Data wrangling with **dplyr**

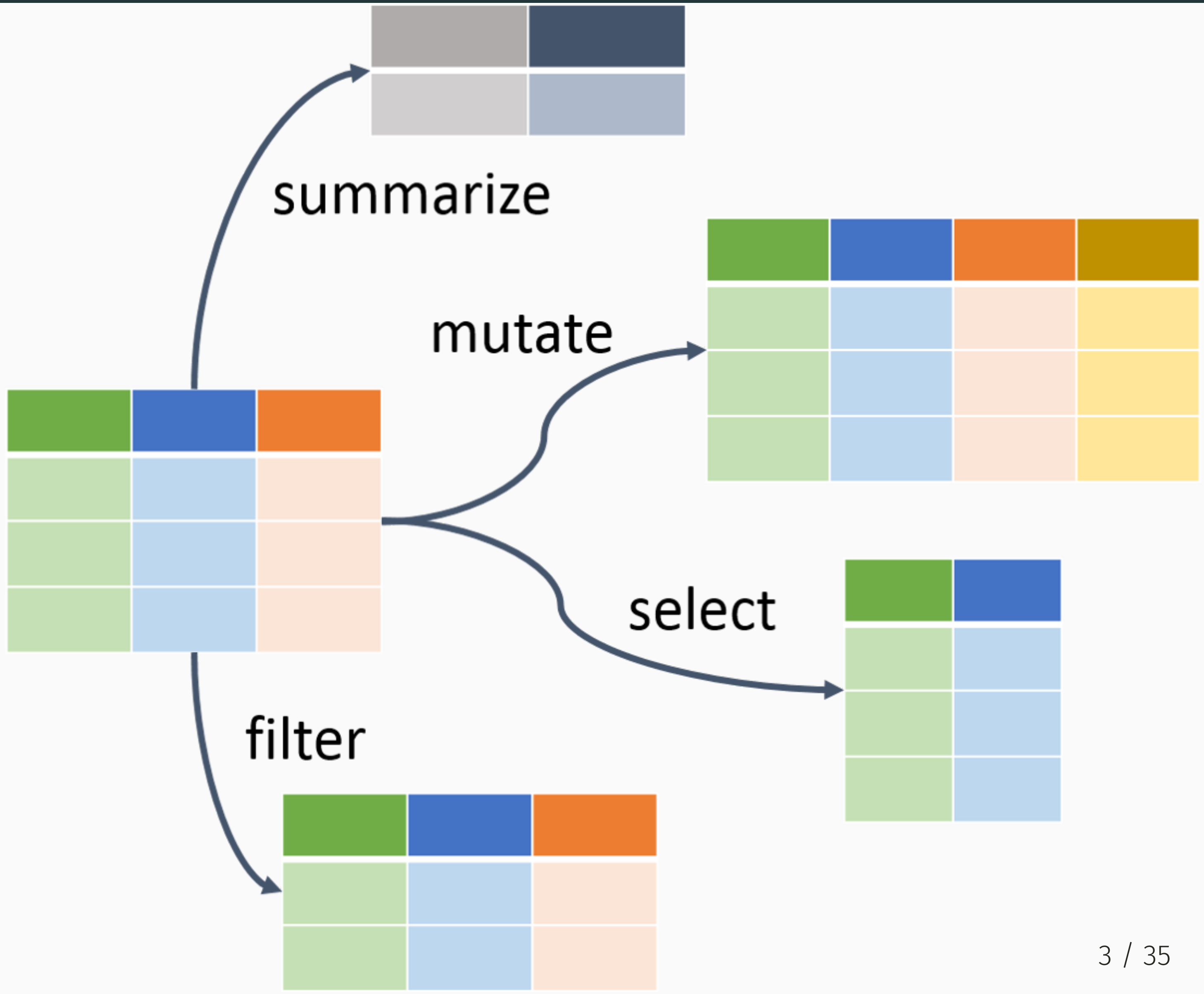
STAT 220

Bastola
January 17 2022

Data Wrangling

the process of cleaning and unifying messy and complex data sets for easy access and analysis

- "data janitor work"
- importing data
- cleaning data
- changing shape of data
- fixing errors and poorly formatted data elements
- transforming columns and rows
- filtering, subsetting


summarize
mutate
select
filter

Essential data wrangling tasks

1. Extract a subset of columns
2. Extract a subset of rows
3. Order the rows from smallest to largest (or largest to smallest)
4. Compute a table of summary statistics, perhaps by group
5. Create new columns

The Five Verbs

The creater of `dplyr`, Hadley Wickham, argues that most of the operations on a data table can be achieved with

- *select()*
- *filter()*
- *mutate()*
- *arrange()*
- *summarize()*

Babynames Dataset

- Names of babies born in the U.S. between 1880 and 2017
- Source: Social Security Administration

```
library(babynames)
glimpse(babynames)
Rows: 1,924,665
Columns: 5
$ year <dbl> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880,…
$ sex  <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", …
$ name <chr> "Mary", "Anna", "Emma", "Elizabeth", "Minnie", "Margaret", "Ida",…
$ n    <int> 7065, 2604, 2003, 1939, 1746, 1578, 1472, 1414, 1320, 1288, 1258,…
$ prop <dbl> 0.07238359, 0.02667896, 0.02052149, 0.01986579, 0.01788843, 0.016…
```

Find a subset of the columns using *select()*:

- `select()`: take a subset of the columns (variables/features)

```
babynames %>%
  select(year, name, n) %>%
  head()
# A tibble: 6 × 3
   year name          n
  <dbl> <chr>     <int>
1  1880 Mary       7065
2  1880 Anna       2604
3  1880 Emma       2003
4  1880 Elizabeth  1939
5  1880 Minnie     1746
6  1880 Margaret   1578
```

Using %>%

- `%>%` passes result on left into first argument of function on right
- `Chaining` functions together lets you read `Left-to-right`, `top-to-bottom`

Using %>%

You can build up a series of pipes

```
babynames %>%                      # dataframe first and then...
  select(year, name, n) %>%
  head()
```

Using %>%

You can build up a series of pipes

```
babynames %>%                   # dataframe first and then...
  select(year, name, n) %>%     # select columns year, name, and n
  head()
```

Using %>%

You can build up a series of pipes

```
babynames %>%                   # dataframe first and then...
  select(year, name, n) %>%     # select columns year, name, and n
  head()                        # display header of the data frame
```

```
# A tibble: 6 × 3
   year name          n
  <dbl> <chr>     <int>
1  1880 Mary       7065
2  1880 Anna       2604
3  1880 Emma       2003
4  1880 Elizabeth  1939
5  1880 Minnie     1746
6  1880 Margaret   1578
```

select() helpers

`:` select range of columns

```
select(gapminder, income:population)
```

`-` select every column but

```
select(gapminder, -c(income,population))
```

select() helpers

`starts_with()` select columns that start with...

```r
select(gapminder, starts_with("p"))
```

`ends_with()` select columns that end with...

```r
select(gapminder, ends_with("y"))
```

`contains()` select columns whose names contain...

```r
select(gapminder, contains("e"))
```

Your Turn 1

- Please git clone the repository on data wrangling activity from the course GitHub organization.
- Which of these is **NOT** a way to select the `name` and `n` columns together?

```
select(babynames, -c(year, sex, prop))

select(babynames, name:n)

select(babynames, starts_with("n"))

select(babynames, ends_with("n"))
```

02:00

Find a subset of the rows using *filter()*:

- `filter()` : take a subset of the rows (observations)

```
babynames %>%
  filter(name == "Bella") %>%
  head()
# A tibble: 6 × 5
   year sex   name      n     prop
  <dbl> <chr> <chr> <int>    <dbl>
1  1880 F     Bella    13 0.000133
2  1881 F     Bella    24 0.000243
3  1882 F     Bella    16 0.000138
4  1883 F     Bella    17 0.000142
5  1884 F     Bella    31 0.000225
6  1885 F     Bella    25 0.000176
```

Use both *filter()* and *select()*

```
bella <- babynames %>%
  filter(name == "Bella") %>%
  select(year, name, sex, n)
```

```
head(bella)
# A tibble: 6 × 4
   year name  sex       n
  <dbl> <chr> <chr> <int>
1  1880 Bella F        13
2  1881 Bella F        24
3  1882 Bella F        16
4  1883 Bella F        17
5  1884 Bella F        31
6  1885 Bella F        25
```

```
dim(bella)
[1] 144   4
class(bella)
[1] "tbl_df"     "tbl"        "data.frame"
```

Some Operators

Operator	Definition
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	exactly equal to
`!=`	not equal to
`x & y`	`x` AND `y`
`x %in% y`	test if `x` is in `y`

Your Turn 2

Use `filter()` with the logical operators to extract:

- All of the names where prop is greater than or equal to 0.08
- All of the babies named "Rose"
- All of the names that have a missing value for n

02:00

summarize() or *summarise()*

If we want to compare summary statistics, we might use `summarize()`.

```
babynames %>%
  filter(name == "Bella", sex == "F") %>%
  summarise(total = sum(n), max = max(n), mean = mean(n))
# A tibble: 1 × 3
  total   max  mean
  <int> <int> <dbl>
1 57411  5121  416.
```

```
babynames %>%
  filter(name == "Bella", sex == "F") %>
  summarize(n = n())
# A tibble: 1 × 1
      n
  <int>
1   138
```

```
babynames %>%
  summarize(nname = n_distinct(name))
# A tibble: 1 × 1
  nname
  <int>
1 97310
```

Your Turn 3

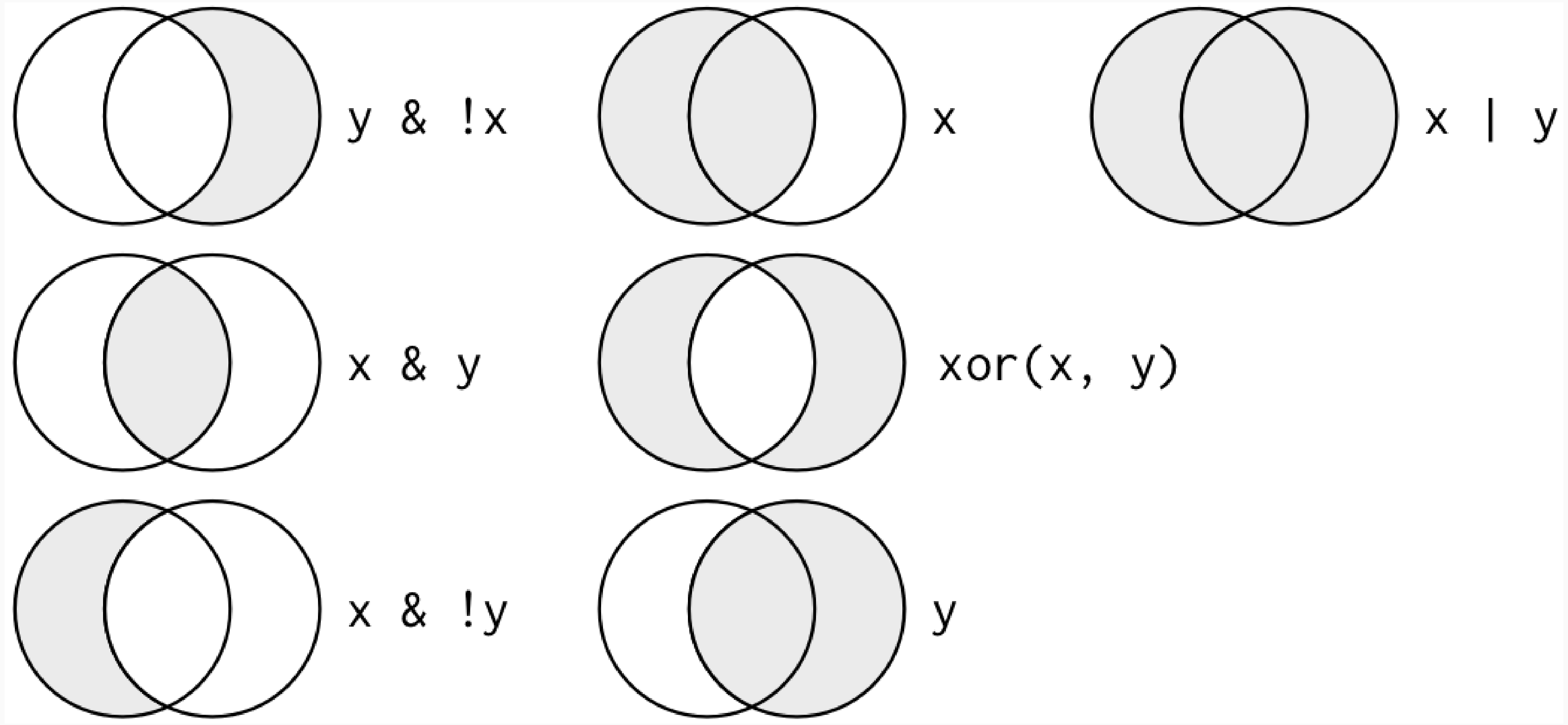
Use the codes mentioned so far to compute three statistics:

1. the total number of children who ever had your name
2. the maximum number of children given your name in a single year
3. the mean number of children given your name per year

04:00

Boolean operators

For help, `?base::Logic`



Source: *R for Data Science*, by Grolemund & Wickham

Your Turn 4

Use Boolean operators to alter the code below to return only the rows that contain:

- Girls named Rhea
- Names that were used by exactly 5 or 6 children in 1990
- Names that are one of Apple, Yoroi, Ada

```
filter(babynames, name == "Ada" | name == "Rhea")
```

04:00

Using *group_by()*

```
babynames %>%
  group_by(year, sex)
# A tibble: 1,924,665 × 5
# Groups:   year, sex [276]
    year sex   name          n   prop
   <dbl> <chr> <chr>     <int>  <dbl>
 1  1880 F     Mary       7065 0.0724
 2  1880 F     Anna       2604 0.0267
 3  1880 F     Emma       2003 0.0205
 4  1880 F     Elizabeth  1939 0.0199
 5  1880 F     Minnie     1746 0.0179
 6  1880 F     Margaret   1578 0.0162
 7  1880 F     Ida        1472 0.0151
 8  1880 F     Alice      1414 0.0145
 9  1880 F     Bertha     1320 0.0135
10  1880 F     Sarah      1288 0.0132
# … with 1,924,655 more rows
```

Using *group_by()* along with *summarize()*

```
babynames %>%
  group_by(year) %>%
  summarise(total = sum(n))
# A tibble: 138 × 2
    year  total
   <dbl>  <int>
 1  1880 201484
 2  1881 192696
 3  1882 221533
 4  1883 216946
 5  1884 243462
 6  1885 240854
 7  1886 255317
 8  1887 247394
 9  1888 299473
10  1889 288946
# … with 128 more rows
```

mutate()

- `mutate()` lets us create new variables based on manipulations of the old variables

```
babynames <- babynames %>%
  group_by(year) %>%
  mutate(percent = prop * 100)
head(babynames)
# A tibble: 6 × 6
# Groups:   year [1]
   year sex   name          n   prop percent
  <dbl> <chr> <chr>     <int>  <dbl>   <dbl>
1  1880 F     Mary       7065 0.0724    7.24
2  1880 F     Anna       2604 0.0267    2.67
3  1880 F     Emma       2003 0.0205    2.05
4  1880 F     Elizabeth  1939 0.0199    1.99
5  1880 F     Minnie     1746 0.0179    1.79
6  1880 F     Margaret   1578 0.0162    1.62
```

arrange()

Order rows from smallest to largest

```
arrange(.data, ...)
```

```
arrange(babynames, n)
```

desc()

Changes ordering from largest to smallest.

```
arrange(babynames, desc(n))
# A tibble: 1,924,665 × 6
# Groups:   year [138]
    year sex   name          n   prop percent
   <dbl> <chr> <chr>     <int>  <dbl>   <dbl>
 1  1947 F     Linda     99686 0.0548    5.48
 2  1948 F     Linda     96209 0.0552    5.52
 3  1947 M     James     94756 0.0510    5.10
 4  1957 M     Michael   92695 0.0424    4.24
 5  1947 M     Robert    91642 0.0493    4.93
 6  1949 F     Linda     91016 0.0518    5.18
 7  1956 M     Michael   90620 0.0423    4.23
 8  1958 M     Michael   90520 0.0420    4.20
 9  1948 M     James     88588 0.0497    4.97
10  1954 M     Michael   88514 0.0428    4.28
# … with 1,924,655 more rows
```

Most common names in 1990

```
babynames %>%
  filter(year == 1990) %>%
  arrange(desc(prop))
# A tibble: 24,719 × 6
# Groups:   year [1]
    year sex   name            n   prop percent
   <dbl> <chr> <chr>       <int>  <dbl>   <dbl>
 1  1990 M     Michael     65282 0.0303    3.03
 2  1990 M     Christopher 52332 0.0243    2.43
 3  1990 F     Jessica     46475 0.0226    2.26
 4  1990 F     Ashley      45558 0.0222    2.22
 5  1990 M     Matthew     44800 0.0208    2.08
 6  1990 M     Joshua      43216 0.0201    2.01
 7  1990 F     Brittany    36538 0.0178    1.78
 8  1990 F     Amanda      34408 0.0168    1.68
 9  1990 M     Daniel      33815 0.0157    1.57
10  1990 M     David       33742 0.0157    1.57
# … with 24,709 more rows
```

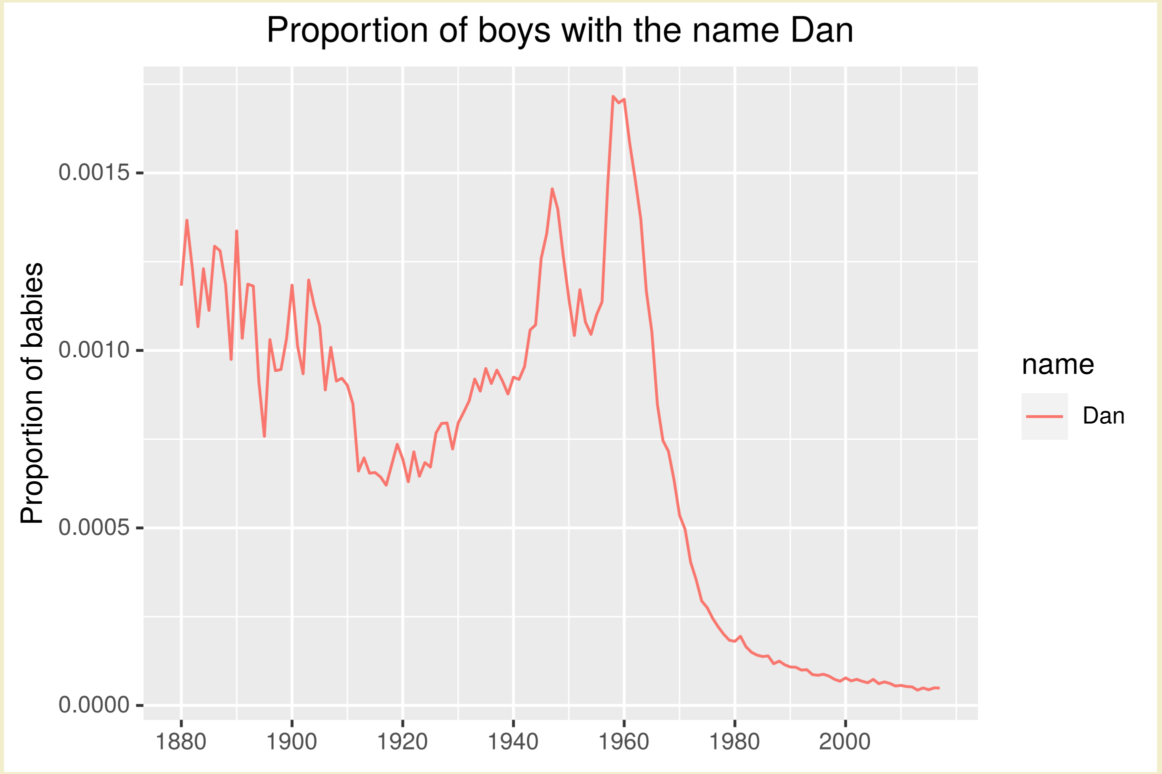
top_n()

Most common name in each year

```
babynames %>%
  group_by(year) %>%
  top_n(1, prop)
# A tibble: 138 × 6
# Groups:   year [138]
    year sex   name      n   prop percent
   <dbl> <chr> <chr> <int>  <dbl>   <dbl>
 1  1880 M     John   9655 0.0815    8.15
 2  1881 M     John   8769 0.0810    8.10
 3  1882 M     John   9557 0.0783    7.83
 4  1883 M     John   8894 0.0791    7.91
 5  1884 M     John   9388 0.0765    7.65
 6  1885 M     John   8756 0.0755    7.55
 7  1886 M     John   9026 0.0758    7.58
 8  1887 M     John   8110 0.0742    7.42
 9  1888 M     John   9247 0.0712    7.12
```

Your Turn 5

Find the year your first name is most common.



03:00

Vectorized function

a function that takes a vector as input, is applied to every element of the vector, and returns a vector (of the original length) as output

min_rank()

A go to ranking function (ties share the lowest rank)

```
min_rank(c(50, 100, 1000))
[1] 1 2 3
```

```
min_rank(desc(c(50, 100, 1000)))
[1] 3 2 1
```

Your Turn 6

- Use `min_rank()` and `mutate()` to rank each row in `babynames` from largest `prop` to smallest `prop`.
- Compute each name's rank within its `year` and `sex`.
- Then compute the median rank for each combination of `name` and `sex`, and arrange the results from highest median rank to lowest.

06:00

Summary

- Extract variables with `select()`
- Extract cases with `filter()`
- Arrange cases, with `arrange()`
- Make tables of summaries with `summarize()`
- Make new variables, with `mutate()`

Acknowledgement: some of the slides are based on previous works of Adam Loy and Katie St. Clair.