An alternative style for ggplots:

```
ggplot(data=mydata) +
   aes(x=v1, y=v2) +
   geom_point()
```

Three lines - three steps - three layers:

1. the data

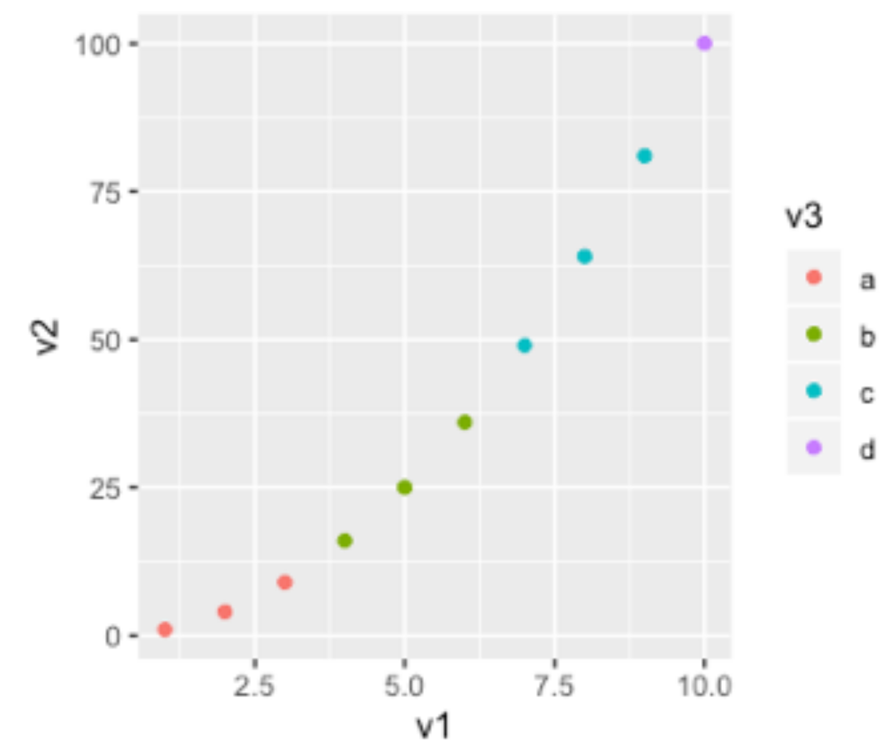
2. the aesthetic mapping of the variables to dimensions of the plot
(what goes on x axis? do I specify the y axis?...)

3. the shape (“geom”) type of plot

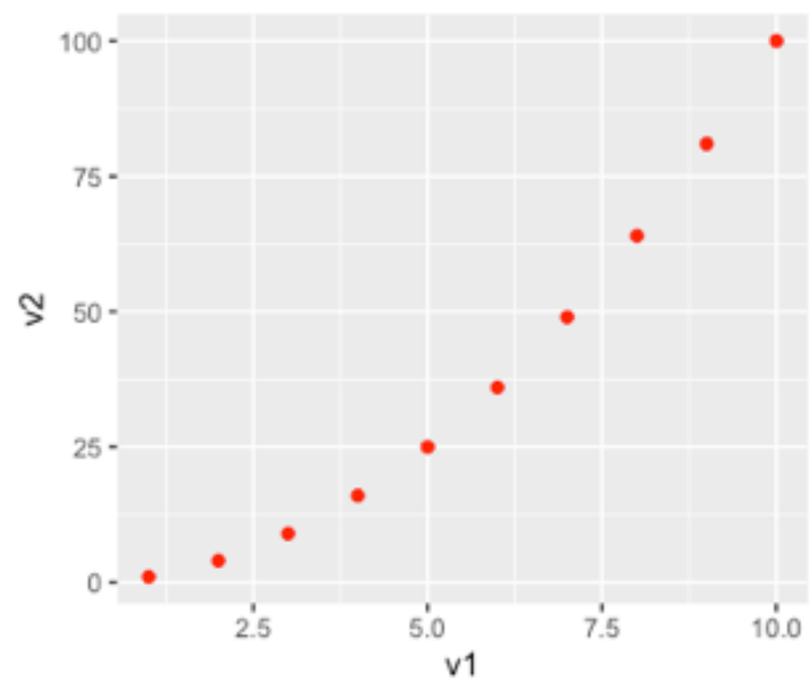
Credit: style from a class taught by Mine Dogcu at UCI.

aes: maps data variables to plot dimensions

```
ggplot(data=mydata) +
   aes(x = v1, y = v2, color = v3)+
   geom_point()
```



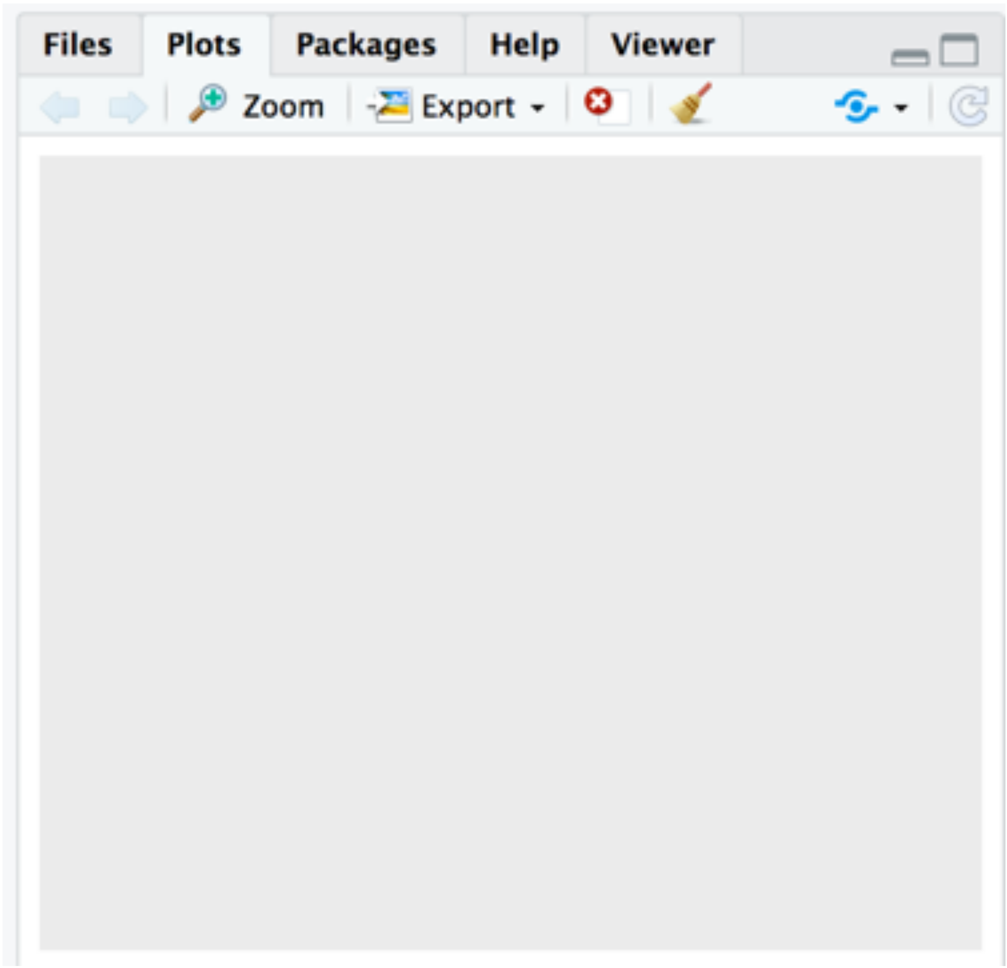

```
ggplot(data=mydata) +
   aes(x = v1, y = v2) +
   geom_point(color = “red”)
```



outside an aes, the color does not change with the value of any variable

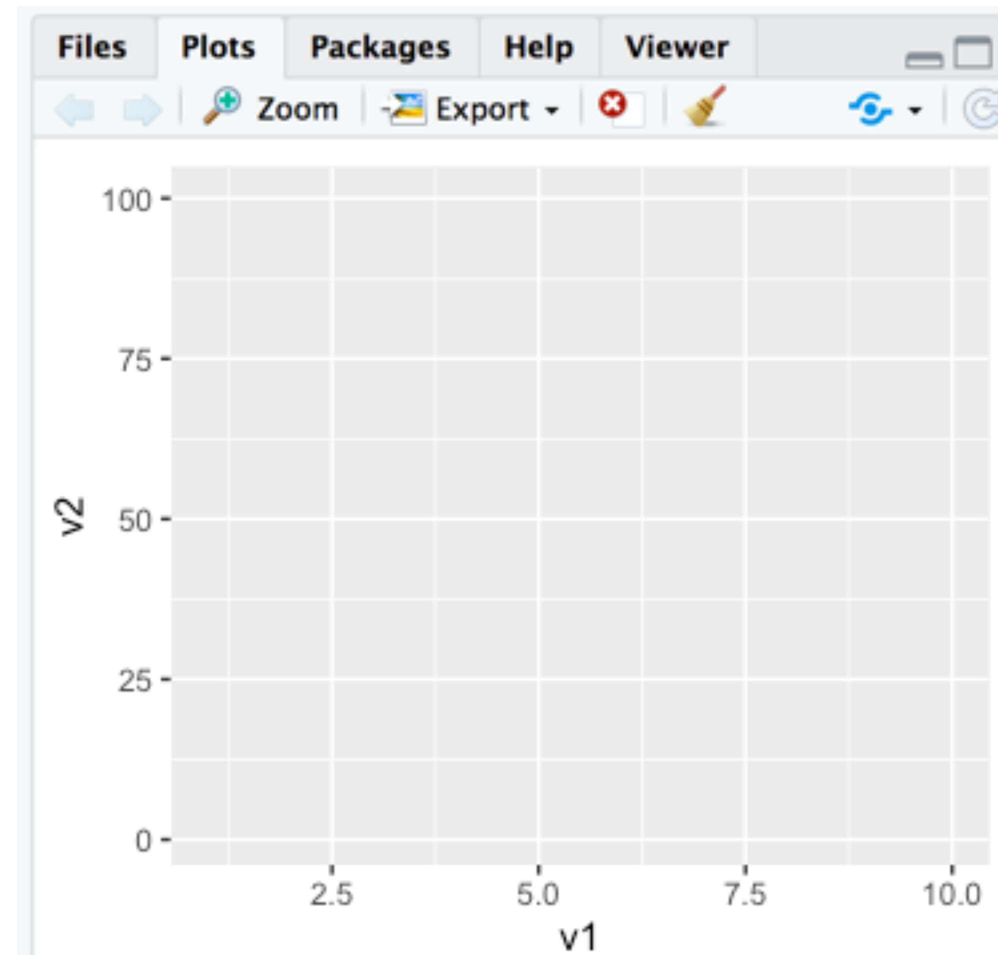
each line corresponds to a plot layer:

```
ggplot(data=mydata)
```



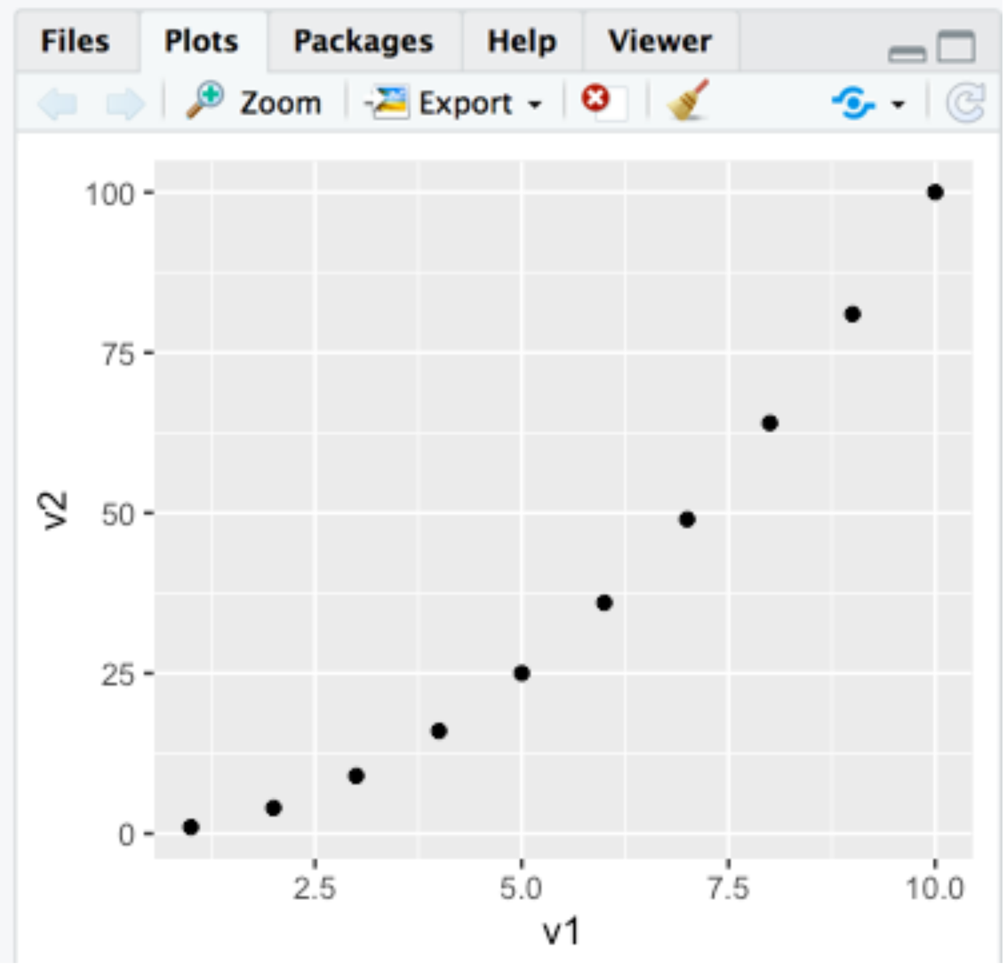
each line corresponds to a plot layer:

```
ggplot(data=mydata) +
    aes(x=v1, y=v2)
```



each line corresponds to a plot layer:

```
ggplot(data=mydata) +
    aes(x=v1, y=v2) +
    geom_point()
```



Wrangle (part 1)

Ch Online/Book

Ch 9/NA: Intro

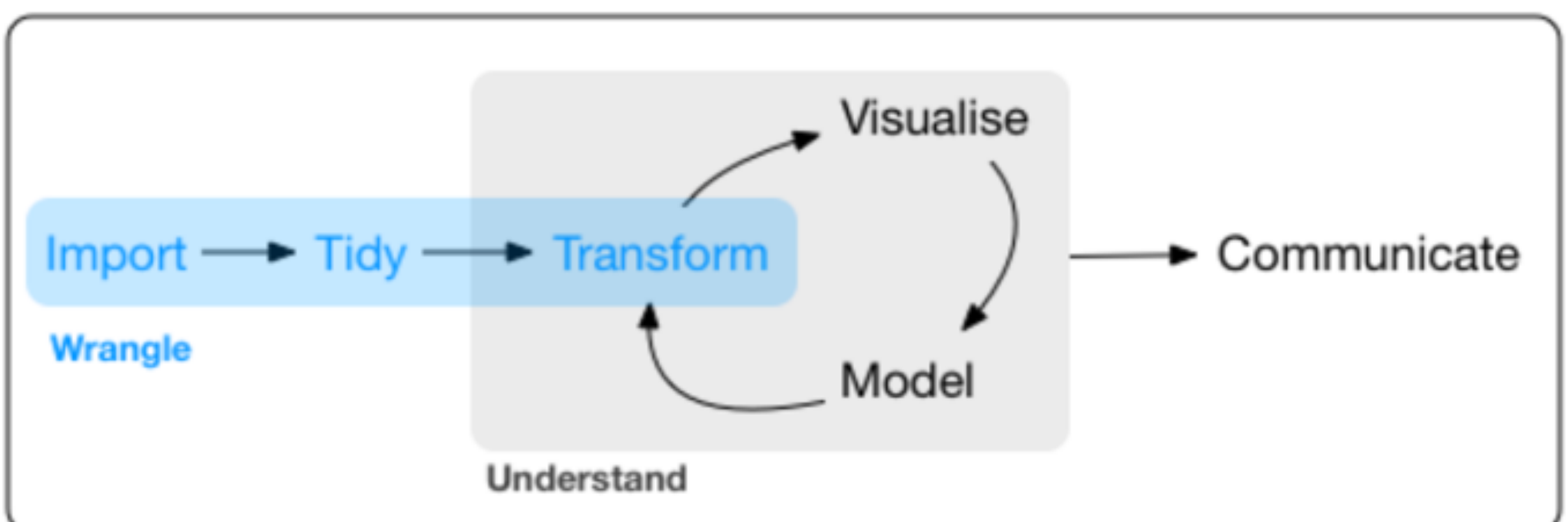
Ch 10/7: Tibbles

Ch 11/8: Data import

Ch 12/9: Tidy data

Linda Palmer

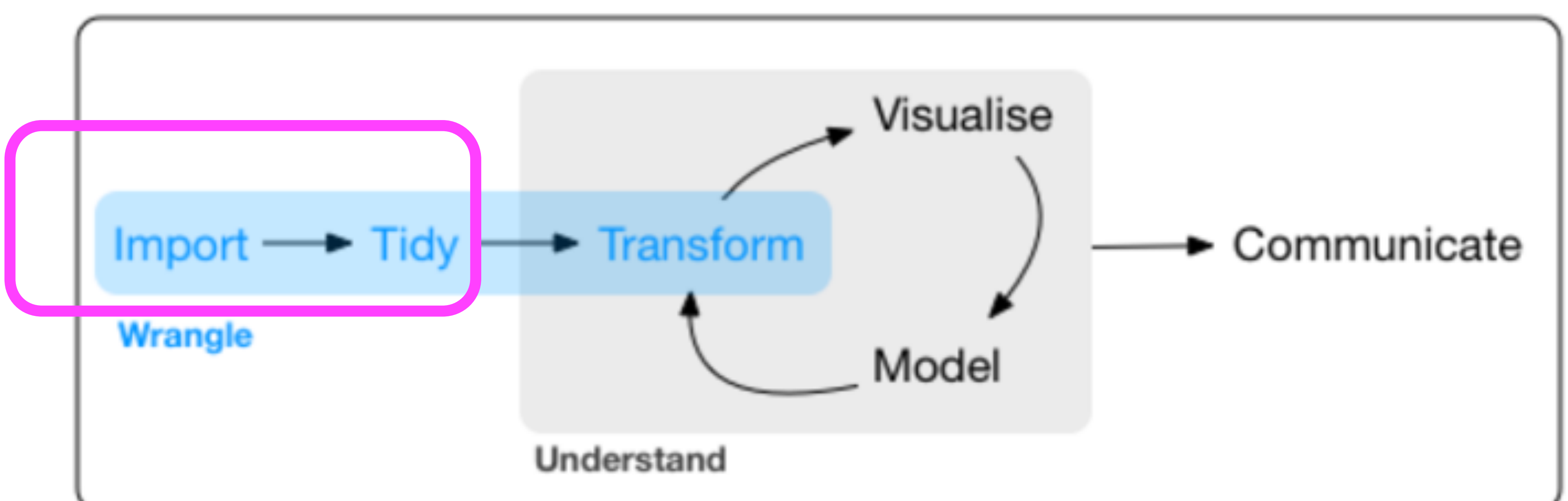
OC R Users Group Bookclub: R for Data Science Feb10, 2021



Wrangle (part 1)

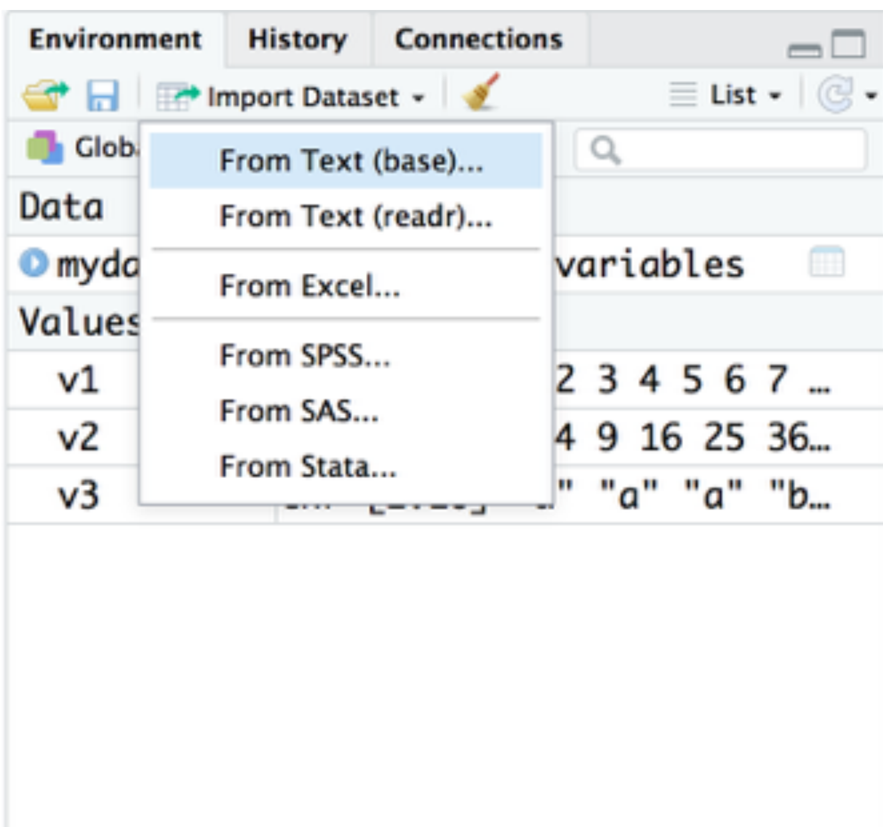
Ch Online/Book
Ch 9/NA: Intro
Ch 10/7: Tibbles
Ch 11/8: Data import
Ch 12/9: Tidy data

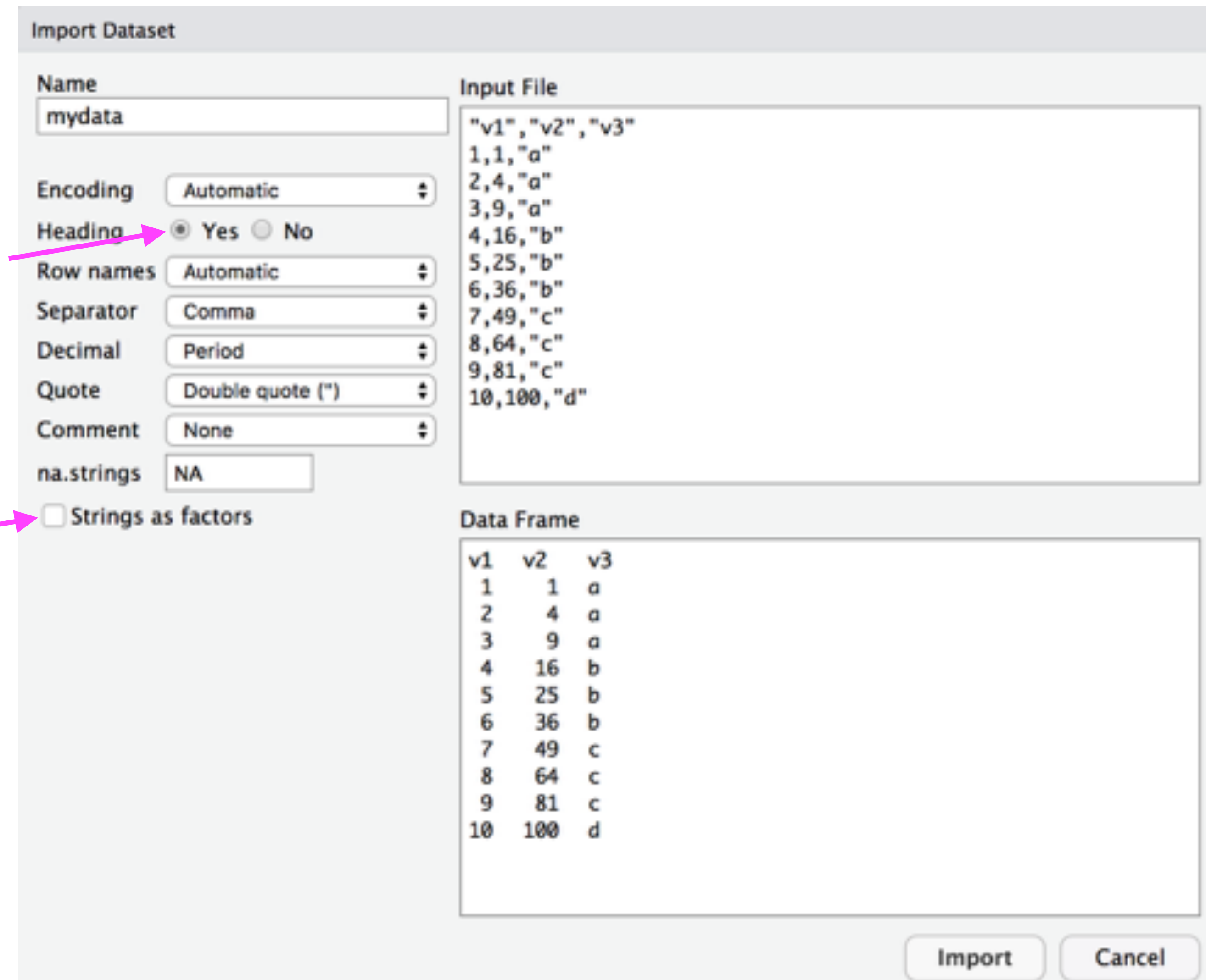
Linda Palmer
OC R Users Group Bookclub: R for Data Science   Feb10, 2021



Ch 7/10: Tibbles

**Confession: I usually use base R data.frame, instead of tibbles,
and base R data import
…. but I *always* set stringsAsFactors = FALSE**






```
mydata <- read.csv("data/mydata.csv", stringsAsFactors=FALSE)

write.csv(mydata, file = "data/mydata.csv", row.names=FALSE)
```

```
head(mydata)
tail(mydata)
glimpse(mydata) #tidyverse
summary(mydata)
```

```
unique(mydata$v3)
as.factor(mydata$v3)
levels(as.factor(mydata$v3))
```

```
> head(mydata)
  v1 v2 v3
1  1  1  a
2  2  4  a
3  3  9  a
4  4 16  b
5  5 25  b
6  6 36  b
> tail(mydata)
   v1  v2 v3
5   5  25  b
6   6  36  b
7   7  49  c
8   8  64  c
9   9  81  c
10 10 100  d
```

```
> summary(mydata)
       v1              v2              v3
 Min.   : 1.00   Min.   :  1.00   Length:10
 1st Qu.: 3.25   1st Qu.: 10.75   Class :character
 Median : 5.50   Median : 30.50   Mode  :character
 Mean   : 5.50   Mean   : 38.50
 3rd Qu.: 7.75   3rd Qu.: 60.25
 Max.   :10.00   Max.   :100.00
> unique(mydata$v3)
[1] "a" "b" "c" "d"
> as.factor(mydata$v3)
 [1] a a a b b b c c c d
Levels: a b c d
> levels(as.factor(mydata$v3))
[1] "a" "b" "c" "d"
```

```
> glimpse(mydata) #tidyverse
Observations: 10
Variables: 3
$ v1 <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
$ v2 <int> 1, 4, 9, 16, 25, 36, 49, 64, 81, 100
$ v3 <chr> "a", "a", "a", "b", "b", "b", "c", "c", "c", "d"
```

Ch 8/10: Tibbles

So: be convinced to use tibbles instead?

```
tdata <- as_tibble(mydata)
```

```
> tdata
# A tibble: 10 x 3
      v1    v2 v3
   <int> <int> <chr>
 1     1     1 a
 2     2     4 a
 3     3     9 a
 4     4    16 b
 5     5    25 b
 6     6    36 b
 7     7    49 c
 8     8    64 c
 9     9    81 c
10    10   100 d
```

- don't have to set "stringsAsFactors=F" √
- can use nonpermissible variable names
- no partial matching!! √

10.3.2 Subsetting tibbles

tibbles: `tdata <- as_tibble(mydata)`

```
> # Subsetting
> tdata$v2
 [1]   1   4   9  16  25  36  49  64  81 100
> tdata[["v2"]]
 [1]   1   4   9  16  25  36  49  64  81 100
> tdata[[2]]
 [1]   1   4   9  16  25  36  49  64  81 100
```

- by name, with $v2 or [["v2"]]
- by position: [[2]]

- What about [ ] ?

```
> tdata[3]
# A tibble: 10 x 1
   v3
   <chr>
 1 a
 2 a
 3 a
 4 b
 5 b
 6 b
 7 c
 8 c
 9 c
10 d
```

```
> tdata["v2"]
# A tibble: 10 x 1
      v2
   <int>
 1     1
 2     4
 3     9
 4    16
 5    25
 6    36
 7    49
 8    64
 9    81
10   100
```

```
> tdata$v3[4:7]
[1] "b" "b" "b" "c"
```

To **pipe** data to subset: use .

```
> tdata %>% .[[3]]
 [1] "a" "a" "a" "b" "b" "b" "c" "c" "c" "d"
> tdata %>% .$v3
 [1] "a" "a" "a" "b" "b" "b" "c" "c" "c" "d"
> tdata %>% .$v2
 [1]   1   4   9  16  25  36  49  64  81 100
```

Turning a tibble into a data frame

Some older functions don't work with tibbles. If you encounter one of these functions, use `as.data.frame()` to turn a tibble back to a `data.frame`:

```
class(as.data.frame(tb))
#> [1] "data.frame"
```

Copy

```
# Convert a tibble to a data.frame:
dfdata <- as.data.frame(tdata)
```

```
> class(dfdata)
[1] "data.frame"
```

10.5 Exercises: Tibbles

1. How can you tell if an object is a tibble?

```
> class(mtcars)
[1] "data.frame"
> class(tdata)
[1] "tbl_df"     "tbl"        "data.frame"
```

2. Compare and contrast the following operations on a `data.frame` and equivalent tibble. What is different? Why might the default data frame behaviours cause you frustration?

```
df <- data.frame(abc = 1, xyz = "a")
df$x
df[, "xyz"]
df[, c("abc", "xyz")]
```

```
> df <- data.frame(abc = 1, xyz = "a")
> df
  abc xyz
1   1   a
> df$x
[1] a
Levels: a
> df[, "xyz"]
[1] a
Levels: a
> df[, c("abc", "xyz")]
  abc xyz
1   1   a
```

2. Compare and contrast the following operations on a `data.frame` and equivalent tibble. What is different? Why might the default data frame behaviours cause you frustration?

```
> df <- data.frame(cat = 3:5, dog=letters[10:12])
> df
  cat dog
1   3   j
2   4   k
3   5   l
```

Bad default behavior 1:

```
> df$b
NULL
> df$d
[1] j k l
Levels: j k l
```

```
> df$dog
[1] j k l
Levels: j k l
```

default behaviors 2 (and 3):

```
> df[, "cat"]
[1] 3 4 5
> df[, "dog"]
[1] j k l
Levels: j k l
> df[, c("cat", "dog")]
  cat dog
1   3   j
2   4   k
3   5   l
```

```
> class( df[, "cat"] )
[1] "integer"
> class( df[, c("cat", "dog")] )
[1] "data.frame"
> class( df[, "dog"] )
[1] "factor"
```

2. Compare and contrast the following operations on a `data.frame` and equivalent tibble. What is different? Why might the default data frame behaviours cause you frustration?

```
> df <- data.frame(cat = 3:5, dog=letters[10:12])
> df
  cat dog
1   3   j
2   4   k
3   5   l
```

Bad default behavior 1:

```
> df$b
NULL
> df$d
[1] j k l
Levels: j k l
Warning message:
In df$d : partial match of 'd' to 'dog'
> df$dog
[1] j k l
Levels: j k l
```

```
options(
  warnPartialMatchAttr = TRUE,
  warnPartialMatchDollar = TRUE,
  warnPartialMatchArgs = TRUE
)
```

default behaviors 2 (and 3):

```
> df[, "cat"]
[1] 3 4 5
> df[, "dog"]
[1] j k l
Levels: j k l
> df[, c("cat", "dog")]
  cat dog
1   3   j
2   4   k
3   5   l
```

```
> class( df[, "cat"] )
[1] "integer"
> class( df[, c("cat", "dog")] )
[1] "data.frame"
> class( df[, "dog"] )
[1] "factor"
```

Exercises: con't

3. If you have the name of a variable stored in an object, e.g. `var <- "mpg"`, how can you extract the reference variable from a tibble?

```
myvar <- "mpg" # the string "mpg" stored in myvar
mtcars$mpg
mtcars[, "mpg"]
mtcars[, myvar]
mtcars[[myvar]]
```

4. ~~Practice referring to non-syntactic names in the following data frame by:~~

5. What does `tibble::enframe()` do? When might you use it?

```
> # converts a named vector to a tibble
> groceries <- c(4, 2, 1, 6)
> names(groceries) <- c('bananas', 'apples', 'bread', 'eggs')
> groceries
bananas  apples   bread    eggs
      4       2       1       6
> enframe(groceries)
# A tibble: 4 x 2
  name    value
  <chr>   <dbl>
1 bananas     4
2 apples      2
3 bread       1
4 eggs        6
```

Ch Online/Book
Ch 9/NA: Intro
Ch 10/7:   Tibbles
Ch 11/8:   Data import
Ch 12/9:   Tidy data

https://jrnold.github.io/r4ds-exercise-solutions/tibbles.html

Ch Online/Book

Ch 9/NA: Intro

Ch 10/7: Tibbles

Ch 11/8: Data import

Ch 12/9: Tidy data

Ch 9/12: Tidy data

<u>Tidy data has:</u>

One row per observation
One variable per column
One value per cell

```
> table1
# 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
```

- It's a matter of judgment to figure out what are the appropriate "observations" and the corresponding variables (some factors or ID's, some measurements) for a given set of data.

- Not all data is best expressed as tidy -- but it's often best! For some counterexamples: <u>http://simplystatistics.org/2016/02/17/non-tidy-data/</u>

examples on tidy data: <u>mutate</u>, <u>count</u>, viz over time

```r
# Compute rate per 10,000:
table1 %>%
  mutate(rate = cases / population * 10000)
# What are the parameters to mutate function?
?mutate
# Not using pipe:
mutate(.data = table1, rate = cases / population * 10000)
```

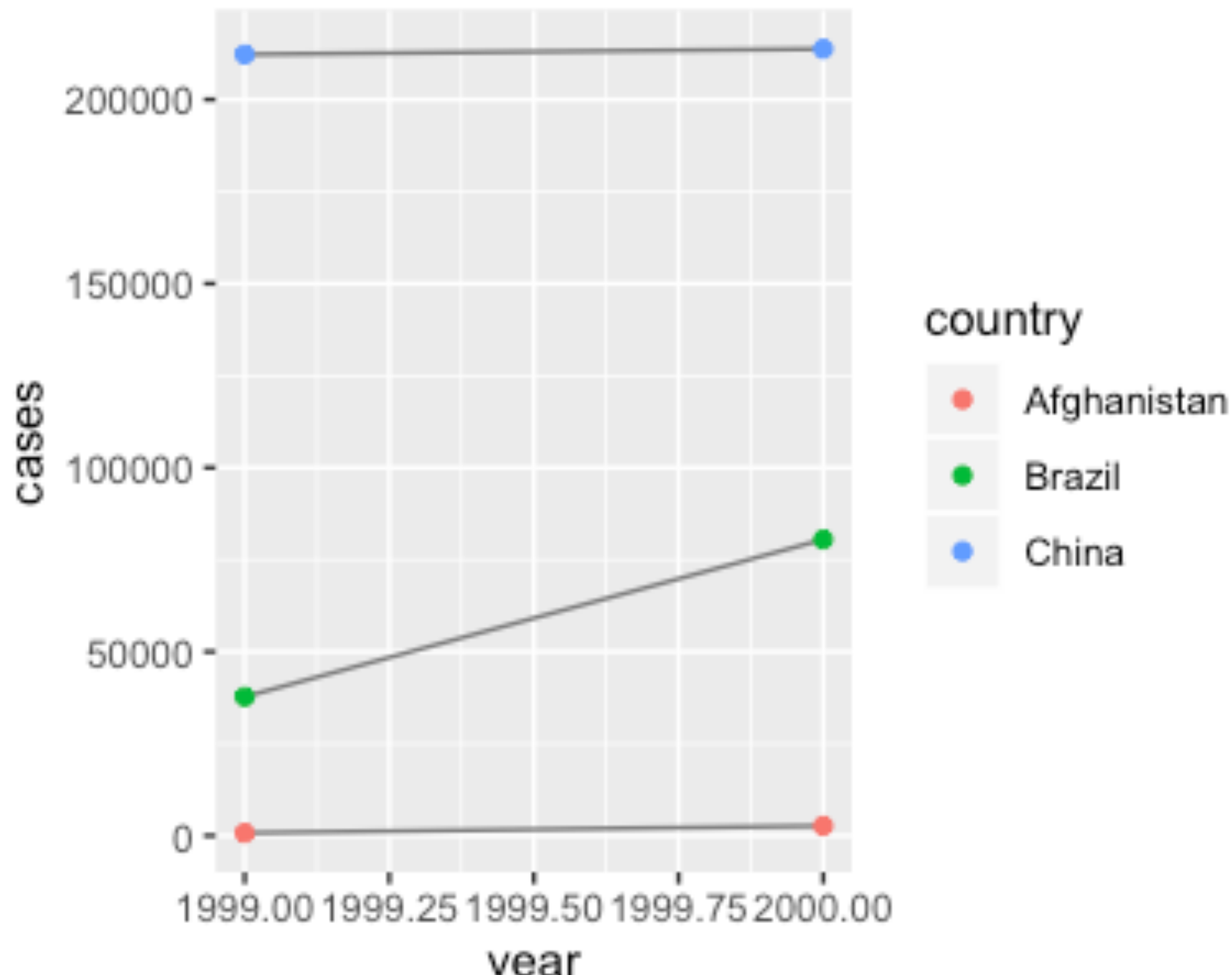
```
> mutate(.data=table1, rate = cases / population * 10000)
# A tibble: 6 x 5
  country      year  cases population  rate
  <chr>       <int>  <int>      <int> <dbl>
1 Afghanistan  1999    745   19987071 0.373
2 Afghanistan  2000   2666   20595360 1.29
3 Brazil       1999  37737  172006362 2.19
4 Brazil       2000  80488  174504898 4.61
5 China        1999 212258 1272915272 1.67
6 China        2000 213766 1280428583 1.67
```

```r
table1 %>% count(year) # Counts the number of data rows per year
# For each unique value of "year" column, calculates sum of "cases" column:
table1 %>% count(year, wt=cases) # Note that the return is unlabeled: it says "n"
count(x=table1, year, wt=cases) # there's no argument name for where "year" goes
?count
```

```
> # For each unique value of "year" column, calculates sum of "cases" column:
> table1 %>% count(year, wt=cases) # Note that the return is unlabeled: it says "n"
# A tibble: 2 x 2
   year      n
  <int>  <int>
1  1999 250740
2  2000 296920
```

examples on tidy data: mutate, count, viz over time

```
ggplot(table1, aes(year, cases)) +
  geom_line(aes(group = country), colour = "grey50") +
  geom_point(aes(colour = country))
# Same plot with alternative style coding:
ggplot(data=table1) +                                    # plot table1
  aes(x=year, y=cases) +                                 # with x~y: year ~ cases
  geom_line( aes(group = country), color = "gray50" ) +  # line per country
  geom_point( aes(color = country) )                     # point per country
```



Big change

Online Ch 12

vs chapter formerly known as Ch 9 (physical book):

pivot_longer(), pivot_wider()

replace

gather(), spread()

Pivot_longer

```
> table4a
# A tibble: 3 x 3
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Here, the two column names 1999 and 2000 would be better expressed as values of a categorical variable (or levels of a factor).
So, I'm going to take the two column names and move them into rows, as appropriate;
this makes the dataframe get less wide and more tall (aka long).
Result has fewer columns and more rows: **pivot_longer.**

```
> pivot_longer( data=table4a, cols=2:3)
# A tibble: 6 x 3
  country     name   value
  <chr>       <chr>  <int>
1 Afghanistan 1999     745
2 Afghanistan 2000    2666
3 Brazil      1999   37737
4 Brazil      2000   80488
5 China       1999  212258
6 China       2000  213766
> # same as:
> table4a %>% pivot_longer(cols=2:3)
> table4a %>%
+   pivot_longer(cols=c(`1999`, `2000`))
```

default new column names chosen by pivot_longer:
"name" , "value".
Let's specify nice names for these instead:

```
> pivot_longer(data = table4a, cols = 2:3,
+              names_to = "Year", values_to = "TB_cases")
# A tibble: 6 x 3
  country     Year  TB_cases
  <chr>       <chr>    <int>
1 Afghanistan 1999       745
2 Afghanistan 2000      2666
3 Brazil      1999     37737
4 Brazil      2000     80488
5 China       1999    212258
6 China       2000    213766
```

Pivot_wider

```
> table2
# 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
```

```
> pivot_wider(data=table2, id=c(country, year),
+             names_from=type, values_from = count ) # here we will lose these 2 colnames
# 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
```

12.3.3 Exercises

Separating

```
> table3
# 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
```

```
separate(data=table3, col=rate, into=c("TBcases", "population"), sep = "/")
```

```
> 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
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

```
> t5 <- separate(data=table3, col=year, into=c("century", "year"), sep = 2)
> table5
# A tibble: 6 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
6 China       20      00    213766/1280428583
```

and Uniting

```
> table5 %>%
+   unite(new, century, year)
# A tibble: 6 x 3
  country     new   rate
  <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
6 China       20_00 213766/1280428583
```

12.6 Case Study