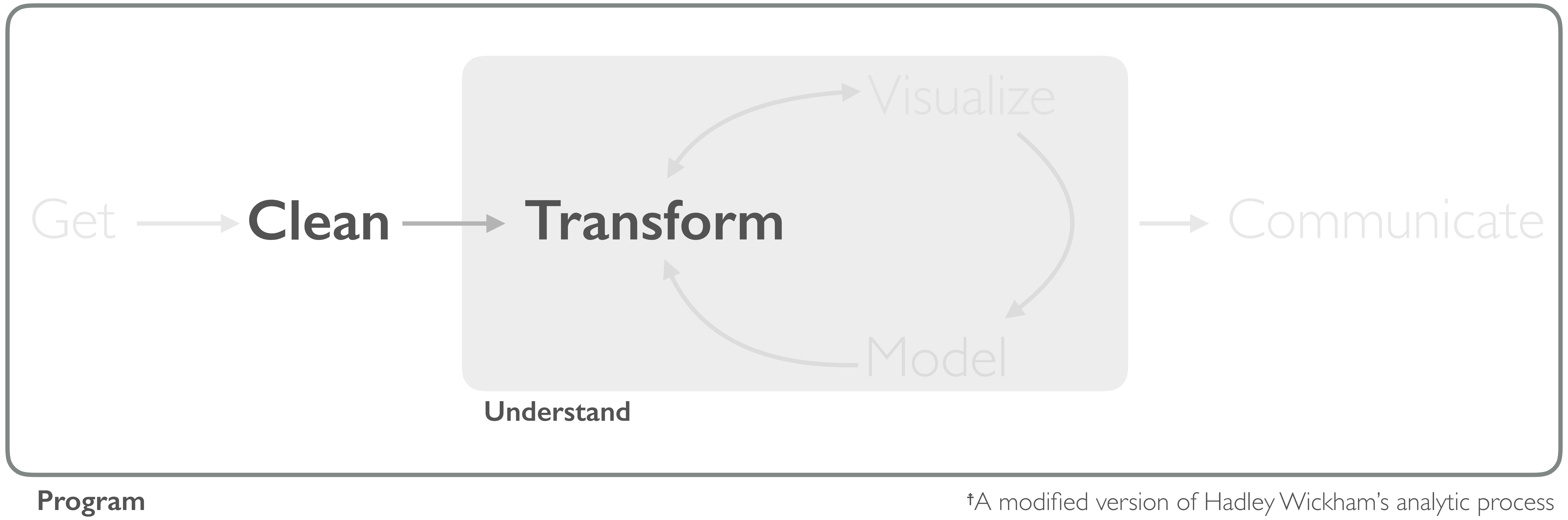
DATA STRUCTURES



†A modified version of Hadley Wickham's analytic process

BASICS

vector

```
0.70 0.86 0.95 0.25 0.52 0.37 0.27 0.80 0.60 0.26
```

matrix

```
      [,1]  [,2]  [,3]  [,4]
[1,]  0.70  0.37  0.70  0.37
[2,]  0.86  0.27  0.86  0.27
[3,]  0.95  0.80  0.95  0.80
[4,]  0.25  0.60  0.25  0.60
[5,]  0.52  0.26  0.52  0.26
```

data frame

```
    Sepal.Length  Sepal.Width  Petal.Width  Species
1            5.1          3.5          0.2   setosa
2            4.9          3.0          0.2   setosa
3            4.7          3.2          0.2   setosa
4            4.6          3.1          0.2   setosa
5            5.0          3.6          0.2   setosa
6            5.4          3.9          0.4   setosa
7            4.6          3.4          0.3   setosa
8            5.0          3.4          0.2   setosa
9            4.4          2.9          0.2   setosa
10           4.9          3.1          0.1   setosa
```

list

```
$item1
[1] 1 2 3

$item2
[1] "a" "b" "c" "d" "e"

$item3
[1]  TRUE FALSE  TRUE  TRUE

$item4
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

PREREQUISITES

Re-start your R session

- **Windows:** Ctrl+Shift+F10
- **Mac:** Command+Shift+F10

Reload `nycflights13` library

```
library(nycflights13)
```

DATA FRAMES

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4

PROPERTIES

- Spreadsheet style data
- 2 dimensions
  - rows
  - columns
- Can contain heterogenous data
- All columns must be of equal length

The `flights` data we've been working with is a data frame

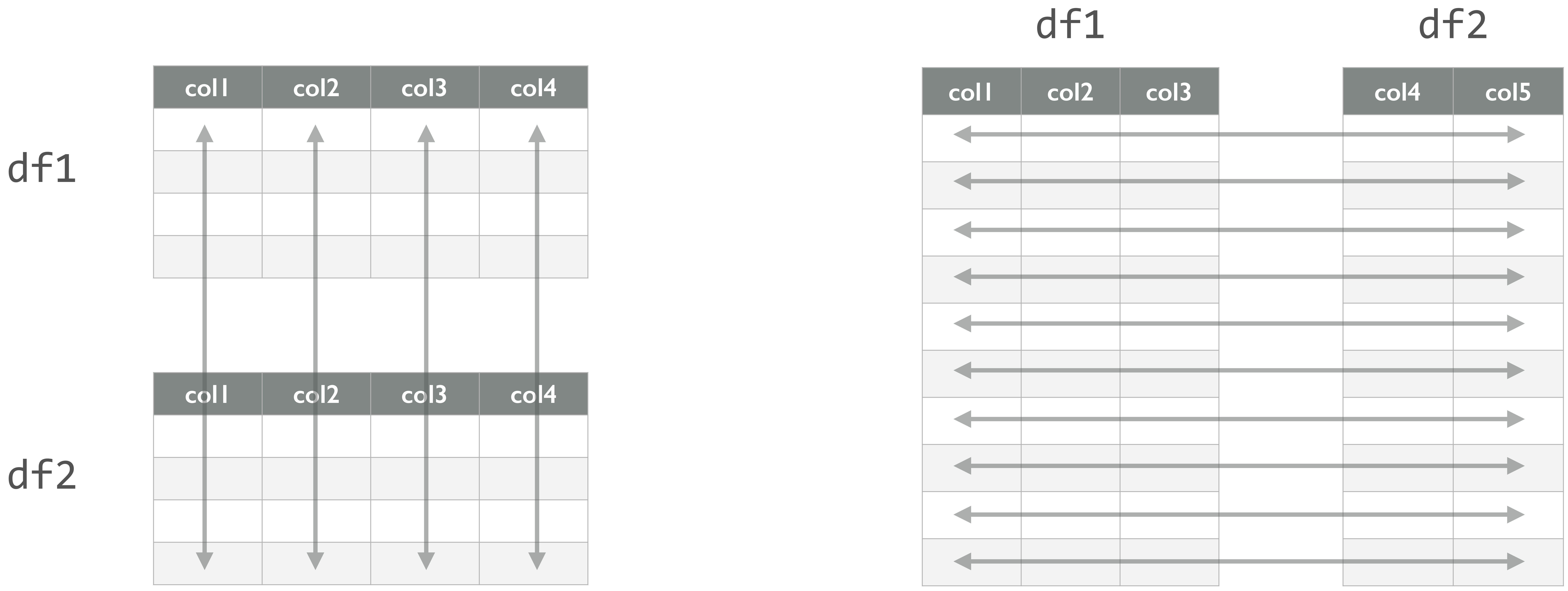
	year	month	day	dep_time	carrier	tailnum	dest	time_hour
1	2013	1	1	517	UA	N14228	IAH	2013-01-01 05:00:00
2	2013	1	1	533	UA	N24211	IAH	2013-01-01 05:00:00
3	2013	1	1	542	AA	N619AA	MIA	2013-01-01 05:00:00
4	2013	1	1	544	B6	N804JB	BQN	2013-01-01 05:00:00
5	2013	1	1	554	DL	N668DN	ATL	2013-01-01 06:00:00
6	2013	1	1	554	UA	N39463	ORD	2013-01-01 05:00:00
7	2013	1	1	555	B6	N516JB	FLL	2013-01-01 06:00:00
8	2013	1	1	557	EV	N829AS	IAD	2013-01-01 06:00:00
9	2013	1	1	557	B6	N593JB	MCO	2013-01-01 06:00:00
10	2013	1	1	558	AA	N3ALAA	ORD	2013-01-01 06:00:00
11	2013	1	1	558	B6	N793JB	PBI	2013-01-01 06:00:00
12	2013	1	1	558	B6	N657JB	TPA	2013-01-01 06:00:00
13	2013	1	1	558	UA	N29129	LAX	2013-01-01 06:00:00
14	2013	1	1	558	UA	N53441	SFO	2013-01-01 06:00:00
15	2013	1	1	559	AA	N3DUAA	DFW	2013-01-01 06:00:00
16	2013	1	1	559	B6	N708JB	BOS	2013-01-01 05:00:00
17	2013	1	1	559	UA	N76515	LAS	2013-01-01 06:00:00
18	2013	1	1	600	B6	N595JB	FLL	2013-01-01 06:00:00
19	2013	1	1	600	MQ	N542MQ	ATL	2013-01-01 06:00:00
20	2013	1	1	601	B6	N644JB	PBI	2013-01-01 06:00:00
21	2013	1	1	602	DL	N971DL	MSP	2013-01-01 06:00:00
22	2013	1	1	602	MQ	N730MQ	DTW	2013-01-01 06:00:00
23	2013	1	1	606	AA	N633AA	MIA	2013-01-01 06:00:00

CREATING

```
df <- data.frame(col1 = 1:3,
                 col2 = c("this", "is", "text"),
                 col3 = c(TRUE, FALSE, TRUE),
                 col4 = c(2.5, 4.2, pi))

df
  col1 col2  col3     col4
1    1 this  TRUE 2.500000
2    2   is FALSE 4.200000
3    3 text  TRUE 3.141593
```

Create this data frame

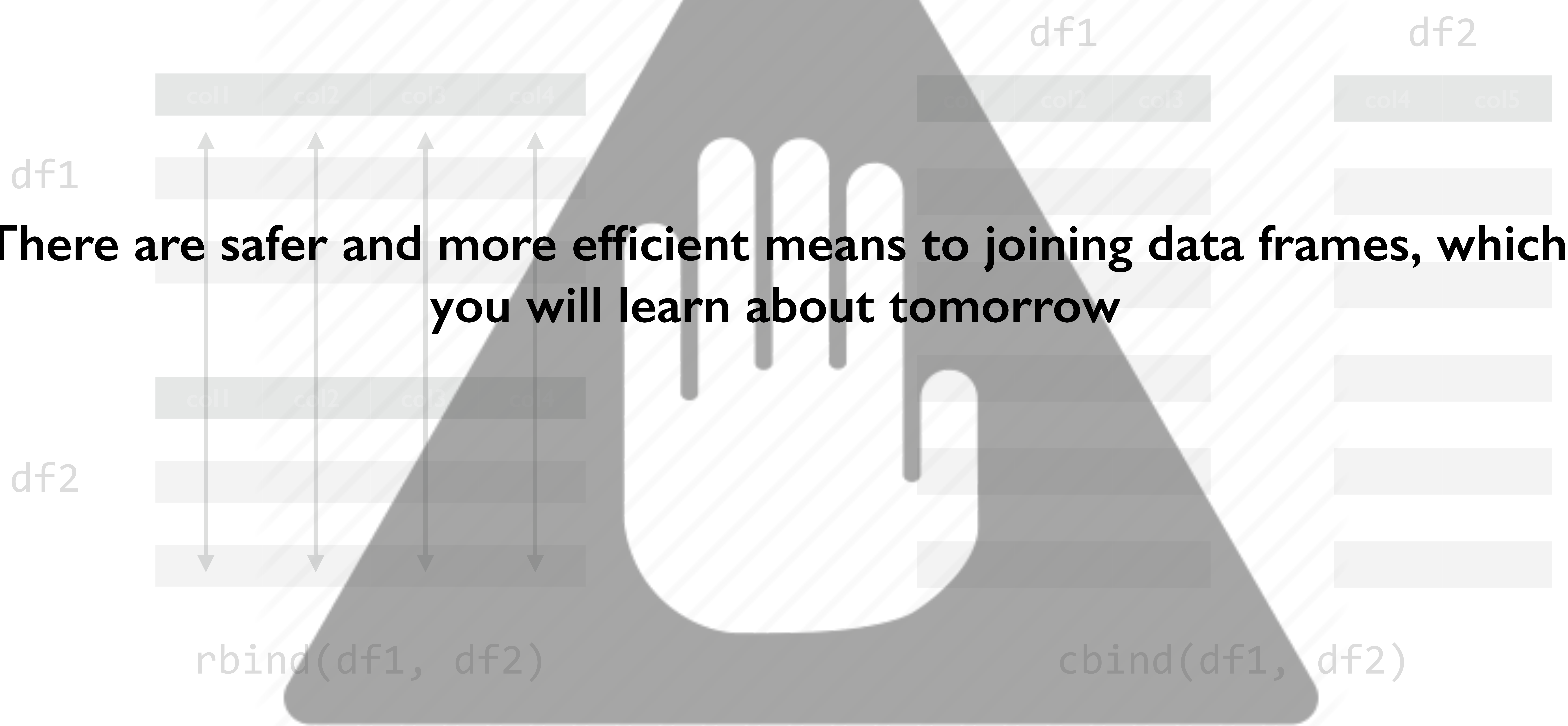
ADDING ON TO



rbind(df1, df2)

cbind(df1, df2)

ADDING ON TO



There are safer and more efficient means to joining data frames, which you will learn about tomorrow

ATTRIBUTES

Data frames have two main attributes that you care about:

```
# you can also get the number of rows or columns individually with nrow() ncol()
dim(df)
[1] 3 4

names(df)
[1] "col1" "col2" "col3" “col4"

names(df) <- c("Col 1", "Col 2", "Col 3", "Col 4")
df
  Col 1 Col 2 Col 3    Col 4
1     1  this  TRUE 2.500000
2     2    is FALSE 4.200000
3     3  text  TRUE 3.141593
```

QUICK SUMMARIES

Get a quick summary of your data frame with `summary()` and `str()`:

For larger data frames you can also use `head(df, n)` and `tail()` to see the first or last *n* rows.

```
summary(df)
     Col 1         Col 2       Col 3               Col 4      
 Min.   :1.0   is  :1   Mode :logical   Min.   :2.500  
 1st Qu.:1.5   text:1   FALSE:1         1st Qu.:2.821  
 Median :2.0   this:1   TRUE :2         Median :3.142  
 Mean   :2.0            NA's :0         Mean   :3.281  
 3rd Qu.:2.5                            3rd Qu.:3.671  
 Max.   :3.0                            Max.   :4.200  

str(df)
'data.frame': 3 obs. of  4 variables:
 $ Col 1: int  1 2 3
 $ Col 2: Factor w/ 3 levels "is","text","this": 3 1 2
 $ Col 3: logi  TRUE FALSE TRUE
```

INDEXING/SUBSETTING

Most of our indexing and subsetting of data frames will be done with `dplyr` functions (`filter` and `select`)

But as you'll see, understanding the `[ ]` functionality is important.

INDEXING/SUBSETTING

```
data.frame[row, col]
```

Try these different forms of indexing & subsetting:

```
# extract the second column and all rows using column indexing or the name
df[, 2]
df[, “Col 2"]

# extract all rows and columns 1 through 3
df[, 1:3]
df[, c(“Col 1”, Col 2", “Col 3")]

# index for first row and all columns
df[1, ]

# subset for rows
subset(df, `Col 3` == TRUE)
subset(df, `Col 3` == TRUE & `Col 4` > 3, c(2, 4))
```

YOUR TURN!

1. Using `[ ]`, select the first 1000 rows and the following columns: **`month, dep_delay, carrier, distance, time_hour`**. Save this as **`small_flights`**.

2. Look at the structure and summary of **`small_flights`**

3. Rename the columns of small_flights to **`c("Month", "Delay", "Carrier", "Distance", “Date-Time")`**

4. Look at the first and last 15 rows

SOLUTION

```
# 1
small_flights <- flights[1:1000, c("month", "dep_delay", "carrier", "distance", “time_hour")]

# 2
str(small_flights)
summary(small_flights)

# 3
names(small_flights) <- c("Month", "Delay", "Carrier", "Distance", “Date-Time”)

# 4
head(small_flights, 15)
tail(small_flights, 15)
```

TIBBLES

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay
   <int> <int> <int>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2
2   2013     1     1      533            529         4
3   2013     1     1      542            540         2
4   2013     1     1      544            545        -1
5   2013     1     1      554            600        -6
6   2013     1     1      554            558        -4
7   2013     1     1      555            600        -5
8   2013     1     1      557            600        -3
9   2013     1     1      557            600        -3
10  2013     1     1      558            600        -2
# ... with 336,766 more rows, and 13 more variables:
#   arr_time <int>, 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>
```

PROPERTIES

Same as data frames but with minor tweaks to make life easier

Compare the outputs by running this code:

```
flights

tibble::as_tibble(flights)
```

What differences do you notice?

PROPERTIES

A tibble is a data frame with a better printing structure

```
tibble::as_tibble(flights)
# A tibble: 336,776 × 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
10  2013     1     1      558            600        -2      753            745         8
# ... with 336,766 more rows, and 10 more variables: carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

CREATE

- tibbles are provided by the **`tibble`** package which is also provided by the **`tidyverse`** package
- to convert data frames to tibbles just apply `as_tibble()`

```
library(tidyverse)
as_tibble(iris)
# A tibble: 150 × 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
# ... with 140 more rows
```

PROPERTIES

All indexing, subsetting, and attribute functions apply to tibbles just as they do to data frames.

MATRICES

```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,] 0.34 0.96 0.36 0.95 0.50 0.98
 [2,] 0.47 0.25 0.68 0.65 0.37 0.53
 [3,] 0.35 0.93 0.60 0.65 0.14 0.71
 [4,] 0.89 0.68 0.07 0.10 0.46 0.20
 [5,] 0.28 0.25 0.70 0.36 0.59 0.26
 [6,] 0.96 0.42 0.93 0.62 0.24 0.82
 [7,] 0.72 0.13 0.47 0.93 0.05 0.23
 [8,] 0.82 0.32 0.70 0.84 0.66 0.70
 [9,] 0.68 0.04 0.06 0.82 0.78 0.84
[10,] 0.13 0.14 0.46 0.91 0.29 0.82
[11,] 0.45 0.29 0.04 0.12 0.92 0.57
[12,] 0.90 0.81 0.74 0.83 0.91 0.29
[13,] 0.89 0.40 0.71 0.12 0.73 0.08
[14,] 0.05 0.52 0.47 0.53 0.53 0.96
[15,] 0.16 0.59 0.43 0.19 0.37 0.54
```

PROPERTIES

- 2 dimensions
  - rows
  - columns
- Can only contain *homogenous* data
- All columns must be of equal length

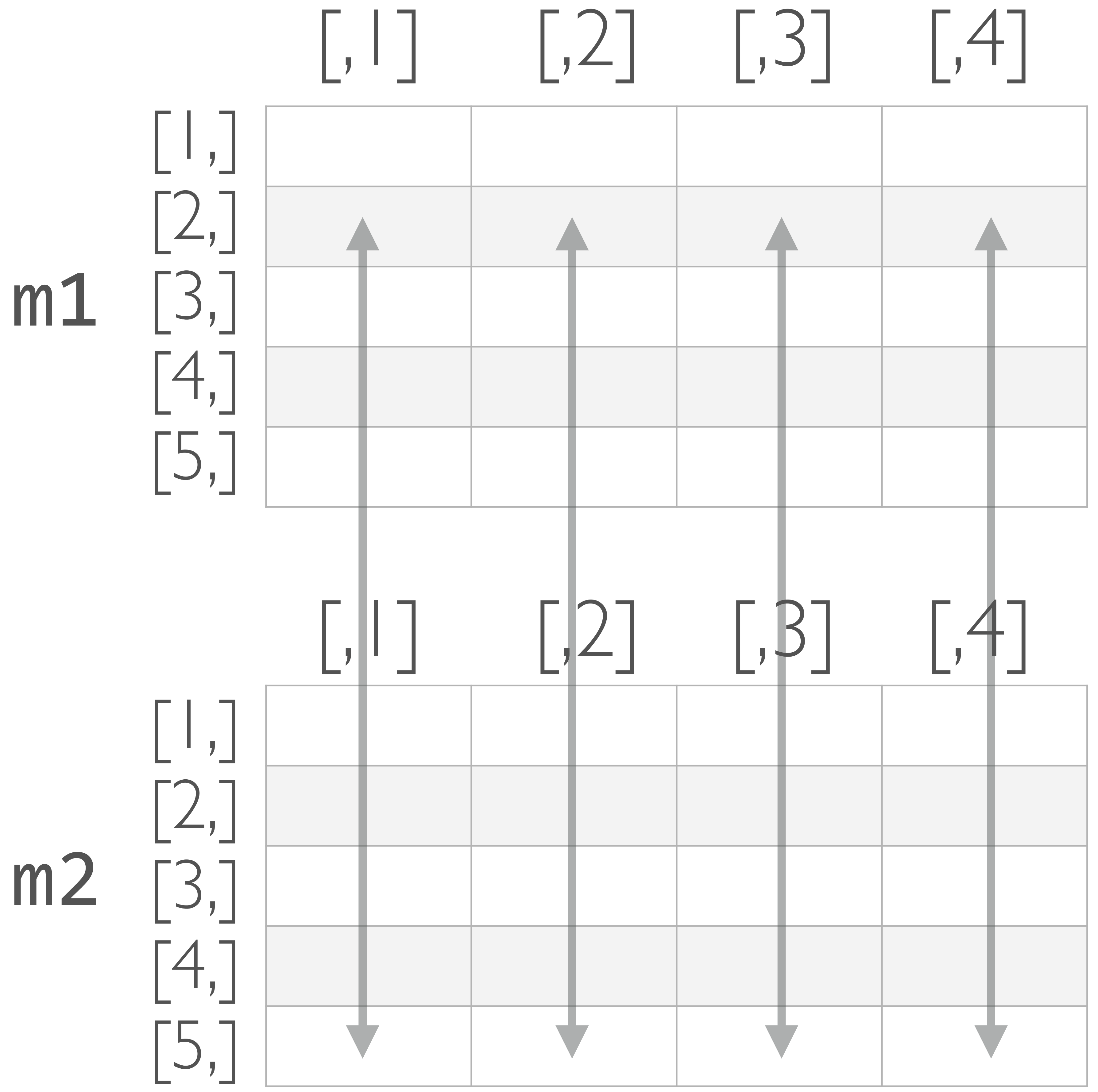
```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,] 0.34 0.96 0.36 0.95 0.50 0.98
 [2,] 0.47 0.25 0.68 0.65 0.37 0.53
 [3,] 0.35 0.93 0.60 0.65 0.14 0.71
 [4,] 0.89 0.68 0.07 0.10 0.46 0.20
 [5,] 0.28 0.25 0.70 0.36 0.59 0.26
 [6,] 0.96 0.42 0.93 0.62 0.24 0.82
 [7,] 0.72 0.13 0.47 0.93 0.05 0.23
 [8,] 0.82 0.32 0.70 0.84 0.66 0.70
 [9,] 0.68 0.04 0.06 0.82 0.78 0.84
[10,] 0.13 0.14 0.46 0.91 0.29 0.82
[11,] 0.45 0.29 0.04 0.12 0.92 0.57
[12,] 0.90 0.81 0.74 0.83 0.91 0.29
[13,] 0.89 0.40 0.71 0.12 0.73 0.08
[14,] 0.05 0.52 0.47 0.53 0.53 0.96
[15,] 0.16 0.59 0.43 0.19 0.37 0.54
```

CREATING

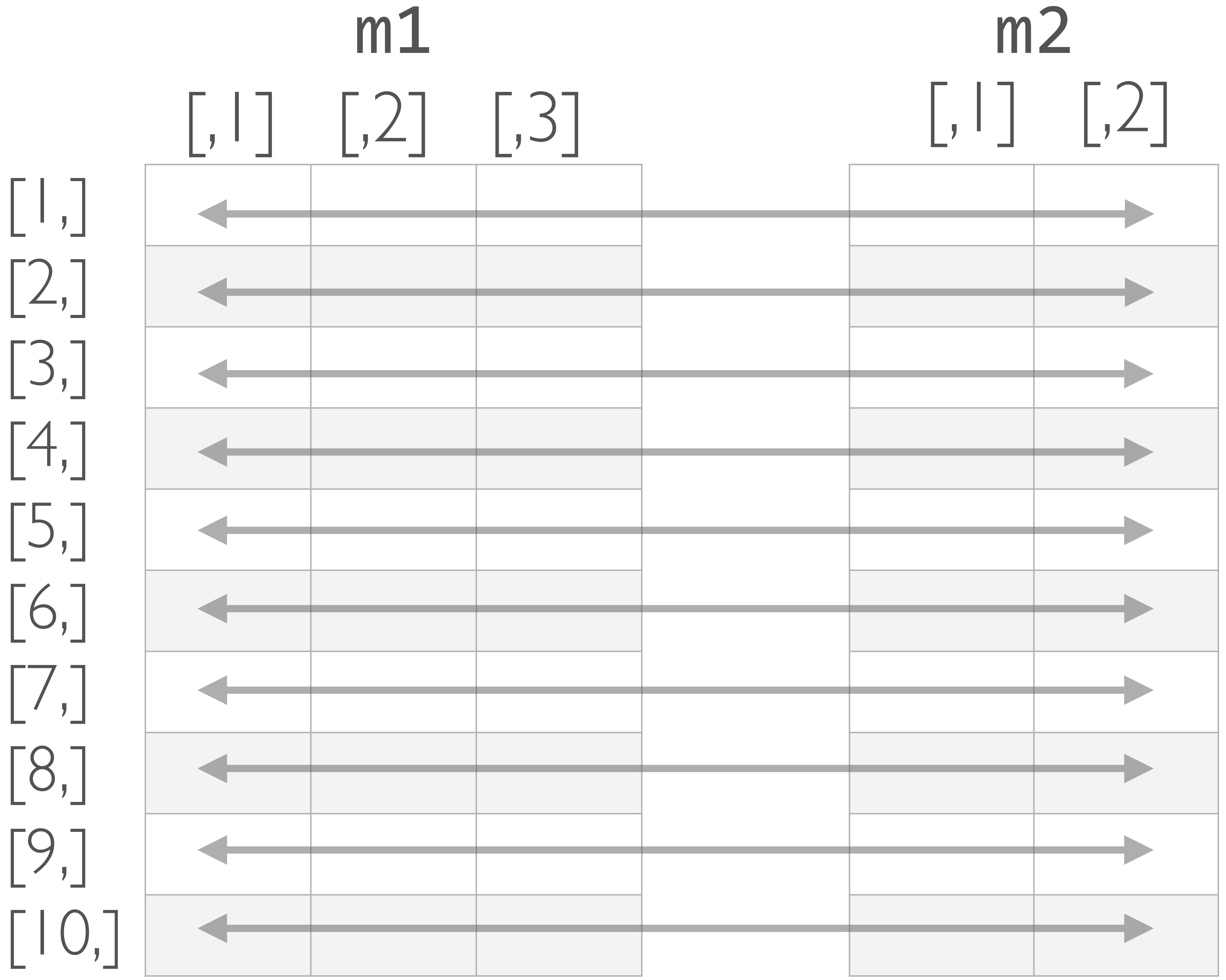
```
set.seed(123)
v1 <- sample(1:10, 25, replace = TRUE)
m1 <- matrix(v1, nrow = 5)

m1
     [,1] [,2] [,3] [,4] [,5]
[1,]    3    1   10    9    9
[2,]    8    6    5    3    7
[3,]    5    9    7    1    7
[4,]    9    6    6    4   10
[5,]   10    5    2   10    7
```

Create this matrix

ADDING ONTO



`rbind(m1, m2)`



`cbind(m1, m2)`

ATTRIBUTES

Matrices have similar attributes as data frames

```
# test these out on your matrix
dim(m1)
length(m1)
str(m1)
colnames(m1) <- paste("col", 1:5)
rownames(m1) <- paste("row", 1:5)
```

INDEXING/SUBSETTING

`matrix[row, col]`

Try these different forms of indexing & subsetting:

```
# extract individual elements
m1[1, 3]
m1["row 4", "col 3"]

# extract all rows and columns 1 through 3
m1[, 1:3]
m1[, c(“col 1”, col 2", “col 3")]

# index for all rows and just the second column
m1[, 2]
m1[, 2, drop = FALSE]
```

QUICK SUMMARIES

Get a quick summary of your matrix with `summary()` or any other math/logical operation:

```
summary(m1)
mean(m1)
mean(m[1,])
rowMeans(m1)
colMeans(m1)
rowSums(m1)
colSums(m1)
m > .5
sum(m > .5)
which(m > .5)
m[m > .5]
```

These same functions can be applied to data frames / tibbles

YOUR TURN!

Using the built-in **`VADeaths`** matrix data:

1. Calculate averages for each column and row
2. Can you figure out how to add these averages to your table so the output looks like:

```
             Rural Male Rural Female Urban Male Urban Female Avg_by_Age
50-54             11.70         8.70      15.40         8.40     11.050
55-59             18.10        11.70      24.30        13.60     16.925
60-64             26.90        20.30      37.00        19.30     25.875
65-69             41.00        30.90      54.60        35.10     40.400
70-74             66.00        54.30      71.10        50.00     60.350
Avg_by_Local      32.74        25.18      40.48        25.28     30.920
```

SOLUTION

```
# Calculate average for each age group and add as a new column
Avg_by_Age <- rowMeans(VADeaths)
VADeaths <- cbind(VADeaths, Avg_by_Age)

# Calculate average for each column and add as a new row
Avg_by_Local <- colMeans(VADeaths)
VADeaths <- rbind(VADeaths, Avg_by_Local)


VADeaths
             Rural Male Rural Female Urban Male Urban Female Avg_by_Age
50-54             11.70         8.70      15.40         8.40     11.050
55-59             18.10        11.70      24.30        13.60     16.925
60-64             26.90        20.30      37.00        19.30     25.875
65-69             41.00        30.90      54.60        35.10     40.400
70-74             66.00        54.30      71.10        50.00     60.350
Avg_by_Local      32.74        25.18      40.48        25.28     30.920
```

VECTORS

```
[1] 0.67149785 0.47398715 0.32813279 0.87295142 0.56274062 0.16796701 0.05765868 0.59618446
[9] 0.94417744 0.83129550 0.38959025 0.99178460
```

PROPERTIES

- 1 dimension
- Can only contain *homogenous* data

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
[14] "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
[18] 18
```

```
 [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
 [9]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
```

CREATING

- Most common way to create a vector is with `c()` or `:`
- For numeric vectors there are numerous ways to generate sequences of numbers

```
# vectors with no set sequence
c("Learning", "to", "create", "character", “vectors")
c(3, 2, 10, 55)
c(TRUE, FALSE, FALSE, FALSE, TRUE)

# numeric vectors with regular sequence
6:15
15.5:-6.75
seq(from = 5, to = 95, by = 15)
seq(from = 5, to = 95, length = 4)

# regular sequence for any type of vector
rep(c(TRUE, TRUE, FALSE), times = 3)
rep(c(TRUE, TRUE, FALSE), each = 3)
```

CREATING

- Most common way to create a vector is with `c()` or `:`
- For numeric vectors there are numerous ways to generate sequences of numbers

```
# vectors with no set sequence
c("Learning", "to", "create", "character", “vectors")
c(3, 2, 10, 55)
c(TRUE, FALSE, FALSE, FALSE, TRUE)

# numeric vectors with regular sequence
6:15
15.5:-6.75
seq(from = 5, to = 95, by = 15)
seq(from = 5, to = 95, length = 4)

# regular sequence for any type of vector
rep(c(TRUE, TRUE, FALSE), times = 3)
rep(c(TRUE, TRUE, FALSE), each = 3)
```

There are also many distribution functions to generate data:

- uniform: `<r,d,p,q>unif`
- normal: `<r,d,p,q>norm`
- binomial: `<r,d,p,q>binom`
- poisson: `<r,d,p,q>pois`
- exponential: `<r,d,p,q>exp`

ADDING ONTO

- Most common way to create a vector is with `c()`
- Combining two different kinds of vectors will coerce the vector to the "simplest" form

```
v1 <- 1:10
v2 <- c(12, 15)
v3 <- c(20, 25:30)
c(v1, v2, v3)
[1]  1  2  3  4  5  6  7  8  9 10 12 15 20 25 26 27 28 29 30

v4 <- c("Counting from")
c(v4, v1)
 [1] "Counting from" "1"             "2"             "3"             "4"
 [6] "5"             "6"             "7"             "8"             "9"
[11] “10"

paste(v4, v1)
 [1] "Counting from 1"  "Counting from 2"  "Counting from 3"  "Counting from 4"
 [5] "Counting from 5"  "Counting from 6"  "Counting from 7"  "Counting from 8"
 [9] "Counting from 9"  "Counting from 10"
```

ATTRIBUTES

Vectors have limited attributes

```
# create this vector
v1 <- 1:10

# try these out on your vector
length(v1)
str(v1)

names(v1)
names(v1) <- paste("Var", LETTERS[1:10])
names(v1)
v1
```

INDEXING / SUBSETTING

`vector[element]`

Try these different forms of indexing & subsetting:

```
v1[4]
v1[4:7]
v1[c(4, 3, 4)]
v1[c("Var A", "Var D", "Var J")]
v1[v1 > 6]
v1[v1 > 8 | v1 <=3]
```

QUICK SUMMARIES

Get a quick summary of your vector with `summary()` or any other math/logical operation:

```
summary(v1)
mean(v1)
median(v1[c("Var A", "Var D", "Var J")])
v1 > 5
sum(v1 > 5)
```

YOUR TURN!

1. *check out the built-in character vector* ***`state.name`***

2. *how many elements are in this vector*

3. *Can you name each vector element with* **`“V1”, “V2”, …, “V50”?`**

4. Subset state.name for those elements with the following names: **`V35, V17, V14, V38`**

SOLUTION

```
# check out state.name
state.name

# how many elements are in state.name
length(state.name)

# name state.name with “V1”, “V2”,…, “V50”
names(state.name) <- paste0("V", 1:50)

# subset state.name for V35, V17, V14, V38
state.name[c("V35", "V17", "V14", "V38")]
```

LISTS

```
$item1
[1] 1 5 3 7

$item2
[1] "g" "b" "q" "v" "d" "z" "w" "i"

$item3
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

$item4
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

PROPERTIES

- 1 dimension
- Can only contain *heterogeneous* data - to include multiple and different objects (i.e. vectors, data frames, matrices, and even lists)

```
$item1
[1] 1 5 3 7

$item2
[1] "g" "b" "q" "v" "d" "z" "w" "i"

$item3
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

$item4
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Lists are very important objects in R!

They may be confusing but they are worth learning

CREATING

- To create a list we use `list()`

```
# list of 4 items
l1 <- list(item1 = 1:3,
           item2 = letters[1:5],
           item3 = c(T, F, T, T),
           item4 = matrix(1:9, nrow = 3))

l1
## $item1
## [1] 1 2 3
##
## $item2
## [1] "a" "b" "c" "d" "e"
##
## $item3
## [1]  TRUE FALSE  TRUE  TRUE
##
## $item4
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

ADDING ONTO

- We can add on to lists a couple different ways

```
# add a 5th (named) list item
l1$item5 <- flights

l1
$item1
[1] 1 2 3

$item2
[1] "a" "b" "c" "d" "e"

$item3
[1]  TRUE FALSE  TRUE  TRUE

$item4
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

$item5
# A tibble: 336,776 × 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
```

UNDERSTANDING YOUR LIST

Lists have a few main attributes that you care about:

```
str(l1)
length(l1)
names(l1)
```

INDEXING / SUBSETTING

- Its important that you know how to index/subset a list
- Elements of lists can be extracted using 3 approaches:

  preserve: `list[component]`

  simplify: `list[[component]]`

  simplify: `list$component`

```
# try these on our l1 list
l1[“item5”]
l1[[“item5”]]
l1$item5
l1[["item5"]][1:20, 1:5]
```

How do they differ?

WHAT YOU NEED TO KNOW

- Many statistical modeling results come in the form of lists
- You need to know how to extract parts of a list to access model results

WHAT YOU NEED TO KNOW

- Many statistical modeling results come in the form of lists
- You need to know how to extract parts of a list to access model results

```
# here's a linear regression model
model <- lm(mpg ~ wt, data = mtcars)

summary(model)
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
##
```

WHAT YOU NEED TO KNOW

- Model is simply a list of statistical results for our regression model

```
# here's a linear regression model
model <- lm(mpg ~ wt, data = mtcars)

names(model)
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"

str(model)
## List of 12
##  $ coefficients : Named num [1:2] 37.29 -5.34
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "wt"
##  $ residuals    : Named num [1:32] -2.28 -0.92 -2.09 1.3 -0.2 ...
##   ..- attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
##  $ effects      : Named num [1:32] -113.65 -29.116 -1.661 1.631 0.111 ...
##   ..- attr(*, "names")= chr [1:32] "(Intercept)" "wt" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:32] 23.3 21.9 24.9 20.1 18.9 ...
```

WHAT YOU NEED TO KNOW

- Model is simply a list of statistical results for our regression model
- So if you want to extract the residuals or fitted values you can just use normal list subsetting procedures

```
# extract the regression model residuals
model$residuals
##           Mazda RX4       Mazda RX4 Wag          Datsun 710
##          -2.2826106          -0.9197704          -2.0859521
##      Hornet 4 Drive   Hornet Sportabout             Valiant
##           1.2973499          -0.2001440          -0.6932545
##          Duster 360           Merc 240D            Merc 230
##          -3.9053627           4.1637381           2.3499593
##            Merc 280           Merc 280C          Merc 450SE
##           0.2998560          -1.1001440           0.8668731
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
##          -0.0502472          -1.8830236           1.1733496
## Lincoln Continental   Chrysler Imperial            Fiat 128
##           2.1032876           5.9810744           6.8727113
```

YOUR TURN!

1. *Create this this regression model:*

```
flight_lm <- lm(arr_delay ~ dep_delay + month + carrier,
                data = flights)
```

2. Extract the residuals from the `flight_lm` list

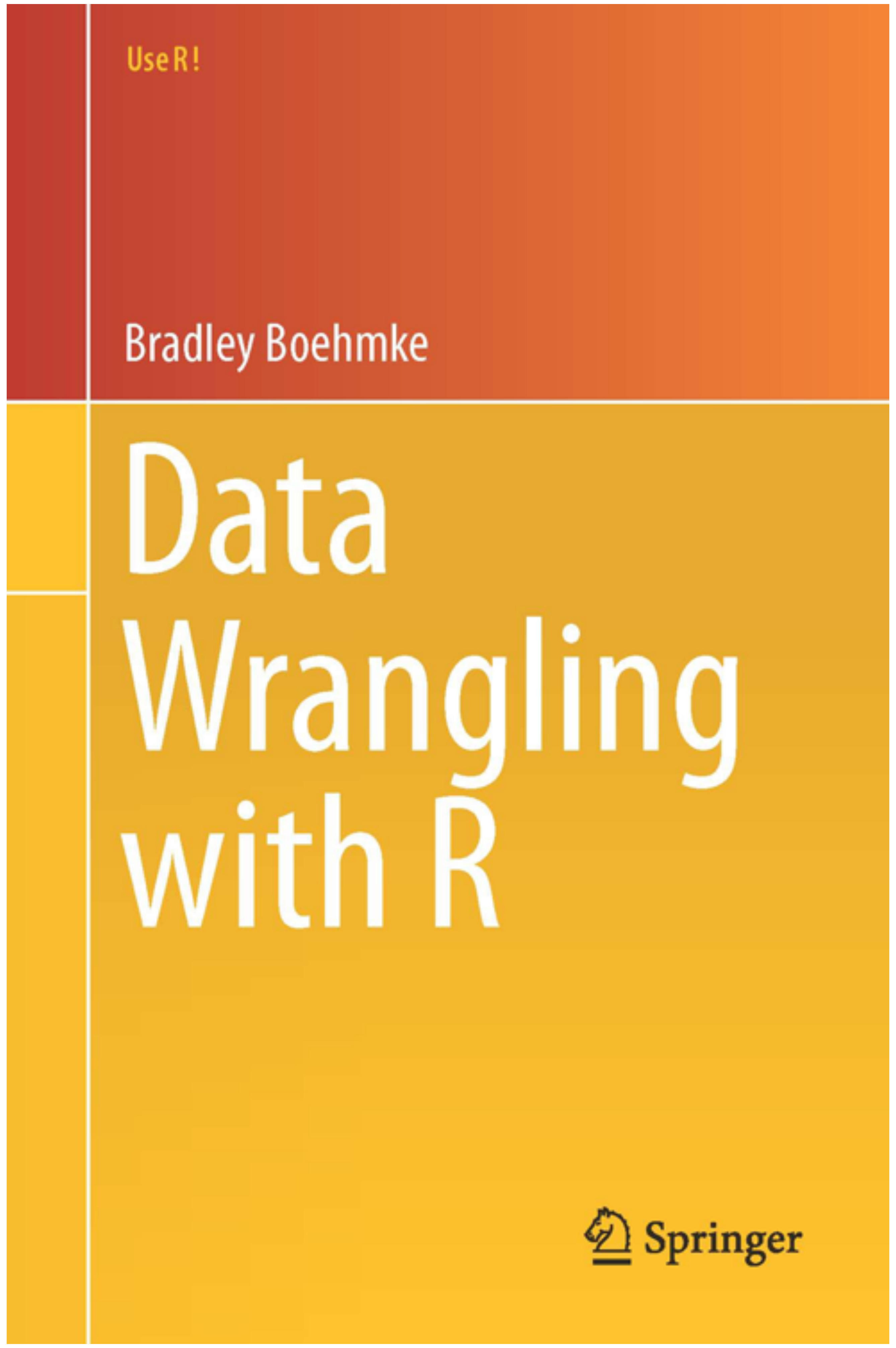
3. What is the min, max, median, and mean of these residuals?

SOLUTION

```
# create regression model
flight_lm <- lm(arr_delay ~ dep_delay + month + carrier, data = flights)

# extract residuals from flight_lm list
residuals <- flight_lm$residuals

# compute summary statistics
summary(residuals)
```

SO LITTLE TIME!

LEARN MORE



WHAT TO REMEMBER

FUNCTIONS TO REMEMBER

Operator/Function	Description
`data.frame, as_tibble, matrix, list, c(), :`	create data frames, tibbles, matrices, etc.
`str, names, colnames, rownames, dim, length, nrow, ncol`	understand attributes of data structures
`summary, mean, median, sum, colSums, rowSums, colMeans, rowMeans`	understand summary statistics of data structures
`[], [[]], $`	index & subset data structures