CHRISTOPHER PRENER, PHD
FALL, 2017

WEEK 02
LECTURE 03

QUANTITATIVE ANALYSIS

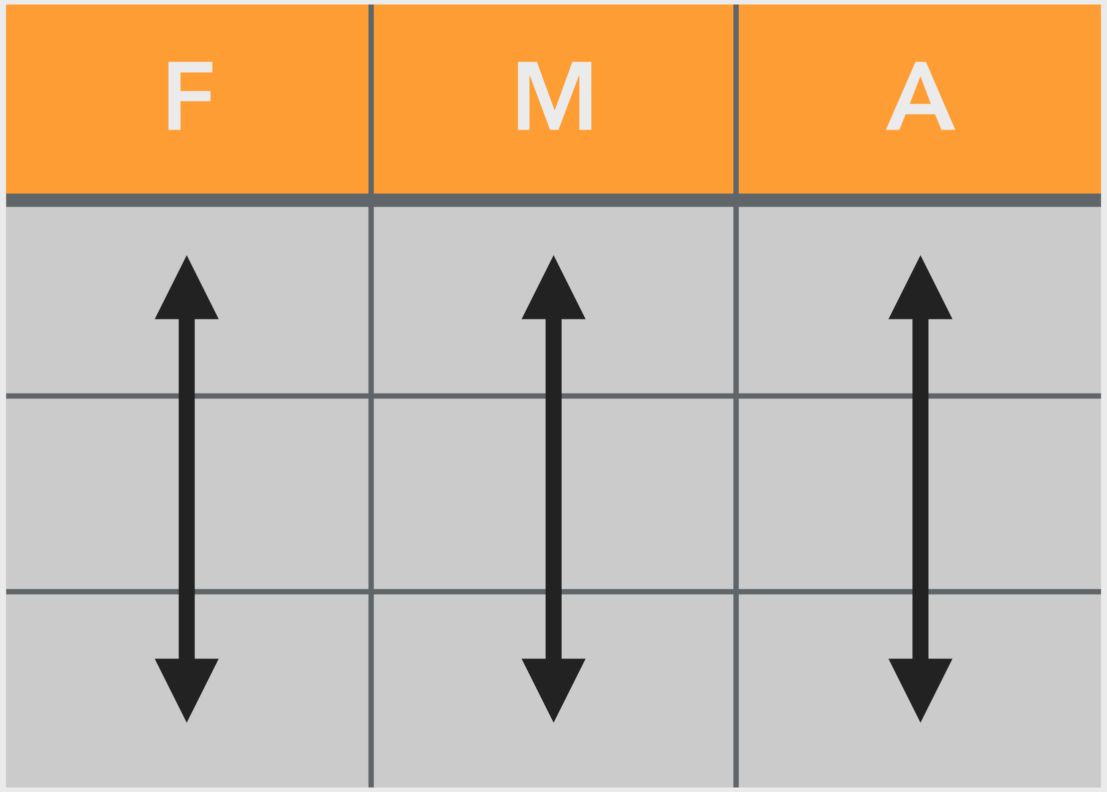
VERBS FOR CLEANING DATA

AGENDA

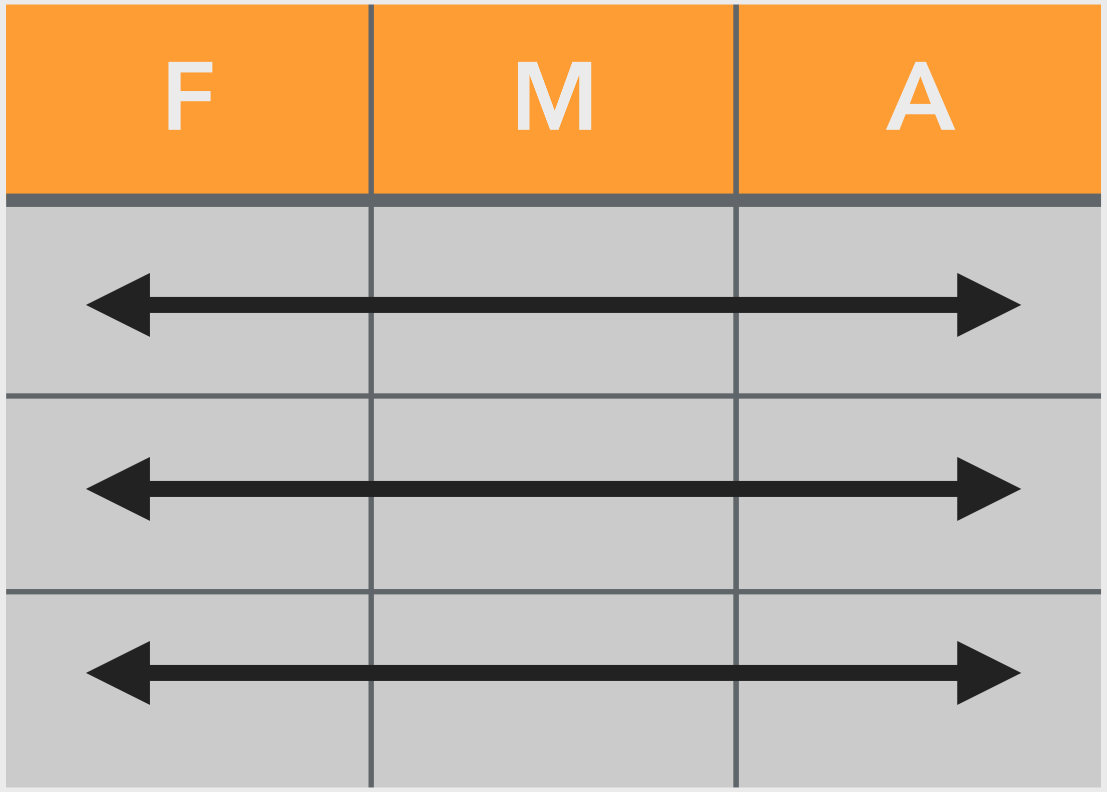
1. Tidy Data: A Review
2. `dplyr` Verbs
3. Piping Functions

1 TIDY DATA: A REVIEW

KEY CHARACTERISTICS



Each **variable** is stored in its own **column**

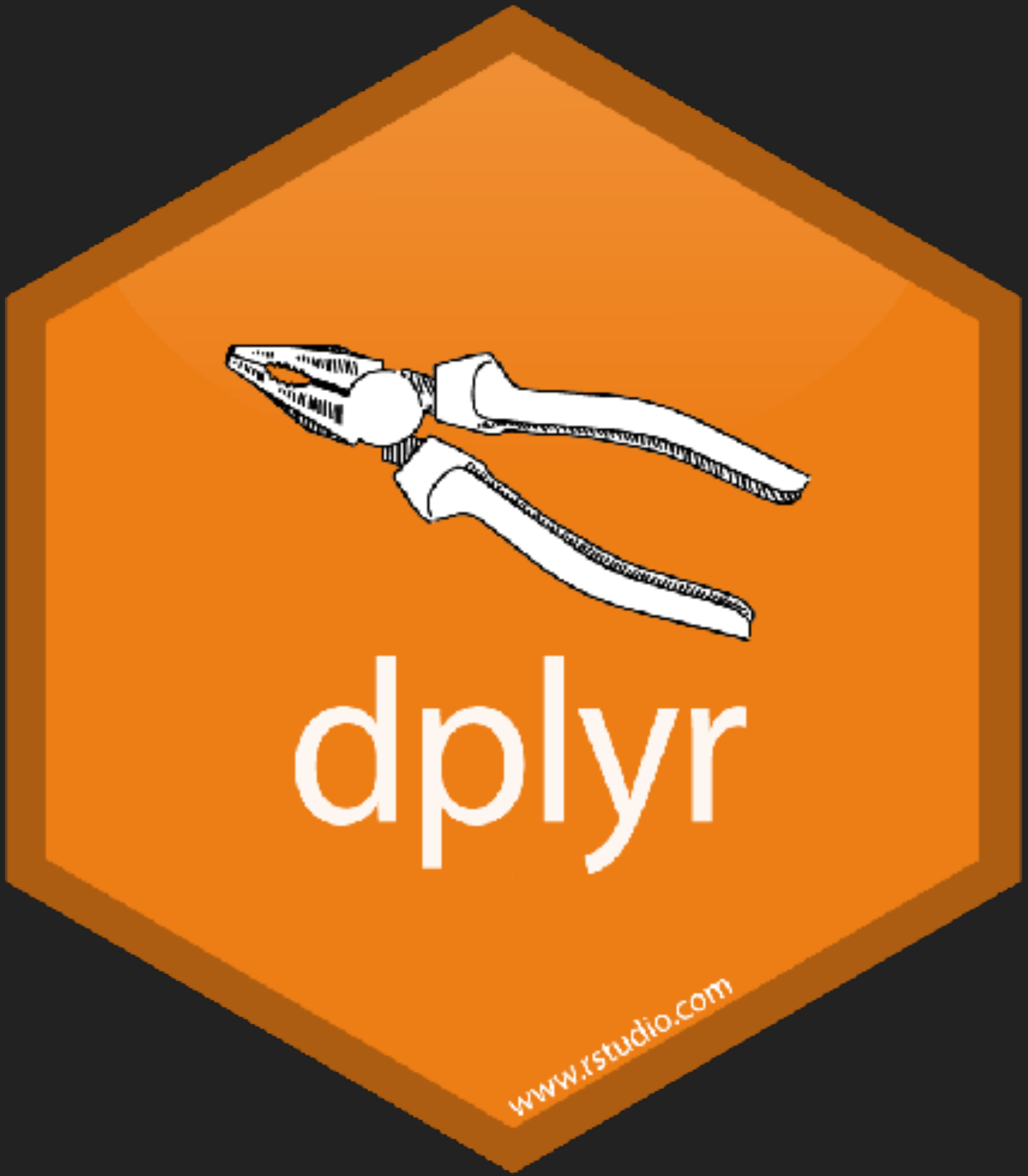


Each **observation** is stored in its own **row**

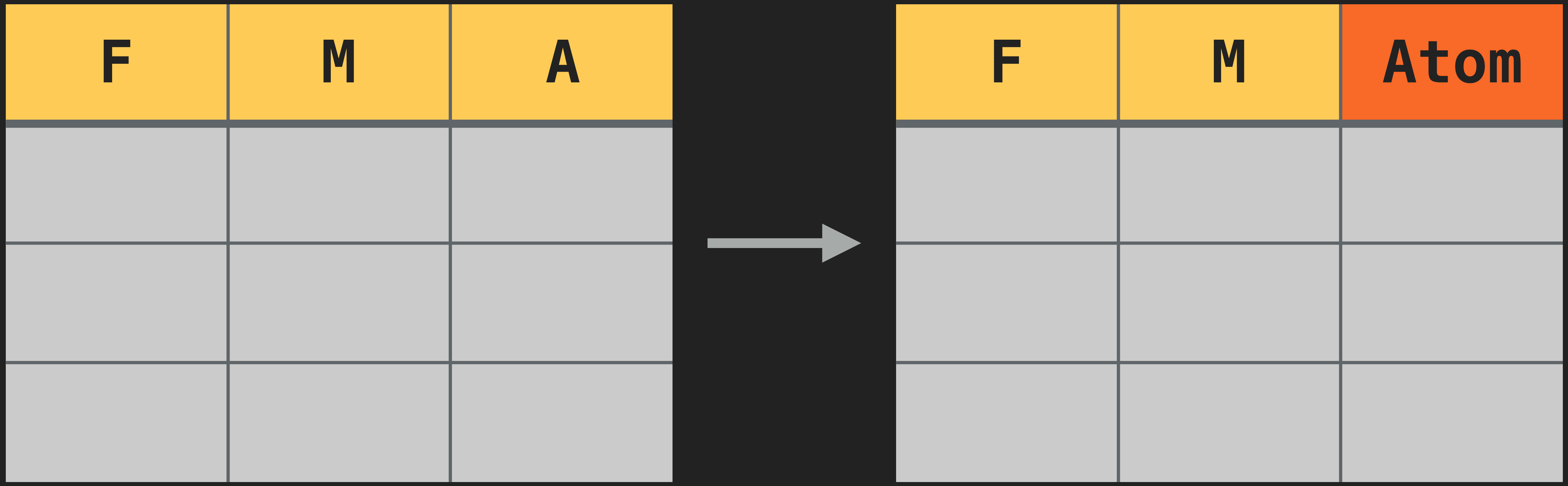
2 DPLYR VERBS

DPLYR

- Like `ggplot2`, `dplyr` is a core part of the `tidyverse`.
- Dplyr specializes in data wrangling, which is the work we put into getting a data set ready for analysis
- It is based around the concept of *verbs* - functions are named for actions that they undertake
- We'll focus on five key functions today



RENAMING VARIABLES



RENAMING VARIABLES

```
dplyr::rename(dataFrame, newName = oldName)
```

Example - the `mpg` data from `ggplot2:`

```
rename(mpg, hwyMpg = hwy)
```

RENAMING VARIABLES

```
dplyr::rename(dataFrame, newName = oldName)
```

Example - the `mpg` data from `ggplot2:`

```
rename(mpg, hwyMpg = hwy)
```

This does not make the change permanent, however. You must *assign* the results of `dplyr` functions back to the original data frame or to a new one.

ASSIGNING CHANGES

```
dataFrame <- rename(dataFrame, newName = oldName)
```

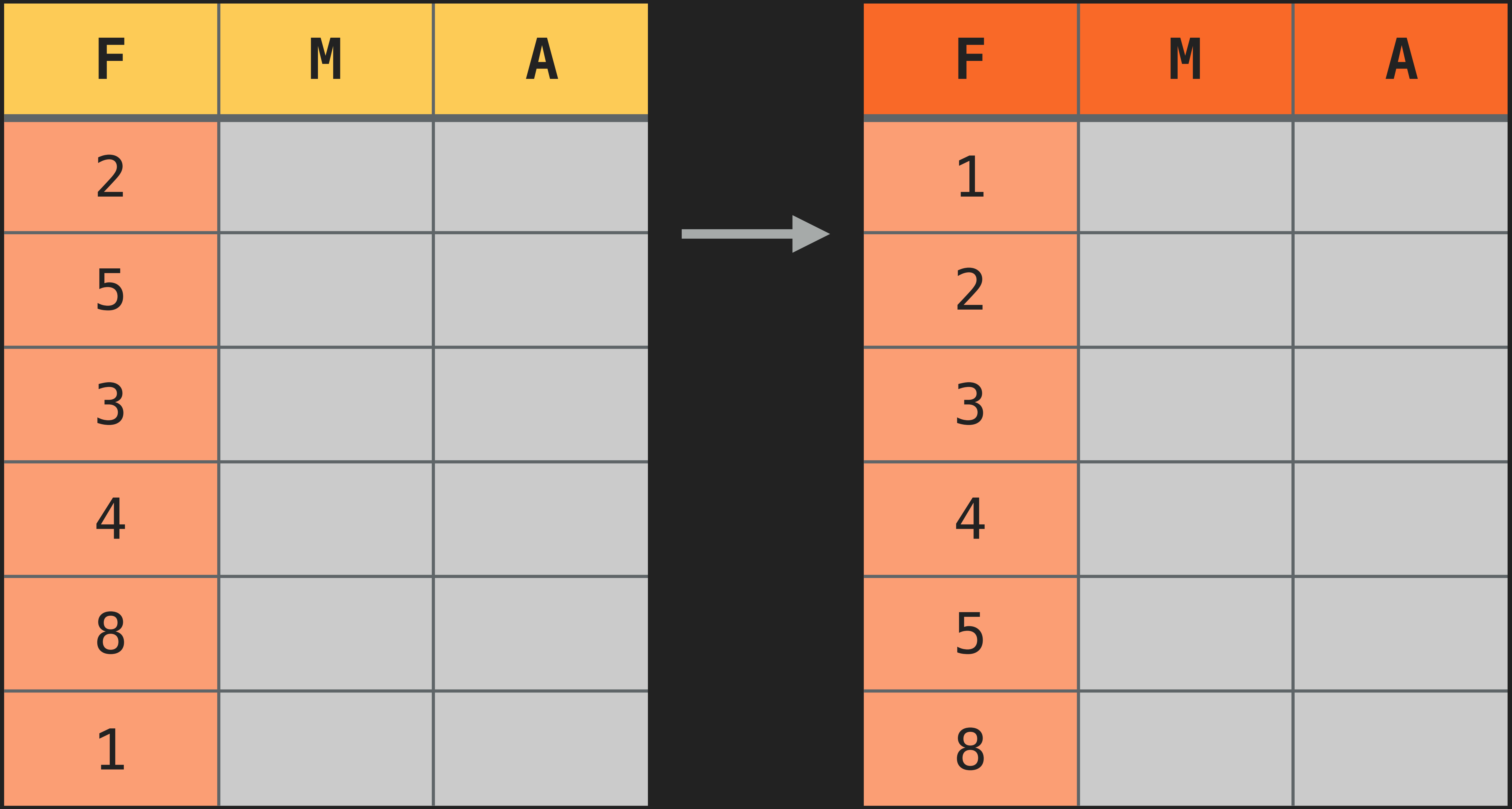
Example 1 - assigning the `mpg` data from `ggplot2` to a new object:

```
autoData <- rename(mpg, hwyMpg = hwy)
```

Example 2 - overwriting the `autoData` data example 1:

```
autoData <- rename(autoData, type = class)
```

REORDERING OBSERVATIONS

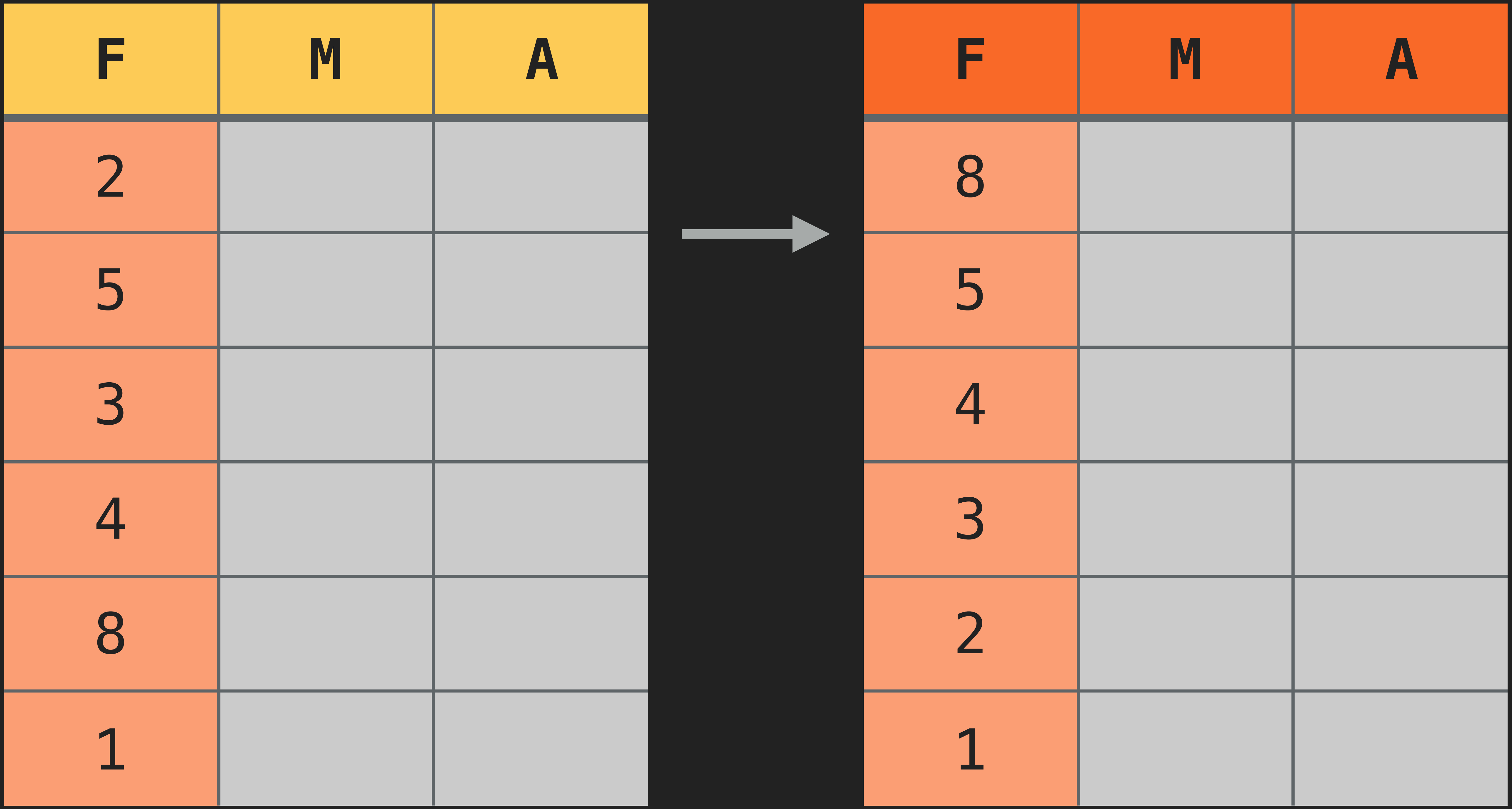


F	M	A
2		
5		
3		
4		
8		
1		

$\rightarrow$

F	M	A
1		
2		
3		
4		
5		
8		

REORDERING OBSERVATIONS



F	M	A
2		
5		
3		
4		
8		
1		

F	M	A
8		
5		
4		
3		
2		
1		

REORDERING OBSERVATIONS

```
dplyr::arrange(dataFrame, varlist)
```

Example - the `mpg` data from `ggplot2` in *ascending* order (lowest first):

```
arrange(mpg, hwy)
```

You can include more than one variable, separated by commas, if you want a list sorted based on more than one condition.

Reordering your data may change how some output looks and how the assignment of ID numbers occurs.

REORDERING OBSERVATIONS

```
dplyr::arrange(dataFrame, desc(varlist))
```

Example - the `mpg` data from `ggplot2` in *descending* order (highest first):

```
arrange(mpg, desc(hwy))
```

You can include more than one variable, separated by commas, if you want a list sorted based on more than one condition.

Reordering your data may change how some output looks and how the assignment of ID numbers occurs.

REORDERING OBSERVATIONS

```
> library(tidyverse)

> autoData <- mpg

> head(autoData)
# A tibble: 6 x 11
  manufacturer model displ  year   cyl      trans   drv   cty   hwy    fl   class
         <chr> <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>   <chr>
1         audi    a4   1.8  1999     4   auto(l5)     f    18    29     p compact
2         audi    a4   1.8  1999     4 manual(m5)     f    21    29     p compact
3         audi    a4   2.0  2008     4 manual(m6)     f    20    31     p compact
4         audi    a4   2.0  2008     4   auto(av)     f    21    30     p compact
5         audi    a4   2.8  1999     6   auto(l5)     f    16    26     p compact
6         audi    a4   2.8  1999     6 manual(m5)     f    18    26     p compact

> View(autoData)
```

REORDERING OBSERVATIONS

```
> tail(autoData)
# A tibble: 6 x 11
  manufacturer  model displ  year   cyl      trans   drv   cty   hwy    fl   class
         <chr>  <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>   <chr>
1   volkswagen passat   1.8  1999     4   auto(l5)     f    18    29     p midsize
2   volkswagen passat   2.0  2008     4   auto(s6)     f    19    28     p midsize
3   volkswagen passat   2.0  2008     4 manual(m6)     f    21    29     p midsize
4   volkswagen passat   2.8  1999     6   auto(l5)     f    16    26     p midsize
5   volkswagen passat   2.8  1999     6 manual(m5)     f    18    26     p midsize
6   volkswagen passat   3.6  2008     6   auto(s6)     f    17    26     p midsize
```

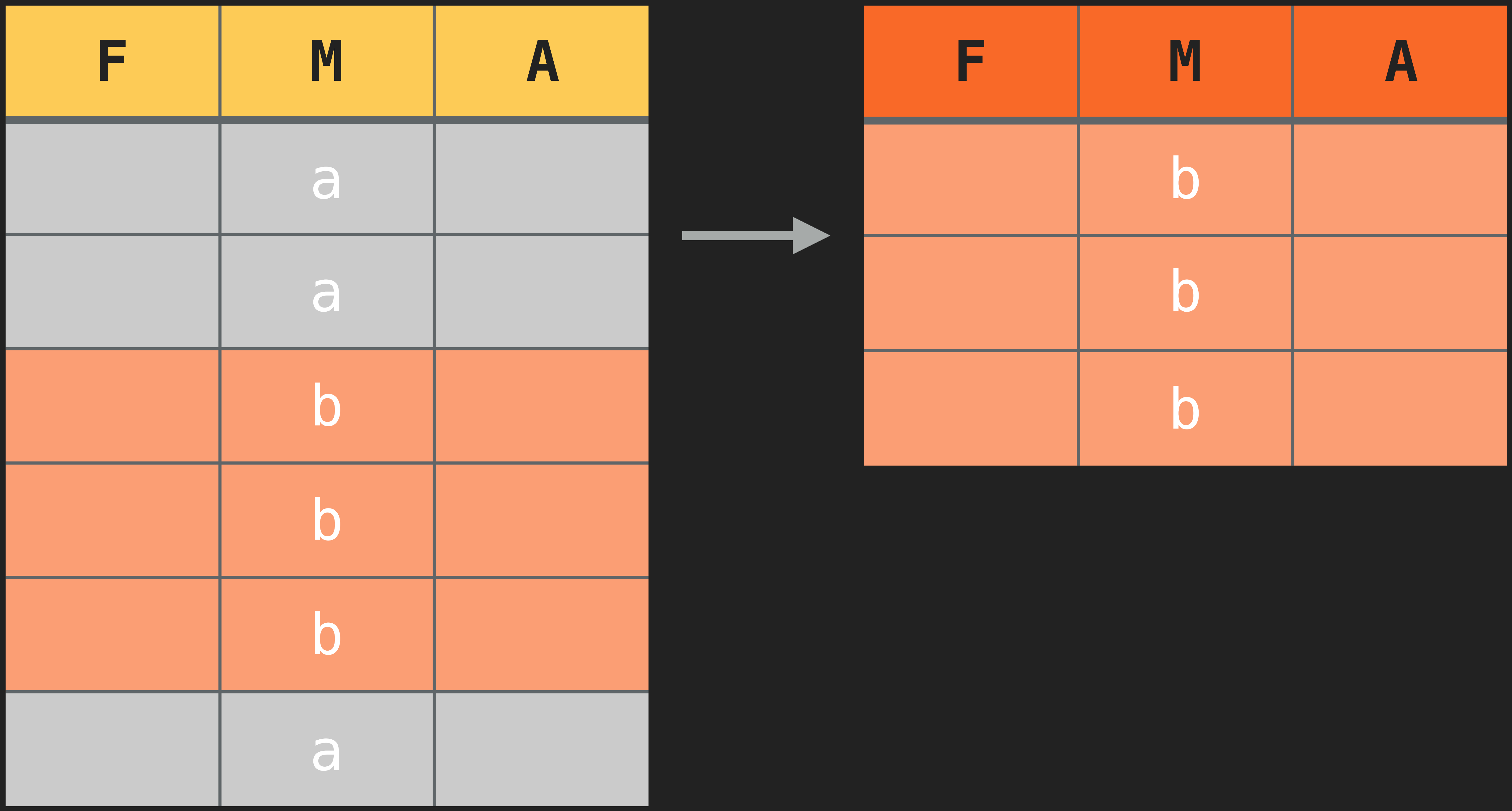
REORDERING OBSERVATIONS

```
> autoData <- arrange(autoData, hwy)

> head(autoData)
# A tibble: 6 x 11
  manufacturer                model displ  year   cyl      trans   drv   cty   hwy    fl  class
         <chr>                <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>  <chr>
1        dodge   dakota pickup 4wd   4.7  2008     8   auto(l5)     4     9    12     e pickup
2        dodge         durango 4wd   4.7  2008     8   auto(l5)     4     9    12     e    suv
3        dodge ram 1500 pickup 4wd   4.7  2008     8   auto(l5)     4     9    12     e pickup
4        dodge ram 1500 pickup 4wd   4.7  2008     8 manual(m6)     4     9    12     e pickup
5         jeep  grand cherokee 4wd   4.7  2008     8   auto(l5)     4     9    12     e    suv
6    chevrolet     k1500 tahoe 4wd   5.3  2008     8   auto(l4)     4    11    14     e    suv
```

REORDERING OBSERVATIONS

```
> autoData <- arrange(autoData, desc(hwy))

> head(autoData)
# A tibble: 6 x 11
  manufacturer      model displ  year   cyl      trans   drv   cty   hwy    fl      class
         <chr>      <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>      <chr>
1   volkswagen      jetta   1.9  1999     4 manual(m5)     f    33    44     d    compact
2   volkswagen new beetle   1.9  1999     4 manual(m5)     f    35    44     d subcompact
3   volkswagen new beetle   1.9  1999     4   auto(l4)     f    29    41     d subcompact
4       toyota    corolla   1.8  2008     4 manual(m5)     f    28    37     r    compact
5        honda      civic   1.8  2008     4   auto(l5)     f    25    36     r subcompact
6        honda      civic   1.8  2008     4   auto(l5)     f    24    36     c subcompact
```

SUBSETTING DATA



SUBSETTING DATA

`dplyr::filter(`*`dataFrame`*`, `*`expression`*`)`

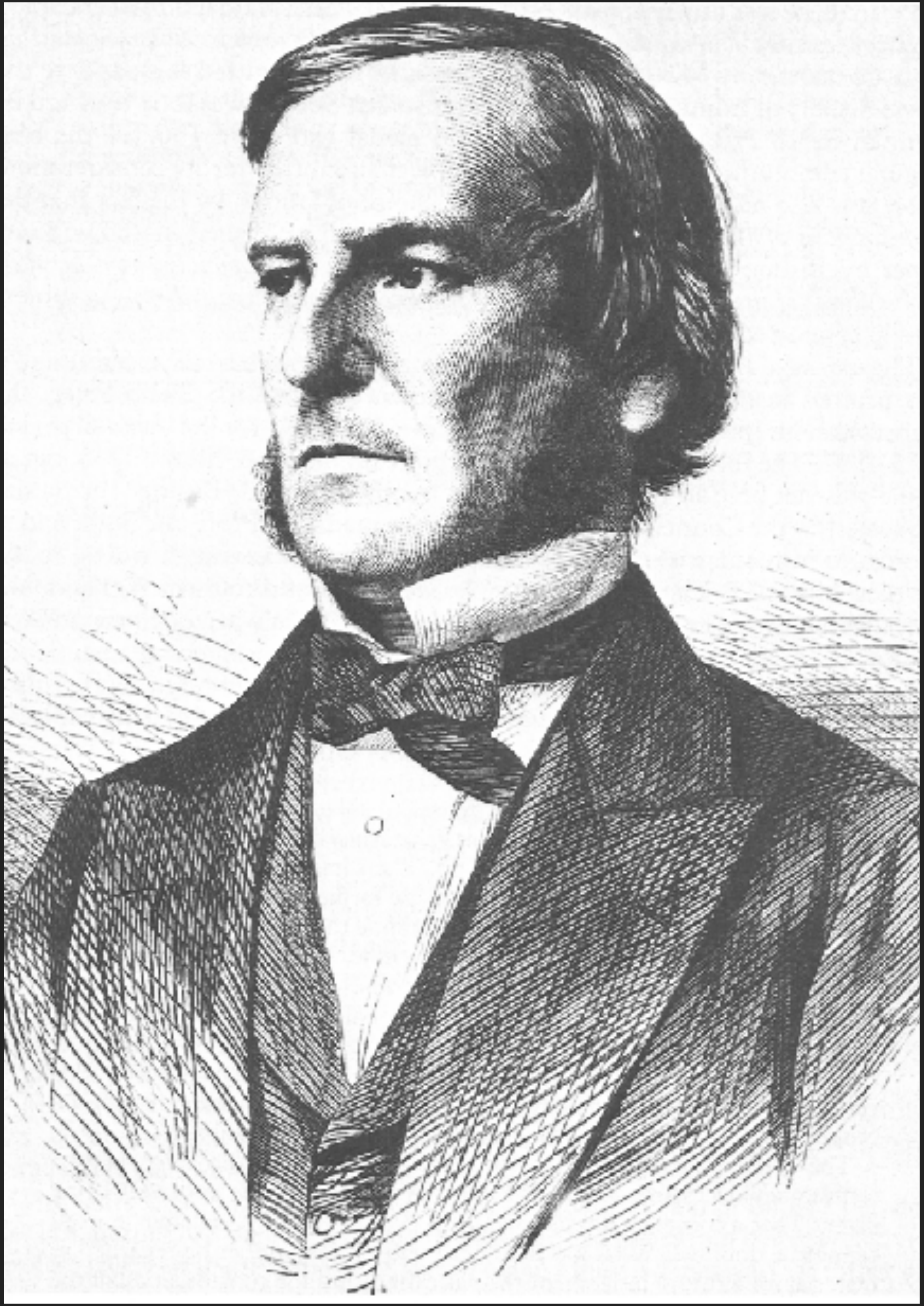
Example - the `mpg` data from `ggplot2` filtered using a numeric value:

```
filter(mpg, hwy >= 30)
```

This will *retain* only observations that are TRUE based on the expression.

GEORGE BOOLE

- British mathematician who was active during the 1840s and 1850s
- Credited with establishing the field of boolean algebra in papers published in 1847 and 1854
- Boolean algebra is premised on the idea that logical relations can be used evaluate expressions as either TRUE or FALSE
- Boolean logic is a fundamental concept for modern computing



BOOLEAN LOGIC

```
filter(mpg, hwy >= 30)
```

↓

model	year	hwy	boolean eval.
a4	1999	29	FALSE
forester awd	2008	23	FALSE
corolla	2008	35	TRUE

↓

model	year	hwy
corolla	2008	35

SUBSETTING DATA

```
dplyr::filter(dataFrame, expression)
```

Example - the `mpg` data from `ggplot2` filtered using a string:

```
filter(mpg, manufacturer == "subaru")
```

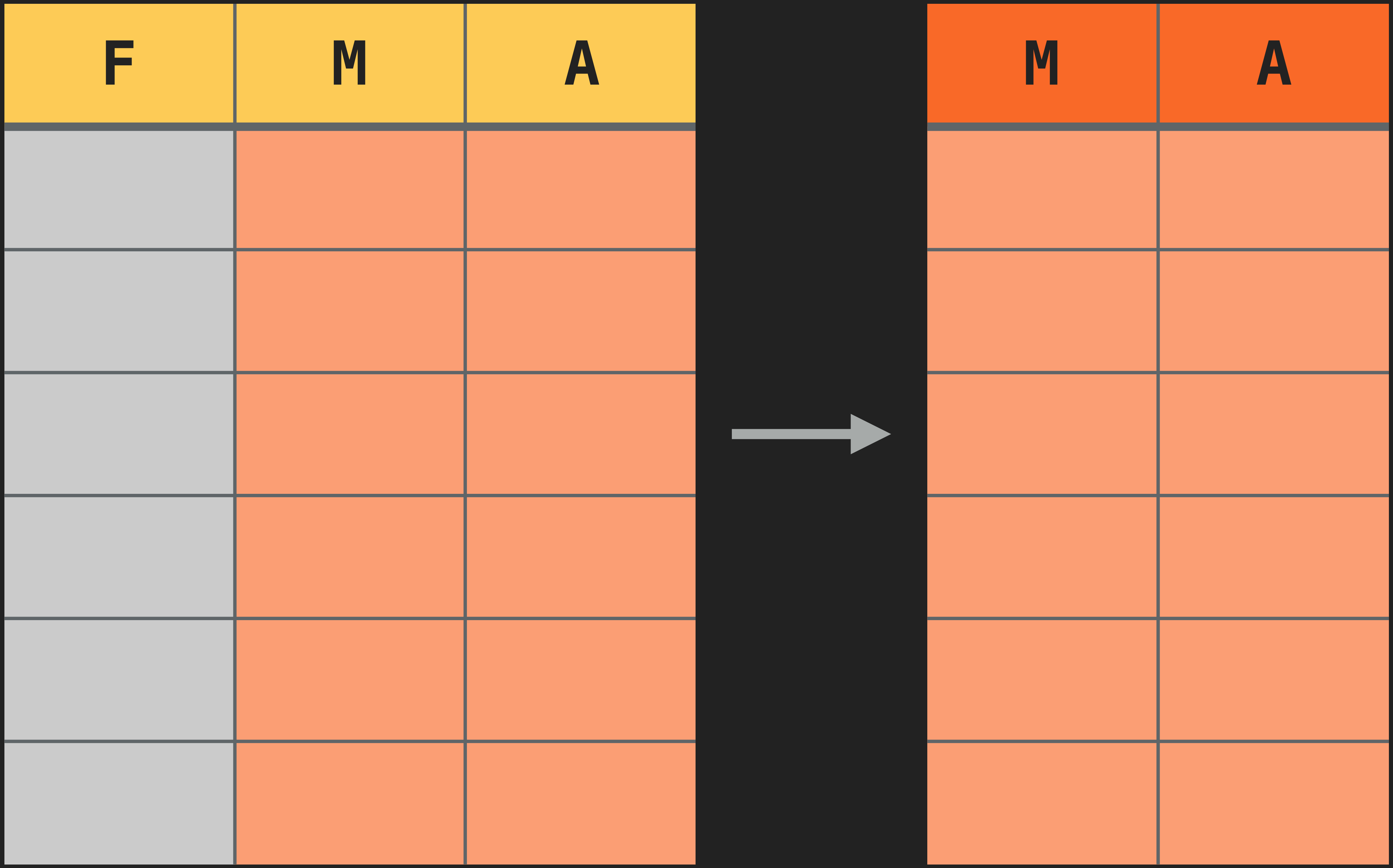
This will *retain* only observations that are TRUE based on the expression.

This method of searching strings is case sensitive and will only evaluate as TRUE for exact matches. There are more flexible ways to search strings as well.

SUBSETTING DATA

```
> library(tidyverse)

> subaru <- filter(mpg, manufacturer == "subaru")

> str(subaru)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   14 obs. of  11 variables:
 $ manufacturer: chr  "subaru" "subaru" "subaru" "subaru" ...
 $ model       : chr  "forester awd" "impreza awd" "impreza awd" "forester awd" ...
 $ displ       : num  2.5 2.5 2.5 2.5 2.2 2.2 2.5 2.5 2.5 2.5 ...
 $ year        : int  2008 2008 2008 2008 1999 1999 1999 1999 1999 2008 ...
 $ cyl         : int  4 4 4 4 4 4 4 4 4 4 ...
 $ trans       : chr  "manual(m5)" "auto(s4)" "manual(m5)" "auto(l4)" ...
 $ drv         : chr  "4" "4" "4" "4" ...
 $ cty         : int  20 20 20 20 21 19 19 19 18 19 ...
 $ hwy         : int  27 27 27 26 26 26 26 26 25 25 ...
 $ fl          : chr  "r" "r" "r" "r" ...
 $ class       : chr  "suv" "compact" "compact" "suv" ...
```

SUBSETTING DATA



SUBSETTING DATA

```
dplyr::select(dataFrame, varlist)
```

Example - the mpg data from `ggplot2`:

```
select(mpg, manufacturer, model, hwy, class)
```

This approach will *retain* only the listed variables.

There are additional helper functions for searching

SUBSETTING DATA

```
> library(tidyverse)

> autoData <- select(mpg, manufacturer, model, hwy, class)

> str(autoData)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   234 obs. of  4 variables:
 $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
 $ model       : chr  "a4" "a4" "a4" "a4" ...
 $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
 $ class       : chr  "compact" "compact" "compact" "compact" ...
```

SUBSETTING DATA

```
dplyr::select(dataFrame, -varlist)
```

Example - the mpg data from `ggplot2`:

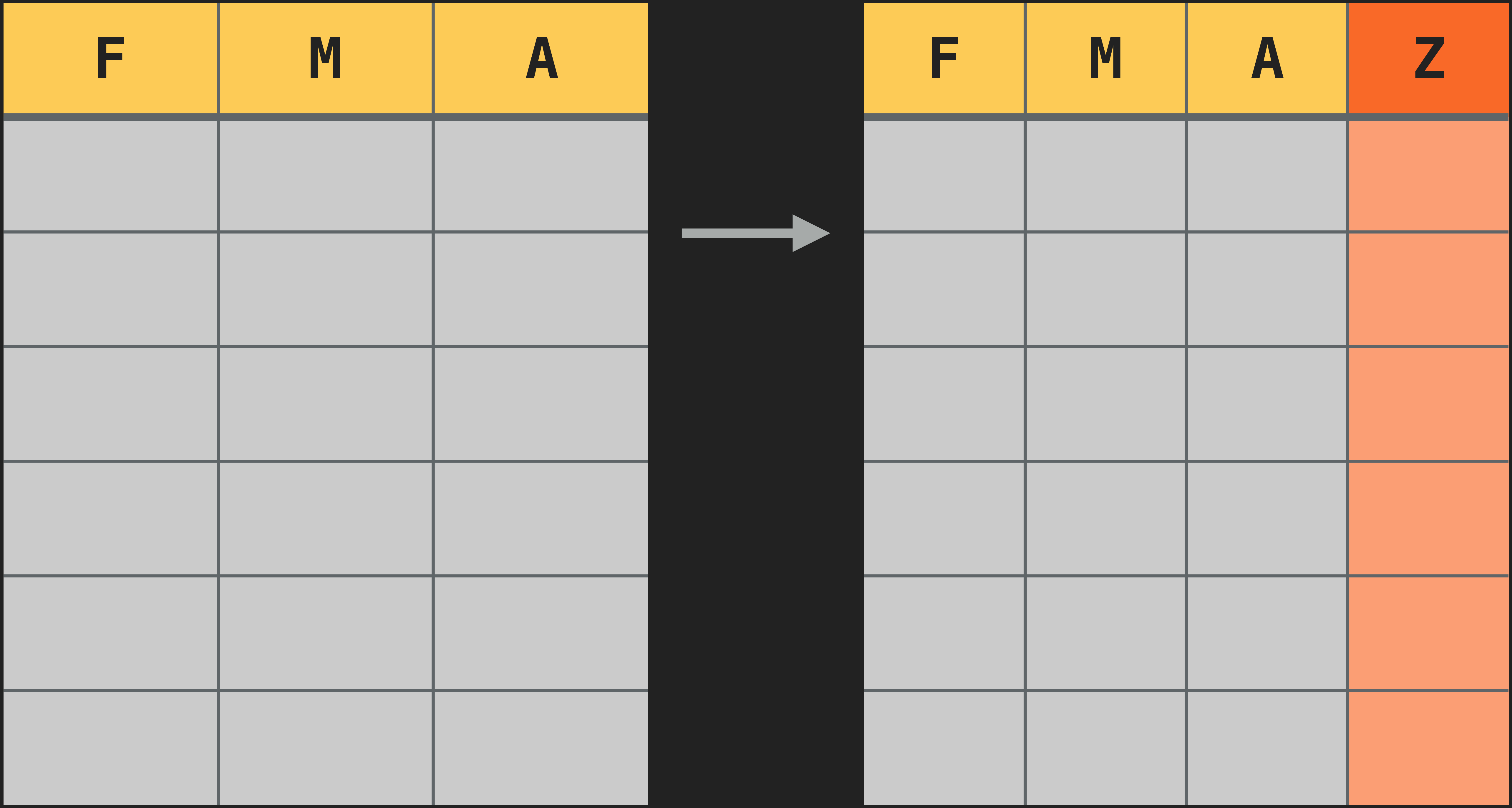
```
select(mpg, -manufacturer, -model, -hwy, -class)
```

This approach will *remove* only the listed variables.

SUBSETTING DATA

```
> library(tidyverse)

> autoData <- select(mpg, -manufacturer, -model, -hwy, -class)

> str(autoData)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   234 obs. of  7 variables:
 $ displ: num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl  : int  4 4 4 4 6 6 6 4 4 4 ...
 $ trans: chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv  : chr  "f" "f" "f" "f" ...
 $ cty  : int  18 21 20 21 16 18 18 18 16 20 ...
 $ fl   : chr  "p" "p" "p" "p" ...
```

CREATING NEW VARIABLES



CREATING NEW VARIABLES

```
dplyr::mutate(dataFrame, newVar = expression)
```

Example - numerical calculation with the `mpg` data from `ggplot2`:

```
mutate(mpg, avgMpg = (cty+hwy)/2)
```

Requires numeric data

CREATING NEW VARIABLES

```
dplyr::mutate(dataFrame, newVar =

       ifelse(expression, trueOutcome, falseOutcome))
```

Example - binary variable creation with the `mpg` data from `ggplot2`:

```
mutate(mpg, highMpg = ifelse(hwy >= 30, TRUE, FALSE))
```

Requires numeric data.

True and false expressions can be either logical, character, or numeric data. You should be consistent in keeping both the true and false expressions as the same data type.

CREATING NEW VARIABLES

```
dplyr::mutate(dataFrame, newVar =
        ifelse(expression, trueOutcome, falseOutcome))
```

Example - binary variable creation with the `mpg` data from `ggplot2`:

```
mutate(mpg, subaru =
  ifelse(manufacturer == “subaru”, TRUE, FALSE))
```

Requires string data.

This method of searching strings is case sensitive and will only evaluate as TRUE for exact matches. There are more flexible ways to search strings as well.

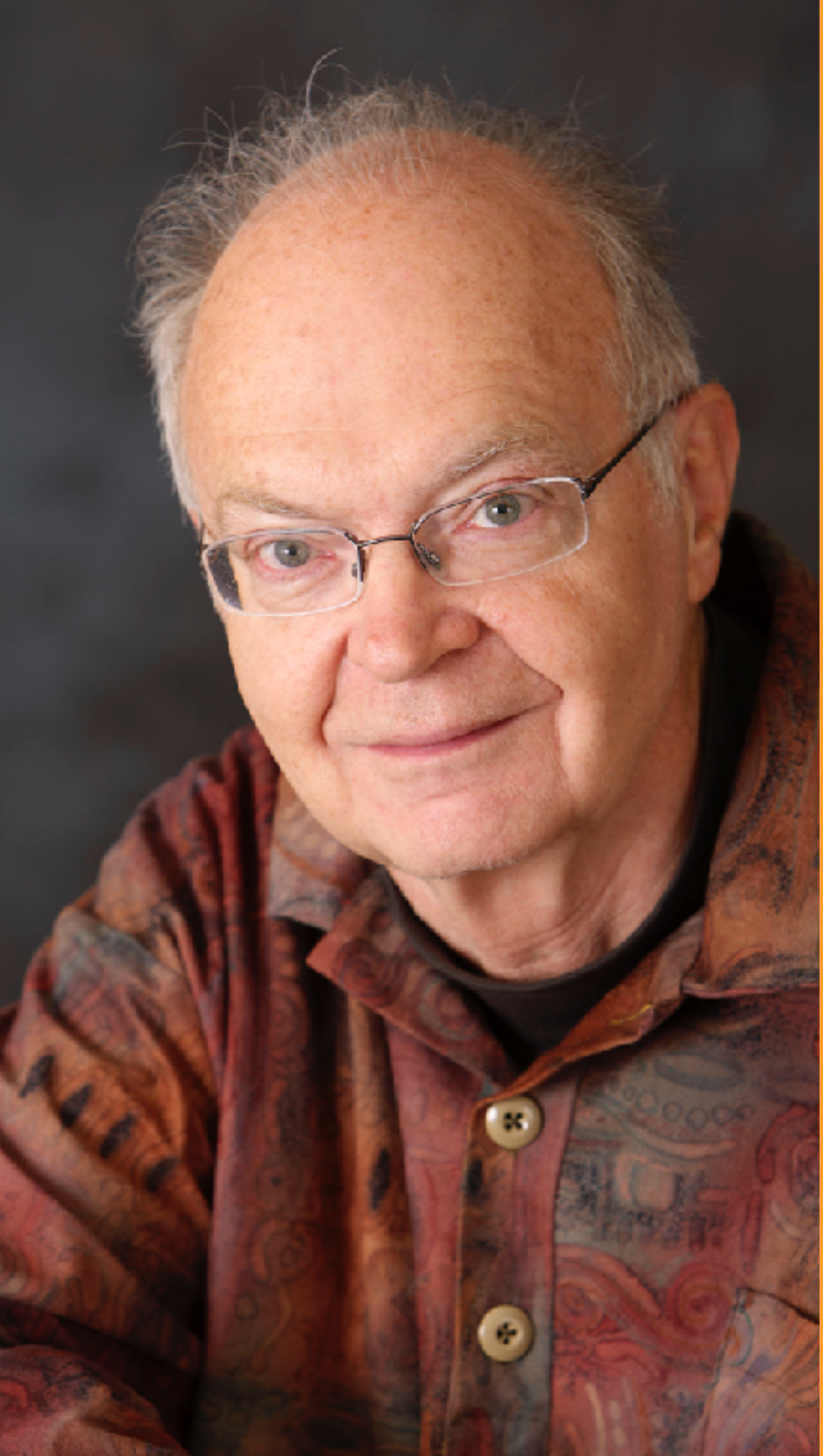
CREATING NEW VARIABLES

```
> library(tidyverse)

> autoData <- mpg

> mutate(autoData, subaru = ifelse(manufacturer == “subaru”, TRUE, FALSE))

> table(autoData$subaru)

FALSE  TRUE
  220    14
```

3 PIPING DATA

ASSIGNING DATA CAN GET CUMBERSOME

```
> library(tidyverse)

> japaneseAutos <- mpg

> japaneseAutos <-
    select(japaneseAutos, model, cty, hwy)

> japaneseAutos <-
    rename(japaneseAutos, cityMpg = cty)

> japaneseAutos <-
    rename(japaneseAutos, hwyMpg = hwy)

> japaneseAutos <-
    filter(japaneseAutos, manufacturer == “honda” |
    manufacturer == “nissan” |
    manufacturer == “subaru” |
    manufacturer == “toyota”)

> japaneseAutos <-
    mutate(japaneseAutos, avgMpg =
    (cityMpg+hwyMpg)/2)

> japaneseAutos <- arrange(japaneseAutos, avgMpg)
```



LET US CHANGE OUR TRADITIONAL ATTITUDE TO THE CONSTRUCTION OF PROGRAMS: INSTEAD OF IMAGINING THAT OUR MAIN TASK IS TO INSTRUCT A COMPUTER WHAT TO DO, LET US CONCENTRATE RATHER ON EXPLAINING TO HUMANS WHAT WE WANT THE COMPUTER TO DO.

Donald E. Knuth

Stanford University Computer Scientist

MAGRITTR PACKAGE

- `dplyr` automatically loads the `magrittr` package
- `magrittr` includes a number of helpful functions, but is most well know for the “pipe”:

  %>%

- Piping data makes it easier to write and more readable for humans

ASSIGNING DATA CAN GET CUMBERSOME

```
> library(tidyverse)

> japaneseAutos <- mpg

> japaneseAutos <-
    select(japaneseAutos, model, cty, hwy)

> japaneseAutos <-
    rename(japaneseAutos, cityMpg = cty)

> japaneseAutos <-
    rename(japaneseAutos, hwyMpg = hwy)

> japaneseAutos <-
    filter(japaneseAutos, manufacturer == “honda” |
    manufacturer == “nissan” |
    manufacturer == “subaru” |
    manufacturer == “toyota”)

> japaneseAutos <-
    mutate(japaneseAutos, avgMpg =
    (cityMpg+hwyMpg)/2)

> japaneseAutos <- arrange(japaneseAutos, avgMpg)
```

```
> library(tidyverse)

> mpg %>%
    select(manufacturer,
        model, cty, hwy) %>%
    rename(cityMpg = cty) %>%
    rename(hwyMpg = hwy) %>%
    filter(manufacturer == “honda” |
        manufacturer == “nissan” |
        manufacturer == “subaru” |
        manufacturer == “toyota”) %>%
    mutate(avgMpg =
        (cityMpg+hwyMpg)/2) %>%
    arrange(avgMpg) -> japaneseAutos
```

READING PIPES

- Pipes can be read in sequential order:
  1. Take the `mpg` data frame, then
  2. select the manufacturer, model, and fuel efficiency variables, then
  3. rename the city gas mileage variable, then
  4. rename the highway gas mileage variable, then
  5. filter observations for Japanese automobile manufacturers, then
  6. create a new average miles per gallon variable, then
  7. arrange observations from high to low based on the new fuel efficiency variable, then
  8. assign these changes to a new data frame named `japaneseAutos`

```
> library(tidyverse)

> mpg %>%
    select(manufacturer,
       model, cty, hwy) %>%
    rename(cityMpg = cty) %>%
    rename(hwyMpg = hwy) %>%
    filter(manufacturer == “honda” |
       manufacturer == “nissan” |
       manufacturer == “subaru” |
       manufacturer == “toyota”) %>%
    mutate(avgMpg =
       (cityMpg+hwyMpg)/2) %>%
    arrange(avgMpg) -> japaneseAutos
```

READING PIPES

- Pipes can be read in sequential order:

1. Take the `mpg` data frame, then
2. select the manufacturer, model, and fuel efficiency variables, then
3. rename the city gas mileage variable, then
4. rename the highway gas mileage variable, then
5. filter observations for Japanese automobile manufacturers, then
6. create a new average miles per gallon variable, then
7. arrange observations from high to low based on the new fuel efficiency variable, then
8. assign these changes to a new data frame named `japaneseAutos`

```
> library(tidyverse)

> mpg %>%
    select(manufacturer,
       model, cty, hwy) %>%
    rename(cityMpg = cty) %>%
    rename(hwyMpg = hwy) %>%
    filter(manufacturer == “honda” |
       manufacturer == “nissan” |
       manufacturer == “subaru” |
       manufacturer == “toyota”) %>%
    mutate(avgMpg =
       (cityMpg+hwyMpg)/2) %>%
    arrange(avgMpg) -> japaneseAutos
```

READING PIPES

- The final assignment can also be made on the first line of code like the example to the right
- I prefer the initial method only because the code “reads” in a linear fashion
- In either case, the data reference in each function can be omitted since it is “passed” by the pipe operator
- Pipes should be *short*

```
> library(tidyverse)

> japaneseAutos <- mpg %>%
    select(manufacturer,
        model, cty, hwy) %>%
    rename(cityMpg = cty) %>%
    rename(hwyMpg = hwy) %>%
    filter(manufacturer == “honda” |
        manufacturer == “nissan” |
        manufacturer == “subaru” |
        manufacturer == “toyota”) %>%
    mutate(avgMpg =
        (cityMpg+hwyMpg)/2) %>%
    arrange(avgMpg)
```

PIPES AND GGPLOT2

- If we remove the data assignment, pipes still work!
- They will temporarily alter the data without making those changes permanent
- This is perfect behavior for making `ggplot` plots on a modified set of data without creating a new data frame
- Note that the data reference is not needed in the `ggplot` function

```
> library(tidyverse)

> mpg %>%
    select(manufacturer,
       model, cty, hwy) %>%
    rename(cityMpg = cty) %>%
    rename(hwyMpg = hwy) %>%
    filter(manufacturer == “honda” |
       manufacturer == “nissan” |
       manufacturer == “subaru” |
       manufacturer == “toyota”) %>%
    mutate(avgMpg =
       (cityMpg+hwyMpg)/2) %>%
    arrange(avgMpg) %>%
    ggplot() +
       geom_histogram(mapping =
          aes(avgMpg))
```