Netebook4

Dataset Link

```
"https://vitacin-my.sharepoint.com/:f:/g/personal/
sunilkumar_vijay_vit_ac_in/EjhRGv25OJRNgiczvAUSeAMBWXjXcmqZklirHeXlosjqKg?e=hLnBfO"
```

```
[1] "https://vitacin-my.sharepoint.com/:f:/g/personal/\nsunilkumar_vijay_vit_ac_in/EjhRGv25O
```

Q.1 Read file (input.csv) and

Q.2 assign column names.

Q.3 Print rows from 10 to 30.

Use Automobile Dataset for following questions

Q.4 Find the most expensive car price and company name.

Q.5 Find the most expensive car for each company.

Q.6 Print all Toyota cars details.

Q.7 Find the count of "convertible" type cars in "alfa-romero" company.

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
#library(tidyr)
```

Loop Functions

R has some functions which implement looping in a compact form to make your life easier.

- ***lapply*()**: Loop over a list and evaluate a function on each element
- ***sapply*()**: Same as lapply but try to simplify the result
- ***apply*()**: Apply a function over the margins of an array
- ***tapply*()**: Apply a function over subsets of a vector
- ***mapply*()**: Multivariate version of lapply

An auxiliary function ***split*** is also useful, particularly in conjunction with lapply.

lapply()

The lapply() function does the following simple series of operations:

1. it loops over a list, iterating over each element in that list
2. it applies a *function* to each element of the list (a function that you specify)
3. and returns a list (the l is for "list").

This function takes three arguments: (1) a list X; (2) a function (or the name of a function) FUN; (3) other arguments.

```r
x <- list(a = 1:5, b = rnorm(10))
x
```

```
$a
[1] 1 2 3 4 5

$b
 [1] -1.0560608 -0.4446010  1.6703738  0.2765620 -0.7538311  0.2642991
 [7] -1.1338471 -0.8010829  0.9693896 -0.2444197
```

```r
lapply(x, mean)
```

```
$a
[1] 3

$b
[1] -0.1253218
```

```r
#lapply(df, max)
```

sapply()

The sapply() function behaves similarly to lapply(); the only real difference is in the return value.

sapply() will try to simplify the result of lapply() if possible.

- If the result is a list where every element is length 1, then a vector is returned
- If the result is a list where every element is a vector of the same length (> 1), a matrix is returned.
- If it can't figure things out, a list is returned.

```r
sapply(x, mean)
```

```
         a          b 
 3.0000000 -0.1253218 
```

split()

The split() function takes a vector or other objects and splits it into groups determined by a factor or list of factors.

```r
str(split)
```

```
function (x, f, drop = FALSE, ...)  
```

```r
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
x
```

```
 [1] -1.96292359 -0.08511854 -1.53981598  1.03018974 -0.26397489  0.14368535
 [7]  0.44365041 -0.34343514 -1.03818038  0.34228892  0.81579083  0.35919275
[13]  0.70722563  0.06474938  0.91324137  0.69567741  0.71254099  0.88693864
[19]  0.95033425  0.74375169  2.28643250  1.18061885  1.24682941  1.27486231
[25]  1.56617893  0.91923936  0.02751513 -0.67855577  1.64288675  1.50917944
```

```r
f
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
Levels: 1 2 3
```

```r
split(x, f)
```

```
$`1`
 [1] -1.96292359 -0.08511854 -1.53981598  1.03018974 -0.26397489  0.14368535
 [7]  0.44365041 -0.34343514 -1.03818038  0.34228892

$`2`
 [1] 0.81579083 0.35919275 0.70722563 0.06474938 0.91324137 0.69567741
 [7] 0.71254099 0.88693864 0.95033425 0.74375169

$`3`
 [1]  2.28643250  1.18061885  1.24682941  1.27486231  1.56617893  0.91923936
 [7]  0.02751513 -0.67855577  1.64288675  1.50917944
```

```r
lapply(split(x, f), mean)
```

```
$`1`
[1] -0.3273634

$`2`
[1] 0.6849443

$`3`
[1] 1.097519
```

```r
airquality$Month
```

```
  [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6
 [38] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7
 [75] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
[112] 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
[149] 9 9 9 9 9
```

```r
s <- split(airquality, airquality$Month)
str(s)
```

```
List of 5
 $ 5:'data.frame':  31 obs. of  6 variables:
  ..$ Ozone  : int [1:31] 41 36 12 18 NA 28 23 19 8 NA ...
  ..$ Solar.R: int [1:31] 190 118 149 313 NA NA 299 99 19 194 ...
  ..$ Wind   : num [1:31] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  ..$ Temp   : int [1:31] 67 72 74 62 56 66 65 59 61 69 ...
  ..$ Month  : int [1:31] 5 5 5 5 5 5 5 5 5 5 ...
  ..$ Day    : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...
 $ 6:'data.frame':  30 obs. of  6 variables:
  ..$ Ozone  : int [1:30] NA NA NA NA NA NA 29 NA 71 39 ...
  ..$ Solar.R: int [1:30] 286 287 242 186 220 264 127 273 291 323 ...
  ..$ Wind   : num [1:30] 8.6 9.7 16.1 9.2 8.6 14.3 9.7 6.9 13.8 11.5 ...
  ..$ Temp   : int [1:30] 78 74 67 84 85 79 82 87 90 87 ...
  ..$ Month  : int [1:30] 6 6 6 6 6 6 6 6 6 6 ...
  ..$ Day    : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
 $ 7:'data.frame':  31 obs. of  6 variables:
  ..$ Ozone  : int [1:31] 135 49 32 NA 64 40 77 97 97 85 ...
  ..$ Solar.R: int [1:31] 269 248 236 101 175 314 276 267 272 175 ...
  ..$ Wind   : num [1:31] 4.1 9.2 9.2 10.9 4.6 10.9 5.1 6.3 5.7 7.4 ...
  ..$ Temp   : int [1:31] 84 85 81 84 83 83 88 92 92 89 ...
  ..$ Month  : int [1:31] 7 7 7 7 7 7 7 7 7 7 ...
  ..$ Day    : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...
 $ 8:'data.frame':  31 obs. of  6 variables:
  ..$ Ozone  : int [1:31] 39 9 16 78 35 66 122 89 110 NA ...
  ..$ Solar.R: int [1:31] 83 24 77 NA NA NA 255 229 207 222 ...
  ..$ Wind   : num [1:31] 6.9 13.8 7.4 6.9 7.4 4.6 4 10.3 8 8.6 ...
  ..$ Temp   : int [1:31] 81 81 82 86 85 87 89 90 90 92 ...
  ..$ Month  : int [1:31] 8 8 8 8 8 8 8 8 8 8 ...
  ..$ Day    : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...
```

```
 $ 9:'data.frame':	30 obs. of  6 variables:
  ..$ Ozone  : int [1:30] 96 78 73 91 47 32 20 23 21 24 ...
  ..$ Solar.R: int [1:30] 167 197 183 189 95 92 252 220 230 259 ...
  ..$ Wind   : num [1:30] 6.9 5.1 2.8 4.6 7.4 15.5 10.9 10.3 10.9 9.7 ...
  ..$ Temp   : int [1:30] 91 92 93 93 87 84 80 78 75 73 ...
  ..$ Month  : int [1:30] 9 9 9 9 9 9 9 9 9 9 ...
  ..$ Day    : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
```

Q. column means for Ozone, Solar.R, and Wind for each sub-data frame.

```r
sapply(s, function(x) { colMeans(x[, c("Ozone", "Solar.R", "Wind")])})
```

```
               5         6          7        8        9
Ozone         NA        NA         NA       NA       NA
Solar.R       NA 190.16667 216.483871       NA 167.4333
Wind    11.62258  10.26667   8.941935 8.793548  10.1800
```

```r
sapply(s, function(x) { colMeans(x[, c("Ozone", "Solar.R", "Wind")], na.rm = T)})
```

```
                5         6          7          8         9
Ozone    23.61538  29.44444  59.115385  59.961538  31.44828
Solar.R 181.29630 190.16667 216.483871 171.857143 167.43333
Wind     11.62258  10.26667   8.941935   8.793548  10.18000
```

apply()

```r
head(airquality)
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

```r
dff <- na.omit(airquality)
```

```r
apply(dff, 2, sum)
```

```
  Ozone Solar.R    Wind    Temp   Month     Day 
 4673.0 20513.0  1103.3  8635.0   801.0  1770.0 
```

```r
dim(dff)
```

```
[1] 111   6
```

Handling Missing Values

- Explicitly, i.e., flagged with NA.
- Implicitly, i.e., simply not present in the data.

```r
stocks <- tibble(
year = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
qtr = c( 1, 2, 3, 4, 2, 3, 4),
return = c(1.88, 0.59, 0.35, NA, 0.92, 0.17, 2.66)
)
stocks
```

```
# A tibble: 7 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4  NA   
5  2016     2   0.92
6  2016     3   0.17
7  2016     4   2.66
```

There are two missing values in this dataset:

- The return for the fourth quarter of 2015 is explicitly missing, because the cell where its value should be instead contains NA.
- The return for the first quarter of 2016 is implicitly missing, because it simply does not appear in the dataset.

```
stocks %>%
spread(year, return)
```

```
# A tibble: 4 x 3
    qtr `2015` `2016`
  <dbl>  <dbl>  <dbl>
1     1   1.88  NA
2     2   0.59   0.92
3     3   0.35   0.17
4     4  NA      2.66
```

complete() takes a set of columns, and finds all unique combinations. It then ensures the original dataset contains all those values, filling in explicit NAs where necessary.

```
stocks %>%
complete(year, qtr)
```

```
# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4  NA
5  2016     1  NA
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

filling missing values

```
#last observation carried forward
stocks %>%
  fill(return)
```

```
# A tibble: 7 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
```

```
2  2015     2   0.59
3  2015     3   0.35
4  2015     4   0.35
5  2016     2   0.92
6  2016     3   0.17
7  2016     4   2.66
```

```
stocks %>%
  fill(return, .direction = 'up')
```

```
# A tibble: 7 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4   0.92
5  2016     2   0.92
6  2016     3   0.17
7  2016     4   2.66
```

```
na_col <- colnames(stocks)[ apply(stocks, 2, anyNA) ]
na_col
```

```
[1] "return"
```

names(average_missing)

```
#mean with the argument na.rm = TRUE
average_missing <- apply(stocks[,na_col],
      2,
      mean,
      na.rm =  TRUE)
average_missing
```

```
return
 1.095
```

```
df_fill <- stocks %>% mutate(
  return  = ifelse(is.na(return), average_missing[1],return)
  )
df_fill
```

```
# A tibble: 7 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4   1.10
5  2016     2   0.92
6  2016     3   0.17
7  2016     4   2.66
```

remove outlier

Q.

1. Select numeric
2. print min max and avg values for each column.
3. filter values in column y outside range (0,20).
4. replace values in column y outside range (0,20) with mean

```
num <- sapply(diamonds, is.numeric)

diamonds[, num]
```

```
# A tibble: 53,940 x 7
   carat depth table price     x     y     z
   <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23  61.5    55   326  3.95  3.98  2.43
 2  0.21  59.8    61   326  3.89  3.84  2.31
 3  0.23  56.9    65   327  4.05  4.07  2.31
 4  0.29  62.4    58   334  4.2   4.23  2.63
 5  0.31  63.3    58   335  4.34  4.35  2.75
 6  0.24  62.8    57   336  3.94  3.96  2.48
 7  0.24  62.3    57   336  3.95  3.98  2.47
 8  0.26  61.9    55   337  4.07  4.11  2.53
 9  0.22  65.1    61   337  3.87  3.78  2.49
```

```
10   0.23  59.4    61   338  4     4.05  2.39
# i 53,930 more rows
```

```r
diamonds1 <- select_if(diamonds, is.numeric)
head(diamonds1)
```

```
# A tibble: 6 x 7
  carat depth table price     x     y     z
  <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23  61.5    55   326  3.95  3.98  2.43
2  0.21  59.8    61   326  3.89  3.84  2.31
3  0.23  56.9    65   327  4.05  4.07  2.31
4  0.29  62.4    58   334  4.2   4.23  2.63
5  0.31  63.3    58   335  4.34  4.35  2.75
6  0.24  62.8    57   336  3.94  3.96  2.48
```

```r
t(sapply(diamonds1, function(x) list(min = min(x), max = max(x),avg = mean(x))))
```

```
      min max   avg
carat 0.2 5.01  0.7979397
depth 43  79    61.7494
table 43  95    57.45718
price 326 18823 3932.8
x     0   10.74 5.731157
y     0   58.9  5.734526
z     0   31.8  3.538734
```

```r
diamonds1 %>%
filter(between(y, 3, 20))
```

```
# A tibble: 53,931 x 7
   carat depth table price     x     y     z
   <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23  61.5    55   326  3.95  3.98  2.43
 2  0.21  59.8    61   326  3.89  3.84  2.31
 3  0.23  56.9    65   327  4.05  4.07  2.31
 4  0.29  62.4    58   334  4.2   4.23  2.63
 5  0.31  63.3    58   335  4.34  4.35  2.75
 6  0.24  62.8    57   336  3.94  3.96  2.48
```

```
 7  0.24  62.3    57   336  3.95  3.98  2.47
 8  0.26  61.9    55   337  4.07  4.11  2.53
 9  0.22  65.1    61   337  3.87  3.78  2.49
10  0.23  59.4    61   338  4     4.05  2.39
# i 53,921 more rows
```

```r
diamonds1 %>%
mutate(y = ifelse(y < 3 | y > 20, mean(y), y))
```

```
# A tibble: 53,940 x 7
   carat depth table price     x     y     z
   <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23  61.5    55   326  3.95  3.98  2.43
 2  0.21  59.8    61   326  3.89  3.84  2.31
 3  0.23  56.9    65   327  4.05  4.07  2.31
 4  0.29  62.4    58   334  4.2   4.23  2.63
 5  0.31  63.3    58   335  4.34  4.35  2.75
 6  0.24  62.8    57   336  3.94  3.96  2.48
 7  0.24  62.3    57   336  3.95  3.98  2.47
 8  0.26  61.9    55   337  4.07  4.11  2.53
 9  0.22  65.1    61   337  3.87  3.78  2.49
10  0.23  59.4    61   338  4     4.05  2.39
# i 53,930 more rows
```

date and time

type coercion

Data Sampling

Setting the random number seed with set.seed() ensures reproducibility of the sequence of random numbers.

```r
rnorm(5)
```

```
[1] -0.5691690  1.1892515  0.6240644 -0.9989750 -1.1752370
```

```r
set.seed(1)
rnorm(5)
```

```
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

```
sample(1:10, 4)
```

```
[1] 3 1 5 8
```

```
sample(1:10, replace = TRUE)
```

```
 [1] 10  6 10  7  9  5  5  9  9  5
```

```
set.seed(1)
idx <- seq_len(nrow(airquality))
samp <- sample(idx, 6)
airquality[samp, ]
```

```
    Ozone Solar.R Wind Temp Month Day
68     77     276  5.1   88     7   7
129    32      92 15.5   84     9   6
43     NA     250  9.2   92     6  12
14     14     274 10.9   68     5  14
51     13     137 10.3   76     6  20
85     80     294  8.6   86     7  24
```

Data Discritization

cut() - The cut function is used in R for cutting a numeric value into bins of continuous values and is specified with cut labels.

Syntax: ***cut (x, breaks, labels, include.lowest, right)***

x: numeric input value (vector)

Break : No of breaks points

Labels: labels to each group/bin include.

lowest: whether to include the lowest (FALSE) or the highest (TRUE) break value or not.

Right: interval should be closed on the right and open on the left or vice versa.

```r
x <- c(3,19,27,49,22,16)

cut(x, breaks = c(0,7,15,24,34,50),
    labels = c("First", "Second", "Third", "Fourth", "Fifth"))
```

```
[1] First  Third  Fourth Fifth  Third  Third
Levels: First Second Third Fourth Fifth
```

```r
x <- 0:20

cut(x, breaks = c(0, 10, 20), include.lowest = FALSE)
```

```
 [1] <NA>    (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]
[10] (0,10]  (0,10]  (10,20] (10,20] (10,20] (10,20] (10,20] (10,20] (10,20]
[19] (10,20] (10,20] (10,20]
Levels: (0,10] (10,20]
```

```r
cut(x, breaks = c(0, 10, 20), include.lowest = TRUE)
```

```
 [1] [0,10]  [0,10]  [0,10]  [0,10]  [0,10]  [0,10]  [0,10]  [0,10]  [0,10]
[10] [0,10]  [0,10]  (10,20] (10,20] (10,20] (10,20] (10,20] (10,20] (10,20]
[19] (10,20] (10,20] (10,20]
Levels: [0,10] (10,20]
```

```r
age <- c(40, 49, 48, 40, 67, 52, 53)

wfact = cut(age, 3, labels = c('Young', 'Medium', 'Aged'))
wfact
```

```
[1] Young  Young  Young  Young  Aged   Medium Medium
Levels: Young Medium Aged
```

```r
table(wfact)
```

```
wfact
 Young Medium   Aged 
     4      2      1 
```

```r
age <- c(40, 49, 48, 40, 67, 52, 53)

wfact = cut(age, breaks = c(10, 30, 50,70), labels = c('Young', 'Medium', 'Aged'))
wfact
```

```
[1] Medium Medium Medium Medium Aged   Aged   Aged  
Levels: Young Medium Aged
```

```r
table(wfact)
```

```
wfact
 Young Medium   Aged 
     0      4      3 
```

Dates in R

Dates are represented by the Date class and can be coerced from a character string using the as.Date() function.

```r
x <- as.Date("1970-01-01")
x
```

```
[1] "1970-01-01"
```

Times in R

```r
x <- Sys.time()
x
```

```
[1] "2023-08-21 11:25:09 IST"
```

```r
class(x)
```

```
[1] "POSIXct" "POSIXt" 
```

```r
p <- as.POSIXlt(x)
```

```r
p$sec
```

```
[1] 9.450327
```

strptime() takes a character vector that has dates and times and converts them into to a POSIXlt object.

```r
datestring <- c("January 10, 2012 10:40", "December 9, 2011 9:10")
x <- strptime(datestring, "%B %d, %Y %H:%M")
x[1]
```

```
[1] "2012-01-10 10:40:00 IST"
```

```r
#formatting strings
#?strptime
```

You can use mathematical operations on dates and times. Well, really just + and -. You can do comparisons too (i.e. ==, <=)

```r
y <- as.Date("1970-01-05")
z <- as.Date("1970-01-01")
y-z
```

```
Time difference of 4 days
```

```r
as.Date("1970/01/05")
```

```
[1] "1970-01-05"
```

```r
x <- as.Date("2012-01-01")
y <- strptime("9 Jan 2011 11:34:21", "%d %b %Y %H:%M:%S")
#x-y
```

```r
as.POSIXlt(x)-y
```

```
Time difference of 356.747 days
```

Year

%Y (4 digits).

%y (2 digits; 00-69 → 2000-2069, 70-99 → 1970-1999).

Month

%m (2 digits).

%b (abbreviated name, like "Jan").

%B (full name, "January").

Day

%d (2 digits).

%e (optional leading space)

Time

%H (0-23 hour format).

%I (0-12, must be used with %p).

%p (a.m./p.m. indicator).

%M (minutes).

%S (integer seconds).

%OS (real seconds).

```r
parse_date("01/02/15", "%m/%d/%y")
```

```
[1] "2015-01-02"
```

```r
parse_date("1 janvier 2015", "%d %B %Y", locale = locale("fr"))
```

```
[1] "2015-01-01"
```

```r
d1 <- "January 1, 2010"
d2 <- "2015-Mar-07"
d3 <- "06-Jun-2017"
```

```r
d4 <- c("August 19 (2015)", "July 1 (2015)")
d5 <- "12/30/14" # Dec 30, 2014
t1 <- "1705"
t2 <- "11:15:10.12 PM"
```

```r
strptime("January 1 2010", "%b %d %Y")
```

```
[1] "2010-01-01 IST"
```

```r
df <- tribble(
~x, ~y,
"A", "1.21",
"B", "2.32",
"C", "4.56"
)
type_convert(df)
```

```
-- Column specification --------------------------------------------------------
cols(
  x = col_character(),
  y = col_double()
)

# A tibble: 3 x 2
  x         y
  <chr> <dbl>
1 A      1.21
2 B      2.32
3 C      4.56
```

tidyr a consistent way to organize your data in R, an organization called **tidy** data.

There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

```r
table1
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
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
```

```r
table2
```

```
# A tibble: 12 x 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 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
```

```r
table3
```

```
# A tibble: 6 x 3
  country      year rate
  <chr>       <dbl> <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
```

```r
table4a
```

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

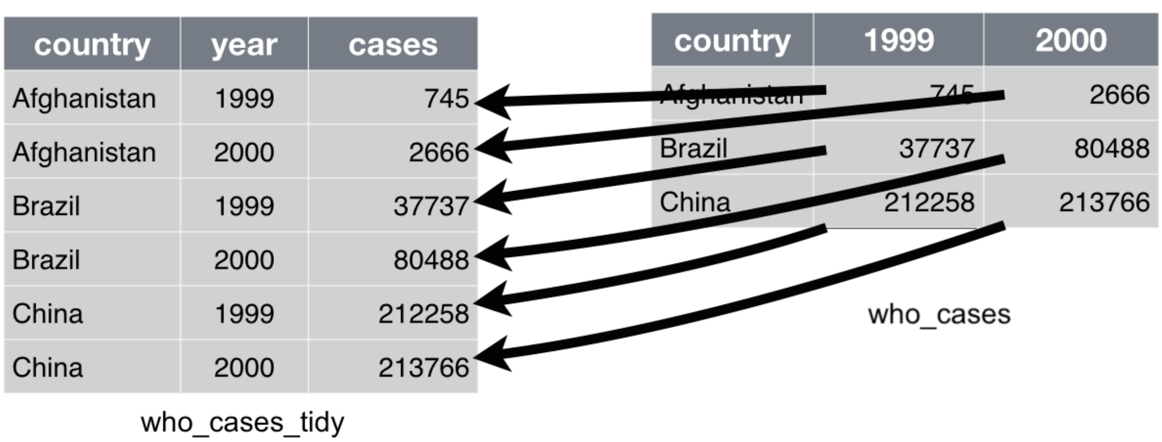
```r
table4b
```

```
# A tibble: 3 x 3
  country         `1999`     `2000`
  <chr>            <dbl>      <dbl>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

two common problems with data:

- One variable might be spread across multiple columns.
- One observation might be scattered across multiple rows.

Spreading and Gathering - tidyr: gather() and spread()



```r
table4a %>%
  gather(`1999`, `2000`, key = "year", value = "cases")
```

```
# A tibble: 6 x 3
  country     year   cases
  <chr>       <chr>  <dbl>
1 Afghanistan 1999     745
2 Brazil      1999   37737
3 China       1999  212258
4 Afghanistan 2000    2666
5 Brazil      2000   80488
6 China       2000  213766
```

```r
table4b %>%
gather(`1999`, `2000`, key = "year", value = "population")
```

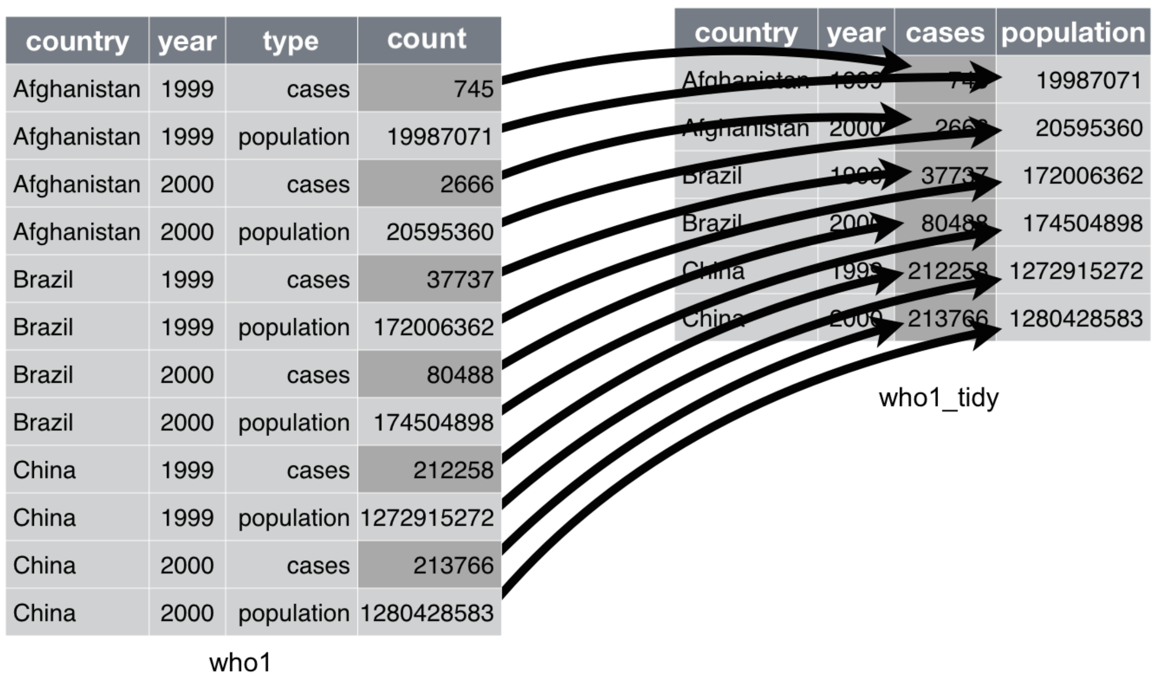
```
# A tibble: 6 x 3
  country     year  population
  <chr>       <chr>      <dbl>
1 Afghanistan 1999    19987071
2 Brazil      1999   172006362
3 China       1999  1272915272
4 Afghanistan 2000    20595360
5 Brazil      2000   174504898
6 China       2000  1280428583
```

Spreading

```r
table2
```

```
# A tibble: 12 x 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 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
```




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

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
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
```

```r
stocks <- tibble(
year = c(2015, 2015, 2016, 2016),
half = c( 1, 2, 1, 2),
return = c(1.88, 0.59, 0.92, 0.17)
)

stocks
```

```
# A tibble: 4 x 3
   year  half return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2016     1   0.92
4  2016     2   0.17
```

```r
stocks %>%
spread(year, return)
```

```
# A tibble: 2 x 3
   half `2015` `2016`
  <dbl>  <dbl>  <dbl>
1     1   1.88   0.92
2     2   0.59   0.17
```

```r
table3
```

```
# A tibble: 6 x 3
  country      year rate
  <chr>       <dbl> <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() pulls apart one column into multiple columns, by split- ting wherever a separator character appears.

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

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <dbl> <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
```

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

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <dbl> <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
```

```r
table3 %>%
separate(
rate,
into = c("cases", "population"),
convert = TRUE
)
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <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
```

```
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
```

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

```
# A tibble: 6 x 3
  country     new   rate
  <chr>       <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
```