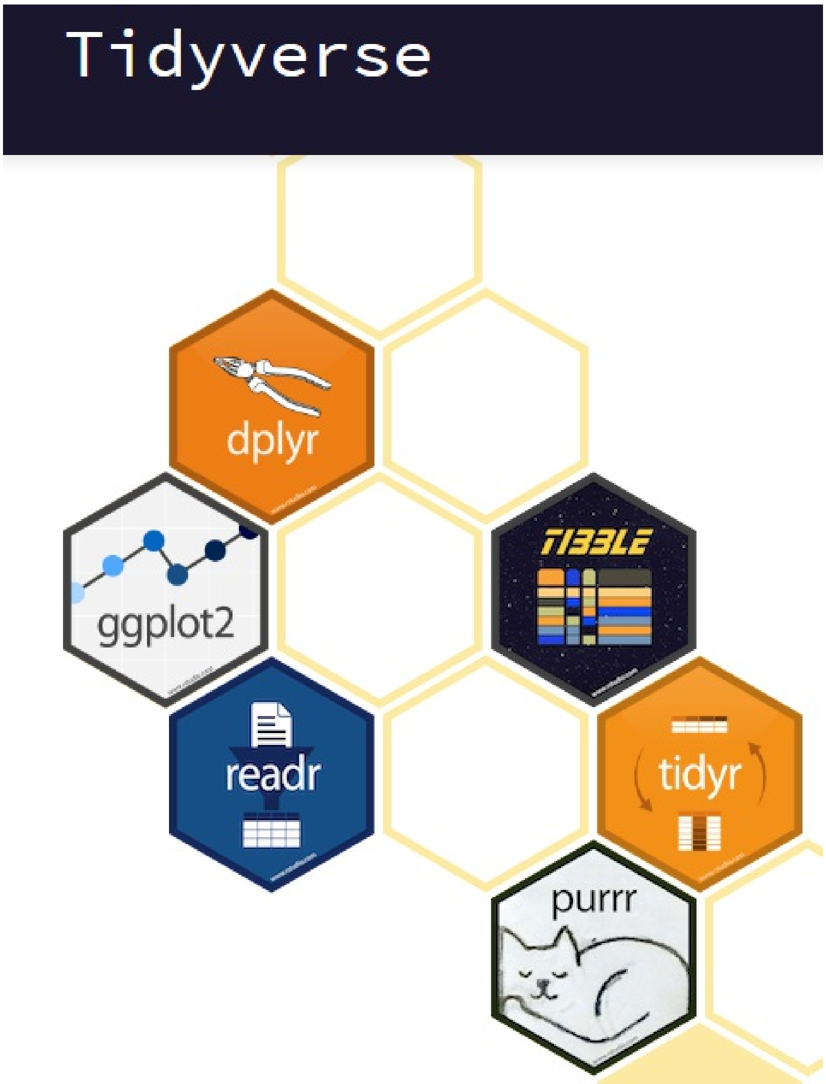
Tidvyerse Basics

Tony

2017-09-21



Introduction

Let’s tackle some of the biggest questions a person who has never heard of the “tidyverse” might ask.

What is the “tidyverse”?

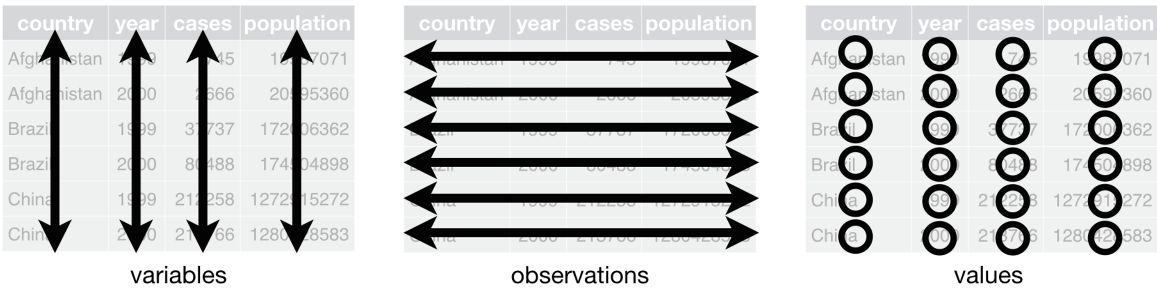
Literally, it is a set of packages that follow “tidy” data principles. The main packages include `ggplot2`, `dplyr` and `tidyr`. More information can be found at the tidyverse website.

What are “tidy” data principles?

Here’s a quote from the vignette for the tidyr package written by Hadley Wickham, who is the author of several of the tidyverse packages and one of the leaders of the tidyverse “movement”. [1]

> Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In **tidy data**:
>
> 1. Each variable forms a column.
> 2. Each observation forms a row.
> 3. Each type of observational unit forms a table.
>
> This is Codd’s 3rd normal form, but with the constraints framed in statistical language, and the focus put on a single dataset rather than the many connected datasets common in relational databases. **Messy data** is any other arrangement of the data.

To help visualize these principles, here’s an image from **R for Data Science**, a free online book co-authored by Garrett Grolemund and Hadley Wickham.



FYI: The principles of tidy data are “formally” outlined in Wickham’s paper **Tidy data**.

What does “untidy” data look like?"

Here’s a list provided by Wickham regarding the most common problems that prevent a data set from being tidy. [2]

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table.
- A single observational unit is stored in multiple tables.

These common issues might be better understood with a couple of examples.

The following example illustrates the first issue listed by Wickham: values encoded as column headers. The data set comes from a report by the Pew Research Center regarding the relationship between religion and income in the U.S.

```
pew <- tbl_df(read.csv("data/pew.csv", stringsAsFactors = FALSE, check.names = FALS
E))
pew
```

```
## # A tibble: 18 x 11
##                   religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k`
##                      <chr>   <int>     <int>     <int>     <int>     <int>
##  1                Agnostic      27        34        60        81        76
##  2                 Atheist      12        27        37        52        35
##  3                Buddhist      27        21        30        34        33
##  4                Catholic     418       617       732       670       638
##  5      Don't know/refused      15        14        15        11        10
##  6        Evangelical Prot     575       869      1064       982       881
##  7                   Hindu       1         9         7         9        11
##  8 Historically Black Prot     228       244       236       238       197
##  9       Jehovah's Witness      20        27        24        24        21
## 10                  Jewish      19        19        25        25        30
## 11           Mainline Prot     289       495       619       655       651
## 12                  Mormon      29        40        48        51        56
## 13                  Muslim       6         7         9        10         9
## 14                Orthodox      13        17        23        32        32
## 15         Other Christian       9         7        11        13        13
## 16            Other Faiths      20        33        40        46        49
## 17   Other World Religions       5         2         3         4         2
## 18            Unaffiliated     217       299       374       365       341
## # ... with 5 more variables: `$50-75k` <int>, `$75-100k` <int>,
## #   `$100-150k` <int>, `>150k` <int>, `Don't know/refused` <int>
```

Nevertheless, it can be tidied fairly easily!

```
pew %>%
  gather(income, frequency, -religion)
```

```
## # A tibble: 180 x 3
##                   religion income frequency
##                      <chr>  <chr>     <int>
##  1                Agnostic  <$10k        27
##  2                 Atheist  <$10k        12
##  3                Buddhist  <$10k        27
##  4                Catholic  <$10k       418
##  5      Don't know/refused  <$10k        15
##  6        Evangelical Prot  <$10k       575
##  7                   Hindu  <$10k         1
##  8 Historically Black Prot  <$10k       228
##  9       Jehovah's Witness  <$10k        20
## 10                  Jewish  <$10k        19
## # ... with 170 more rows
```

This next example illustrates the second issue: multiple variables in one column. This data set reflects tuberculosis information gathered by the World Health Organization.

```
tb <- tbl_df(read.csv("data/tb.csv", stringsAsFactors = FALSE))
tb
```

```
## # A tibble: 5,769 x 22
##     iso2  year   m04  m514  m014 m1524 m2534 m3544 m4554 m5564   m65    mu
##    <chr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
##  1    AD  1989    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##  2    AD  1990    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##  3    AD  1991    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##  4    AD  1992    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##  5    AD  1993    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##  6    AD  1994    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##  7    AD  1996    NA    NA     0     0     0     4     1     0     0    NA
##  8    AD  1997    NA    NA     0     0     1     2     2     1     6    NA
##  9    AD  1998    NA    NA     0     0     0     1     0     0     0    NA
## 10    AD  1999    NA    NA     0     0     0     1     1     0     0    NA
## # ... with 5,759 more rows, and 10 more variables: f04 <int>, f514 <int>,
## #   f014 <int>, f1524 <int>, f2534 <int>, f3544 <int>, f4554 <int>,
## #   f5564 <int>, f65 <int>, fu <int>
```

Although the variables for country and year (i.e. `iso2` and `year`) are already correctly encoded as columns, the variables/columns for demographics are combined together across several columns. (i.e. `m04`, `m514`, etc.) These columns implicitly store information regarding gender and age. The first letter `m` or `f` indicates male or female, and the digits indicate age ranges (e.g. `m514` indicates males who are ages 5 through 14).

Two operations are needed to tidy this data set.[3]

```
tb2 <-
  tb %>%
  gather(demo, n, -iso2, -year, na.rm = TRUE)
tb2
```

```
## # A tibble: 35,750 x 4
##     iso2  year  demo     n
##  *  <chr> <int> <chr> <int>
##  1    AD  2005   m04     0
##  2    AD  2006   m04     0
##  3    AD  2008   m04     0
##  4    AE  2006   m04     0
##  5    AE  2007   m04     0
##  6    AE  2008   m04     0
##  7    AG  2007   m04     0
##  8    AL  2005   m04     0
##  9    AL  2006   m04     1
## 10    AL  2007   m04     0
## # ... with 35,740 more rows
```

```
tb3 <-
  tb2 %>%
  separate(demo, c("sex", "age"), 1)
tb3
```

```
## # A tibble: 35,750 x 5
##     iso2  year   sex   age     n
##   * <chr> <int> <chr> <chr> <int>
##  1    AD  2005     m    04     0
##  2    AD  2006     m    04     0
##  3    AD  2008     m    04     0
##  4    AE  2006     m    04     0
##  5    AE  2007     m    04     0
##  6    AE  2008     m    04     0
##  7    AG  2007     m    04     0
##  8    AL  2005     m    04     0
##  9    AL  2006     m    04     1
## 10    AL  2007     m    04     0
## # ... with 35,740 more rows
```

More examples can be found in the tidyr vignette.

A Quick Aside… "What is this 'tibble' that I see in some examples?"

A tibble is just a data frame that is "smart" about how it is printed to the RStudio console.[4] The **R for Data Science** book explain this feature in the following manner.

> Tibbles have a refined print method that shows only the first 10 rows, and all the columns that fit on screen. This makes it much easier to work with large data. In addition to its name, each column reports its type, a nice feature borrowed from `str()`:

Why should I use tidy data principles?

1. It defines a framework for structuring data that makes analysis easier. It facilitates and, in fact, mandates **consistency**.

2. In terms of programming syntax, tidy data facilitates the use of R's vectorized programming principles. This means that performing operations (e.g. via functions like `summarise()`) on large sets of data and transforming data quickly (e.g. via functions like `mutate()`) is natural and easy.

Related to the first point is the idea of having a singular, definitive method of performing a single task. Tidyverse functions and tidy data principles promote this idea. [5] Although one may argue that having many ways of performing the same operation can be an advantage, this can also easily lead to "sloppy", irreproducible data storage and manipulation.

For example, note that there are several valid ways of manipulating a column without using tidyverse functions. (Here, I'm using the `mtcars` dataset that is automatically loaded when R is loaded.)

```
mtcars$pounds <- mtcars$wt * 1000
mtcars[["pounds"]] <- mtcars[["wt"]] / 1000
mtcars[, "pounds"] <- mtcars[, "wt"] / 1000
```

The singular "tidy" way of doing the same task is arguably easier to comprehend.

```
mtcars <-
  mtcars %>%
  mutate(pounds = wt / 1000)
```

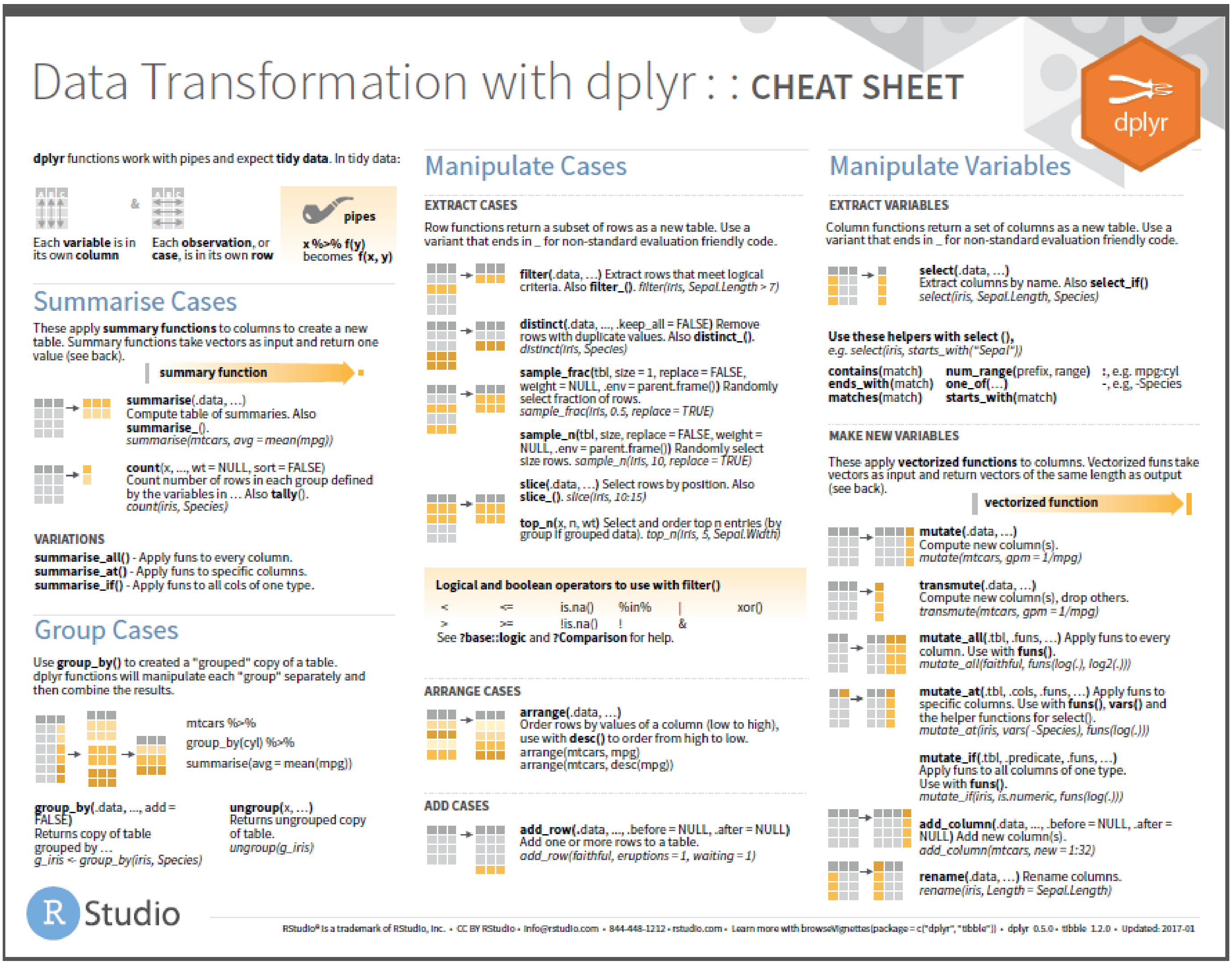
What else should I know about the tidyverse and tidy data?

In my opinion, the tidyverse is much more than just a set of packages, and tidy data is more than just data structured according to a set of principles. These are underlying constructs of a larger, more abstract **mentality** that emphasizes **readability** and **reproducibility**. These principles are coveted in the realm of data science and analysis.

By **readability**, I'm alluding to the manner in which the tidyverse naturally emphasizes data manipulation and analysis actions with verbs. For example, the vignette for the dplyr package describes its implicit implementation of this principle.

> Dplyr aims to provide a function for each basic verb of data manipulation:
>
> - `filter()` to select cases based on their values.
> - `arrange()` to reorder the cases.
> - `select()` and `rename()` to select variables based on their names.
> - `mutate()` and `transmute()` to add new variables that are functions of existing variables.
> - `summarise()` to condense multiple values to a single value.
> - `sample_n()` and `sample_frac()` to take random samples.

FYI: All of dplyr's functionality is captured succintly in the "Data Transformation" "cheat sheet" on RStudio's website. [6]



The readability provided by the verbs in tidyverse functions is complemented/facilitated by the "pipe" operation. "Piping" can easily be done using magrittr's `%>%` operator with functions in tidyverse packages. The notion of “piping” may not be completely unfamiliar to those accustomed to programming. (For example, the “+=” operator implements “piped” addition in C++.) To those unfamiliar, with piping, it is essentially the composition mathematical function (i.e. `x %>% f(y)` is equivalent to `f(x, y)`).

Combined with good code style, piping can make code self-expanatory. For example, see the following comparison of dplyr operations using the nycflight13 package. [7]

```
library("nycflights13")
dim(flights)
flights
```

```
## [1] 336776     19
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Even though the data is tidy and the code uses dplyr functions, the second set of commands using the pipe operator are inarguably easier to interpret because the operations are performed in an ordered fashion (i.e. left-to-right, top-to-bottom). (The first method is less comprehensible because the operations are performed from “inside to out”.)

```
filter(
  summarise(
    select(
      group_by(flights, year, month, day),
      arr_delay, dep_delay
    ),
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ),
  arr > 30 | dep > 30
)
```

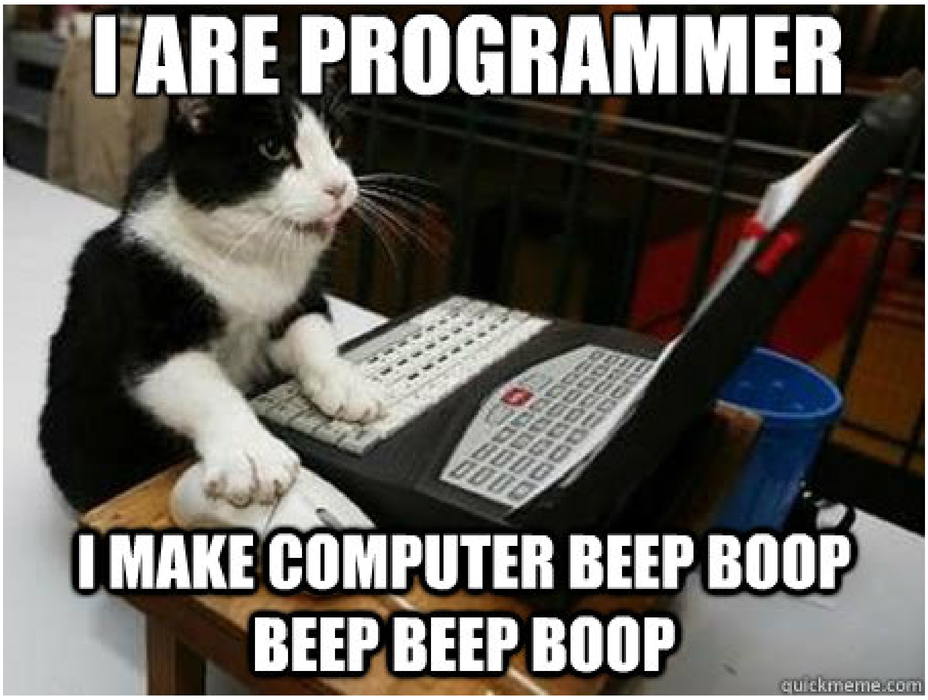
```
## # A tibble: 49 x 5
## # Groups:   year, month [11]
##      year month   day      arr      dep
##     <int> <int> <int>    <dbl>    <dbl>
##  1  2013     1    16 34.24736 24.61287
##  2  2013     1    31 32.60285 28.65836
##  3  2013     2    11 36.29009 39.07360
##  4  2013     2    27 31.25249 37.76327
##  5  2013     3     8 85.86216 83.53692
##  6  2013     3    18 41.29189 30.11796
##  7  2013     4    10 38.41231 33.02368
##  8  2013     4    12 36.04814 34.83843
##  9  2013     4    18 36.02848 34.91536
## 10  2013     4    19 47.91170 46.12783
## # ... with 39 more rows
```

```
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
```

```
## # A tibble: 49 x 5
## # Groups:   year, month [11]
##      year month   day      arr      dep
##     <int> <int> <int>    <dbl>    <dbl>
##  1  2013     1    16 34.24736 24.61287
##  2  2013     1    31 32.60285 28.65836
##  3  2013     2    11 36.29009 39.07360
##  4  2013     2    27 31.25249 37.76327
##  5  2013     3     8 85.86216 83.53692
##  6  2013     3    18 41.29189 30.11796
##  7  2013     4    10 38.41231 33.02368
##  8  2013     4    12 36.04814 34.83843
##  9  2013     4    18 36.02848 34.91536
## 10  2013     4    19 47.91170 46.12783
## # ... with 39 more rows
```

By **reproducibility**, I'm referring to the many tools available to make code documentation easy. For example, this document is actually a .Rmd document that is converted to html via the `knitr` package. It allows for code to be written alongside text, images, and other kinds of figures. Notably, I could have just as easily turned the same underlying document to a pdf simply by changing a single variable.

Why is this all even important? Why can't I just start coding?



Diving into R and learning as much as possible as quickly as possible is highly encouraged! However, it is a good idea to learn good data science principles early on in your learning so that you can completely avoid frustrations that are easily avoidable and mistakes that you might make otherwise.

Conclusion

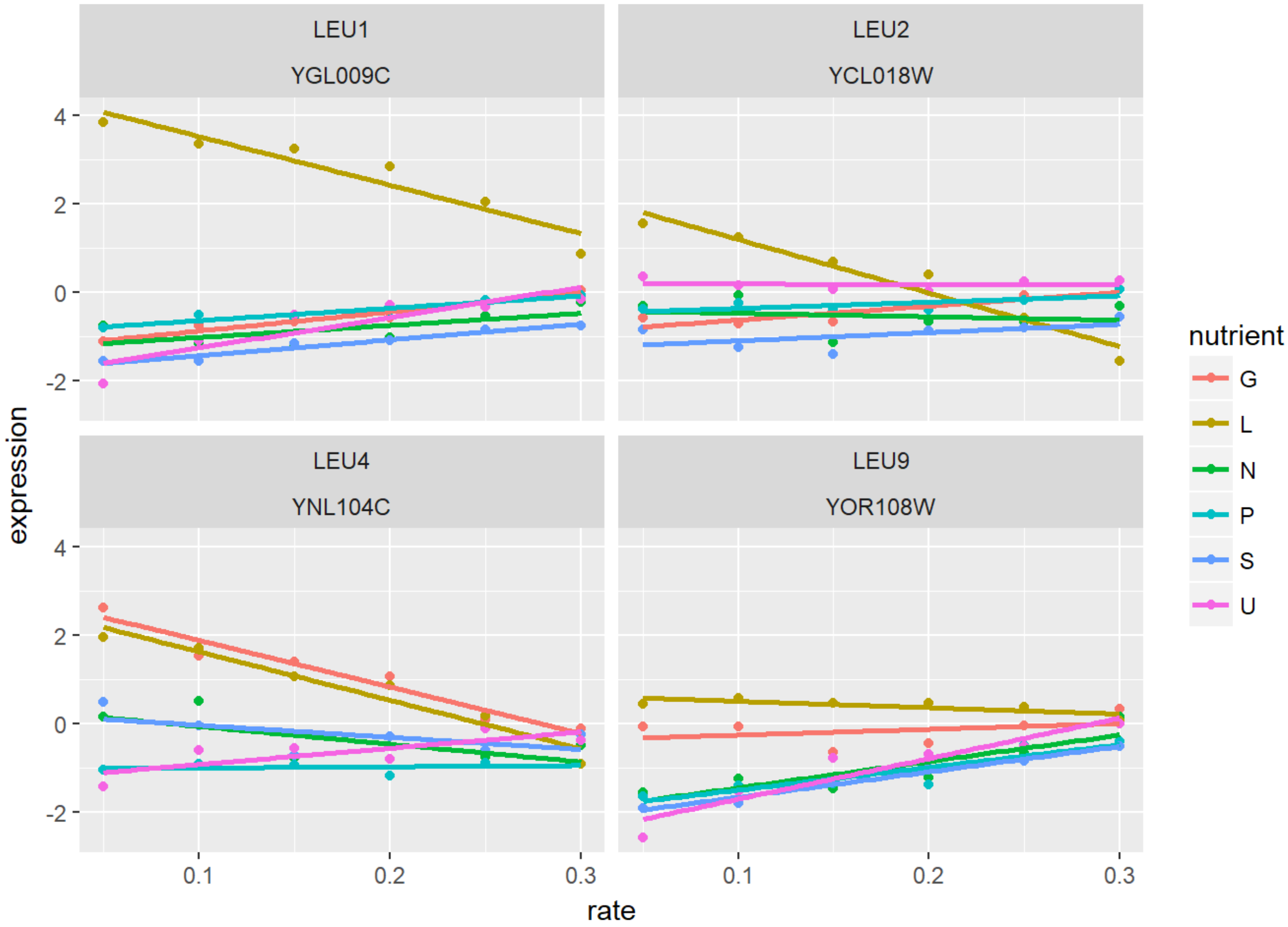
To exhibit the power and elegance of using tidyverse functions and tidy data principles, here are a couple of final examples.

This example comes from an article on David Robinson's blog.[8] The data set concerns gene expression. Note how he is able to easily manipulate and visualize the data set to gain meaningful insight.

```
url <- "http://varianceexplained.org/files/Brauer2008_DataSet1.tds"


# Clean and tidy the data
cleaned_data <- read_delim(url, delim = "\t") %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"), sep = "\\|\\|
") %>%
  mutate_at(vars(name:systematic_name), funs(trimws)) %>%
  select(-number, -GID, -YORF, -GWEIGHT) %>%
  gather(sample, expression, G0.05:U0.3) %>%
  separate(sample, c("nutrient", "rate"), sep = 1, convert = TRUE) %>%
  filter(!is.na(expression), systematic_name != "")

# Visualize a set of four genes
cleaned_data %>%
  filter(BP == "leucine biosynthesis") %>%
  ggplot(aes(rate, expression, color = nutrient)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~name + systematic_name)
```

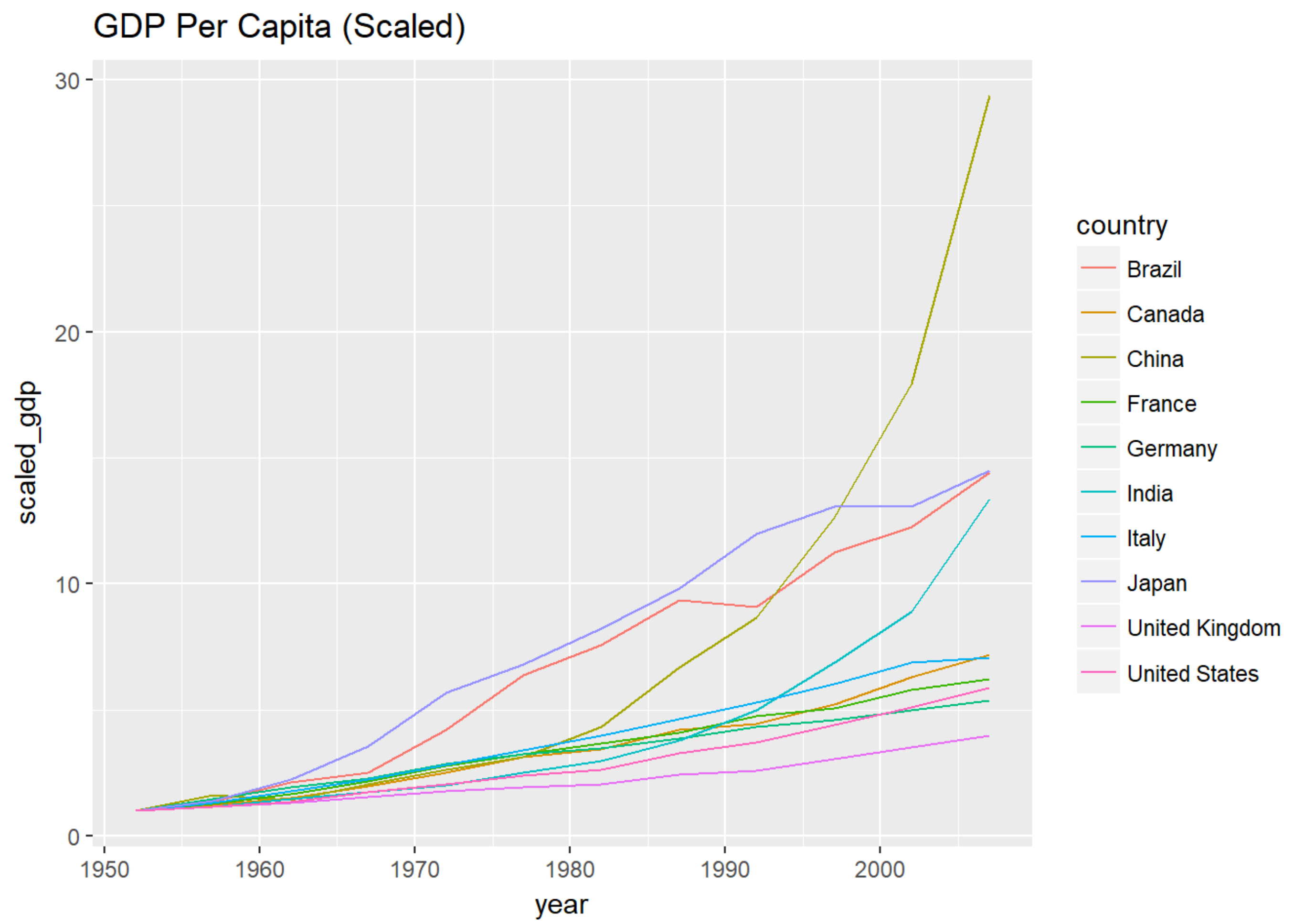


Even if you know nothing about gene expression, it should not be too difficult to understand the steps that are taken to generate a deliverable that can be used to gain understanding. The same could not necessarily be said for applying other coding styles/techniques.

This next example is adapted from a very recent RStudio webinar on the exact same topic that is discussed here–the tidyverse! [9] It uses the famous "gapminder" data set that is seen in some of Hans Rosling TEDTalks.

```
library("gapminder")
top_10 <-
  gapminder %>%
  filter(year == 1952) %>%
  mutate(gdp = pop * gdpPercap) %>%
  arrange(desc(gdp)) %>%
  top_n(10, gdp) %>%
  pull(country)
# top_10


gapminder %>%
  filter(country %in% top_10) %>%
  mutate(gdp = pop * gdpPercap) %>%
  group_by(country) %>%
  mutate(scaled_gdp = gdp / first(gdp)) %>%
  ggplot() +
    geom_line(mapping = aes(x = year, y = scaled_gdp, color = country)) +
  labs(title = "GDP Per Capita (Scaled)")
```



Again, note how powerful (yet simplistic), the tidyverse can be!

---

1. A vignette is a document that comes with a package that explains some of the functions/use cases of a package.↩
2. Guidance for the structure and code for this section is provided by the tidyr vignette.↩
3. For illustrative purposes, the intermediate result is stored. This does not exactly represent a "best practice".↩
4. Subsetting is also slightly different with tibbles.↩
5. See David Robinson's breakdown of the "base R vs. tidyr" argument for more information.↩
6. Many more helpful R "cheat sheets" can be found at https://www.rstudio.com/resources/cheatsheets/.↩
7. This nycflights13 code is borrowed from the dplyr vignette.↩
8. This article discusses the "base R vs. tidyr" argument in regards to how to teach beginners. He makes a strong case in favor of tidyr principles, or, at the least, not avoiding it completely.↩
9. See https://github.com/rstudio/webinars/tree/master/46-tidyverse-visualisation-and-manipulation-basics to download the materials. Also, see https://www.rstudio.com/resources/webinars/ for RStudio's webinars.↩