TidyVerse

Marco Torchiano

Version 1.1.0 - April 2021

License 

Introduction

What is Tidyverse?

The `tidyverse` is a collection of R packages designed for data science.

All packages share

- an underlying design philosophy,
- grammar, and
- data structures.

Tidyverse core packages

- `dplyr` grammar for data manipulation
- `tidyr` set of functions to tidy data
- `readr` read rectangular data
- `stringr` functions for string manipulation
- `forcats` tools to handle factors
- `tibble` evolution of data.frame
- `purrr` functional programming support
- `ggplot2` system for declarative graphics

Pipes

Pipes

When the recipe to process your data is made up of several different stages:

- Save each intermediate step as a new object.
- Overwrite the original object many times.
- Compose the functions.
- Use the pipe (in library `magrittr`)

Save as new object / overwrite

```
mixed    <- mix(ingredients)
form     <- pour(mixed, into=baking_form)
tb_baked <- put(form, into=oven)
ready    <- bake(tb_baked, time=30)
sliced   <- slice(ready, pieces=8)
eat(sliced, 1)
```

```
preparation <- mix(ingredients)
preparation <- pour(preparation, into=baking_form)
preparation <- put(preparation, into=oven)
preparation <- bake(preparation, time=30)
preparation <- slice(preparation, pieces=8)
eat(sliced, 1)
```

Compose the functions

```
eat(
  slice(
    bake(
      put(
        pour(
          mix(ingredients),
          into=baking_form),
        into=oven),
    time=30),
  pieces=8),
  num=1)
```


Use the pipe

```
ingredients %>%
  mix() %>%
  pour(into=baking_form) %>%
  put(into=oven)  %>%
  bake(time=30) %>%
  slice(pieces=8) %>%
  eat(1)
```

Source: https://twitter.com/dmi3k/status/1191824875842879489?s=20

Basic pipe `%>%`

- `%>%` allows passing an object as first argument of a function
  - `x %>% f(y)` equals to `f(x, y)`
  - shortcut `Ctrl/Cmd` + `Shift` + `m`

```
numbers <- c(1, 1, 2, 3, 5, 8, 13, 21, 34, 55 )
sum( sqrt( numbers ), na.rm=TRUE )
```

```
## [1] 31.64604
```

```
numbers %>% sqrt() %>% sum(na.rm=TRUE)
```

```
## [1] 31.64604
```


Placeholder `.`

In a pipe-line `.` represents the first element.

```
numbers[numbers>5 | numbers==1]
```

```
## [1]  1  1  8 13 21 34 55
```

```
numbers %>% .[.>5 | .==1]
```

```
## [1]  1  1  8 13 21 34 55
```

When `.` appears in the expression the first argument is not passed.

Exposition `%$%`

- `%$%` makes the columns/variables of the first element immediately available

```
logarithm <- data.frame(x=1:10,y=log(1:10))
cor(logarithm$x, logarithm$y)
```

```
## [1] 0.9516624
```

```
with(logarithm, cor(x,y) )
```

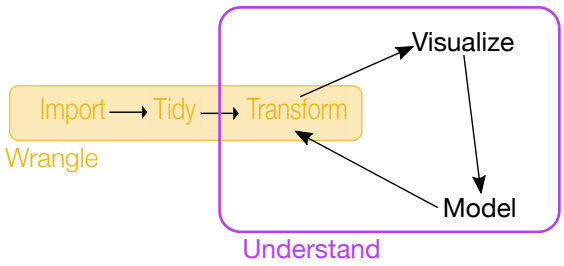
```
## [1] 0.9516624
```

```
logarithm %$% cor(x,y)
```

```
## [1] 0.9516624
```


Wrangling

Wrangling



Wrangling is the preparation part preceding visualization:

- Import: get your data from the disk
- Tidy: put your data in order
- Transform: summarize, reorder, compute

Tabular Data

Tables are structured into *rows*

- a row contains information about some item
- all rows consist of the same number of *cells* (possibly empty)
- cells in the same position describe the same property of the items
- the first row contains the *headers*, that identify the name of the properties

Tibble

- `tibble` is an evolution of the `data.frame` structure
- it is used throughout the tidyverse to store tabular data
- it has improved printing:
  - only the first 10 rows are printer
  - type of columns is printed below the name
- subsetting (`[`) always returns a tibble (e.g. `tb[1,1]`)
  - data frames sometimes with `[` return vectors, e.g. `df[1,1]`

Import

Using package `readr`

Focus on tabular (rectangular data) data

- `read_csv()` read a csv content
- `read_tsv()` read a tab-separated content
- `read_fwf()` read a fixed width fields content
- `read_log()` read a log file

CSV

Comma Separated Values

Textual representation of tabular data

- text file containing records separated by newlines
- fields inside a record separated by commas `,`
- first record may contain *headers*
- double-quote (`"`) are used to enclose text
  - ddq: `"This ""is"" quoted"`
- mime type is `text/csv`

Representation in CSV

ID	Surname	Name	Animal
4321	Snow	Jon	Wolf
5765	Lannister	Tyrion	Lion
4663	Targaryen	Daenerys	Dragon
9896	Stark	Arya	Wolf

Csv:

```
ID,Surname,Name,Animal
4321,Snow,Jon,Wolf
5765,Lannister,Tyrion,Lion
4663,Targaryen,Daenerys,Dragon
9896,Stark,Arya,Wolf
```

Read CSV

Function `read_csv()`

- file path (or content)
- `skip` number of lines to skip (`0`)
- `col_name` whether first row contains names (`TRUE`)
- `comment` character for comment lines
- `na` values to be considered as `NA`

Faster than `read.csv()` and not locale dependent

Read CSV - Example

```
read_csv( csv_got )
```

```
## # A tibble: 4 x 4
##      ID Surname   Name      Animal
##   <dbl> <chr>     <chr>     <chr>
## 1  4321 Snow      Jon       Wolf
## 2  5765 Lannister Tyrion    Lion
## 3  4663 Targaryen Daenerys  Dragon
## 4  9896 Stark     Arya      Wolf
```

Column types

Column types are guessed based on the first 1k rows

It is possible to specify the type of columns.

```
read_csv(csv_got,
     col_type = cols(
       ID = col_integer(),
       Animal = col_factor()  ))
```

```
## # A tibble: 4 x 4
##      ID Surname   Name     Animal
##   <int> <chr>     <chr>    <fct>
## 1  4321 Snow      Jon      Wolf
## 2  5765 Lannister Tyrion   Lion
## 3  4663 Targaryen Daenerys Dragon
## 4  9896 Stark     Arya     Wolf
```


Tidy data

Tidy data is a structured approach to organize data where:

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

Tidy data:

- are easier to understand
- make manipulate simpler by using a uniform approach
- leverage vectorial features of R

Example dataset

First formatting `table1`:

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	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
C ıa	2000	cases	213766

Another one `table3`:

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

Same data spread over two tables.

Cases `table4a`:

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

Population `table4b`:

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

Exercise

Which tables are tidy?

Which are not tidy, and why?

Tidying data

- `pivot_longer()` takes multiple columns (variables) and merge them
  - solves the issue of same variable spread over multiple columns
- `pivot_wider()` merge together different rows by adding new columns (variables)
  - solves the issues of observations spread on many rows
- `separate()` split the values contained in a column at a given separator
  - the opposite is `unite()`

Pivot longer

A variable is spread over two columns (that are not variables) that should be merged together.

```
table4a
```

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


Pivot longer

```
pivot_longer(table4a, cols=c("1999","2000"),
             names_to="year", values_to="cases" )
```

```
## # A tibble: 6 x 3
##   country     year   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

Observations are spread over two rows that should be merged together.

```
head(table2)
```

```
## # A tibble: 6 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
```


Pivot wider

```
pivot_wider(table2, names_from = type,
                    values_from = 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

Two distinct variables are combined in single column that should be split.

```
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

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

```
## # 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
```

Missing values

- Explicit: represented by `NA` values.
- Implicit: not present in the data.

An explicit missing value is the presence of an absence; an implicit missing value is the absence of a presence.

Depending on the goal one alternative is preferable.

Transformation

Transformation

Operate on (tidy) data:

- `filter()`: picks rows/observation matching condition
- `select()`: picks columns/variables matching conditions
- `mutate()`: create or replace a variable
- `summarize()`: aggregates rows for each variable
- `arrange()`: reorder rows

They can be used with `group_by()`

The functions are defined in package `dplyr` (part of the core tidyverse)

Transformation convention

The six *verbs* for data manipulation share a uniform approach:

- the first argument is the data frame
  - suitable for the pipe `%>%`
- names of variables can be used without quotes
- result is the modified data frame

Moreover they:

- exist in both British and american version (e.g. summarize and summarise)
- provide warnings for common programming errors

Filtering

Function `filter()` accepts a list of conditions that are considered in AND.

```
courses %>% filter(semester==2,credits==10)
```

```
##    code          course semester credits
## 1 01RKC Linear Algebra        2      10
## 2 17AXO      Physics I        2      10
```


Conditions

When mistakenly `=` is used instead of `==` a warning is issued.

```
try(
courses %>% filter(semester = 2,credits==10)
)
```

```
## Error : Problem with `filter()` input `..1`.
## x Input `..1` is named.
## ℹ This usually means that you've used `=` instead
## ℹ Did you mean `semester == 2`?
```

Floating point approximation

Warning `numeric` (double) values are approximate representation of Real numbers.

Be careful in comparing with `==`, function `near()` can be used instead:

```
approx <- c( 1/49*49, sqrt(2)^2 )
approx == 1:2
```

```
## [1] FALSE FALSE
```

```
near( approx, 1:2 )
```

```
## [1] TRUE TRUE
```


Selection

Function `select()` picks a subset of the variables in the data frame in the given order

```
courses %>% select(code,credits,course)
```

```
##    code credits           course
## 1 15AHM       8        Chemistry
## 2 12BHD       8 Computer science
## 3 16ACF      10       Calculus I
## 4 01PNN       6     Free Credits
## 5 01RKC      10   Linear Algebra
## 6 17AXO      10        Physics I
```

Selection by exclusion

It is also possible to exclude variables with a `-` before the column name

```
courses %>% select( -semester )
```

```
##    code            course credits
## 1 15AHM         Chemistry       8
## 2 12BHD Computer science       8
## 3 16ACF       Calculus I      10
## 4 01PNN     Free Credits       6
## 5 01RKC   Linear Algebra      10
## 6 17AXO         Physics I      10
```


Selection helpers

A few helper functions can be used:

- `starts_with()`: all names starting with the given prefix
- `ends_with()`: all names ending with the given suffix
- `contains()`: all names including the given sub string
- `num_range(n,r)`: all names with the prefix and the items in the range

Renaming

It is possible to change names to variables with `rename()`:

- `rename()` keeps all unmentioned variables
- `select()` drops all unmentioned variables

```
courses %>% rename( period = semester )
```

```
##    code           course period credits
## 1 15AHM        Chemistry      1       8
## 2 12BHD Computer science      1       8
## 3 16ACF       Calculus I      1      10
## 4 01PNN     Free Credits      2       6
## 5 01RKC   Linear Algebra      2      10
## 6 17AXO        Physics I      2      10
```


Mutation

Function `mutate()` allows defining new variables and modifying existing ones.

```
courses %>% mutate( lecture_h = credits*10,
                    semester = factor(semester,1:2,c("I","I
```

```
##    code           course semester credits lecture_
## 1 15AHM        Chemistry        I       8        8
## 2 12BHD Computer science        I       8        8
## 3 16ACF       Calculus I        I      10       10
## 4 01PNN     Free Credits       II       6        6
## 5 01RKC   Linear Algebra       II      10       10
## 6 17AXO        Physics I       II      10       10
```

Summarization

Function `summarize()` (or `summarise()`) combines rows according to the provided expressions

```
courses %>% summarize(
              tot_credits = sum(credits),
              num_courses = length(course))
```

```
##   tot_credits num_courses
## 1          52           6
```


Grouping

Summarization is mostly useful when combined with `group_by()` that allows summarizing by groups instead of the whole dataset.

```
courses %>% group_by(semester) %>%
  mutate( semester = factor(semester,1:2,c("I","II"))
  summarize( tot_credits = sum(credits),
             num_courses = n())
```

```
## # A tibble: 2 x 3
##   semester tot_credits num_courses
## * <fct>          <dbl>       <int>
## 1 I                 26           3
## 2 II                26           3
```

Summarization functions

All functions returning a single value from an array.

A few useful shortcut functions:

- `n()` count of elements in the group
- `n_disting()` count of elements in the group
- `first()`, `last()`, `nth()`

Sorting

Function `arrange()` sorts the rows by ascending values in given variables

- `desc()` can be used to have descenting order

```
courses %>% arrange(desc(credits), semester)
```

```
##    code           course semester credits
## 1 16ACF        Calculus I        1      10
## 2 01RKC   Linear Algebra        2      10
## 3 17AXO         Physics I        2      10
## 4 15AHM         Chemistry        1       8
## 5 12BHD Computer science        1       8
## 6 01PNN      Free Credits        2       6
```

Grouped filtering

When applying a filtering to a grouped dataset, filtering is applied to each group

```
courses %>% group_by(semester) %>%
            filter( credits == min(credits) )
```

```
## # A tibble: 3 x 4
## # Groups:   semester [2]
##   code  course           semester credits
##   <chr> <chr>               <int>   <dbl>
## 1 15AHM Chemistry               1       8
## 2 12BHD Computer science        1       8
## 3 01PNN Free Credits            2       6
```


Grouped mutation

Mutation of grouped data, mutates group-wise

```
courses %>% group_by(semester) %>%
            mutate( prop_cfu = credits/sum(credits) )
```

```
## # A tibble: 6 x 5
## # Groups:   semester [2]
##   code  course           semester credits prop_cfu
##   <chr> <chr>               <int>   <dbl>    <dbl>
## 1 15AHM Chemistry               1       8    0.308
## 2 12BHD Computer science        1       8    0.308
## 3 16ACF Calculus I              1      10    0.385
## 4 01PNN Free Credits            2       6    0.231
## 5 01RKC Linear Algebra          2      10    0.385
## 6 17AXO Physics I               2      10    0.385
```

References

- Hadley Wickham and Garrett Grolemund. “R for Data Science”, 2017
  - https://r4ds.had.co.nz
- Hadley Wickham. “Tidy Data”, Journal of Statistical Software, 59(10), 2014.
  - http://www.jstatsoft.org/v59/i10/paper