Sane's STATS
Academy of Statistics

Package *tidyr* in R

Introduction

A same data can be represented or captured in multiple ways as shown here. Also all are not equally easy to use

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

country	year	rate
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

country	year	Type	Count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

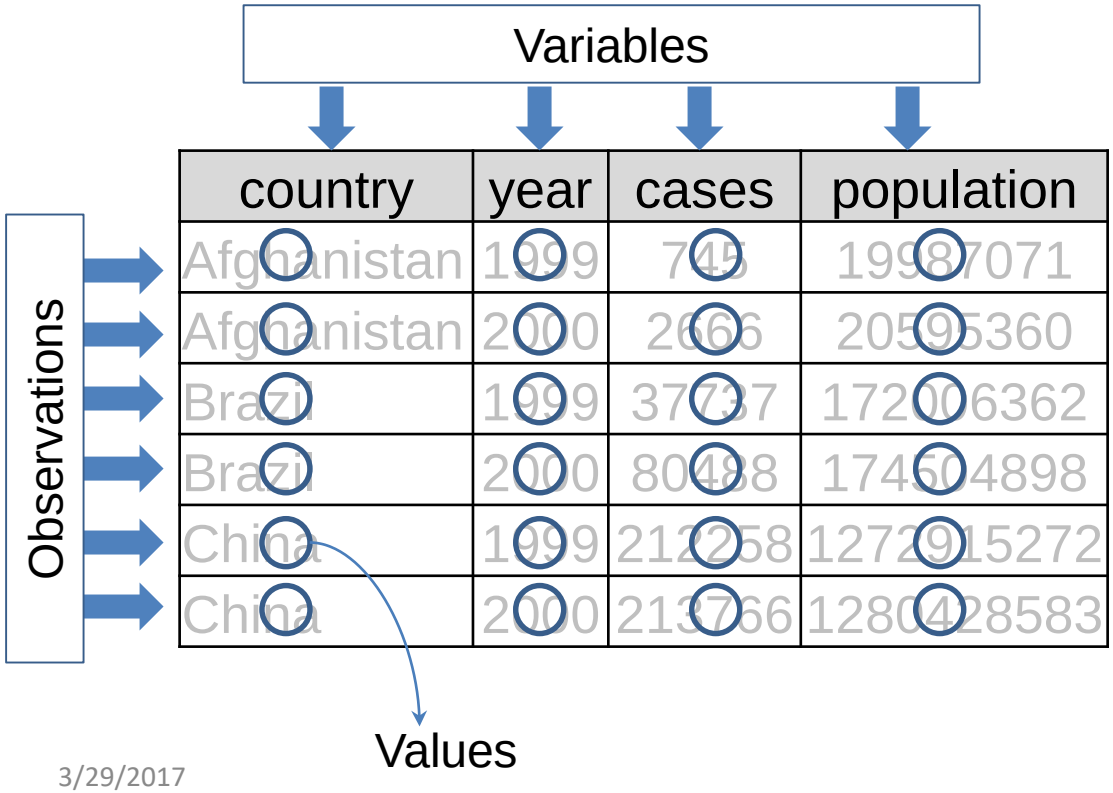
country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

country	1999	2000
Afghanistan	19987071	20595360
Brazil	172006362	174504898
China	1272915272	1280428583

Tidy data

Following are the three interrelated rules which makes a data set tidy

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



That interrelationship leads to an even simpler set of practical instructions

1. Put each dataset in a tibble
2. Put each variable in a column

Prerequisites

tidyr is a member of the core tidyverse

```
install.packages("tidyverse")
library(tidyverse)
```

Spreading and gathering

We need to resolve two common problems in the data

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

1. the column names 1999 and 2000 represent values of the year variable

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

country	year	Type	Count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

2. Observation is scattered across multiple rows. an observation is a country in a year, but each observation is spread across two rows

Gathering

We need to gather where

1. some of the column names are not names of variables, but values of a variable i.e. one variable spread across multiple columns
2. Each row represents two observations, not one

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

column names 1999 and 2000
represent values of the year variable

To **gather** those columns into a new pair of variables. we need three parameters:

1. The set of columns that represent values, not variables.
2. The name of the variable whose values form the column names. It is called the **Key**
3. The name of the variable whose values are spread over the cells. It is called **Value**

Syntax: gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

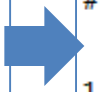
Gathering

Syntax: gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

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

```
> table4a %>% gather(`1999`, `2000`, key= "year", value= "cases")
# A tibble: 6 x 3
      country  year  cases
        <chr> <chr>  <int>
1 Afghanistan  1999    745
2      Brazil  1999  37737
3       China  1999 212258
4 Afghanistan  2000   2666
5      Brazil  2000  80488
6       China  2000 213766
```

```
> table4b
# A tibble: 3 x 3
      country     `1999`     `2000`
*       <chr>      <int>      <int>
1 Afghanistan   19987071   20595360
2      Brazil  172006362  174504898
3       China 1272915272 1280428583
```



```
> table4b %>% gather(`1999`, `2000`, key= "year", value= "population")
# A tibble: 6 x 3
      country  year population
        <chr> <chr>      <int>
1 Afghanistan  1999   19987071
2      Brazil  1999  172006362
3       China  1999 1272915272
4 Afghanistan  2000   20595360
5      Brazil  2000  174504898
6       China  2000 1280428583
```

Note that “1999” and “2000” are non-syntactic names so need to surround them in backticks

Gathering

```
> tidy4a <- table4a %>% gather(`1999`, `2000`, key= "year", value= "cases")
> tidy4b <- table4b %>% gather(`1999`, `2000`, key= "year", value= "population")
> left_join(tidy4a, tidy4b)
Joining, by = c("country", "year")
# A tibble: 6 x 4
      country  year  cases population
        <chr> <chr>  <int>      <int>
1 Afghanistan  1999    745   19987071
2      Brazil  1999  37737  172006362
3       China  1999 212258 1272915272
4 Afghanistan  2000   2666   20595360
5      Brazil  2000  80488  174504898
6       China  2000 213766 1280428583
```

Spreading

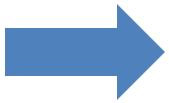
Spreading is the opposite of gathering. We use it when an observation is scattered across multiple rows. we need two parameters:

1. The column that contains variable names, the **key** column
2. The column that contains values forms multiple variables, the **value** column

Syntax: spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

an observation is a country in a year, but each observation is spread across two rows

```
> table2
# A tibble: 12 × 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
```



```
> spread(table2, key = "type", value = "count")
# A tibble: 6 × 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
```

gather() makes wide tables narrower and longer;
spread() makes long tables shorter and wider

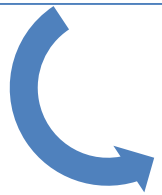
Separate

separate() pulls apart one column into multiple columns, by splitting wherever a separator character appears. We can use this when one column contains two variables

Syntax: separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, ...)

```
> table3
# A tibble: 6 × 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
```

By default, separate() will split values wherever it sees a non-alphanumeric character. We can convert to better types using convert = TRUE



```
> table3 %>% separate(rate, into = c("cases", "population"), sep ="/", convert = TRUE)
# A tibble: 6 × 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 cntd…

•You can also pass a vector of integers to **sep**.
•separate() will interpret the integers as positions to split at. Positive values start at 1 on the far-left of the strings; negative value start at -1 on the far-right of the strings.
•When using integers to separate strings, the length of **sep** should be one less than the number of names in the **into** option

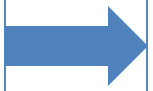
```
> table3 %>% separate(year, into = c("century", "year"), sep =2)
# A tibble: 6 x 4
      country century  year              rate
*       <chr>   <chr> <chr>             <chr>
1 Afghanistan      19    99       745/19987071
2 Afghanistan      20    00      2666/20595360
3      Brazil      19    99    37737/172006362
4      Brazil      20    00    80488/174504898
5       China      19    99 212258/1272915272
6       China      20    00 213766/1280428583
```

Unite

unite() is the inverse of separate(): it combines multiple columns into a single column

Syntax: unite(data, col, ..., sep = "_", remove = TRUE)

```
> table5
# A tibble: 6 × 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 × 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
```

The default will place an underscore (_) between the values from different columns. Here we don't want any separator so we use ""

Missing values

Value can be missing in one of two possible ways

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

	year	qtr	return
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. As the cell contains *NA*
- The return for the first quarter of 2016 is implicitly missing. Does not appear in the dataset.

Missing values cntd..

We can make the implicit missing value explicit by putting years in the columns

```
> stocks %>%  spread(key = year, value = 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
```

We can also use drop_na() function

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

Complete function for Missing values

This is another important tool to turns implicit missing values into explicit missing values. 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

Syntax: complete(data, ..., fill = list())

```
> 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
```

```
> # Imputing by Mean
> mu_return <- mean(stocks$return,na.rm = T)
> stocks %>% complete(year,qtr,
+                     fill = list(return=mu_return))
# 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   1.10
5  2016     1   1.10
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

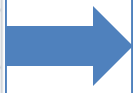
fill function for Missing values

Sometimes missing values indicate that the previous value should be carried forward.

We can fill in these missing values with fill(). Fills missing values in using the previous entry

Syntax: fill(data, ..., .direction = c("down", "up"))

	Channel	Program	Adrate
1	SAB TV	Tarak Mehta	600
2	*NA*	Chidiya Ghar	450
3	*NA*	FIR	250
4	Star Plus	Chandra	750
5	*NA*	Namkaran	550



```
> TVrate %>% fill(Channel)
# A tibble: 5 x 3
    Channel      Program Adrate
      <chr>        <chr>  <dbl>
1    SAB TV  Tarak Mehta    600
2    SAB TV Chidiya Ghar    450
3    SAB TV          FIR    250
4 Star Plus      Chandra    750
5 Star Plus     Namkaran    550
```