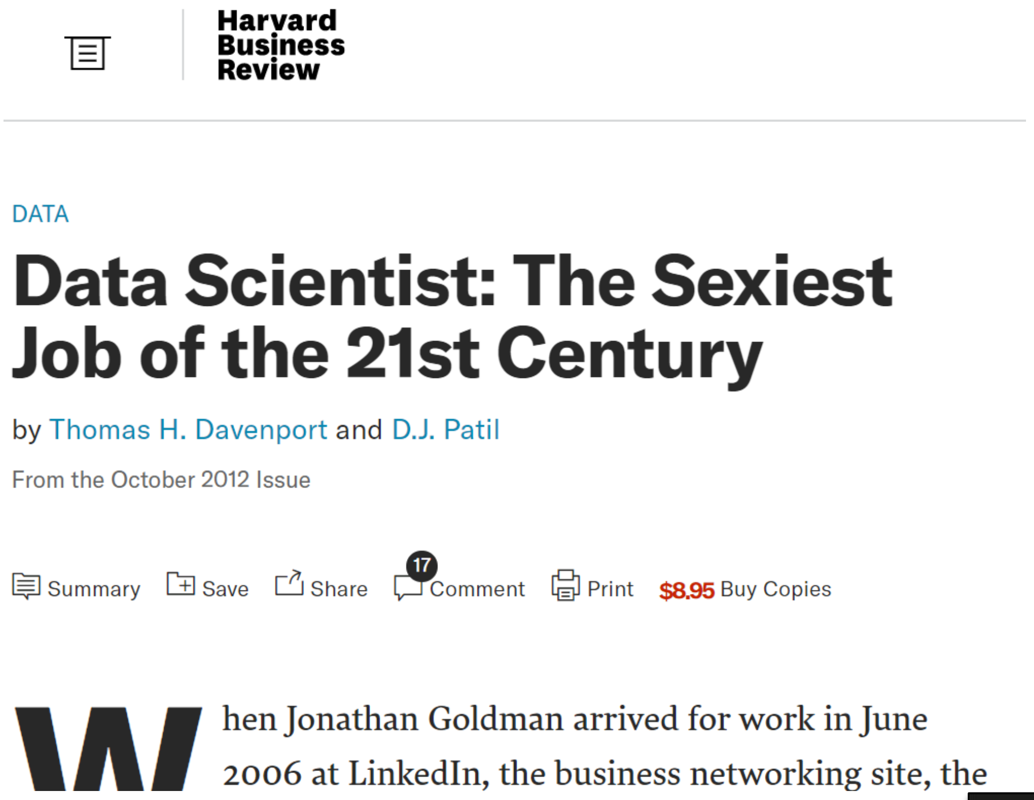
Tidyverse introduction

Akitaka Matsuo

Introduction

- Data manipulation (wrangling) takes the vast majority of time in data science
  - That is more true when the data are big
- **tidyverse** provides a unified method for data wrangling
  - ``The tidyverse is an **opinionated collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.”


Harvard Business Review

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

From the October 2012 Issue

Summary Save Share 17 Comment Print $8.95 Buy Copies

When Jonathan Goldman arrived for work in June 2006 at LinkedIn, the business networking site, the

Why or why not **tidyverse**?

Pros:

- Once getting used to it, you can do a lot of complicated data management work efficiently
- Improved reproducibility of research
  - Easy to read data manipulation process
- Unified syntax to access to many background options
  - **dbplyr**: SQL databases
  - **dtplyr**: data.table
  - **sparklyr**: Apache Spark (Day 5)

Cons:

- Opinionated
- Steep learning curve

tidyverse packages

tidyverse (tidy + universe) is a group of packages working together.

- **dplyr**: ``a grammar of data manipulation''
- **tidyr**: data reshaping to make the data *tidy*
- **readr**: read various data sets (also **readxl**)
- **purrr**: `map()`
- **ggplot2**: data visualization

We will mostly look at **dplyr** and **tidyr**.

Load it

```
library(tidyverse)
## -- Attaching packages ------------------------ tidyverse 1.3.0 --
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.2     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts --------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

dplyr

A package to provide a new grammar to carry out data manipulation

With `dplyr` we don't have to...

```
mtcars_sub <- subset(mtcars, cyl == 4)

mtcars_sub$log_hp <- log(mtcars_sub$hp)

model_regression <- lm(mpg ~ log_hp + wt, data =
mtcars_sub)

summary(model_regression)
```

The grammer of `dplyr`

Basics:

- `%>%`: pipe operator
- `filter()`: select rows
- `select()`: select variables
- `mutate()`: manipulate variables

Group based operators:

- `group_by()`: group the dataset
- `summarise()`: get a summary value for the group
- `count()`: table-like function

%>% - pipe operator

- **%>%** sends the results from one function to the another. For example

```
x <- 1:10
x %>% sum()
```

  - The pipe command, send `x` to `sum()` as the first argument of the function (i.e. `sum(x)`)
- When you want to send to the second or later arugment of a function, you can use `.` indicator

```
mtcars %>% lm(mpg ~ wt + cyl + hp, data = .)
```

- You can use multiple `%>%` to create a chain of operations

```
mtcars %>% lm(mpg ~ wt + cyl + hp, data = .) %>%
summary()
```

- The pipe operator is one of the main reasons why tidrverse codes look different from base-R

%>% - pipe outputs

```
mtcars %>% lm(mpg ~ wt + cyl + hp, data = .) %>% summary()
##
## Call:
## lm(formula = mpg ~ wt + cyl + hp, data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9290 -1.5598 -0.5311  1.1850  5.8986
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 38.75179    1.78686  21.687  < 2e-16 ***
## wt          -3.16697    0.74058  -4.276 0.000199 ***
## cyl         -0.94162    0.55092  -1.709 0.098480 .
## hp          -0.01804    0.01188  -1.519 0.140015
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.512 on 28 degrees of freedom
## Multiple R-squared:  0.8431, Adjusted R-squared:  0.8263
## F-statistic: 50.17 on 3 and 28 DF,  p-value: 2.184e-11
```

filter()

- `filter()` subset the data, using the conditional statement.

```
filter(mtcars, cyl == 4)
```

- Or, using pipe operator

```
mtcars %>% filter(cyl == 4)
```

select()

`select()` is used for selecting variables to keep or drop from a data.frame.

Selecting

```
mtcars <- mtcars %>% rownames_to_column(var = "car_name")

mtcars_sub <- mtcars %>%
  select(car_name, mpg, cyl, wt, hp)

mtcars_sub %>% head()
##             car_name  mpg cyl    wt  hp
## 1          Mazda RX4 21.0   6 2.620 110
## 2      Mazda RX4 Wag 21.0   6 2.875 110
## 3         Datsun 710 22.8   4 2.320  93
## 4     Hornet 4 Drive 21.4   6 3.215 110
## 5  Hornet Sportabout 18.7   8 3.440 175
## 6            Valiant 18.1   6 3.460 105
```

select()

Dropping

```
mtcars_sub2 <- mtcars %>%
  select(-c(gear, carb, disp, qsec))

mtcars_sub2 %>% head()
##                car_name  mpg cyl  hp drat    wt vs am
## 1             Mazda RX4 21.0   6 110 3.90 2.620  0  1
## 2         Mazda RX4 Wag 21.0   6 110 3.90 2.875  0  1
## 3            Datsun 710 22.8   4  93 3.85 2.320  1  1
## 4        Hornet 4 Drive 21.4   6 110 3.08 3.215  1  0
## 5 Hornet Sportabout 18.7   8 175 3.15 3.440  0  0
## 6               Valiant 18.1   6 105 2.76 3.460  1  0
```

mutate()

`mutate()`: manipulation (either generation or alteration of variable). Let's say you want to create a dummy variable to indicate a car with above averate `mpg`.

```
mtcars <- mtcars %>%
  mutate(good_mpg = (mpg > mean(mpg)))
  #or mutate(good_mpg = (mpg > mean(mpg)) %>% as.integer())

mtcars %>% select(car_name, mpg, good_mpg) %>% head()
##             car_name  mpg good_mpg
## 1          Mazda RX4 21.0     TRUE
## 2      Mazda RX4 Wag 21.0     TRUE
## 3         Datsun 710 22.8     TRUE
## 4     Hornet 4 Drive 21.4     TRUE
## 5  Hornet Sportabout 18.7    FALSE
## 6            Valiant 18.1    FALSE
```

arrange, slice

```
mtcars %>% arrange(desc(mpg)) %>%
  select(car_name, mpg, good_mpg) %>%
  slice(1:5)
```

```
##          car_name  mpg good_mpg
## 1  Toyota Corolla 33.9     TRUE
## 2        Fiat 128 32.4     TRUE
## 3     Honda Civic 30.4     TRUE
## 4    Lotus Europa 30.4     TRUE
## 5       Fiat X1-9 27.3     TRUE
```

group_by, summarise

`dplyr` also provides a very powerful functionality to get an aggregate measure for specific groups of the interest. For instance suppose that you want to know an average `mpg` by number of `cyl`. You can achieve that with the combination of `group_by()` and `summarise()`

```
mtcars %>%
  group_by(cyl) %>% # this can be more than one grouping variable
  summarise(avg_mpg = mean(mpg, na.rm = T),
            count = n()) # and you can generate more than one
variable
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## # A tibble: 3 x 3
##     cyl avg_mpg count
##   <dbl>   <dbl> <int>
## 1     4    26.7    11
## 2     6    19.7     7
## 3     8    15.1    14
```

ecpr

group_by, mutate

- `group_by()` can work with `mutate()` and others

```
mtcars %>%
  group_by(cyl) %>%
  mutate(good_mpg = (mpg > mean(mpg))) %>%
  arrange(desc(mpg)) %>%
  slice(1) %>%
  select(car_name, mpg, good_mpg)
```

```
## Adding missing grouping variables: `cyl`
## # A tibble: 3 x 4
## # Groups:   cyl [3]
##     cyl car_name           mpg good_mpg
##   <dbl> <chr>            <dbl> <lgl>
## 1     4 Toyota Corolla    33.9 TRUE
## 2     6 Hornet 4 Drive    21.4 TRUE
## 3     8 Pontiac Firebird  19.2 TRUE
```

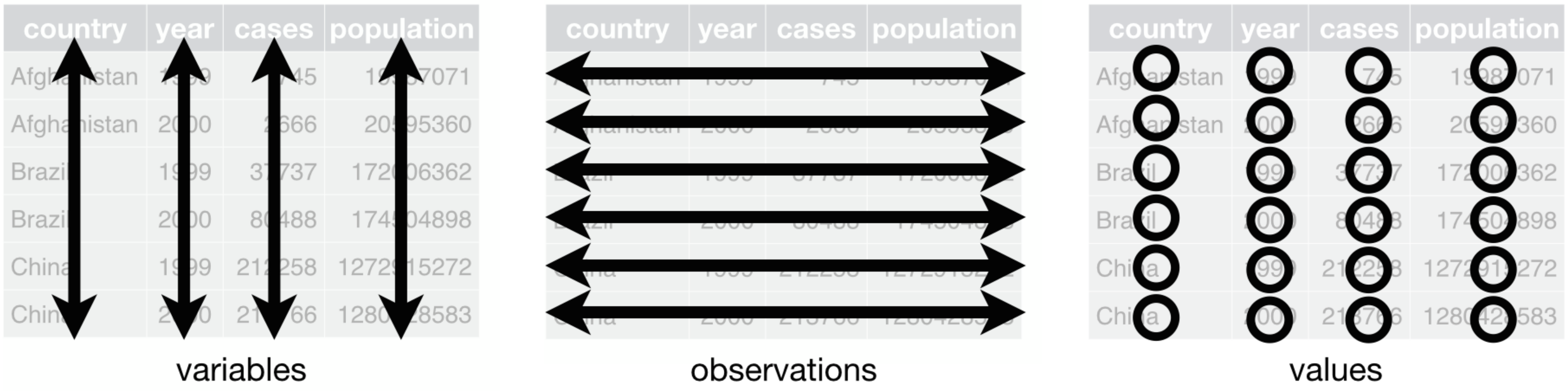
ecpr

What is tidy data?

Essentially speaking, the tidy data are

- Columns are variables
- Rows are observations

What is tidy data?



What is NOT tidy data?

These are historical household data in the US States.

```
data_income <-
read_csv("data/data_state_income.csv")
data_income %>%
select(1:5) %>%
  slice(1:6) %>%
  knitr::kable()
```

How to make it tidy?

state	Y2018	Y2017	Y2016	Y2015
Alabama	49936	50865	47221	44509
Alaska	68734	77987	75723	75112
Arizona	62283	59700	57100	52248
Arkansas	49781	49751	45907	42798
California	70489	70038	66637	63636
Colorado	73034	74984	70566	66596

The data in tidy format

This is the same data in tidy format. For that kind of reshaping between wide to long format, we can use `tidyr::pivot_longer`

state	year	income
Alabama	2018	49936
Alabama	2017	50865
Alabama	2016	47221
Alabama	2015	44509
Alabama	2014	42278
Alabama	2013	47320

The data in tidy format

– This is an code example to generate the previous pages output

```
data_income_long <- data_income %>%
  pivot_longer(Y2018:Y1984, names_to = "year",
               values_to = "income") %>%
  mutate(year = str_sub(year, 2) %>% as.integer())

data_income_long %>%
  slice(1:3)
## # A tibble: 3 x 3
##   state    year income
##   <chr>   <int>  <dbl>
## 1 Alabama  2018  49936
## 2 Alabama  2017  50865
## 3 Alabama  2016  47221
```

Going back to the wide format

```
data_income_long %>%
  pivot_wider(names_from = year, values_from = income) %>%
  slice(1:10) %>% select(1:8)
## # A tibble: 10 x 8
##    state       `2018` `2017` `2016` `2015` `2014` `2013` `2012`
##    <chr>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1 Alabama      49936  50865  47221  44509  42278  47320  43464
##  2 Alaska       68734  77987  75723  75112  67629  72472  63648
##  3 Arizona      62283  59700  57100  52248  49254  52611  47044
##  4 Arkansas     49781  49751  45907  42798  44922  39376  39018
##  5 California   70489  70038  66637  63636  60487  60794  57020
##  6 Colorado     73034  74984  70566  66596  60940  67912  57255
##  7 Connecticut  72812  74304  75923  72889  70161  69291  64247
##  8 Delaware     65012  64961  58046  57756  57522  54091  48972
##  9 D.C.         85750  81282  70982  70071  68277  60057  65246
## 10 Florida      54644  53086  51176  48825  46140  48532  46071
```

Summary

What we've learned:

- What is `tidyverse`
- Why we should learn
- `dplyr`
  - `filter(),select(),mutate()`
  - `group_by(),summarise()`
- `tidyr`
  - `pivot_wider()`

Next: data.table, reading data efficiently