22   16 2012-06-22
## 3 2013    11  13    9 2013-11-13
## 4 2013     4  30   22 2013-04-30
## 5 2013     1   9    4 2013-01-09
## 6 2014    10   5   15 2014-10-05
## 7 2016     9  16   17 2016-09-16
```

# Dates

- Process dates with `make_datetime()`

```
dat <- data.frame(
  year = c(2012, 2012, 2013, 2013, 2013, 2014, 2016),
  month = c(5, 6, 11, 4, 1, 10, 9),
  day = c(11, 22, 13, 30, 9, 5, 16),
  hour = c(2, 16, 9, 22, 4, 15, 17),
  minute = c(13, 24, 25, 33, 14, 53, 37),
  second = c(36, 5, 19, 4, 34, 43, 18)
)
```

```
dat_dt <- dat %>%
  mutate(datetime = make_datetime(year, month, day, hour, minute, second))
dat_dt
```

```
##   year month day hour minute second            datetime
## 1 2012     5  11    2     13     36 2012-05-11 02:13:36
## 2 2012     6  22   16     24      5 2012-06-22 16:24:05
## 3 2013    11  13    9     25     19 2013-11-13 09:25:19
## 4 2013     4  30   22     33      4 2013-04-30 22:33:04
## 5 2013     1   9    4     14     34 2013-01-09 04:14:34
## 6 2014    10   5   15     53     43 2014-10-05 15:53:43
## 7 2016     9  16   17     37     18 2016-09-16 17:37:18
```

# Converting between dates and date-times

```r
today()
```

```
## [1] "2022-06-02"
```

```r
as_datetime(today())
```

```
## [1] "2022-06-02 UTC"
```

```r
now()
```

```
## [1] "2022-06-02 08:50:31 CDT"
```

```r
as_date(now())
```

```
## [1] "2022-06-02"
```

# Dates: Extracting components

- `year()`
- `month()`
- `mday()` (day of the month)
- `yday()` (day of the year)
- `wday()` (day of the week)
- `hour()`
- `minute()`
- `second()`

# Dates: Extracting components

```r
year(dat_dt$datetime)
```

```
## [1] 2012 2012 2013 2013 2013 2014 2016
```

```r
mday(dat_dt$datetime)
```

```
## [1] 11 22 13 30  9  5 16
```

```r
yday(dat_dt$datetime)
```

```
## [1] 132 174 317 120   9 278 260
```

```r
wday(dat_dt$datetime)
```

```
## [1] 6 6 4 3 4 1 6
```

```r
wday(dat_dt$datetime, label = TRUE)
```

```
## [1] Fri Fri Wed Tue Wed Sun Fri
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

# Dates

- rounding of dates with `floor_date()`, `round_date()`, and `ceiling_date()`

```r
dat_dt %>%
  select(datetime) %>%
  mutate(week_date = floor_date(datetime, "week"))
```

```
##              datetime  week_date
## 1 2012-05-11 02:13:36 2012-05-06
## 2 2012-06-22 16:24:05 2012-06-17
## 3 2013-11-13 09:25:19 2013-11-10
## 4 2013-04-30 22:33:04 2013-04-28
## 5 2013-01-09 04:14:34 2013-01-06
## 6 2014-10-05 15:53:43 2014-10-05
## 7 2016-09-16 17:37:18 2016-09-11
```

```r
dat_dt %>%
  select(datetime) %>%
  mutate(month_date = floor_date(datetime, "month"))
```

```
##              datetime month_date
## 1 2012-05-11 02:13:36 2012-05-01
## 2 2012-06-22 16:24:05 2012-06-01
## 3 2013-11-13 09:25:19 2013-11-01
## 4 2013-04-30 22:33:04 2013-04-01
## 5 2013-01-09 04:14:34 2013-01-01
## 6 2014-10-05 15:53:43 2014-10-01
## 7 2016-09-16 17:37:18 2016-09-01
```

```r
dat_dt %>%
  select(datetime) %>%
  mutate(week_date = ceiling_date(datetime, "week"))
```

```
##              datetime  week_date
## 1 2012-05-11 02:13:36 2012-05-13
## 2 2012-06-22 16:24:05 2012-06-24
## 3 2013-11-13 09:25:19 2013-11-17
## 4 2013-04-30 22:33:04 2013-05-05
## 5 2013-01-09 04:14:34 2013-01-13
## 6 2014-10-05 15:53:43 2014-10-12
## 7 2016-09-16 17:37:18 2016-09-18
```

```r
dat_dt %>%
  select(datetime) %>%
  mutate(month_date = ceiling_date(datetime, "month"))
```

```
##              datetime month_date
## 1 2012-05-11 02:13:36 2012-06-01
## 2 2012-06-22 16:24:05 2012-07-01
## 3 2013-11-13 09:25:19 2013-12-01
## 4 2013-04-30 22:33:04 2013-05-01
## 5 2013-01-09 04:14:34 2013-02-01
## 6 2014-10-05 15:53:43 2014-11-01
## 7 2016-09-16 17:37:18 2016-10-01
```

# Time spans

- durations

```
# how long since the Big Lebowski was released?
dude_abiding <- today() - ymd("1998-03-06")
dude_abiding
```

```
## Time difference of 8854 days
```

```
str(dude_abiding)
```

```
##  'difftime' num 8854
##  - attr(*, "units")= chr "days"
```

```
as.duration(dude_abiding)
```

```
## [1] "764985600s (~24.24 years)"
```

# Time spans

- duration constructors: `dseconds()`, `dminutes()`, `dhours()`, `ddays()`, `dweeks()`, and `dyears()`

```r
ddays(3)
```

```
## [1] "259200s (~3 days)"
```

```r
dyears(1)
```

```
## [1] "31557600s (~1 years)"
```

- durations can give strange results

```r
# Why still in 2016?
ymd_hms("2016-01-01 01:00:00") + dyears(1)
```

```
## [1] "2016-12-31 07:00:00 UTC"
```

```r
# Why did the time change?
ymd_hms("2020-03-08 01:00:00", tz = "America/Chicago") + ddays(1)
```

```
## [1] "2020-03-09 02:00:00 CDT"
```

# Time periods

- periods are human-defined terms like weeks and months
- periods can be constructed with: `seconds()`, `minutes()`, `hours()`, `days()`, `weeks()`, and `years()`

```r
# Now in 2017
ymd_hms("2016-01-01 01:00:00") + years(1)
```

```
## [1] "2017-01-01 01:00:00 UTC"
```

```r
# Time didn't change
ymd_hms("2020-03-08 01:00:00", tz = "America/Chicago") + days(1)
```

```
## [1] "2020-03-09 01:00:00 CDT"
```

# Intervals

- What should the result of `dyears(1) / ddays(365)` be?

```
dyears(1) / ddays(365)
```

```
## [1] 1.00068493150684934001
```

- What should the result of `years(1) / days(1)` be?

```
years(1) / days(1)
```

```
## [1] 365.25
```

- Could change based on the year!

```
next_year <- today() + years(1)
(today() %--% next_year) ## define a date interval
```

```
## [1] 2022-06-02 UTC--2023-06-02 UTC
```

```
(today() %--% next_year) / days(1)
```

```
## [1] 365
```

```
(today() %--% next_year) / ddays(1)
```

# Time zones

- Time zones are really complex
- Time zones are typically tied to cities
- Get your timzone with `Sys.timezone()`
- Default timezone is UTC
- Get timezones with `OlsonNames()`

```
head(OlsonNames())
```

```
## [1] "Africa/Abidjan"     "Africa/Accra"       "Africa/Addis_Ababa"
## [4] "Africa/Algiers"     "Africa/Asmara"      "Africa/Asmera"
```

# Time zones

```r
x <- ymd_hms("2020-10-31 19:00:00", tz = "America/Denver")
y <- ymd_hms("2020-10-31 20:00:00", tz = "America/Chicago")
z <- ymd_hms("2020-10-31 21:00:00", tz = "America/New_York")
```

```r
x - y
```

```
## Time difference of 0 secs
```

```r
x - z
```

```
## Time difference of 0 secs
```

```r
times <- c(x, y, z)
times
```

```
## [1] "2020-10-31 19:00:00 MDT" "2020-10-31 19:00:00 MDT"
## [3] "2020-10-31 19:00:00 MDT"
```

```r
with_tz(times, tzone = "America/Los_Angeles")
```

```
## [1] "2020-10-31 18:00:00 PDT" "2020-10-31 18:00:00 PDT"
## [3] "2020-10-31 18:00:00 PDT"
```