Module 2: Reporting, Data Wrangling and Graphing (I)

Siyue Yang

04/26/2022

Outline

We will review R, Rstudio, and Syntax of R together.

- LaTeX/Markdown
- Tidy data, processing (tidyverse)
- Graphing (ggplot2)

LaTeX and Markdown

LaTeX is useful for documents with mathematical formulas.

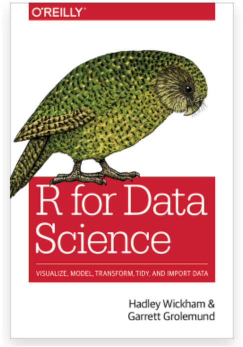
- Overleaf - an online, collaborative LaTeX editor
- LaTeX mathematical symbols
- Inline equation e.g. (`$\alpha$`) returns $\alpha$
- Equation e.g. (`$$e = mc^2$$`) returns

$$e = mc^2$$

Markdown is appealing for formatting, e.g. headings, bold text, text with codes, . . .

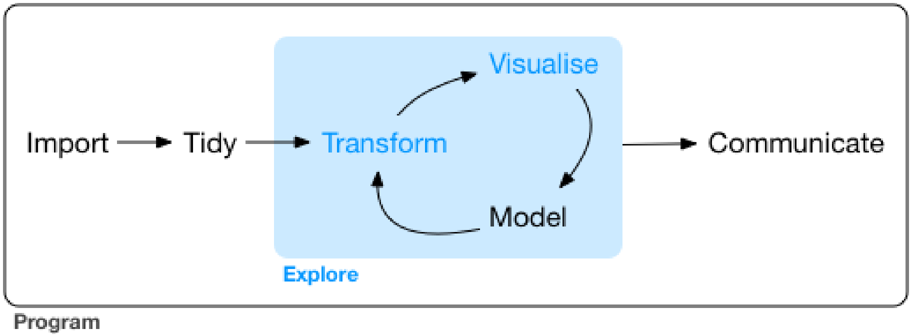
Resources

"R for Data Science: Import, Tidy, Transform, Visualize, and Model Data" by Hadley Wickham.



Let's code!

Data science project workflow:



Data import

```r
df <- read.table("mtcars.txt", header = TRUE)
head(df) # Show the first 6 rows.
```

```
##   Cntry lper100k weight length
## 1    US     19.8   2178   5.92
## 2 Japan      9.9   1026   4.32
## 3    US     10.8   1188   4.27
## 4    US     12.5   1444   5.11
## 5    US     12.5   1485   5.03
## 6    US     12.5   1485   5.03
```

Other options

CSV files.

- read.csv() in the base r.
- read.csv() in "readr" package (much faster).
- fread() in "data.table" package (much more faster).

Rdata.

- load() in the base r.

Tidy data

The goal is to clean the dataset so it is much easier to use.

Specifically,

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

We will focus on the functions from “tidyverse” package.

```
library(tidyverse)
```

Tidy data 1: pivoting

For a dataset having column names are not names of variables, but values of a variable, e.g.

```
table4a
```

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

- Need to change 1999, 2000 to a column named as “year”.
- Need to change the values of 1999, 2000 as “cases”.

We can use `pivot_longer()` from the “tidyverse” package.

Pivot longer

```
table4a %>%
  pivot_longer(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
```

Another example

```
table2 %>% head(5)
```

```
## # A tibble: 5 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
```

- case and population are two variables and should be converted into columns.

We can use `pivot_wider()`.

Pivot wider

```r
table2 %>%
    pivot_wider(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
```

Transform data

Use the “pipes” from the “tidyverse” package, a powerful tool for clearly expressing a sequence of multiple operations, with the combination of the following functions:

- `select()`
- `filter()`
- `arrange()`
- `mutate()`
- `summarise()`
- `group_by()`

Dataset - Diamonds

A dataset containing the prices and other attributes of almost 54,000 diamonds.

```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

Select

Use `select()` to get a column, e.g. "color"

```
diamonds %>%
  select(color) %>%
  head()
```

```
## # A tibble: 6 x 1
##   color
##   <ord>
## 1 E
## 2 E
## 3 E
## 4 I
## 5 J
## 6 J
```

```
# Equivalent to...
head(diamonds$color)
```

```
## [1] E E E I J J
## Levels: D < E < F < G < H < I < J
```

Select

Use `select()` to remove a column, e.g. “color”

```
diamonds %>%
  select(-color)
```

```
## # A tibble: 53,940 x 9
##    carat cut       clarity depth table price     x     y     z
##    <dbl> <ord>     <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

```
# Need to assign the change to the original dataset, otherwise, the deletion won't affect the dataset.
diagmonds <- diamonds %>%
  select(-color)
```

Filter

Use `filter()` to filter by some condition, e.g. filter all price > 335

```
diamonds %>%
  filter(price > 335)
```

```
## # A tibble: 53,935 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  2  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  3  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  4  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
##  5  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
##  6  0.3  Good      J     SI1      64      55   339  4.25  4.28  2.73
##  7  0.23 Ideal     J     VS1      62.8    56   340  3.93  3.9   2.46
##  8  0.22 Premium   F     SI1      60.4    61   342  3.88  3.84  2.33
##  9  0.31 Ideal     J     SI2      62.2    54   344  4.35  4.37  2.71
## 10  0.2  Premium   E     SI2      60.2    62   345  3.79  3.75  2.27
## # ... with 53,925 more rows
```

Filters with multiple conditions

```
diamonds %>%
  filter(price > 335 & depth < 64)
```

```
## # A tibble: 51,849 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  2  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  3  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  4  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
##  5  0.23 Ideal     J     VS1      62.8    56   340  3.93  3.9   2.46
##  6  0.22 Premium   F     SI1      60.4    61   342  3.88  3.84  2.33
##  7  0.31 Ideal     J     SI2      62.2    54   344  4.35  4.37  2.71
##  8  0.2  Premium   E     SI2      60.2    62   345  3.79  3.75  2.27
##  9  0.32 Premium   E     I1       60.9    58   345  4.38  4.42  2.68
## 10  0.3  Ideal     I     SI2      62      54   348  4.31  4.34  2.68
## # ... with 51,839 more rows
```

```
diamonds %>%
  filter(cut == "Very Good" | cut == "Fair")
```

```
## # A tibble: 13,692 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  2  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  3  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  4  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
##  5  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
##  6  0.3  Very Good J     SI1      62.7    59   351  4.21  4.27  2.66
##  7  0.23 Very Good E     VS2      63.8    55   352  3.85  3.92  2.48
##  8  0.23 Very Good H     VS1      61      57   353  3.94  3.96  2.41
##  9  0.31 Very Good J     SI1      59.4    62   353  4.39  4.43  2.62
```

Filter after select

This is an example of “a sequence of operations”.

```
diamonds %>%
  select(price) %>%
  filter(price > 335)
```

```
## # A tibble: 53,935 x 1
##    price
##    <int>
##  1   336
##  2   336
##  3   337
##  4   337
##  5   338
##  6   339
##  7   340
##  8   342
##  9   344
## 10   345
## # ... with 53,925 more rows
```

Arrange

Use `arrange()` to order data.

```
diamonds %>%
  arrange(price)
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

Arrange descending order

e.g. from the cheapest!

```
diamonds %>%
  arrange(-price)
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  2.29 Premium   I     VS2      60.8    60 18823  8.5   8.47  5.16
##  2  2    Very Good G     SI1      63.5    56 18818  7.9   7.97  5.04
##  3  1.51 Ideal     G     IF       61.7    55 18806  7.37  7.41  4.56
##  4  2.07 Ideal     G     SI2      62.5    55 18804  8.2   8.13  5.11
##  5  2    Very Good H     SI1      62.8    57 18803  7.95  8     5.01
##  6  2.29 Premium   I     SI1      61.8    59 18797  8.52  8.45  5.24
##  7  2.04 Premium   H     SI1      58.1    60 18795  8.37  8.28  4.84
##  8  2    Premium   I     VS1      60.8    59 18795  8.13  8.02  4.91
##  9  1.71 Premium   F     VS2      62.3    59 18791  7.57  7.53  4.7 
## 10  2.15 Ideal     G     SI2      62.6    54 18791  8.29  8.35  5.21
## # ... with 53,930 more rows
```

Arrange by multiple conditions

```
diamonds %>%
  arrange(price, cut)
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  2  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
##  9  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

Filter, select, arrange

```r
diamonds %>%
  filter(table < 340) %>%
  select(carat, cut, price) %>%
  arrange(price, cut)
```

```
## # A tibble: 53,940 x 3
##    carat cut       price
##    <dbl> <ord>     <int>
##  1  0.21 Premium     326
##  2  0.23 Ideal       326
##  3  0.23 Good        327
##  4  0.29 Premium     334
##  5  0.31 Good        335
##  6  0.24 Very Good   336
##  7  0.24 Very Good   336
##  8  0.22 Fair        337
##  9  0.26 Very Good   337
## 10  0.23 Very Good   338
## # ... with 53,930 more rows
```

Mutate

Create new variables using `mutate()`.

- Create a boolean variable, 0 = not affordable, 1 = affordable.

```
diamonds %>%
  mutate(affordable = price < 400)
```

```
## # A tibble: 53,940 x 11
##    carat cut       color clarity depth table price     x     y     z affordable
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <lgl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43 TRUE
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31 TRUE
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31 TRUE
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63 TRUE
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75 TRUE
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48 TRUE
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47 TRUE
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53 TRUE
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49 TRUE
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39 TRUE
## # ... with 53,930 more rows
```

Mutate (cont'd)

- Create a variable containing string with `case_when()`:

```
diamonds %>%
  mutate(affordable = case_when(price<400 ~ "affordable",
                                TRUE ~ "not affordable"))
```

```
## # A tibble: 53,940 x 11
##    carat cut       color clarity depth table price     x     y     z affordable
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43 affordable
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31 affordable
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31 affordable
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63 affordable
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75 affordable
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48 affordable
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47 affordable
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53 affordable
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49 affordable
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39 affordable
## # ... with 53,930 more rows
```

Group by and Summarise

Use `group_by` and `summarise` to group variables:

```
diamonds %>%
  group_by(cut) %>%
  summarise(n = n())
```

```
## # A tibble: 5 x 2
##   cut           n
##   <ord>     <int>
## 1 Fair       1610
## 2 Good       4906
## 3 Very Good 12082
## 4 Premium   13791
## 5 Ideal     21551
```

More examples

```
diamonds %>%
  group_by(cut) %>%
  summarise(n = n(), price_avg = mean(price))
```

```
## # A tibble: 5 x 3
##   cut           n price_avg
##   <ord>     <int>     <dbl>
## 1 Fair       1610     4359.
## 2 Good       4906     3929.
## 3 Very Good 12082     3982.
## 4 Premium   13791     4584.
## 5 Ideal     21551     3458.
```

Proportions

```
diamonds %>%
  group_by(cut) %>%
  summarise(n = n(), price_avg = mean(price)) %>%
  ungroup() %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 5 x 4
##   cut           n price_avg   prop
##   <ord>     <int>     <dbl>  <dbl>
## 1 Fair       1610     4359. 0.0298
## 2 Good       4906     3929. 0.0910
## 3 Very Good 12082     3982. 0.224
## 4 Premium   13791     4584. 0.256
## 5 Ideal     21551     3458. 0.400
```

With percentage

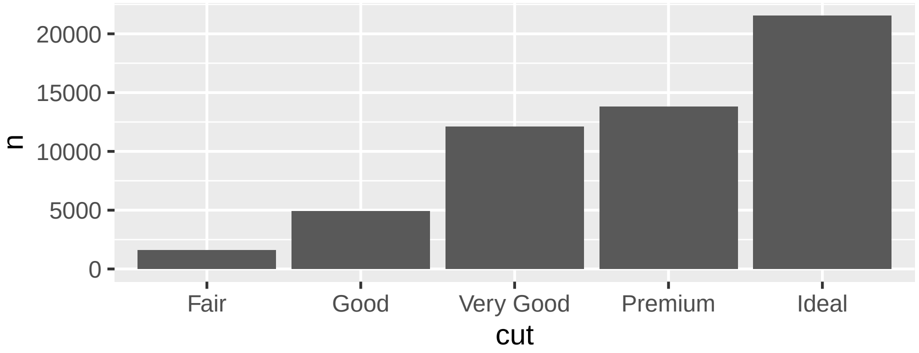
Use `scales::percent()` to add %.

```
diamonds %>%
  group_by(cut) %>%
  summarise(n = n(), price_avg = mean(price)) %>%
  ungroup() %>%
  mutate(prop = scales::percent(n/sum(n)))
```

```
## # A tibble: 5 x 4
##   cut           n price_avg prop
##   <ord>     <int>     <dbl> <chr>
## 1 Fair       1610     4359. 3.0%
## 2 Good       4906     3929. 9.1%
## 3 Very Good 12082     3982. 22.4%
## 4 Premium   13791     4584. 25.6%
## 5 Ideal     21551     3458. 40.0%
```

Graphing after transformation

```r
diamonds %>%
  group_by(cut) %>%
  summarise(n = n(), price_avg = mean(price)) %>%
  ggplot() +
  geom_bar(aes(x = cut, y = n), stat = "identity")
```



ggplot

Here we used functions from "ggplot2" package. Same pattern as "tidyverse", but using "+" to connect.

How to write?

- Specify the data using `ggplot(data = diamonds)`
- Specify the x-/y-axis, `ggplot(data = diamonds, mapping = aes(x = cut))`
- Specify the types of plots with `geom`, e.g. `+ geom_bar()`

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```

More plots

- `geom_histogram()`, `geom_density()`, `geom_line()`, `geom_point()`
- `geom_facet()` generates subplots
- color package
  - "RColorBrewer"
  - "ggsci"

Resources

This module is based on

- Brendan R. E. Ansell's “Introduction to R - tidyverse” [link]