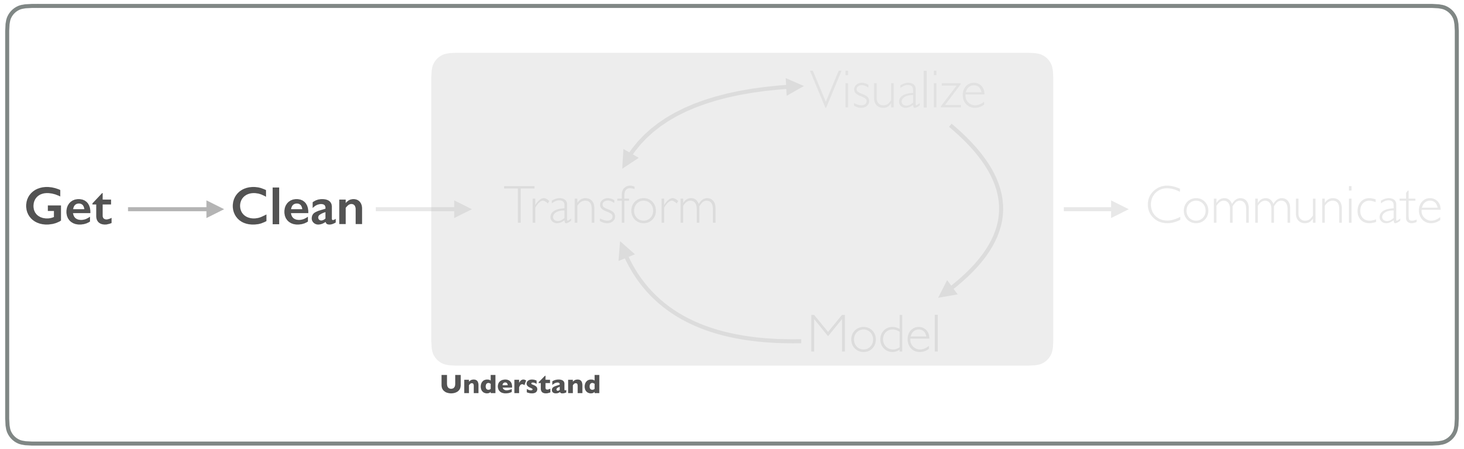
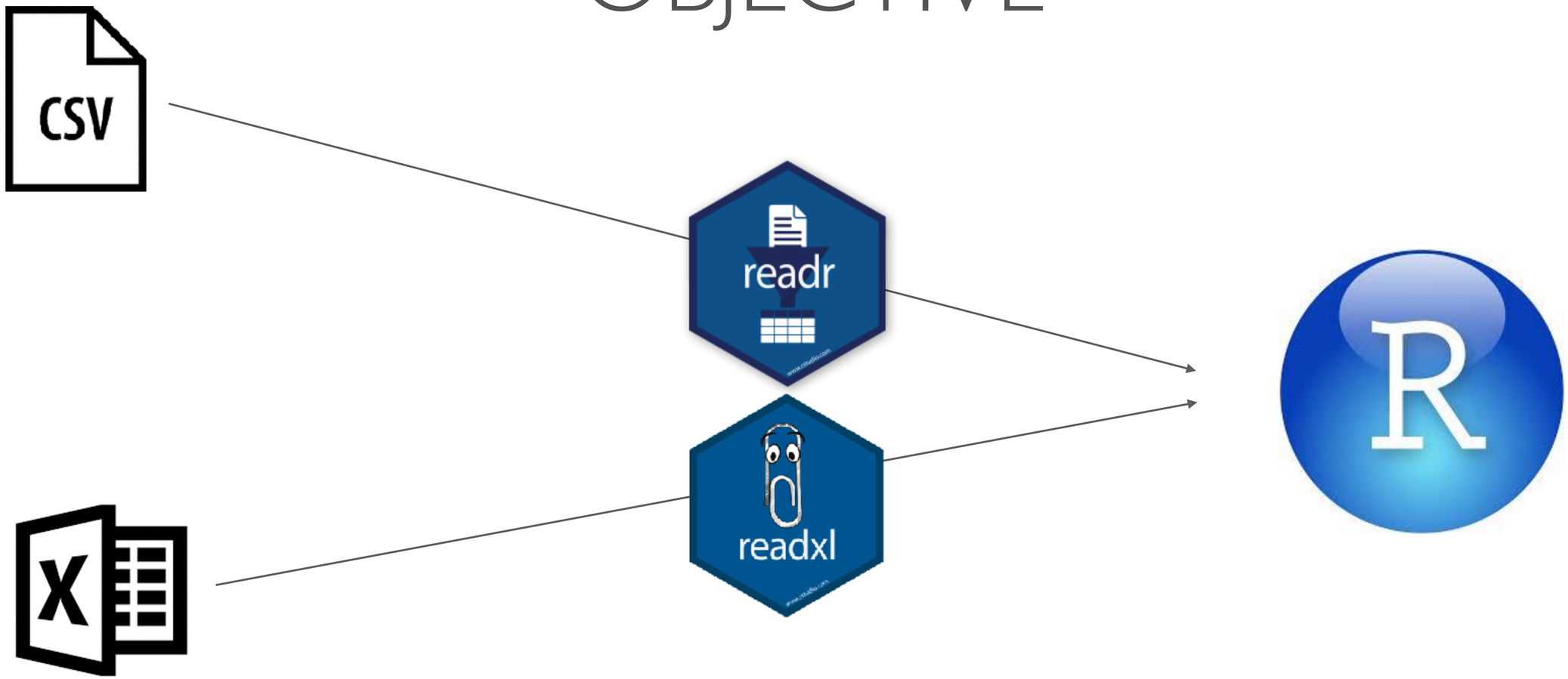
DATA PREP



†A modified version of Hadley Wickham's analytic process

OBJECTIVE



Its important to note that base R functions exist for reading and writing files; however, their syntax is inconsistent. **`readr`** *and* **`readxl`** *help to standardize these functions.*

PREREQUISITES

PACKAGE PREREQUISITE

```
library(readxl)
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
```

READING IN DATA

READING IN A .CSV FILE

- Text files are a popular way to hold and exchange tabular data
- Text file formats use **delimiters** to separate the different elements (.csv, .tsv, .txt, etc.)
- .csv most common - use `read_csv()` to read in

```
read_csv("data/mydata.csv")
Parsed with column specification:
cols(
  `variable 1` = col_integer(),
  `variable 2` = col_character(),
  `variable 3` = col_logical()
)
# A tibble: 3 × 3
  `variable 1` `variable 2` `variable 3`
         <int>        <chr>        <lgl>
1           10         beer         TRUE
2           25         wine         TRUE
3            8       cheese        FALSE
```

READING IN A .CSV FILE

- Text files are a popular way to hold and exchange tabular data
- Text file formats use **delimiters** to separate the different elements (.csv, .tsv, .txt, etc.)
- .csv most common - use `read_csv()` to read in

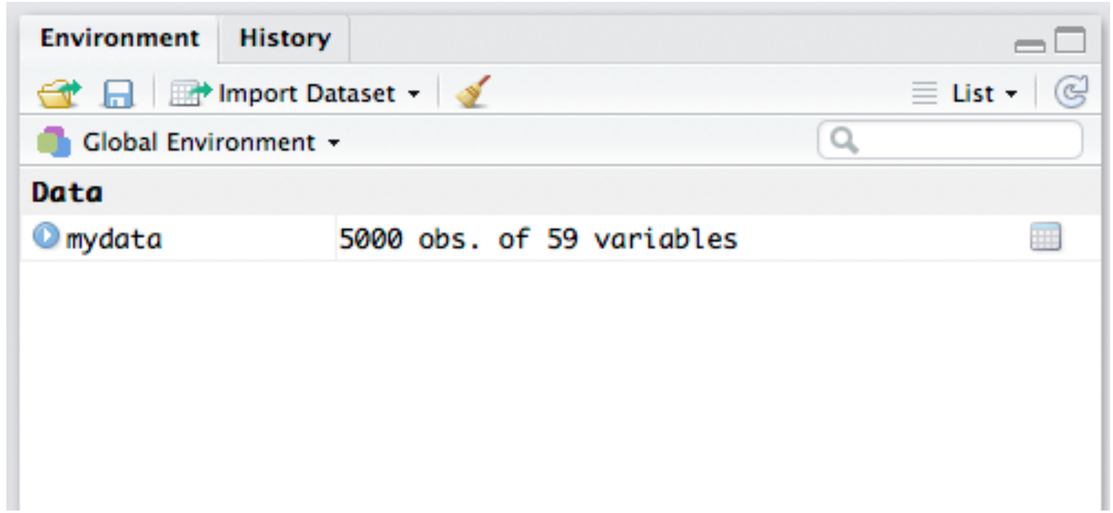
```
read_csv("data/mydata.csv")
Parsed with column specification:
cols(
  `variable 1` = col_integer(),
  `variable 2` = col_character(),
  `variable 3` = col_logical()
)
# A tibble: 3 × 3
  `variable 1` `variable 2` `variable 3`
         <int>        <chr>        <lgl>
1           10         beer         TRUE
2           25         wine         TRUE
3            8       cheese        FALSE
```

- Parsing information
- Resulting data read in

READING IN A .CSV FILE

- To use this data in R you need to save it to an R object
- In this case I save it as an object called **`mydata`**

```
mydata <- read_csv("data/mydata.csv")
```



READING IN AN EXCEL FILE

Excel is still the spreadsheet software of choice

You need to understand both the workbook *and* the sheet that you want to read in

```
# identify the sheet you want
excel_sheets(“data/mydata.xlsx")
[1] "PICK_ME_FIRST!" "Sheet2"          "extra_header"    "functions"
[5] "date_time"      “unique_NA”
```

READING IN AN EXCEL FILE

Excel is still the spreadsheet software of choice

You need to understand both the workbook *and* the sheet that you want to read in

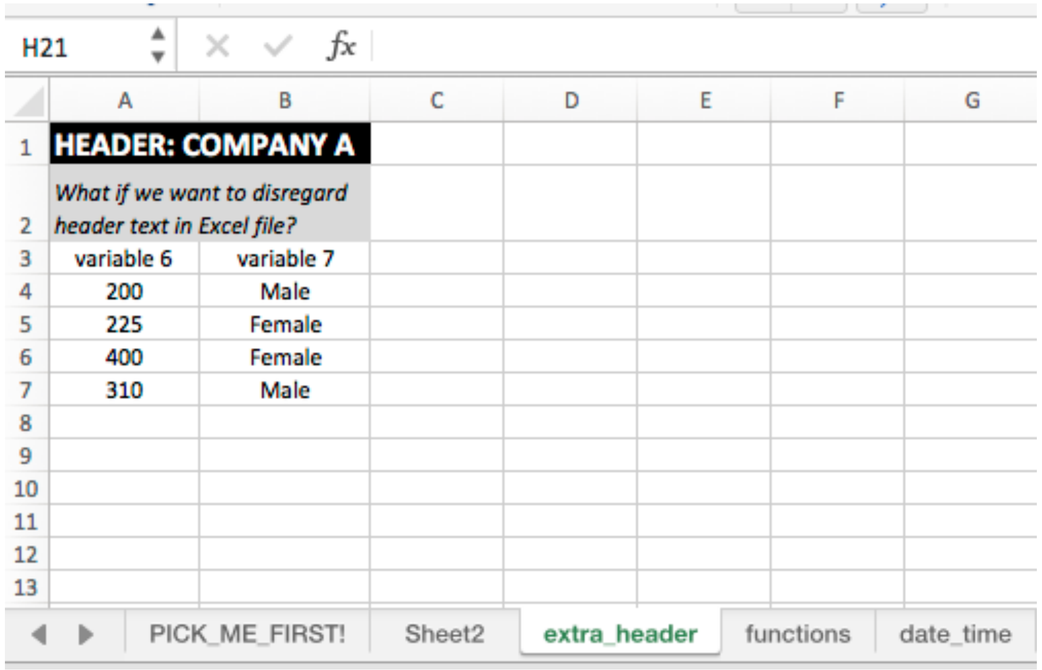
```
# identify the sheet you want
excel_sheets("data/mydata.xlsx")
[1] "PICK_ME_FIRST!" "Sheet2"         "extra_header"   "functions"
[5] "date_time"      "unique_NA"

# now read in the data
read_excel("data/mydata.xlsx", sheet = "PICK_ME_FIRST!")
# A tibble: 3 × 3
  `variable 1` `variable 2` `variable 3`
         <dbl>        <chr>        <dbl>
1           10         beer            1
2           25         wine            1
3            8       cheese            0
```

ADDITIONAL SPECIFICATIONS

Similar specifications exist with `readr::read_csv`

```
read_excel("data/mydata.xlsx",
           sheet = "extra_header",
           skip = 2)
# A tibble: 4 × 2
  `variable 6` `variable 7`
         <dbl>        <chr>
1          200         Male
2          225       Female
3          400       Female
4          310         Male
```



H21

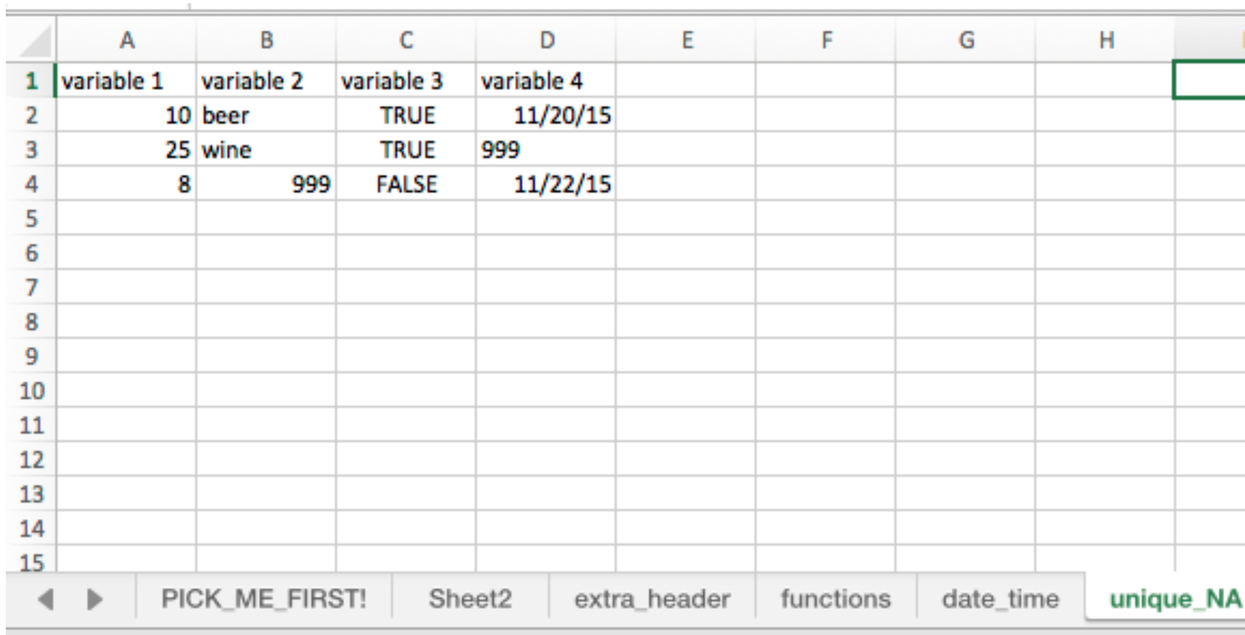
	A	B	C	D	E	F	G
1	**HEADER: COMPANY A**						
2	*What if we want to disregard header text in Excel file?*						
3	variable 6	variable 7					
4	200	Male					
5	225	Female					
6	400	Female					
7	310	Male					
8							
9							
10							
11							
12							
13							

PICK_ME_FIRST! | Sheet2 | extra_header | functions | date_time

ADDITIONAL SPECIFICATIONS

Similar specifications exist as we saw with `readr::read_csv`

```
read_excel("data/mydata.xlsx",
           sheet = "unique_NA",
           na = "999")
# A tibble: 3 × 4
  `variable 1` `variable 2` `variable 3` `variable 4`
         <dbl>        <chr>        <dbl>        <dbl>
1           10         beer            1        42328
2           25         wine            1           NA
3            8         <NA>            0        42330
```



	A	B	C	D	E	F	G	H
1	variable 1	variable 2	variable 3	variable 4				
2	10	beer	TRUE	11/20/15				
3	25	wine	TRUE	999				
4	8	999	FALSE	11/22/15				
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								

PICK_ME_FIRST! | Sheet2 | extra_header | functions | date_time | unique_NA

YOUR TURN!

1. *Read in the CustomerData.csv file and save as* **`raw_data`**

2. *What spreadsheets are contained in the "CustomerData.xlsx" file?*

3. *Read in the spreadsheet that contains the data.*

SOLUTION

```
# read in .csv
raw_data <- read_csv(“data/CustomerData.csv")


# read in spreadsheet names
excel_sheets(“data/CustomerData.xlsx")
[1] "Metadata" "Data"     "Sheet3"


# read in spreadsheet that contains the data
raw_data <- read_excel("data/CustomerData.xlsx", sheet = “Data")
```

UNDERSTANDING OUR DATA

WHAT ARE THE DIMENSIONS?

```
# load built-in data set
data(mtcars)

nrow(mtcars)
[1] 32

ncol(mtcars)
[1] 11

dim(mtcars)
[1] 32   11
```

WHAT ARE THE VARIABLES?

```
names(mtcars)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"
 [9] "am"   "gear" "carb"

glimpse(mtcars)
Observations: 32
Variables: 11
$ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4...
$ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8...
$ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360....
$ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123,...
$ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69...
$ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.57...
$ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.8...
$ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0...
$ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
$ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3...
$ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4…
```

Alternatively, you can just type:

```
View(mtcars)
```

YOUR TURN!

1. *What are the dimensions of our customer data (`raw_data`)?*

2. *What are the variable names?*

3. *Take a peak at the entire data set.*

SOLUTION

```
# dimension
dim(raw_data)


# names of variables
names(raw_data)
 [1] "CustomerID"          "Region"
 [3] "TownSize"            "Gender"
 [5] "Age" …


# take a peak at the entire data
View(raw_data)
```

ARE THERE MISSING VALUES?

```
# load built-in data set
data(airquality)

is.na(airquality)
       Ozone Solar.R  Wind  Temp Month   Day
  [1,] FALSE   FALSE FALSE FALSE FALSE FALSE
  [2,] FALSE   FALSE FALSE FALSE FALSE FALSE
  [3,] FALSE   FALSE FALSE FALSE FALSE FALSE
  [4,] FALSE   FALSE FALSE FALSE FALSE FALSE
  [5,]  TRUE    TRUE FALSE FALSE FALSE FALSE
  [6,] FALSE    TRUE FALSE FALSE FALSE FALSE
  [7,] FALSE   FALSE FALSE FALSE FALSE FALSE
  [8,] FALSE   FALSE FALSE FALSE FALSE FALSE
  [9,] FALSE   FALSE FALSE FALSE FALSE FALSE
 [10,]  TRUE   FALSE FALSE FALSE FALSE FALSE
 [11,] FALSE    TRUE FALSE FALSE FALSE FALSE
 [12,] FALSE   FALSE FALSE FALSE FALSE FALSE
```

We identify missing values with:

`is.na()`

ARE THERE MISSING VALUES?

```
How many missing values are there?
sum(is.na(airquality))
[1] 44

Where are these missing values?
colSums(is.na(airquality))
Ozone Solar.R    Wind    Temp   Month     Day
   37       7       0       0       0       0

Remove missing values?
clean_data <- na.omit(airquality)
```

Luckily:

TRUE = 1
FALSE = 0

So we can do simple math with missing values

YOUR TURN!

1. *How many missing values are in our customer data (`raw_data`)?*

2. *Which variables have the most missing data?*

3. *How would you omit missing values so we have a complete data set?*

SOLUTION

```
# how many missing values are in our customer data?
sum(is.na(raw_data))
 [1] 124

# which variables have the most missing values?
missing <- colSums(is.na(raw_data))
sort(missing, decreasing = TRUE)
        NumberBirds              Gender         JobCategory
                 34                  33                  15
          HomeOwner       HouseholdSize          NumberDogs
                 13                   8                   8
         NumberCats           NumberPets          CustomerID
                  7                   6                   0
             Region            TownSize                 Age

# how would you delete missing observations?
clean_data <- na.omit(raw_data)
dim(clean_data)
[1] 4893   59
```

So, we're ready to do some exploratory data analysis but before we do let's talk about data structures!

DATA STRUCTURE

DATA STRUCTURE 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
```

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
```

For an example type **`state.abb`** in your console

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
```

There are many other ways to create vectors (sequential & distribution data) but these are the two most common.

INDEXING

`vector[element]`

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


# Try these
 v1[4]
 v1[4:7]
 v1[c(4, 3, 4)]
 v1[v1 > 6]
 v1[v1 > 8 | v1 <=3]
```

Try these different forms of indexing

QUICK SUMMARIES

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

```
length(v1)
summary(v1)
mean(v1)
median(v1)
v1 > 5
sum(v1 > 5)
```

What do these do?

YOUR TURN!

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

2. how many elements are in this vector

4. Subset state.name for elements 35, 38, 14, 17. Which states are these?

SOLUTION

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

# how many elements are in state.name
length(state.name)
[1] 50

# subset state.name for V35, V17, V14, V38
state.name[c(35, 38, 14, 17)]
[1] "OH" "PA" "IN" "KY"
```

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

INDEXING/SUBSETTING

`matrix[row, col]`

```
# extract individual elements
m1[1, 3]


# extract all rows and columns 1 through 3
m1[, 1:3]


# index for all rows and just the second column
m1[1:3, ]
```

Try these different forms of indexing & subsetting:

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]
```

YOUR TURN!

1. Compute the column means of the built-in matrix named ***`VADeaths`.***

2. Compute the row means of the built-in matrix named ***`VADeaths`.***

4. Index ***`VADeaths`*** for females aged 55-64.

SOLUTION

```
# compute column means
colMeans(VADeaths)
  Rural Male Rural Female   Urban Male Urban Female
       32.74         25.18         40.48         25.28

# compute row means
rowMeans(VADeaths)
 50-54  55-59  60-64  65-69  70-74
11.050 16.925 25.875 40.400 60.350

# Index for females aged 55-64
VADeaths[2:3, c(2, 4)]
      Rural Female Urban Female
55-59         11.7         13.6
60-64         20.3         19.3
```

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 `customer data` 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

INDEXING

`data.frame[row, col]`

```
# extract the second column and all rows using column indexing or the name
raw_data[, 4]
raw_data[, “Gender"]

# extract all rows and columns 1 through 3
raw_data[, 1:3]
raw_data[, c(“CustomerID”, “Region”, "TownSize")]

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

Try these different forms of indexing & subsetting:

INDEXING

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

```
# extract the second column and all rows using colum
raw_data[, 4]
raw_data[, “Gender"]

# extract all rows and columns 1 th
raw_data[, 1:3]
raw_data[, c(“CustomerID”                    )]

# index for firs
raw_data[1
```

We will learn more efficient ways to index and manipulate data frames in the next module

Try these different forms of indexing & subsetting:

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

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 ...
```

INDEXING

- 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
model[“residuals”]
model[[“residuals”]]
model$residuals
model[["residuals"]][1:20]
```

How do they differ?

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
```

WHAT TO REMEMBER

FUNCTIONS TO REMEMBER

Operator/Function	Description
`read_csv, excel_sheets, read_excel`	import data
`data.frame, vector, 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