Week 4 Demonstration

Tidy & Manipulate: Part I - Tidy

Dr. Anil Dolgun

21/03/2018

Course Announcements

- Suggested solutions to Week 3 Worksheet are available here

Course Announcements

- Suggested solutions to Week 3 Worksheet are available here
- Answers to Module 3 Skill Builders are available here

Course Announcements

- Suggested solutions to Week 3 Worksheet are available here
- Answers to Module 3 Skill Builders are available here
- Assignment 1 is due this Sunday! Details are available here.

Course Announcements

- Suggested solutions to Week 3 Worksheet are available here
- Answers to Module 3 Skill Builders are available here
- Assignment 1 is due this Sunday! Details are available here.
- Any other questions or concerns?

Course Feedback

- Thank you to the students who have provided ongoing feedback. The comments have been very helpful and encouraging. Current feedback provided:
- Positives:
  - Well structured lecture delivery,
  - The mix of lecture material,
  - The multiple opportunities to practice material with tutors nearby
  - Plenty of supplementary coursework including interactive learning (i.e., Datacamp modules, swirl)
- Needs improvement:
  - There is a frequent mutter from the students --> Strict no noise in lecture!
  - Better if DataCamp modules and skill builders are graded --> In semester 2, I'll be able to integrate DataCamp to LMS Canvas.
  - When a student asks a question in the lecture, it can't be heard in the recording --> I'll repeat the question before answering it.
  - Printable version of Module notes --> Coming soon!

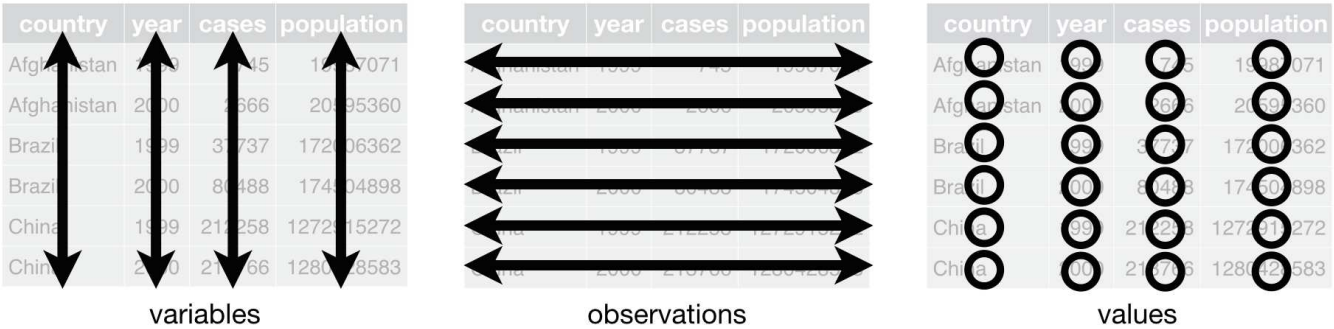
Tidy & Manipulate: Part I - Tidy

Recall

- Now that you have imported your data and you understand the basics of managing your data structure
- The next thing you probably want to do is jump into exploratory data analysis (i.e. basic descriptive statistics or data visualisations).
- However, prior to that, it is important to make sure your data frame is properly prepared for analysis.
- This may require you to do some basic manipulation and ensure your data is in a format.

What is Tidy data?

- Hadley Wickham's tidy data publication



- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.

Why Tidy data ?

- Consistent data structure allows easier learning of related tools because they have an underlying uniformity.

Why Tidy data ?

- Consistent data structure allows easier learning of related tools because they have an underlying uniformity.
- Placing variables in columns takes advantage of R's vectorized nature. One can extract variables in a simple, standard way.

Why Tidy data ?

- Consistent data structure allows easier learning of related tools because they have an underlying uniformity.
- Placing variables in columns takes advantage of R's vectorized nature. One can extract variables in a simple, standard way.
- Have a look at the following illustration. Which would you rather work with?

Data frame (df)

No	Student.Name	Subject	Grade
1	Anna	Math	86
2	John	Math	43
3	Catherine	Math	80
4	Anna	English	90
5	John	English	75
6	Catherine	English	82

Variable 1 Variable 2 Variable 3

R codes to extract variables

```
df$Student.Name
df$Subject
df$Grade
```

Student.Name	Math	English
Anna	86	90
John	43	
Catherine	80	2

Variable 1 Variable 2 Variable 3

```
df[[1]]
names(df)
c(df[2,2],df[3,2],df[2,3],df[3,3])
```

Class Activity: Tidy vs. Untidy?

- Visit https://b.socrative.com/login/student/
- Room Name: MATH2349

- Enter a nickname
- Work individually or in groups
- Have a look at the data frames provided in each question.
- Decide whether the given data is Tidy or not.

Q1. 50 states of the United States of America

- The following data frame includes some characteristics of 50 states of the United States of America for 1977.

```
##             Population Income Illiteracy Life Exp Murder HS Grad Frost
## Alabama           3615   3624        2.1    69.05   15.1    41.3    20
## Alaska             365   6315        1.5    69.31   11.3    66.7   152
## Arizona           2212   4530        1.8    70.55    7.8    58.1    15
## Arkansas          2110   3378        1.9    70.66   10.1    39.9    65
## California       21198   5114        1.1    71.71   10.3    62.6    20
## Colorado          2541   4884        0.7    72.06    6.8    63.9   166
## Connecticut       3100   5348        1.1    72.48    3.1    56.0   139
## Delaware           579   4809        0.9    70.06    6.2    54.6   103
```

- Is this Tidy?

Q2. Province population in Canada

- The following data frame includes province population in different provinces of Canada.

```
##    source  pops66  pops71
## 1     PEI  108535  111641
## 2      NS  756039  788960
## 3      NB  616788  534557
## 4     QUE 5780845 6027764
## 5     ONT 6960870 7703106
## 6     MAN  963066  988247
## 7    SASK  955344  926242
## 8    ALTA 1463203 1627874
## 9      BC 1873674 2184621
## 10   NFLD  493396  522104
```

- Is this data Tidy?

Q3. WHO TB rates

- The following data frame includes Tuberculosis rates of different countries:

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

- Is this data Tidy?

Long vs. wide format data

- A single data set can be rearranged in many different ways.
- One of the ways is called " ". In this layout, the data set is arranged in such a way that a single subject's information is stored in multiple rows.
- In the , a single subject's information is stored in multiple columns.
- The main difference between a wide layout and a long layout is that the wide layout contains all the measured information in different columns.

Wide format

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

=

Long format

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

The `tidyr` package

- `tidyr` is a one such package which was built for the sole purpose of simplifying the process of creating tidy data.

- There are four fundamental functions of data tidying that `tidyr` provides:

- **`gather()`** makes "wide" data longer
- **`spread()`** makes "long" data wider
- **`separate()`** splits a single column into multiple columns
- **`unite()`** combines multiple columns into a single column

%>% Operator

- Although not required, the `tidyr` package makes use of the pipe operator `%>%` developed by Stefan Milton Bache in the R package magrittr.
- `%>%` moves or "pipes" the result forward into the next function call/expression.

$$f(x) \text{ is the same as } x \text{ \%>\% } f()$$

- For instance, regular code chunks work from inside out like this:

```
finally_last_step(
  and_then_third(
    then_second(
      do_first(data)
    )
  )
)
```

- Piping uses intuitive ordering:

```
data %>%
  do_first() %>% then_second() %
```

`gather()` function

Reshaping wide format to long format

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

gather()

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

`gather()` function arguments

```
Function:       gather(data, key, value, ..., na.rm = FALSE, convert
Same as:        data %>% gather(key, value, ..., na.rm = FALSE, conve

Arguments:
        data:           data frame
        key:            column name representing new variable
        value:          column name representing variable values
        ...:            names of columns to gather (or not gather)
        na.rm:          option to remove observations with missing va
        convert:        if TRUE will automatically convert values to
```

`gather()` function

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

```
cases %>% gather(Year, n, 2:4)
```

dataframe to reshape

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

`gather()` function

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

```
cases %>% gather(Year, n, 2:4)
```

name of the new "key" column

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

`gather()` function

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

```
cases %>% gather(Year, n, 2:4)
```

name of the new "value" column

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

`gather()` function

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

key

value

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

```
cases %>% gather(Year, n, 2:4)
```

columns to collapse

Example `gather()`

We'll start with the following data set:

```
##    Group Year Qtr.1 Qtr.2 Qtr.3 Qtr.4
## 1      1 2006    15    16    19    17
## 2      1 2007    12    13    27    23
## 3      1 2008    22    22    24    20
## 4      1 2009    10    14    20    16
## 5      2 2006    12    13    25    18
## 6      2 2007    16    14    21    19
## 7      2 2008    13    11    29    15
## 8      2 2009    23    20    26    20
## 9      3 2006    11    12    22    16
## 10     3 2007    13    11    27    21
## 11     3 2008    17    12    23    19
## 12     3 2009    14     9    31    24
```

- Wide or long format data ?

Example `gather()`

We'll start with the following data set:

```
##    Group Year Qtr.1 Qtr.2 Qtr.3 Qtr.4
## 1      1 2006    15    16    19    17
## 2      1 2007    12    13    27    23
## 3      1 2008    22    22    24    20
## 4      1 2009    10    14    20    16
## 5      2 2006    12    13    25    18
## 6      2 2007    16    14    21    19
## 7      2 2008    13    11    29    15
## 8      2 2009    23    20    26    20
## 9      3 2006    11    12    22    16
## 10     3 2007    13    11    27    21
## 11     3 2008    17    12    23    19
## 12     3 2009    14     9    31    24
```

- Wide or long format data ?
- This data is considered wide since the *time* variable (represented as quarters) is structured such that each quarter represents a variable.

Example `gather()` Cont.

- We need to *gather* each quarter within one column variable and also *gather* the values associated with each quarter in a second column variable.

```
long_DF <- DF %>% gather(Quarter, Revenue, Qtr.1:Qtr.4)
head(long_DF, 24)  # note, for brevity, I only show the data for the


##    Group Year Quarter Revenue
## 1      1 2006   Qtr.1      15
## 2      1 2007   Qtr.1      12
## 3      1 2008   Qtr.1      22
## 4      1 2009   Qtr.1      10
## 5      2 2006   Qtr.1      12
## 6      2 2007   Qtr.1      16
## 7      2 2008   Qtr.1      13
## 8      2 2009   Qtr.1      23
## 9      3 2006   Qtr.1      11
## 10     3 2007   Qtr.1      13
## ..   ...  ...     ...     ...
```

Example `gather()` Cont.

```
##    Group Year Qtr.1 Qtr.2 Qtr.3 Qtr.4
## 1      1 2006    15    16    19    17
## 2      1 2007    12    13    27    23
## 3      1 2008    22    22    24    20
...
...
```

- Note that all of these produce the same result:

```
DF %>% gather(Quarter, Revenue, Qtr.1:Qtr.4)
DF %>% gather(Quarter, Revenue, -Group, -Year)
DF %>% gather(Quarter, Revenue, 3:6)
DF %>% gather(Quarter, Revenue, Qtr.1, Qtr.2, Qtr.3, Qtr.4)
```

`spread()` function

Reshaping long format to wide format.

- This function is a complement to `gather()`.

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16
Beijing	121	56

`spread()` function arguments

- There are times when we are required to turn long formatted data into wide formatted data.
- When multiple variables are stored in rows, the `spread()` function generates columns from rows. This function spreads a key-value pair across multiple columns.

```
Function:       spread(data, key, value, fill = NA, convert = FALSE)
Same as:        data %>% spread(key, value, fill = NA, convert = FALS

Arguments:
        data:           data frame
        key:            column values to convert to multiple columns
        value:          single column values to convert to multiple c
        fill:           If there isn't a value for every combination
                        column, this value will be substituted
        convert:        if TRUE will automatically convert values to
                        factor as appropriate
```

`spread()` function

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

```
cases %>% spread(Year, n)
```

dataframe to reshape

`spread()` function

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

```
cases %>% spread(Year, n)
```

column to use as keys
(new column names)

`spread()` function

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

```
cases %>% spread(Year, n)
```

column to use as values
(new column cells)

Example `spread()`

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

- Suppose you want to calculate tuberculosis rate from (rate = cases/population).
- To achieve this, `cases` and `population` needs to be separately given in columns.

Example `spread()` Cont.

```
## # A tibble: 5 x 4
##   country      year type          count
##   <chr>       <int> <chr>         <int>
## 1 Afghanistan  1999 cases           745
## 2 Afghanistan  1999 population 19987071
## 3 Afghanistan  2000 cases          2666
## 4 Afghanistan  2000 population 20595360
## 5 Brazil       1999 cases         37737
```

```
spread(table2, key = type, value = count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

`separate( )` function

Splitting a single variable into two

Many times multiple variables are stored in one column and you want to split them according to a separator character.

```
Function:       separate(data, col, into, sep = " ", remove = TRUE, c
Same as:        data %>% separate(col, into, sep = " ", remove = TRUE

Arguments:
        data:           data frame
        col:            column name representing current variable
        into:           names of variables representing new variables
        sep:            how to separate current variable (char, num,
        remove:         if TRUE, remove input column from output data
        convert:        if TRUE will automatically convert values to
                        factor as appropriate
```

`separate( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% separate(date, c("year", "month", "day"), sep = "-")
```

dataframe to reshape

`separate( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% separate(date, c("year", "month", "day"), sep = "-")
```

column to split into multiple columns

`separate( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% separate(date, c("year", "month", "day"), sep = "-")
```

names of the new variable columns

`separate( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% separate(date, c("year", "month", "day"), sep = "-")
```

how to separate current variable

Example `seperate()` function

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

- The rate column contains both cases and population variables, and we need to split it into two variables.

```
table3 %>%
separate(rate, into = c("cases", "population"), sep = "/")
```

```
## # A tibble: 6 x 4
##   country      year cases  population
## * <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
```

`unite( )` function

Merging two variables into one

There may be a time in which we would like to combine the values of two variables. The `unite()` function combine multiple columns into a single column.

```
Function:       unite(data, col, ..., sep = " ", remove = TRUE)
Same as:        data %>% unite(col, ..., sep = " ", remove = TRUE)

Arguments:
        data:           data frame
        col:            column name of new "merged" column
        ...:            names of columns to merge
        sep:            separator to use between merged values
        remove:         if TRUE, remove input column from output data
```

`unite( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% unite(date, year, month, day, sep = "-")
```

dataframe to reshape

`unite( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% unite(date, year, month, day, sep = "-")
```

name of new "merged" column

`unite( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% unite(date, year, month, day, sep = "-")
```

columns to merge

`unite( )` function

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

```
storms %>% unite(date, year, month, day, sep = "-")
```

separator to use btwn merged values

Example `unite()` function

```
## # A tibble: 5 x 4
##   country     century year  rate
##   <chr>       <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
## 3 Brazil      19      99    37737/172006362
## 4 Brazil      20      00    80488/174504898
## 5 China       19      99    212258/1272915272
```

- Assume that we want to combine the `century` and `year` variables into one variable called `new_year`.

```
table5 %>%
unite(new_year, century, year, sep="")
```

```
## # A tibble: 6 x 3
##   country     new_year rate
##   <chr>       <chr>    <chr>
## 1 Afghanistan 1999     745/19987071
## 2 Afghanistan 2000     2666/20595360
## 3 Brazil      1999     37737/172006362
```

What do you need to know by Week 4

- Distinguish tidy vs untidy data sets.
- Distinguish wide vs long format data.
- Understand tidy data principles.
- Use `tidyr` package functions
- Practice!

Class Worksheet

- Working in small groups, complete the following class worksheet

Week 4 Class Worksheet

- Once completed, feel free to work on your Assignment and/or Skill Builders