QTM 151

Week 7 – tidyr

Umberto Mignozzetti
Mar 12

Recap

We learned:

- `qplot`: quick way to make ggplot graphs.
- `ggplotly`: transform ggplot objects into nice plotly viz.
- `plot_ly`: create nice plotly graphs.
- `dplyr` methods: data wrangling
- `dplyr *_join` methods: joining data

Do you have any questions about any of these contents?

No quiz this week

Our GitHub page is: https://github.com/umbertomig/qtm151

Getting Started

Getting Started: loading packages

```
# Loading tidyverse
library(tidyverse)
```

Loading data

```
# population table
table1←tibble(
  `country`=c("Afghanistan","Brazil","China"),
  `1999`=c(19987071, 172006362, 1272915272),
  `2000`=c(20595360, 174504898, 1280428583)
)
table1
```

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

```
# case table
table2←tibble(
```

Loading data

```
table3← tibble(
  `country` = c("Afghanistan", "Afghanistan", "Afghanistan", "Afgh
   `year` = c(1999,1999,2000,2000,1999,1999,2000,2000,1999,1999,20
   `type` = c("case", "population","case", "population","case", "p
   `count` = c(745, 19987071, 2666, 20595360, 37737, 172006362, 80
)
table3
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  1 Afghanistan  1999 case              745
##  2 Afghanistan  1999 population    19987071
##  3 Afghanistan  2000 case             2666
##  4 Afghanistan  2000 population    20595360
##  5 Brazil       1999 case            37737
##  6 Brazil       1999 population   172006362
##  7 Brazil       2000 case            80488
```

Loading data

```
table4←tibble(
    `country` = c("Afghanistan", "Afghanistan", "Brazil", "Brazil'
     `year` = c(1999,2000,1999,2000,1999,2000),
    `rate`= c("745/19987071", "2666/20595360", "37737/172006362",
)
table4
```

```
## # 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
```

Loading data

```
# population table
tbl1←tibble(
  `county`=c("DeKalb","Fulton","Cobb"),
  `2010`=c(691961, 920581, 690688),
  `2011`=c(693961, 921581, 691688)
)
tbl1
```

```
## # A tibble: 3 x 3
##   county `2010` `2011`
##   <chr>   <dbl>  <dbl>
## 1 DeKalb 691961 693961
## 2 Fulton 920581 921581
## 3 Cobb   690688 691688
```

```
# veterans table
tbl2←tibble(
```

Loading data

```
tbl3← tibble(
  `country` = c("DeKalb", "DeKalb", "DeKalb", "DeKalb", "Fulton",
   `year` = c(2010,2010,2011,2011,2010,2010,2011,2011,2010,2010,20
   `type` = c("veterans", "population","veterans", "population","\
   `count` = c(36189, 691961, 36389, 693961, 42448, 920581, 42648,
)
tbl3
```

```
## # A tibble: 12 x 4
##    country  year type        count
##    <chr>   <dbl> <chr>       <dbl>
##  1 DeKalb   2010 veterans    36189
##  2 DeKalb   2010 population 691961
##  3 DeKalb   2011 veterans    36389
##  4 DeKalb   2011 population 693961
##  5 Fulton   2010 veterans    42448
##  6 Fulton   2010 population 920581
##  7 Fulton   2011 veterans    42648
```

Loading data

```
tbl4← tibble(
  `country` = c("DeKalb", "DeKalb", "Fulton", "Fulton", "Cobb", "C
   `year` = c(2010,2011,2010,2011,2010,2011),
   `prop` = c("36189/691961", "36389/693961", "42448/920581", "426
)
tbl4
```

```
## # A tibble: 6 x 3
##   country  year prop
##   <chr>   <dbl> <chr>
## 1 DeKalb   2010 36189/691961
## 2 DeKalb   2011 36389/693961
## 3 Fulton   2010 42448/920581
## 4 Fulton   2011 42648/921581
## 5 Cobb     2010 41345/690688
## 6 Cobb     2011 41545/691688
```

tidyr

tidyr

The *tidyr* package helps tidy up messy datasets. 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

There are a few key functions in the *tidyr* package, *gather()*, *spread()*, *separate()*, *unite()*, *complete()*, *fill()*.

gather

gather

To tidy a dataset, we need to *gather* multiple columns, and gathers them into key-value pairs: it makes "wide" data longer.

Syntax:

```
gather(dataset, set_of_columns,
       key="name of variable whose values form the column names",
       value="name of variable whose values are spread over the cells"
```

gather

Example:

```
gather(table1, `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
```

gather

Example:

```
gather(table2, "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
```

Your Turn: Do the same with the `tbl1` and `tbl2` datasets. Save the results and join the datasets.

spread

spread

spread() is the opposite of *gather()*.

gather() makes wide tables narrower and longer, *spread()* makes long tables shorter and wider.

Syntax:

```
spread(dataset,
       key="the column that contains variable name",
       value="the column that contains values forms multiple variables
```

spread

Example:

```
spread(table3, key= "type", value= "count")
```

```
## # A tibble: 6 x 4
##   country      year   case 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
```

Your Turn: Check out `tbl3`. Then, spread it by `type`.

separate and unite

separate and unite

separate() pulls apart one column into multiple columns, *unite()* is the inverse of *separate()*.

Check `table4`. Note that the rate variable has two variables inside it: cases and population. To separate them:

Syntax for *separate()*:

```
separate(table, variable_separate, into=c('v1','v2'), sep="/")
```

separate and unite

Example:

```
table5 ← separate(table4, rate, into=c('case','population'), sep=
table5
```

```
## # A tibble: 6 x 4
##   country      year case   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
```

Your Turn: Do the same separate for `tbl4`.

separate and unite

We can also separate by position:

```
separate(table3, year, into = c("century", "year"), sep=2)
```

```
## # A tibble: 12 x 5
##    country     century year  type            count
##    <chr>       <chr>   <chr> <chr>           <dbl>
##  1 Afghanistan 19      99    case              745
##  2 Afghanistan 19      99    population   19987071
##  3 Afghanistan 20      00    case             2666
##  4 Afghanistan 20      00    population   20595360
##  5 Brazil      19      99    case            37737
##  6 Brazil      19      99    population  172006362
##  7 Brazil      20      00    case            80488
##  8 Brazil      20      00    population  174504898
##  9 China       19      99    case           212258
## 10 China       19      99    population 1272915272
## 11 China       20      00    case           213766
```

separate and unite

Syntax for *unite()*:

```
unite(table, name_col, col1, col2, ... , sep="/")
```

separate and unite

Example:

```
unite(table5, col = rate, case, population, sep="/")
```

```
## # 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
```

complete

complete

complete() is useful to fill up the columns with missing data, based on a given pattern.

Suppose we have the following dataset:

```
df ← tibble(
  group = c(1:2, 1), item_id = c(1:2, 2),
  item_name = c("a", "b", "b"),
  value1 = 1:3, value2 = 4:6
)
df
```

```
## # A tibble: 3 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl> <chr>      <int>  <int>
## 1     1       1 a              1      4
## 2     2       2 b              2      5
```

complete

We can complete by group, item_id and item_name:

```
df %>% complete(group, nesting(item_id, item_name))
```

```
## # A tibble: 4 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl> <chr>      <int>  <int>
## 1     1       1 a              1      4
## 2     1       2 b              3      6
## 3     2       1 a             NA     NA
## 4     2       2 b              2      5
```

Your Turn: Do the same complete for `stocks1`. What happened?

Questions?

Have a great weekend!