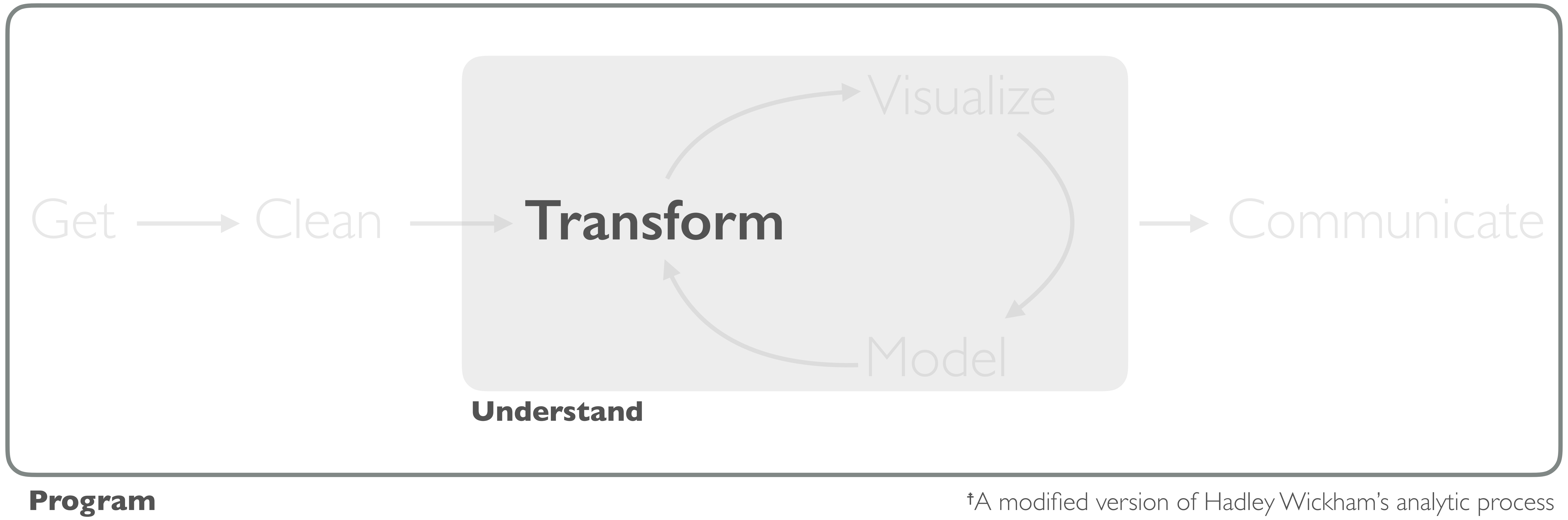
DATA TRANSFORMATION



†A modified version of Hadley Wickham's analytic process

`dplyr`

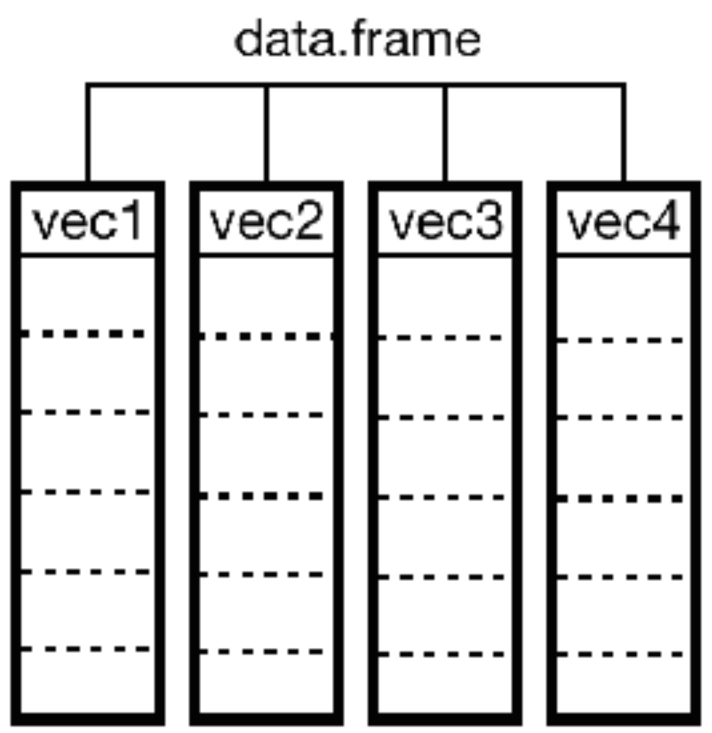
You are going to learn the five key **`dplyr`** functions that allow you to solve the vast majority of your data manipulation challenges:

- `filter`: pick observations based on values
- `arrange`: reorder data
- `select`: pick variables
- `mutate`: create new variables
- `summarise`: summarize data by functions of choice

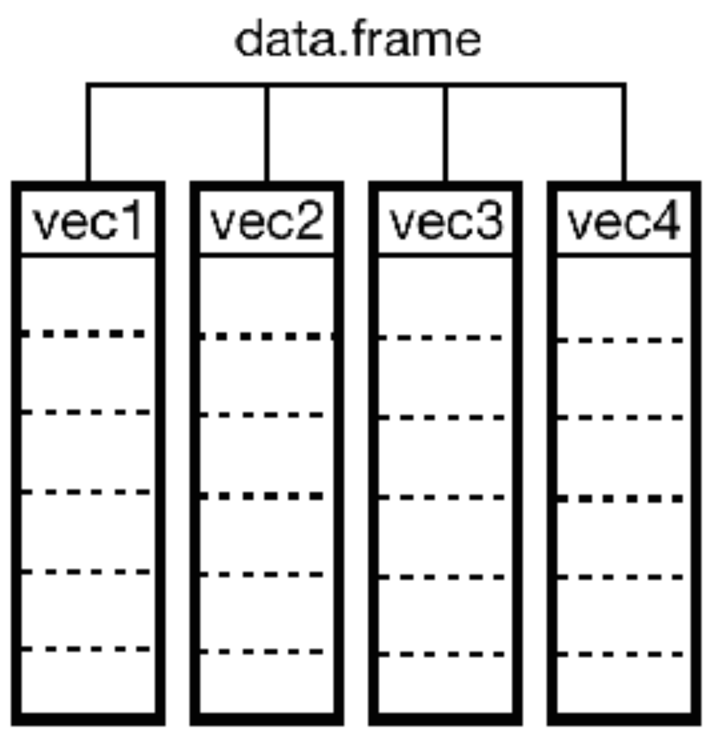
BASICS

All functions work similarly:

- The first argument is a data frame
- Subsequent arguments describe what to do
- Output is a new data frame



`f(data, arguments)`



PREREQUISITES

PACKAGE PREREQUISITE

```
library(nycflights13)
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages ----------------------------------------------
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

DATA PREREQUISITE

```
flights
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013     1     1      517            515         2      830            819        11      UA   1545
2   2013     1     1      533            529         4      850            830        20      UA   1714
3   2013     1     1      542            540         2      923            850        33      AA   1141
4   2013     1     1      544            545        -1     1004           1022       -18      B6    725
5   2013     1     1      554            600        -6      812            837       -25      DL    461
6   2013     1     1      554            558        -4      740            728        12      UA   1696
7   2013     1     1      555            600        -5      913            854        19      B6    507
8   2013     1     1      557            600        -3      709            723       -14      EV   5708
9   2013     1     1      557            600        -3      838            846        -8      B6     79
10  2013     1     1      558            600        -2      753            745         8      AA    301
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

YOUR TURN!

Are there vignettes for the **`dplyr`** *package?*

Can you find additional documentation explaining the **`flights`** *data set?*

SOLUTION

```
# are there vignettes for the dplyr package -> yes, 8 of them
vignette(package = "dplyr")


# additional documentation for the mpg data set
?flights
```

filter

Filter values based on defined conditions

BASIC FILTERING

Filter based on one or more variables

```
filter(flights, month == 1)
# A tibble: 27,004 × 19
     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1    2013     1     1      517            515         2      830            819        11
2    2013     1     1      533            529         4      850            830        20
3    2013     1     1      542            540         2      923            850        33
4    2013     1     1      544            545        -1     1004           1022       -18
5    2013     1     1      554            600        -6      812            837       -25
6    2013     1     1      554            558        -4      740            728        12
7    2013     1     1      555            600        -5      913            854        19
8    2013     1     1      557            600        -3      709            723       -14
9    2013     1     1      557            600        -3      838            846        -8
```

BASIC FILTERING

Filter based on one or more variables

```
filter(flights, month == 1, day == 1)
# A tibble: 842 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1  2013     1     1      517            515         2      830            819        11
2  2013     1     1      533            529         4      850            830        20
3  2013     1     1      542            540         2      923            850        33
4  2013     1     1      544            545        -1     1004           1022       -18
5  2013     1     1      554            600        -6      812            837       -25
6  2013     1     1      554            558        -4      740            728        12
7  2013     1     1      555            600        -5      913            854        19
8  2013     1     1      557            600        -3      709            723       -14
9  2013     1     1      557            600        -3      838            846        -8
```

BASIC FILTERING

Filter based on one or more variables

```
filter(flights, month == 1, day == 1, dep_delay > 0)
# A tibble: 352 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2      830            819        11
2   2013     1     1      533            529         4      850            830        20
3   2013     1     1      542            540         2      923            850        33
4   2013     1     1      601            600         1      844            850        -6
5   2013     1     1      608            600         8      807            735        32
6   2013     1     1      611            600        11      945            931        14
7   2013     1     1      613            610         3      925            921         4
8   2013     1     1      623            610        13      920            915         5
9   2013     1     1      632            608        24      740            728        12
```

SAVE NEW DATA FRAME

- Save filter data frame using assignment operator (`<-`)

```
dec25 <- filter(flights, month == 12, day == 25)
dec25
# A tibble: 719 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1  2013    12    25      456            500        -4      649            651        -2
2  2013    12    25      524            515         9      805            814        -9
3  2013    12    25      542            540         2      832            850       -18
4  2013    12    25      546            550        -4     1022           1027        -5
5  2013    12    25      556            600        -4      730            745       -15
6  2013    12    25      557            600        -3      743            752        -9
7  2013    12    25      557            600        -3      818            831       -13
```

LOGICAL TESTS

```r
12 == 12
[1] TRUE

12 <= c(12, 11)
[1]  TRUE FALSE

12 %in% c(12, 11, 8)
[1] TRUE

x <- c(12, NA, 11, NA, 8)
is.na(x)
[1] FALSE  TRUE FALSE  TRUE FALSE
```

?Comparison

`<`	Less than
`>`	Greater than
`==`	Equal to
`<=`	Less than or equal to
`>=`	Greater than or equal to
`!=`	Not equal to
`%in%`	Group membership
`is.na`	Is NA
`!is.na`	Is not NA

COMPARISON

What will these operations produce?

```
filter(flights, month == 12)
filter(flights, month != 12)
filter(flights, month %in% c(11, 12))
filter(flights, arr_delay <= 120)
filter(flights, !(arr_delay <= 120))
filter(flights, is.na(tailnum))
```

MULTIPLE LOGICAL TESTS

```
12 == 12 & 12 < 14
[1] TRUE

12 == 12 & 12 < 10
[1] FALSE

12 == 12 | 12 < 10
[1] TRUE

any(12 == 12, 12 < 10)
[1] TRUE

all(12 == 12, 12 < 10)
[1] FALSE
```

`?base::Logic`

`&`	boolean and
`|`	boolean or
`xor`	exclusively x or y
`!`	not
`any`	any true
`all`	all true

MULTIPLE COMPARISONS

Using comma is same as using `&`

```
filter(flights, month == 12, day == 25)
filter(flights, month == 12 & day == 25)
```

Use `%in%` as a shortcut for `|`

```
filter(flights, month == 11 | month == 12)
filter(flights, month %in% c(11, 12))
```

Are these the same????

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

YOUR TURN!

1. Import the CustomerData.csv file.
2. Filter for female customers only.
3. Filter for female customers that are greater than 45 years old ***and*** live in region 3.
4. Filter for female customers that are greater than 45 years old ***or*** live in region 3.

SOLUTION

```
# 1: import the data
customer <- read_csv("data/CustomerData.csv")

# 2: filter for female customers only
filter(customer, Gender == "Female")

# 3: filter for female customers that are greater than 45 years old and live in region 3
filter(customer, Gender == "Female", Age > 45, Region == 3)

# 4: filter for female customers that are greater than 45 years old or live in region 3
filter(customer, Gender == "Female", Age > 45 | Region == 3)
```

select

Select variables of concern

SELECTING VARIABLES

Select one or more variables

```
select(flights, year, month, day)
# A tibble: 336,776 × 3
   year month   day
  <int> <int> <int>
1  2013     1     1
2  2013     1     1
3  2013     1     1
4  2013     1     1
5  2013     1     1
6  2013     1     1
7  2013     1     1
8  2013     1     1
9  2013     1     1
```

Same → Results

```
select(flights, year:day)
# A tibble: 336,776 × 3
   year month   day
  <int> <int> <int>
1  2013     1     1
2  2013     1     1
3  2013     1     1
4  2013     1     1
5  2013     1     1
6  2013     1     1
7  2013     1     1
8  2013     1     1
9  2013     1     1
```

SELECTING VARIABLES

Deselect one or more variables

```
select(flights, -(year:day))
# A tibble: 336,776 × 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
      <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1       517            515         2      830            819        11      UA   1545
2       533            529         4      850            830        20      UA   1714
3       542            540         2      923            850        33      AA   1141
4       544            545        -1     1004           1022       -18      B6    725
5       554            600        -6      812            837       -25      DL    461
6       554            558        -4      740            728        12      UA   1696
7       555            600        -5      913            854        19      B6    507
8       557            600        -3      709            723       -14      EV   5708
9       557            600        -3      838            846        -8      B6     79
10      558            600        -2      753            745         8      AA    301
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>,
```

USEFUL `select` FUNCTIONS

* Blue functions come in `dplyr`

`-`	Select everything but
`:`	Select range
`contains()`	Select columns whose name contains a character string
`ends_with()`	Select columns whose name ends with a string
`everything()`	Select every column
`matches()`	Select columns whose name matches a regular expression
`num_range()`	Select columns named `x1, x2, x3, x4, x5`
`one_of()`	Select columns whose names are in a group of names
`starts_with()`	Select columns whose name starts with a character string

SELECTING VARIABLES

Select variables based on name patterns

```
select(flights, ends_with("time"))
# A tibble: 336,776 × 5
   dep_time sched_dep_time arr_time sched_arr_time air_time
      <int>          <int>    <int>          <int>    <dbl>
1       517            515      830            819      227
2       533            529      850            830      227
3       542            540      923            850      160
4       544            545     1004           1022      183
5       554            600      812            837      116
6       554            558      740            728      150
7       555            600      913            854      158
8       557            600      709            723       53
9       557            600      838            846      140
10      558            600      753            745      138
```

SELECTING VARIABLES

Select variables based on multiple name patterns

```
select(flights, c(carrier, ends_with("time"), contains("delay")))
# A tibble: 336,776 × 8
   carrier dep_time sched_dep_time arr_time sched_arr_time air_time dep_delay arr_delay
     <chr>    <int>          <int>    <int>          <int>    <dbl>     <dbl>     <dbl>
1       UA      517            515      830            819      227         2        11
2       UA      533            529      850            830      227         4        20
3       AA      542            540      923            850      160         2        33
4       B6      544            545     1004           1022      183        -1       -18
5       DL      554            600      812            837      116        -6       -25
6       UA      554            558      740            728      150        -4        12
7       B6      555            600      913            854      158        -5        19
8       EV      557            600      709            723       53        -3       -14
9       B6      557            600      838            846      140        -3        -8
10      AA      558            600      753            745      138        -2         8
```

VARIABLE PLACEMENT

Sometimes we just want to change the order of variables

```
select(flights, time_hour, air_time, everything())
# A tibble: 336,776 × 19
             time_hour air_time  year month   day dep_time sched_dep_time dep_delay arr_time
                <dttm>    <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013-01-01 05:00:00      227  2013     1     1      517            515         2      830
2  2013-01-01 05:00:00      227  2013     1     1      533            529         4      850
3  2013-01-01 05:00:00      160  2013     1     1      542            540         2      923
4  2013-01-01 05:00:00      183  2013     1     1      544            545        -1     1004
5  2013-01-01 06:00:00      116  2013     1     1      554            600        -6      812
6  2013-01-01 05:00:00      150  2013     1     1      554            558        -4      740
7  2013-01-01 06:00:00      158  2013     1     1      555            600        -5      913
8  2013-01-01 06:00:00       53  2013     1     1      557            600        -3      709
9  2013-01-01 06:00:00      140  2013     1     1      557            600        -3      838
10 2013-01-01 06:00:00      138  2013     1     1      558            600        -2      753
```

RENAMING VARIABLES

Other times we just want to rename our variables:

```
rename(flights, ANNOYING = dep_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time ANNOYING arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>    <dbl>    <int>          <int>     <dbl>
1   2013     1     1      517            515        2      830            819        11
2   2013     1     1      533            529        4      850            830        20
3   2013     1     1      542            540        2      923            850        33
4   2013     1     1      544            545       -1     1004           1022       -18
5   2013     1     1      554            600       -6      812            837       -25
6   2013     1     1      554            558       -4      740            728        12
7   2013     1     1      555            600       -5      913            854        19
8   2013     1     1      557            600       -3      709            723       -14
9   2013     1     1      557            600       -3      838            846        -8
10  2013     1     1      558            600       -2      753            745         8
```

YOUR TURN!

1. Using the customer data, select all columns between CustomerID and Gender.

2. Now select all columns other than those between columns between CustomerID and Gender.

3. Select CustomerID and all variables that contain the word "Card".

SOLUTION

```
# 1. select all variables between CustomerID and Gender
select(customer, CustomerID:Gender)

# 2. select all variables except for those between CustomerID and Gender
select(customer, -(CustomerID:Gender))

#3. select CustomerID and all variables that contain the word “Card”
select(customer, CustomerID, contains(“Card”))
```

arrange

Reorder data

ORDERING YOUR DATA

Order data based on one or more variables

```
arrange(flights, dep_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013    12     7     2040           2123       -43       40           2352        48      B6     97
2   2013     2     3     2022           2055       -33     2240           2338       -58      DL   1715
3   2013    11    10     1408           1440       -32     1549           1559       -10      EV   5713
4   2013     1    11     1900           1930       -30     2233           2243       -10      DL   1435
5   2013     1    29     1703           1730       -27     1947           1957       -10      F9    837
6   2013     8     9      729            755       -26     1002            955         7      MQ   3478
7   2013    10    23     1907           1932       -25     2143           2143         0      EV   4361
8   2013     3    30     2030           2055       -25     2213           2250       -37      MQ   4573
9   2013     3     2     1431           1455       -24     1601           1631       -30      9E   3318
10  2013     5     5      934            958       -24     1225           1309       -44      B6    375
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>
```

ORDERING YOUR DATA

Order data based on one or more variables

```
arrange(flights, dep_delay, arr_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013    12     7     2040           2123       -43       40           2352        48      B6     97
2   2013     2     3     2022           2055       -33     2240           2338       -58      DL   1715
3   2013    11    10     1408           1440       -32     1549           1559       -10      EV   5713
4   2013     1    11     1900           1930       -30     2233           2243       -10      DL   1435
5   2013     1    29     1703           1730       -27     1947           1957       -10      F9    837
6   2013     8     9      729            755       -26     1002            955         7      MQ   3478
7   2013     3    30     2030           2055       -25     2213           2250       -37      MQ   4573
8   2013    10    23     1907           1932       -25     2143           2143         0      EV   4361
9   2013     5     5      934            958       -24     1225           1309       -44      B6    375
10  2013     9    18     1631           1655       -24     1812           1845       -33      AA   2223
# … with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>
```

ORDERING YOUR DATA

Reverse the order by using **`desc()`**

```
arrange(flights, desc(dep_delay))
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>   <chr>  <int>
1   2013     1     9      641            900      1301     1242           1530      1272      HA     51
2   2013     6    15     1432           1935      1137     1607           2120      1127      MQ   3535
3   2013     1    10     1121           1635      1126     1239           1810      1109      MQ   3695
4   2013     9    20     1139           1845      1014     1457           2210      1007      AA    177
5   2013     7    22      845           1600      1005     1044           1815       989      MQ   3075
6   2013     4    10     1100           1900       960     1342           2211       931      DL   2391
7   2013     3    17     2321            810       911      135           1020       915      DL   2119
8   2013     6    27      959           1900       899     1236           2226       850      DL   2007
9   2013     7    22     2257            759       898      121           1026       895      DL   2047
10  2013    12     5      756           1700       896     1058           2020       878      AA    172
# … with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
```

ORDERING YOUR DATA

Note that missing values are always sorted at the end:

```
df <- tibble(x = c(5, 2, 5, NA))
# A tibble: 4 × 1
      x
  <dbl>
1     5
2     2
3     5
4    NA
```

```
arrange(df, x)
# A tibble: 4 × 1
      x
  <dbl>
1     2
2     5
3     5
4    NA
```

```
arrange(df, desc(x))
# A tibble: 4 × 1
      x
  <dbl>
1     5
2     5
3     2
4    NA
```

YOUR TURN!

1. Select the variables CustomerID, Region, Gender, Age, HHIncome, Cardspend and save this as **`sub_cust`**.

2. Order **`sub_cust`** data by Age and CardSpendMonth (ascending order)

3. Order **`sub_cust`** data by Age (oldest to youngest) and CardSpendMonth (least to most)

SOLUTION

```
# 1: select variables
sub_cust <- select(customer, CustomerID, Region, Gender, Age, HHIncome, CardSpendMonth)

# 2: Order sub_cust data by Age and CardSpendMonth (ascending order)
arrange(customer, Age, CardSpendMonth)

# 3: Order sub_cust data by Age (oldest to youngest) and CardSpendMonth (least to most)
arrange(customer, desc(Age), CardSpendMonth)
```

`mutate`

Create new variables with functions of existing variables

REDUCE OUR DATA

Lets work with a smaller data set

```
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)

flights_sml
# A tibble: 336,776 × 7
    year month   day dep_delay arr_delay distance air_time
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
1   2013     1     1         2        11     1400      227
2   2013     1     1         4        20     1416      227
3   2013     1     1         2        33     1089      160
4   2013     1     1        -1       -18     1576      183
5   2013     1     1        -6       -25      762      116
6   2013     1     1        -4        12      719      150
```

CREATE NEW VARIABLES

`mutate()` creates new variables with functions of existing variables:

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60
)
# A tibble: 336,776 × 9
    year month   day dep_delay arr_delay distance air_time  gain    speed
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>    <dbl>
1   2013     1     1         2        11     1400      227     9 370.0441
2   2013     1     1         4        20     1416      227    16 374.2731
3   2013     1     1         2        33     1089      160    31 408.3750
4   2013     1     1        -1       -18     1576      183   -17 516.7213
5   2013     1     1        -6       -25      762      116   -19 394.1379
6   2013     1     1        -4        12      719      150    16 287.6000
7   2013     1     1        -5        19     1065      158    24 404.4304
8   2013     1     1        -3       -14      229       53   -11 259.2453
9   2013     1     1        -3        -8      944      140    -5 404.5714
```

CREATE NEW VARIABLES

Note: you can create variables based on columns that you've just created:

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
# A tibble: 336,776 × 10
   year month   day dep_delay arr_delay distance air_time  gain     hours gain_per_hour
  <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>     <dbl>         <dbl>
1  2013     1     1         2        11     1400      227     9 3.7833333      2.378855
2  2013     1     1         4        20     1416      227    16 3.7833333      4.229075
3  2013     1     1         2        33     1089      160    31 2.6666667     11.625000
4  2013     1     1        -1       -18     1576      183   -17 3.0500000     -5.573770
5  2013     1     1        -6       -25      762      116   -19 1.9333333     -9.827586
6  2013     1     1        -4        12      719      150    16 2.5000000      6.400000
7  2013     1     1        -5        19     1065      158    24 2.6333333      9.113924
8  2013     1     1        -3       -14      229       53   -11 0.8833333    -12.452830
```

CREATE NEW VARIABLES

If you only want to keep the new variables use `transmute()`:

```
transmute(flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
# A tibble: 336,776 × 3
    gain     hours gain_per_hour
   <dbl>     <dbl>         <dbl>
1      9 3.7833333      2.378855
2     16 3.7833333      4.229075
3     31 2.6666667     11.625000
4    -17 3.0500000     -5.573770
5    -19 1.9333333     -9.827586
6     16 2.5000000      6.400000
7     24 2.6333333      9.113924
8    -11 0.8833333    -12.452830
```

MANY USEFUL CREATION FUNCTIONS

There are a wide variety of functions you can use with `mutate()`

Must be vectorized functions - meaning the function must take a vector of values as input and return the same number of values as output.

Functions	**Description**
`+, -, *, /, ^`	arithmetic
`x / sum(x)`	arithmetic w/aggregate functions
`%/%, %%`	modular arithmetic
`log, exp, sqrt`	transformations
`lag, lead`	offsets
`cumsum, cumprod, cum…`	cum/rolling aggregates
`>, >=, <, <=, !=, ==`	logical comparisons
`min_rank, dense_rank,`	ranking
`between`	are values between a and b?
`ntile`	bin values into buckets

CREATE NEW VARIABLES

```
transmute(flights,
  normalized_delay = dep_delay / mean(dep_delay, na.rm = TRUE))
# A tibble: 336,776 × 1
   normalized_delay
              <dbl>
1        0.15823949
2        0.31647898
3        0.15823949
4       -0.07911974
5       -0.47471846
6       -0.31647898
7       -0.39559872
8       -0.23735923
9       -0.23735923
10      -0.15823949
# ... with 336,766 more rows
```

Functions	Description
+, -, *, /, ^	arithmetic
x / sum(x)	arithmetic w/aggregate functions
%/%, %%	modular arithmetic
log, exp, sqrt	transformations
lag, lead	offsets
cumsum, cumprod, cum…	cum/rolling aggregates
>, >=, <, <=, !=, ==	logical comparisons
min_rank, dense_rank,	ranking
between	are values between a and b?
ntile	bin values into buckets

CREATE NEW VARIABLES

```
transmute(flights,
  log_air_time = log2(air_time),
  exp_delay = exp(dep_delay))
# A tibble: 336,776 x 2
   log_air_time    exp_delay
          <dbl>        <dbl>
1      7.826548  7.389056099
2      7.826548 54.598150033
3      7.321928  7.389056099
4      7.515700  0.367879441
5      6.857981  0.002478752
6      7.228819  0.018315639
7      7.303781  0.006737947
8      5.727920  0.049787068
9      7.129283  0.049787068
10     7.108524  0.135335283
# ... with 336,766 more rows
```

Functions	Description
+, -, *, /, ^	arithmetic
x / sum(x)	arithmetic w/aggregate functions
%/%, %%	modular arithmetic
log, exp, sqrt	transformations
lag, lead	offsets
cumsum, cumprod, cum…	cum/rolling aggregates
>, >=, <, <=, !=, ==	logical comparisons
min_rank, dense_rank,	ranking
between	are values between a and b?
ntile	bin values into buckets

CREATE NEW VARIABLES

```
transmute(flights,
  dep_delay = dep_delay,
  lag_delay = lag(dep_delay),
  sum_delay = cumsum(dep_delay))
# A tibble: 336,776 × 3
   dep_delay lag_delay sum_delay
       <dbl>     <dbl>     <dbl>
1          2        NA         2
2          4         2         6
3          2         4         8
4         -1         2         7
5         -6        -1         1
6         -4        -6        -3
7         -5        -4        -8
8         -3        -5       -11
9         -3        -3       -14
10        -2        -3       -16
```

Functions	Description
+, -, *, /, ^	arithmetic
x / sum(x)	arithmetic w/aggregate functions
%/%, %%	modular arithmetic
log, exp, sqrt	transformations
lag, lead	offsets
cumsum, cumprod, cum…	cum/rolling aggregates
>, >=, <, <=, !=, ==	logical comparisons
min_rank, dense_rank,	ranking
between	are values between a and b?
ntile	bin values into buckets

YOUR TURN!

1. With `sub_cust`, create a *ratio* variable that computes the ratio of *CardSpendMonth* to *HHIncome*

2. Create two variables:
   i. *ratio1* = *CardSpendMonth / HHIncome*
   ii. *ratio2* = *CardSpendMonth / Age*

SOLUTION

```
#1: create a ratio variable that computes the ratio of CardSpendMonth to HHIncome
mutate(sub_cust, ratio = CardSpendMonth / HHIncome)

#2: create 2 variables:
#        i) ratio1 = CardSpendMonth / HHIncome
#       ii) ratio2 = CardSpendMonth / Age
mutate(sub_cust,
  ratio1 = CardSpendMonth / HHIncome,
  ratio2 = CardSpendMonth / Age
  )
```

summarise

Collapse many values down to a single summary statistic

SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

```
summarise(flights, dep_delay_mean = mean(dep_delay, na.rm = TRUE))
# A tibble: 1 × 1
  dep_delay_mean
           <dbl>
1       12.63907
```

Important, try this without `na.rm = TRUE` and see what happens. Why does this happen?

SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

```
summarise(flights,
  dep_delay_mean = mean(dep_delay, na.rm = TRUE),
  dep_delay_sd = sd(dep_delay, na.rm = TRUE)
)
# A tibble: 1 × 2
  dep_delay_mean dep_delay_sd
           <dbl>        <dbl>
1       12.63907     40.21006
```

SUMMARIZING OUR DATA

We can create summary statistics of one or more variables:

```
summarise(flights,
  dep_delay_mean = mean(dep_delay, na.rm = TRUE),
  dep_delay_sd = sd(dep_delay, na.rm = TRUE),
  n = n()
)
# A tibble: 1 × 3
  dep_delay_mean dep_delay_sd      n
           <dbl>        <dbl>  <int>
1       12.63907     40.21006 336776
```

SUMMARY FUNCTIONS

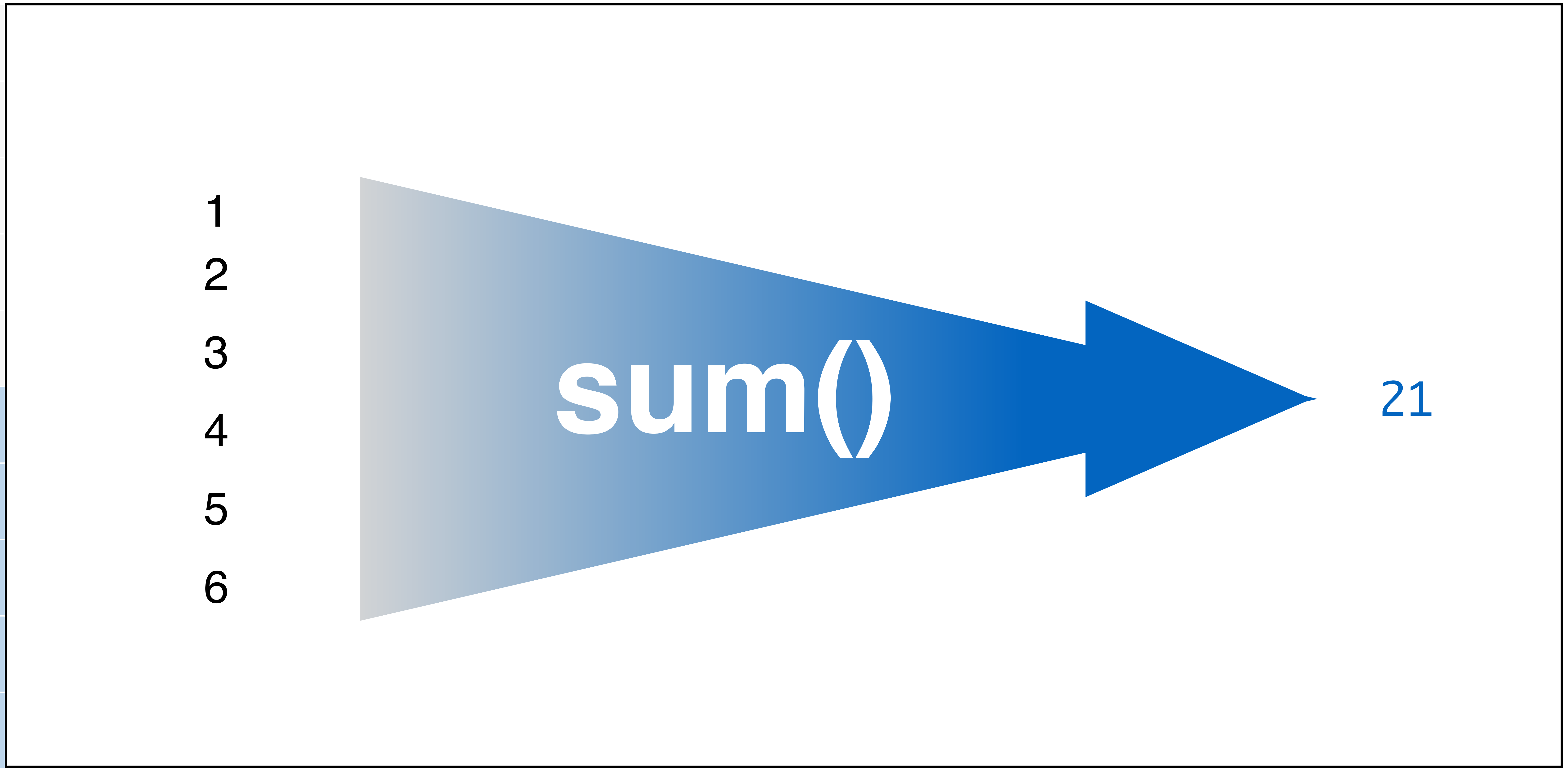
* All take a vector of values and return a single value
** Blue functions come in `dplyr`

`min(), max()`	Minimum and maximum values
`mean()`	Mean value
`median()`	Median value
`sum()`	Sum of values
`var, sd()`	Variance and standard deviation of a vector
`first()`	First value in a vector
`last()`	Last value in a vector
`nth()`	Nth value in a vector
`n()`	The number of values in a vector
`n_distinct()`	The number of distinct values in a vector

SUMMARY FUNCTIONS

* All take a vector of values and return a single value

min(), max()
mean()
median()
sum()
var, sd()
first()
last()
nth()
n()
n_distinct()



SUMMARIZING GROUPED DATA

Summary statistics become more powerful when we can compare groups:

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
Source: local data frame [365 x 4]
Groups: year, month [?]

   year month   day     delay
  <int> <int> <int>     <dbl>
1  2013     1     1 11.548926
2  2013     1     2 13.858824
3  2013     1     3 10.987832
4  2013     1     4  8.951595
5  2013     1     5  5.732218
6  2013     1     6  7.148014
```

SUMMARIZING GROUPED DATA

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

```
group_by(data, country)
```

SUMMARIZING GROUPED DATA

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

```
group_by(data, country, year)
```

SUMMARIZING GROUPED DATA

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

country	year	sex	case
Afghanistan	1999	female	1
Afghanistan	1999	male	1
Afghanistan	2000	female	1
Afghanistan	2000	male	1
Brazil	1999	female	2
Brazil	1999	male	2
Brazil	2000	female	2
Brazil	2000	male	2
China	1999	female	3
China	1999	male	3
China	2000	female	3
China	2000	male	3

```
ungroup(data)
```

YOUR TURN!

1. In our `sub_cust` data, compute the average CardSpendMonth across all customers.

2. Now compute the average CardSpendMonth for each gender.

3. Now compute the average CardSpendMonth for each gender and region. Which gender and region have the highest average spend?

SOLUTION

```
#1: Avg spend across all customers
summarize(sub_cust, Avg_spend = mean(CardSpendMonth, na.rm = TRUE)

#2: Now compute the average CardSpendMonth for each gender.
by_gender <- group_by(sub_cust, Gender)
summarize(by_gender, Avg_spend = mean(CardSpendMonth, na.rm = TRUE))

#3: Now compute the average CardSpendMonth for each gender and region.
#   Which gender and region have the highest average spend?
by_gdr_rgn <- group_by(sub_cust, Gender, Region)
avg_gdr_rgn <- summarize(by_gdr_rgn, Avg_spend = mean(CardSpendMonth, na.rm = TRUE))
arrange(avg_gdr_rgn, desc(Avg_spend))

   Gender Region Avg_spend
    <chr>  <int>     <dbl>
1    Male      3  3692.818
2    Male      5  3617.054
```

%>%

pipe operator

Chaining functions together with the pipe operator

STREAMLINING OUR ANALYSIS

Going back to our last problem, our code was doing three things:

1. grouping by gender and region
2. summarizing average spend
3. sorting spend by greatest to least

```
by_gdr_rgn <- group_by(sub_cust, Gender, Region)

avg_gdr_rgn <- summarize(by_gdr_rgn, Avg_spend = mean(CardSpendMonth, na.rm = TRUE))

arrange(avg_gdr_rgn, desc(Avg_spend))
```

STREAMLINING OUR ANALYSIS

We can streamline our code to make it more ***efficient*** and ***legible***

```
library(magrittr)
x <- 1:15
sum(x)
x %>% sum()
```

These do the same thing

Try it!

%>%

x

sum(___)

STREAMLINING OUR ANALYSIS

- Lets re-write our code using the pipe **(%>%)** operator
- This code does four things in a very *efficient* & *readable* manner

```
sub_cust %>%
  group_by(Gender, Region) %>%
  summarize(Avg_spend = mean(CardSpendMonth, na.rm = TRUE)) %>%
  arrange(desc(Avg_spend))

  Gender Region Avg_spend
     <chr>  <int>      <dbl>
  1     Male       3   3692.818
  2     Male       5   3617.054
  3     Male       4   3535.671
```

YOUR TURN!

Using the pipe operator follow these steps with the **`sub_cust`** data:

1. filter for *male* customers only
2. create a new variable: *ratio* = *CardSpendMonth* / *HHIncome*
3. group this data by *age*
4. compute the mean of the new *ratio* variable by *age*
5. sort this output to find the *age* with the highest *ratio* of expenditures to income.

SOLUTION

```
sub_cust %>%
  filter(Gender == “Male") %>%
  mutate(ratio = CardSpendMonth / HHIncome) %>%
  group_by(Age) %>%
  summarize(Avg_ratio = mean(ratio, na.rm = TRUE)) %>%
  arrange(desc(Avg_ratio))

      Age Avg_ratio
    <int>     <dbl>
1      20 0.1470240
2      18 0.1452089
3      79 0.1440063
4      19 0.1425964
5      24 0.1363957
6      75 0.1296193
```

WHAT TO REMEMBER

FUNCTIONS TO REMEMBER

Operator/Function	Description
`filter`	pick observations based on their values
↳ `>, >=, <, <=, !=, ==`	comparison operators
↳ `is.na`	identify missing values
`arrange`	re-order rows
↳ `desc`	order in descending order
`select`	select variables
↳ `starts_with, ends_with, contains, etc.`	select variables based on patterns
`rename`	rename select variables
`mutate, transmute`	create new variables
`summarise`	summarize data
`group_by, ungroup`	group/ungroup based on categorical variables
`%>%`	pipe operator to chain together functions