Welcome to DATA 151

I'm so glad you're here!

DATA 151: CLASS 4B
INTRODUCTION TO DATA SCIENCE (WITH R)

DATA WRANGLING

UPDATED
DATA 2022

NOTES PREPARED BY PROF. KITADA SMALLEY (FALL 2022)

ANNOUNCEMENTS

RELEVANT READING

Introduction to Data Science:

- Tuesday:
  - Ch 3: Programming basics
- Thursday:
  - Ch 4: Tidyverse

HOMEWORK REMINDER

Due next week:

- *HW #4: DC Introduction to Tidyverse*
  - ***No submission on WISE necessary, do on DataCamp***
- *Project Milestone #2: Group Meetings*
  - Required to approve data set
  - Weeks 5 and 6

DATA151: `dplyr` Verbs

Kitada Smalley

Learning Objectives

In this session students will learn the basics of working with `dplyr` verbs from the `tidyverse` as well as employ the pipe operator `%>%`.

- Use the piping operator `%>%` in your code to improve readability
- Employ `dplyr` Verbs
    - `filter()`
    - `count()`
    - `arrange()`
    - `group_by` and `summarise()`
    - `select()`
    - `mutate()`

You will need to start by calling the `tidyverse` library.

```
library(tidyverse)
```

PART I: MLB DATA

LAHMAN PACKAGE IN R

Directions/Introduction: In this section we will use the `People` dataset contained in the `Lahman` package.

Source: Lahman, S. (2020) Lahman's Baseball Database, 1871-2019

```
# Import the package
install.packages("Lahman")
library(Lahman)

# Import the data for baseball players
data(People)
head(People)
```

LAHMAN PACKAGE IN R

Learn a little bit about the dataset before you start. *What are the variables contained in this dataset? How would you classify each variables' type?*

```
?People
```

Description

This database contains pitching, hitting, and fielding statistics for Major League Baseball from 1871 through 2021. It includes data from the two current leagues (American and National), the four other "major" leagues (American Association, Union Association, Players League, and Federal League), and the National Association of 1871-1875.

This database was created by Sean Lahman, who pioneered the effort to make baseball statistics freely available to the general public. What started as a one man effort in 1994 has grown tremendously, and now a team of researchers have collected their efforts to make this the largest and most accurate source for baseball statistics available anywhere.

This database, in the form of an R package offers a variety of interesting challenges and opportunities for data processing and visualization in R.

In the current version, the examples make extensive use of the `dplyr` package for data manipulation (tabulation, queries, summaries, merging, etc.), reflecting the original relational database design and `ggplot2` for graphics.

5. Data Structure

What are the variables contained in this dataset? How would you classify each variables' type?

```
str(People)
```

```
## 'data.frame':    20093 obs. of  26 variables:
##  $ playerID    : chr  "aardsda01" "aaronha01" "aaronto01" "aasedo01" ...
##  $ birthYear   : int  1981 1934 1939 1954 1972 1985 1850 1877 1869 1866 ...
##  $ birthMonth  : int  12 2 8 9 8 12 11 4 11 10 ...
##  $ birthDay    : int  27 5 5 8 25 17 4 15 11 14 ...
##  $ birthCountry: chr  "USA" "USA" "USA" "USA" ...
##  $ birthState  : chr  "CO" "AL" "AL" "CA" ...
##  $ birthCity   : chr  "Denver" "Mobile" "Mobile" "Orange" ...
##  $ deathYear   : int  NA 2021 1984 NA NA NA 1905 1957 1962 1926 ...
##  $ deathMonth  : int  NA 1 8 NA NA NA 5 1 6 4 ...
##  $ deathDay    : int  NA 22 16 NA NA NA 17 6 11 27 ...
##  $ deathCountry: chr  NA "USA" "USA" NA ...
##  $ deathState  : chr  NA "GA" "GA" NA ...
##  $ deathCity   : chr  NA "Atlanta" "Atlanta" NA ...
##  $ nameFirst   : chr  "David" "Hank" "Tommie" "Don" ...
##  $ nameLast    : chr  "Aardsma" "Aaron" "Aaron" "Aase" ...
##  $ nameGiven   : chr  "David Allan" "Henry Louis" "Tommie Lee" "Donald William"
...
##  $ weight      : int  215 180 190 190 184 235 192 170 175 169 ...
##  $ height      : int  75 72 75 75 73 74 72 71 71 68 ...
##  $ bats        : Factor w/ 3 levels "B","L","R": 3 3 3 3 2 2 3 3 3 2 ...
##  $ throws      : Factor w/ 3 levels "L","R","S": 2 2 2 2 1 1 2 2 2 1 ...
##  $ debut       : chr  "2004-04-06" "1954-04-13" "1962-04-10" "1977-07-26" ...
##  $ finalGame   : chr  "2015-08-23" "1976-10-03" "1971-09-26" "1990-10-03" ...
##  $ retroID     : chr  "aardd001" "aaroh101" "aarot101" "aased001" ...
##  $ bbrefID     : chr  "aardsda01" "aaronha01" "aaronto01" "aasedo01" ...
##  $ deathDate   : Date, format: NA "2021-01-22" ...
##  $ birthDate   : Date, format: "1981-12-27" "1934-02-05" ...
```

PART II: Pipe `%>%` Operator

Ceci n'est pas une pipe

The pipe `%>%` operator takes a data set from the left and passes it into a function on the right. This operator can be used to help you read your code. When you are reading your code out loud try replacing `%>%` with saying *"and then"*. These operators can be used to make your code more efficient (i.e. use less lines and less typing). They can also be used sequentially refine your data.

ASIDE TO ART HISTORY: SURREALISM

The Treachery of Images

From Wikipedia, the free encyclopedia

The Treachery of Images (French: *La Trahison des Images*) is a 1929 painting by Belgian surrealist painter René Magritte. It is also known as *This Is Not a Pipe*[2] and *The Wind and the Song*.[3] Magritte painted it when he was 30 years old. It is on display at the Los Angeles County Museum of Art.[1]

The painting shows an image of a pipe. Below it, Magritte painted, "*Ceci n'est pas une pipe*", French for "This is not a pipe".

> The famous pipe. How people reproached me for it! And yet, could you stuff my pipe? No, it's just a representation, is it not? So if I had written on my picture "This is a pipe", I'd have been lying!
>
> — René Magritte[4]

The theme of pipes with the text "Ceci n'est pas une pipe" is extended in *Les Mots et Les Images*,[5] *La Clé des Songes*,[6] *Ceci n'est pas une pipe (L'air et la chanson)*,[7] *The Tune and Also the Words*,[8] *Ceci n'est pas une pomme*,[9] and *Les Deux Mystères*.[10]

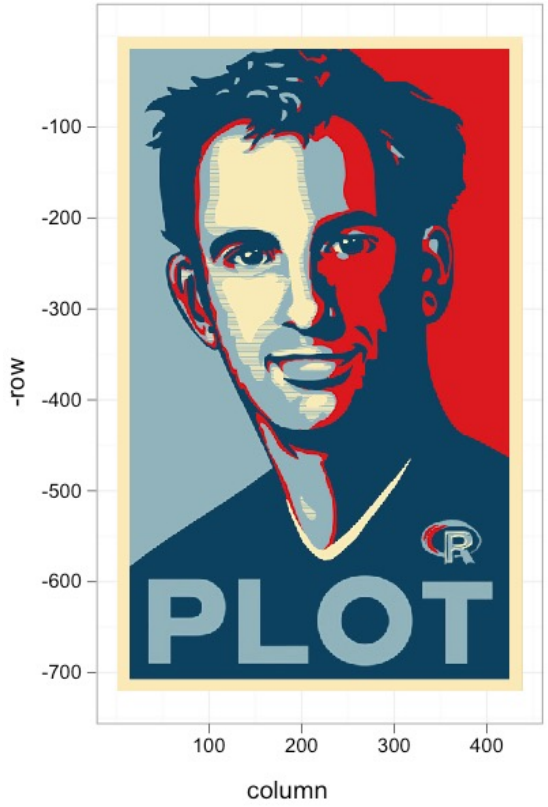
The Treachery of Images

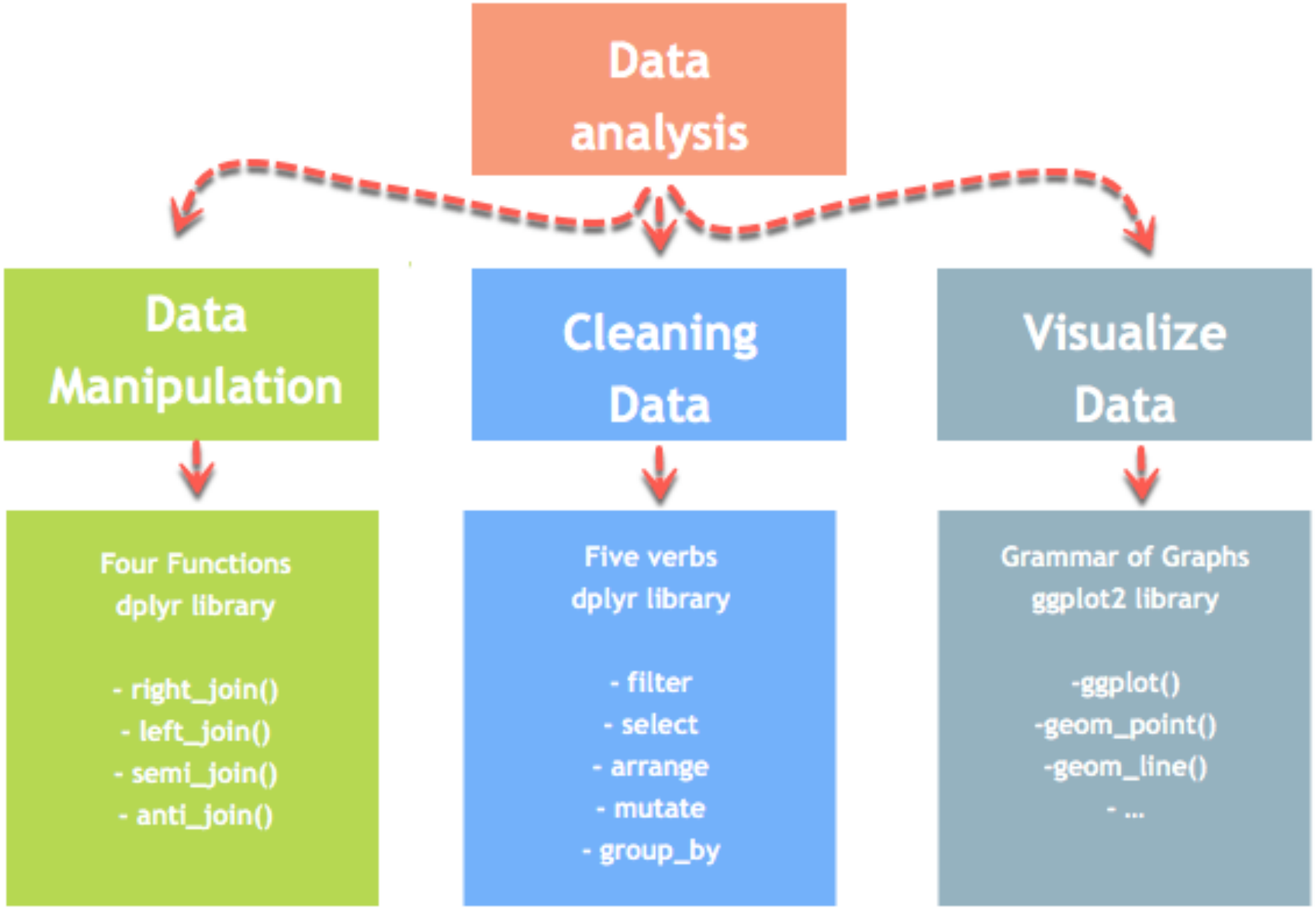
Artist	René Magritte
Year	1929
Medium	Oil on canvas
Movement	Surrealism
Dimensions	60.33 cm × 81.12 cm (23.75 in × 31.94 in)
Location	Los Angeles County Museum of Art[1]

PART III: DATA WRANGLING

DATA WRANGLING

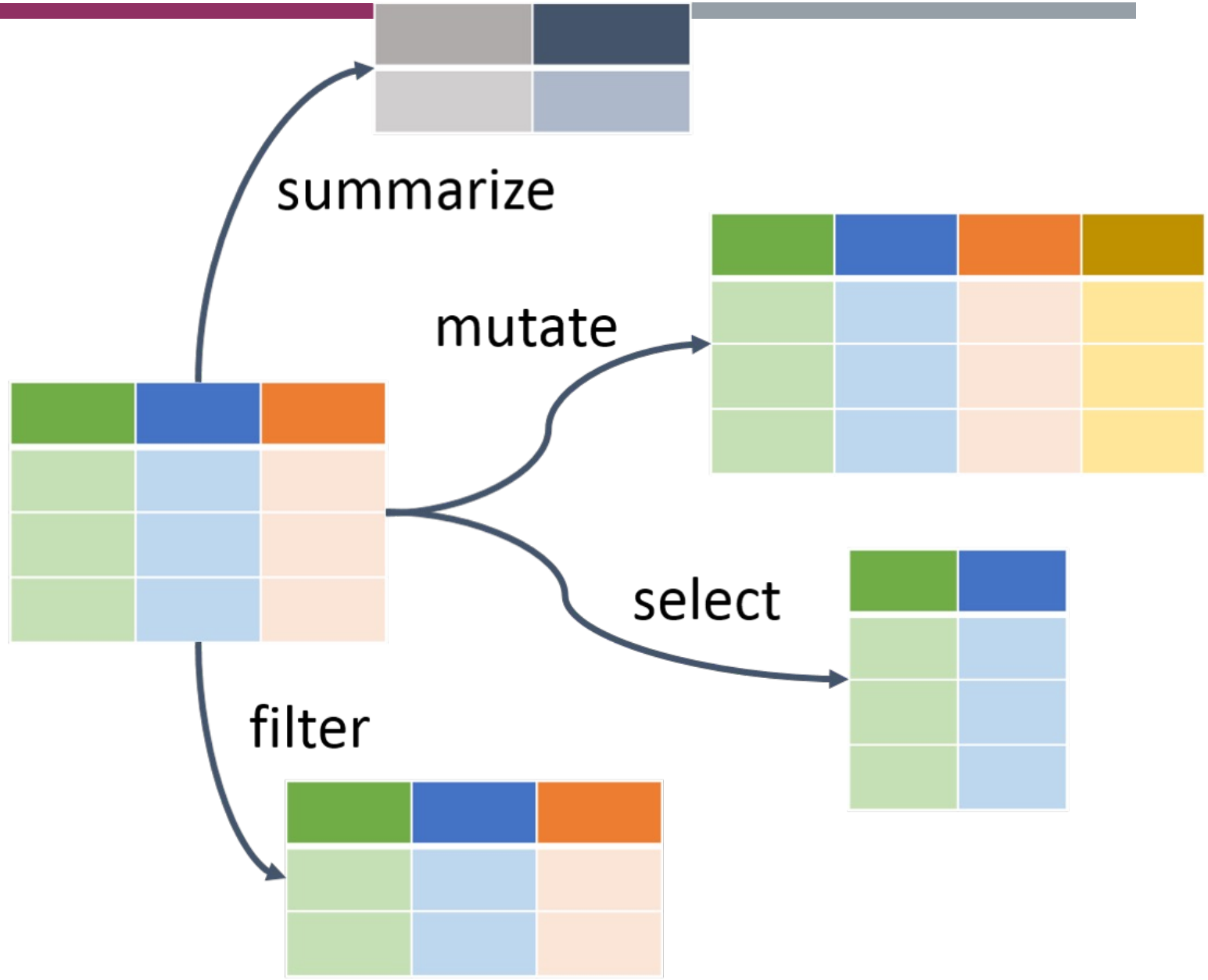
- Luckily for us… we have the tidyverse …




Data analysis
Data Manipulation
Cleaning Data
Visualize Data
Four Functions
dplyr library
- right_join()
- left_join()
- semi_join()
- anti_join()
Five verbs
dplyr library
- filter
- select
- arrange
- mutate
- group_by
Grammar of Graphs
ggplot2 library
-ggplot()
-geom_point()
-geom_line()
- ...

DPLYR VERBS


dplyr
www.rstudio.com


summarize
mutate
select
filter

DPLYR VERBS

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

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

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

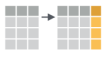
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)
ends_with(match)
matches(match)
num_range(prefix, range)
one_of(…)
starts_with(match)
:, e.g. mpg:cyl
-, e.g, -Species

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

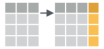


mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

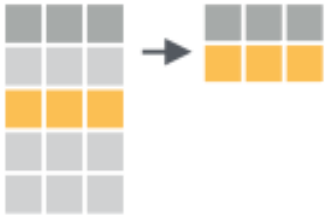
rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

DPLYR VERBS:FILTER

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



filter(.data, …**)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

Q1:

How many MLB players are from the United States?

1. `filter`

We can use the `filter` verb to identify the observations that adhere to a specification.

Filtering with strings

Let's look at only baseball players that are from the United States of America.

Note: You don't want to print the whole data set, unless you want to scroll for a long time...

```
## FILTER USA
usa<-People %>%
  filter(birthCountry == "USA")

## How big is this?
## How many players are from the USA?
dim(usa)
```

```
## [1] 17601    26
```

Try it!

What countries are represented in these data?

```
unique(People$birthCountry)
```

```
##  [1] "USA"           "D.R."          "Venezuela"      "Cuba"
##  [5] "Mexico"        "Panama"        "CAN"            "P.R."
##  [9] "Russia"        "Japan"         "Curacao"        "Colombia"
## [13] "Nicaragua"     "Germany"       "Norway"         "Ireland"
## [17] "Italy"         "Bahamas"       "United Kingdom" "South Korea"
## [21] "Australia"     "Czech Republic" "V.I."          "Netherlands"
## [25] "France"        "Aruba"         NA               "Sweden"
## [29] "Hong Kong"     "Afghanistan"   "Spain"          "Greece"
## [33] "Taiwan"        "Philippines"   "Jamaica"        "Poland"
## [37] "Honduras"      "Brazil"        "Viet Nam"       "Guam"
## [41] "Denmark"       "Switzerland"   "Austria"        "Singapore"
## [45] "China"         "Belgium"       "Peru"           "Belize"
## [49] "Indonesia"     "Finland"       "Lithuania"      "South Africa"
## [53] "At Sea"        "Slovakia"      "American Samoa" "Saudi Arabia"
## [57] "Portugal"      "Latvia"
```

Pick your favorite country from this list repeat the example. How many baseball players are from your chosen country?

```
## INSERT YOUR CODE HERE ##
```

Q2:

How many MLB players are from Oregon?

Think about it…

What do you think the code would look like to only retain the rows for baseball players from Oregon?

Hint: Use the variable `birthState`

```
## INSERT CODE HERE ##
```

SOLUTION

Think about it...

What do you think the code would look like to only retain the rows for baseball players from Oregon?

Hint: Use the variable `birthState`

```
oregon<-People %>%
  filter(birthCountry == "USA")%>%
  filter(birthState == "OR")

dim(oregon)
```

```
## [1] 138  26
```

Q3:

How many MLB players were shorter than 5ft?

FILTERING WITH NUMBERS

Filtering with numbers

Create a new dataframe that only includes players less than 60 inches tall. Show the dataset. How many players are there who are less than 60 inches tall? What are their names?

```
short <- People %>%
  filter(height < 60)

short
```

FILTERING WITH NUMBERS

```
## PULLING OUT JUST THE FIRST AND LAST NAME
## AND COMBINING INTO A STRING
paste(short$nameFirst, short$nameLast)
```

```
## [1] "Eddie Gaedel" "Tom Healey"
```

Q4:

How many MLB players are from each state?

DPLYR VERBS: COUNT

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

2. count

```
state<-People %>%
  filter(birthCountry == "USA")%>%
  count(birthState)

head(state)
```

```
##   birthState    n
## 1         AK   12
## 2         AL  347
## 3         AR  161
## 4         AZ  130
## 5         CA 2386
## 6         CO   98
```

Q5:

How many MLB players are bat right, left, or both?

Learn by doing!

How many players bat right (R), left (L), or both (B)?

Hint: Use the variable `bats`.

```
## INSERT CODE HERE ##
```

What does `NA` mean?

```
ANSWER HERE:
```

SOLUTION

Learn by doing!

How many players bat right (R), left (L), or both (B)?

Hint: Use the variable `bats`.

```
batSide<-People %>%
  count(bats)

batSide
```

```
##   bats     n
## 1    B  1242
## 2    L  5321
## 3    R 12626
## 4 <NA>  1181
```

Q6:

What state generates the most MLB players?

3. arrange

```
stateArr<-People %>%
  filter(birthCountry == "USA")%>%
  count(birthState)%>%
  arrange(desc(n))

head(stateArr)
```

```
##   birthState    n
## 1         CA 2386
## 2         PA 1458
## 3         NY 1260
## 4         IL 1093
## 5         OH 1068
## 6         TX  980
```

Q6:

Which country has the tallest baseball players on average?

DPLYR VERBS:GROUP_BY

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

DPLYR: GROUP_BY AND SUMMARISE

- We are often interested in data aggregation to a certain level, this can be done with group_by() and summarise()
- We will string this operations together with the pipe operator %>%

4. `group_by` and `summarise`

Create subgroups of your data using `group_by` with a categorical variable. Once the data has been grouped you can perform numerical summaries on them.

Make a dataframe that displays just the average height of players from each country. What country has the tallest players?

```
tallest <- People %>%
  group_by(birthCountry) %>%
  summarise(avg_height=mean(height,na.rm = TRUE)) %>%
  arrange(desc(avg_height))

tallest
```

```
## # A tibble: 58 × 2
##    birthCountry avg_height
##    <chr>              <dbl>
```

```
## # A tibble: 58 × 2
##    birthCountry avg_height
##    <chr>              <dbl>
##  1 Indonesia           78
##  2 Belgium             77
##  3 Hong Kong           76
##  4 Jamaica             75.2
##  5 Afghanistan         75
##  6 Lithuania           75
##  7 Guam                74.5
##  8 Brazil              74.2
##  9 Singapore           74
## 10 Australia           73.5
## # … with 48 more rows
```

This is surprising!

```{r}
tallest <- People %>%
  group_by(birthCountry) %>%
  summarise(avg_height=mean(height,na.rm = TRUE),
            n=n()) %>%
  arrange(desc(avg_height))

tallest
```

birthCountry <chr>	avg_height <dbl>	n <int>
Indonesia	78.00000	1
Belgium	77.00000	1
Hong Kong	76.00000	1
Jamaica	75.25000	4
Afghanistan	75.00000	1
Lithuania	75.00000	1
Guam	74.50000	2
Brazil	74.20000	5
Singapore	74.00000	1
Australia	73.54545	33

Thomas Raymond Mastny (born February 4, 1981) is a former Major League Baseball right-handed relief pitcher. He stands 6 feet, 6 inches in height and weighs 220 pounds. Mastny is the only Indonesian-born player in Major League history. He was raised in Zionsville, Indiana, where he played for Zionsville Community High School. He made his major league debut with the Indians on July 25, 2006.[1]

Contents [hide]

Career [edit]

Tom Mastny	
 Mastny signing autographs at Indians' Spring training 2008 in Winter Haven, Florida.	
Relief pitcher	
Born: February 4, 1981 (age 41) Bontang, Indonesia	
Batted: Right	**Threw:** Right
MLB debut	
July 30, 2006, for the Cleveland Indians	

5. `select`

Rather than keeping all of the variables, we can select the ones that we are interested in using.

```
peopleS<-People%>%
  select(c(nameFirst, nameLast, weight, height))

str(peopleS)
```

```
## 'data.frame':    20093 obs. of  4 variables:
##  $ nameFirst: chr  "David" "Hank" "Tommie" "Don" ...
##  $ nameLast : chr  "Aardsma" "Aaron" "Aaron" "Aase" ...
##  $ weight   : int  215 180 190 190 184 235 192 170 175 169 ...
##  $ height   : int  75 72 75 75 73 74 72 71 71 68 ...
```

Q7:

Can we calculate each players BMI?

DPLYR VERBS:MUTATE

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

6. `mutate`

The `mutate` function can be used to create a new variable (a new column) as a function of other columns.

Example BMI

Body Mass Index (BMI) is a convenient rule of thumb used to broadly categorize a person as underweight, normal weight, overweight, or obese based on tissue mass (muscle, fat, and bone) and height.

$$BMI = \frac{weight(lbs) \times 703}{height^2(in^2)}$$

$$BMI = \frac{weight(lbs) \times 703}{height^2(in^2)}$$

```
peopleBMI<-People %>%
  select(c(nameFirst, nameLast, weight, height))%>%
  mutate(bmi=(weight*703)/(height^2))

## OBSERVE THAT THE NEW COLUMN IS ADDED ON
str(peopleBMI)
```

```
## 'data.frame':    20093 obs. of  5 variables:
##  $ nameFirst: chr  "David" "Hank" "Tommie" "Don" ...
##  $ nameLast : chr  "Aardsma" "Aaron" "Aaron" "Aase" ...
##  $ weight   : int  215 180 190 190 184 235 192 170 175 169 ...
##  $ height   : int  75 72 75 75 73 74 72 71 71 68 ...
##  $ bmi      : num  26.9 24.4 23.7 23.7 24.3 ...
```

Q8:

Which MLB player has the highest BMI?

The CDC defines a BMI greater than 30 “within an obese range”. Create a dataframe of only the obese players. How many players are in this dataset? Which player has the highest BMI?

```
bmi30<-People %>%
  select(c(nameFirst, nameLast, weight, height))%>%
  mutate(bmi=(weight*703)/(height^2))%>%
  filter(bmi>30)%>%
  arrange(desc(bmi))

head(bmi30)
```

```
##   nameFirst nameLast weight height      bmi
## 1 Alejandro     Kirk    265     68 40.28871
## 2   Bartolo    Colon    285     71 39.74509
## 3     Pablo Sandoval    268     70 38.44980
## 4    Prince  Fielder    275     71 38.35053
## 5     Jumbo     Diaz    315     76 38.33882
## 6     Reyes  Moronta    265     70 38.01939
```

Alejandro Kirk

Baseball catcher

Overview | Stats | Videos | News | Contracts

About

Alejandro Kirk is a Mexican professional baseball catcher for the Toronto Blue Jays of Major League Baseball.

Wikipedia

Example Dates

If you look back at the structure of the data set, you might notice that `birthDate` and `deathDate` are `Date` variables. This is a new type of variable for us! If you subtract `Date` variables you will get the number of data between the dates in question.

Q9:

Which MLB player lived the longest?

Creating a dataset for only baseball players who have deceased, add a new column to the data set for a player's lifetime (in years). You can assume that each year has 365 days. Which baseball player lived the longest? How many years old was he?

```
oldest <- People %>%
  select(c(nameFirst, nameLast, birthDate, deathDate, deathYear))%>%
  filter(is.na(deathYear)==FALSE) %>%
  mutate(lifetime=as.numeric((deathDate-birthDate)/365) )%>%
  arrange(desc(lifetime))

head(oldest)
```

```
##   nameFirst  nameLast  birthDate  deathDate deathYear lifetime
## 1       Red      Hoff 1891-05-08 1998-09-17      1998 107.4329
## 2    Connie   Marrero 1911-04-25 2014-04-23      2014 103.0658
## 3       Bob    Wright 1891-12-13 1993-07-30      1993 101.6959
## 4       Ace    Parker 1912-05-17 2013-11-06      2013 101.5425
## 5      Tony Malinosky 1909-10-07 2011-02-08      2011 101.4082
## 6      Karl   Swanson 1900-12-17 2002-04-03      2002 101.3616
```

Try it!

Convert the debut and finalGame variables to dates using the `as.Date()` function. Add a new column to the data set for a player's career (in years). You can assume that each year has 365 days. Which baseball player had the longest career? How long was it?

```
## INSERT CODE HERE ##
```

Q10:

Which MLB player had the longest career?

SOLUTION

Try it!

Convert the debut and finalGame variables to dates using the `as.Date()` function. Add a new column to the data set for a player's career (in years). You can assume that each year has 365 days. Which baseball player had the longest career? How long was it?

```
longest <- People %>%
  select(c(nameFirst, nameLast, finalGame, debut))%>%
  mutate(career=as.numeric((as.Date(finalGame)-as.Date(debut))/365)) %>%
  arrange(desc(career))

head(longest)
```

```
##   nameFirst nameLast  finalGame      debut   career
## 1      Nick  Altrock 1933-10-01 1898-07-14 35.23836
## 2       Jim O'Rourke 1904-09-22 1872-04-26 32.42740
## 3    Minnie   Minoso 1980-10-05 1949-04-19 31.48493
## 4   Charley  O'Leary 1934-09-30 1904-04-14 30.48219
## 5     Arlie   Latham 1909-09-30 1880-07-05 29.25479
## 6    Deacon  McGuire 1912-05-18 1884-06-21 27.92329
```

Nick Altrock

American baseball player

Overview | Stats

About

Nicholas Altrock was an American professional baseball player and coach. He played in Major League Baseball as a left-handed pitcher between 1898 and 1919. After the 1919 season he continued to make periodic appearances as a pinch hitter for many years, until his final game at the age of 57.

Wikipedia