R 프로그래밍 #8

2019. 05. 03

한국생명공학연구원
김하성

In previous lectures

- To understand what's programming
  - Built the function for reading excel files

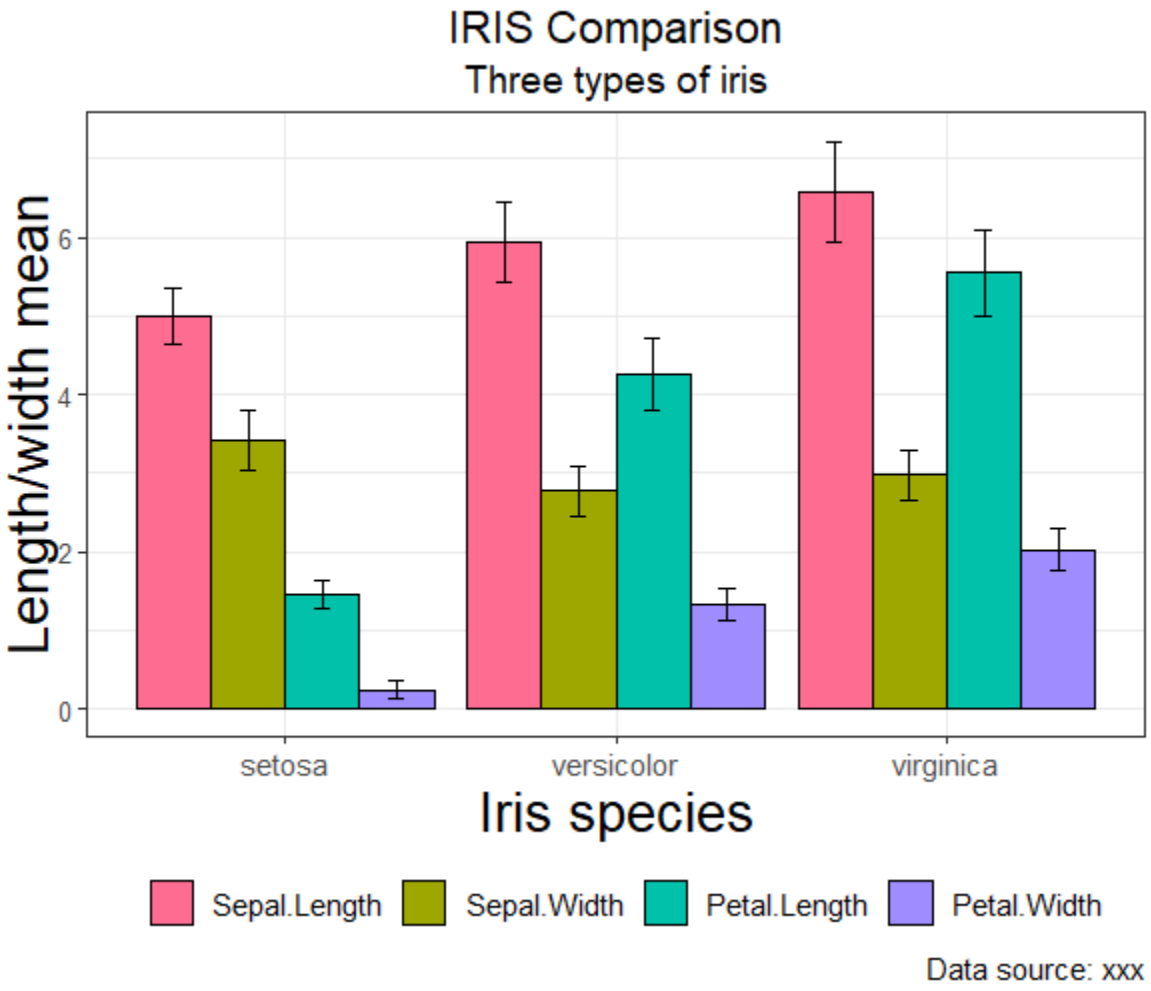
- Exercised how to manipulate and to visualize a dataset
  - ggplot2
    - geom_bar, geom_line
  - dplyr
    - %>%, group_by, summarize

In today's lecture

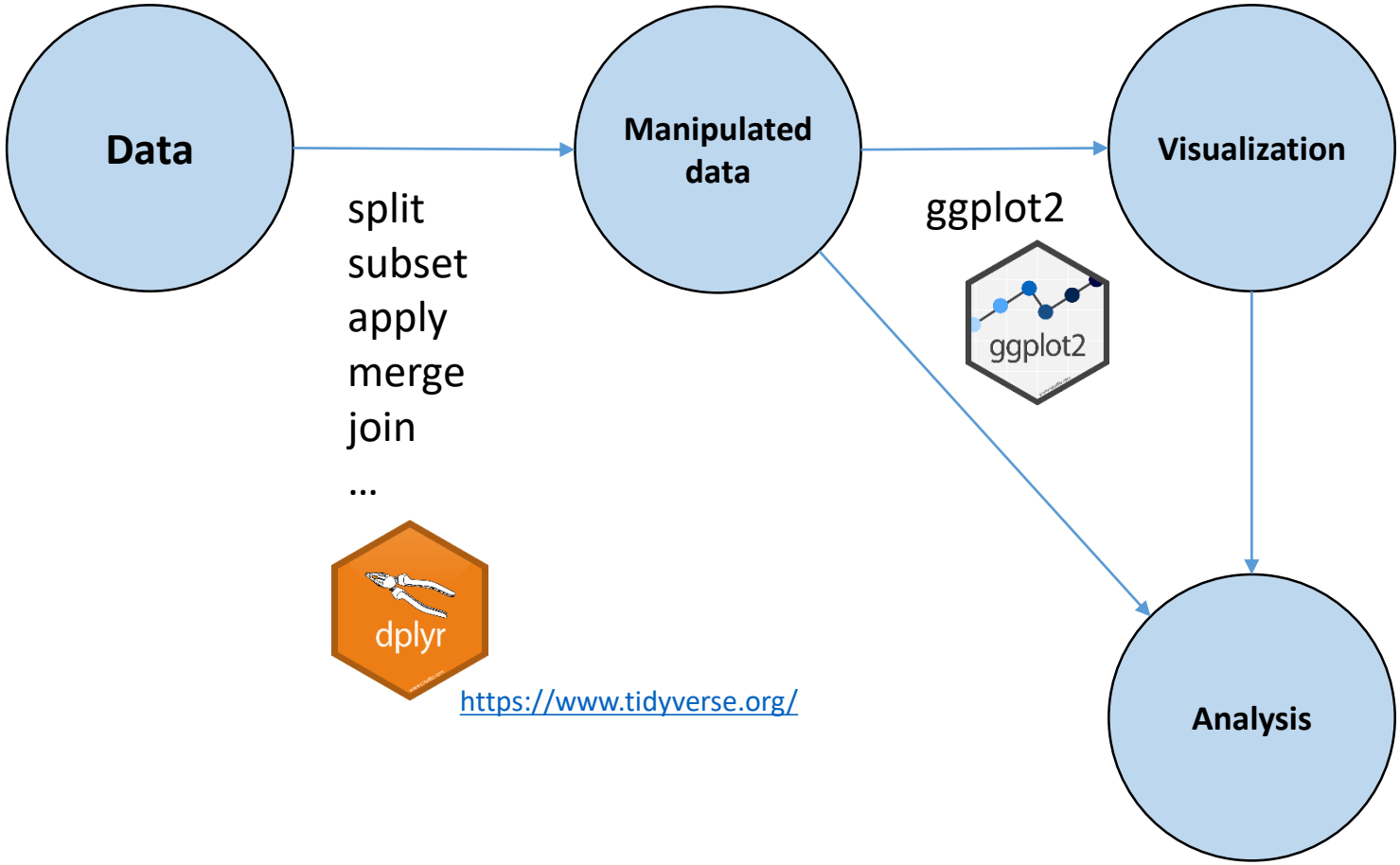
- Exercised how to manipulate and to visualize a dataset
  - ggplot2
    - geom_bar, geom_line
    - geom_errorbar, scale, theme
  - dplyr
    - %>%, group_by, summarize
    - mutate, select, join
  - reshape2

https://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html

Start at the end



Data analysis in R



Introducing dplyr

Hadley Wickham

2014-01-17

Categories: Packages

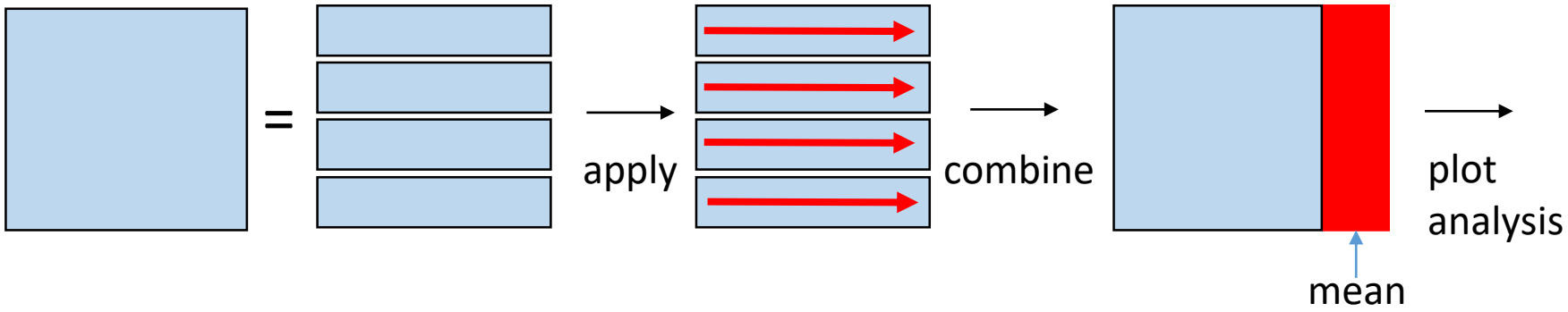
`dplyr` is a new package which provides a set of tools for efficiently manipulating datasets in R. `dplyr` is the next iteration of `plyr`, focussing on only data frames. `dplyr` is faster, has a more consistent API and should be easier to use. There are three key ideas that underlie `dplyr`:

1. Your time is important, so Romain Francois has written the key pieces in Rcpp to provide blazing fast performance. Performance will only get better over time, especially once we figure out the best way to make the most of multiple processors.

2. Tabular data is tabular data regardless of where it lives, so you should use the same functions to work with it. With `dplyr`, anything you can do to a local data frame you can also do to a remote database table. PostgreSQL, MySQL, SQLite and Google bigquery support is built-in; adding a new backend is a matter of implementing a handful of S3 methods.

3. The bottleneck in most data analyses is the time it takes for you to figure out what to do with your data, and dplyr makes this easier by having individual functions that correspond to the most common operations (`group_by`, `summarise`, `mutate`, `filter`, `select` and `arrange`). Each function does one only thing, but does it well.

The Pipe Operator: %>% (dplyr package)



- %>% takes the output of its lhs statement and makes it the input of the rhs (next) statement

$$f(x) == x \%>\% f$$

- Short cut in Rstudio: Shift + Ctl + m (Alt+_ for <-)
- Placeholder . operator

exercise 8-1) basic plot / dplyr

```
df <- data.frame(x=rnorm(100)+1, y=rnorm(100)+3)
```

1. plot two histogram of x and y with -4 < x < 8 scale
2. can you plot two histogram together in a canvas?
3. make a new variable std_x by standardizing x
4. make a new variable std_y by standardizing y
5. make a new data.frame variable std_df by combining std_x and std_y
6. Use mutate() and select () for 3~4
7. Use %>% for 6

```
standardized values = (values – mean(values)) / sd(values)
```

ggplot grammar

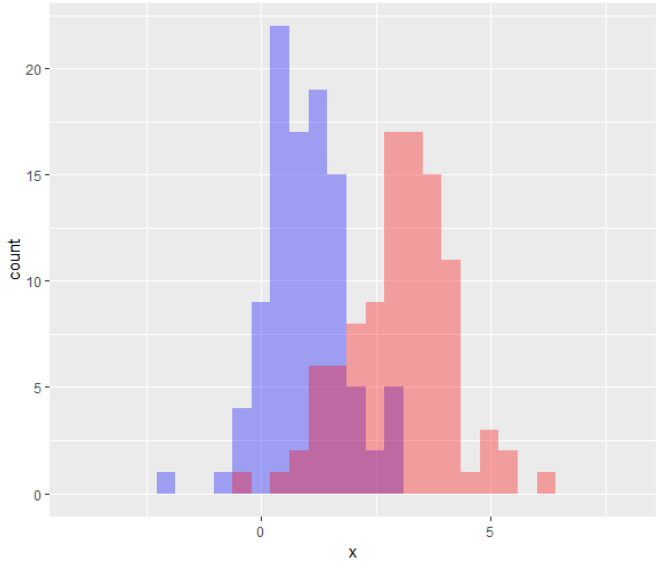
- Components
  - data frame (ggplot)
  - aesthetic factors such as color, size, etc (aes)
  - geometric factors such as point, line, bar, etc (geoms)
  - statistical factors (stats)
  - theme or scale to be used in aes
- Aesthetic Mapping (something you can see)
  - position (x and y axes)
  - color (outside color)
  - fill (inside color)
  - shape (of points)
  - linetype
  - size

  ** Each type of geom accepts only a subset of all aesthetics

exercise 8-2) ggplot

1. plot two histogram of x and y with -4 < x < 8 scale using ggplot
2. can you plot two histogram together in a canvas?

X axis	Height of bar represents	Common name
Continuous	Count (bin)	Histogram
Discrete	Count	Bar graph
Continuous	Value (identity)	Bar graph
Discrete	Value (identity)	Bar graph



iris dataset

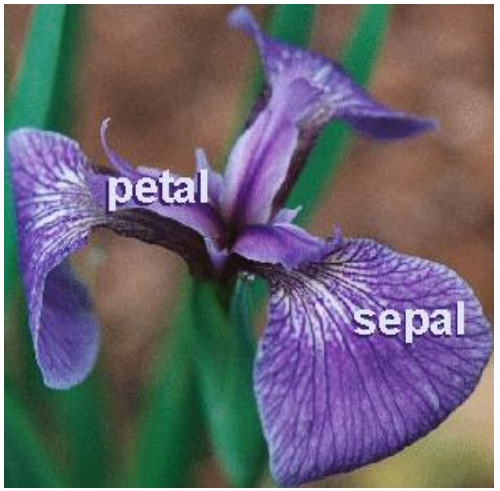
Iris Versicolor

Iris Setosa

Iris Virginica

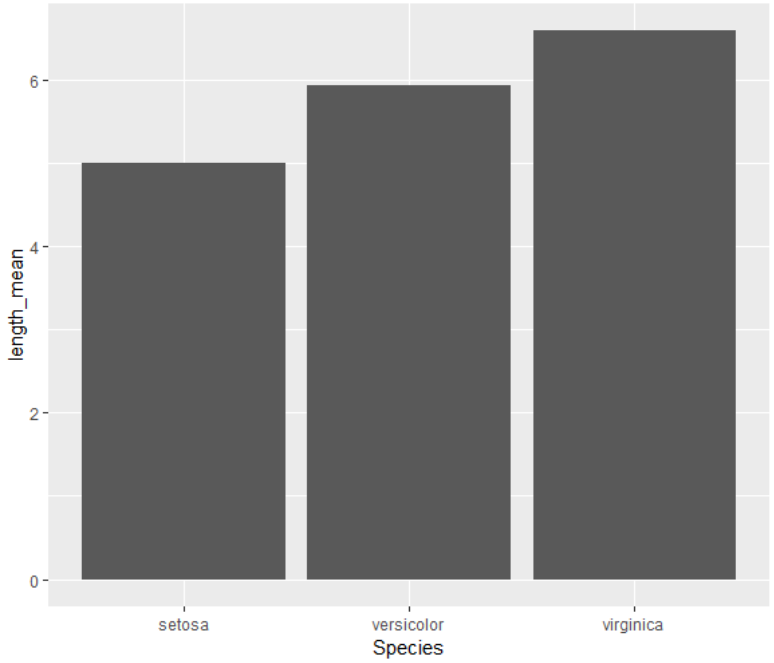
Fisher's/Anderson's iris data set:
measurements (cm) of the sepal length and width and petal length and width (4 features) for 50 flowers from each of 3 species (*Iris setosa*, *versicolor*, and *virginica*)

```
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```



Mean of sepal length

```
iris_grouped <- group_by(iris, Species)
iris_grouped_summary <- summarize(iris_grouped,
                                  length_mean=mean(Sepal.Length),
                                  length_sd=sd(Sepal.Length))

ggplot(iris_grouped_summary, aes(x=Species, y=length_mean)) +
  geom_bar(stat="identity")
```



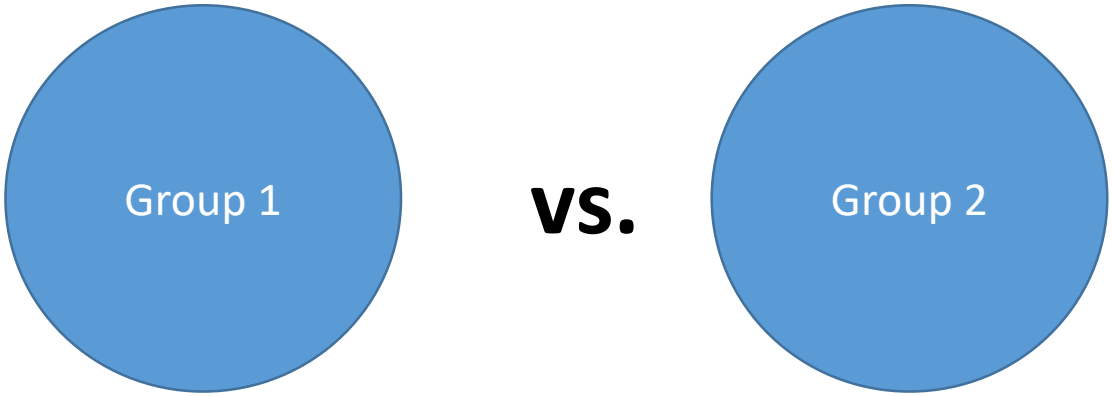
summarize_all

```
iris_grouped <- group_by(iris, Species)
iris_grouped_summary <- summarize_all(iris_grouped, mean)

iris_grouped <- iris %>% group_by(Species)
iris_grouped_summary <- iris %>% group_by(Species) %>% summarize_all(mean)
```

```
> iris_grouped_summary
# A tibble: 3 x 5
  Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>             <dbl>       <dbl>        <dbl>       <dbl>
1 setosa             5.01        3.43         1.46       0.246
2 versicolor         5.94        2.77         4.26       1.33
3 virginica          6.59        2.97         5.55       2.03
```

Quiz 8-1) Data structure

- Which object or data structure will you use to compare two groups of datasets?
- How many variables do we need for the comparison in this example?



Data structure for data analysis

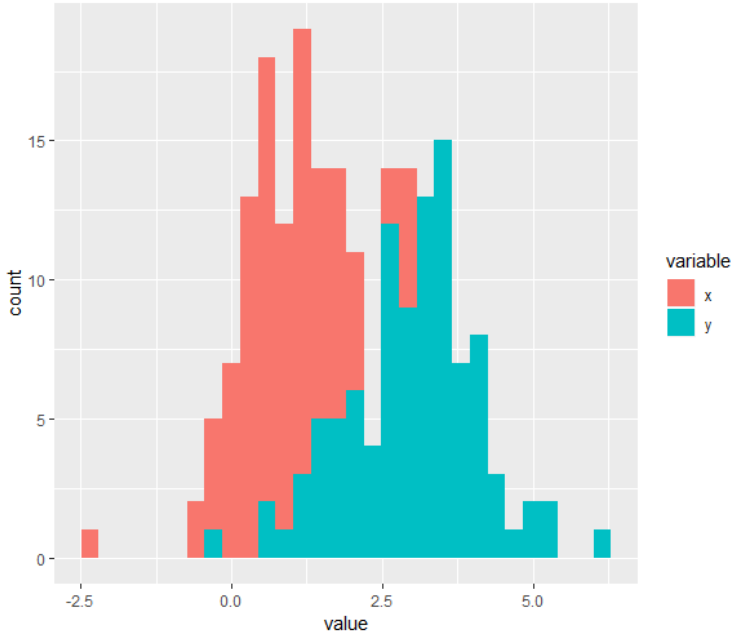
x	y
1	1
2	6
5	7
7	8

=

variable	value
x	1
x	2
x	5
x	7
y	1
y	6
y	7
y	8

Melt data

```
library(reshape2)

class(df)
df_melt <- melt(df)
ggplot(df_melt, aes(x=value, fill=variable)) +
  geom_bar(stat="bin")
```

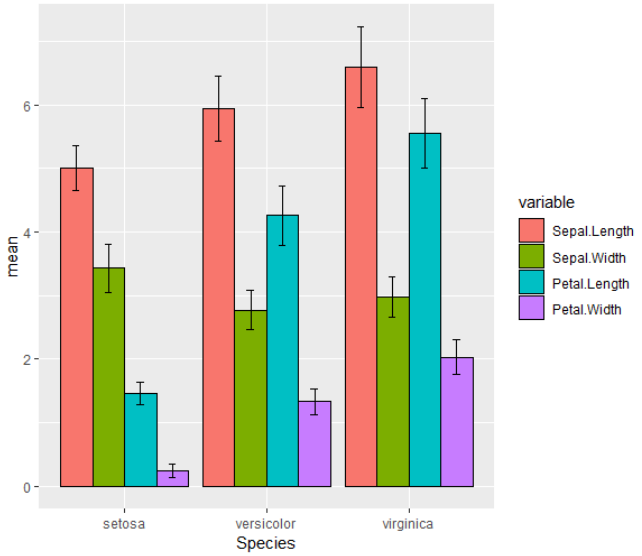


iris data summary and melt

```
library(reshape2)

iris_grouped <- group_by(iris, Species)
iris_grouped_summary <- summarize_all(iris_grouped, mean)

iris_grouped <- iris %>% group_by(Species)
iris_grouped_summary <- iris %>% group_by(Species) %>% summarize_all(mean)

iris_grouped_summary2 <- melt(iris_grouped_summary)
?melt.data.frame

ggplot(iris_grouped_summary2, aes(x=Species, y=value, group=variable)) +
  geom_bar(stat="identity", position="dodge")

ggplot(iris_grouped_summary2, aes(x=Species, y=value, group=variable, fill=variable)) +
  geom_bar(stat="identity", position="dodge")
```

Draw error bars

```
## sd
iris_grouped <- iris %>% group_by(Species)
iris_grouped_mean <- iris_grouped %>% summarize_all(mean) %>% melt(value.name="mean")
iris_grouped_sd <- iris_grouped %>% summarize_all(sd) %>% melt(value.name="sd")
iris_final <- inner_join(iris_grouped_mean, iris_grouped_sd, by=c("Species", "variable"))

ggplot(iris_final, aes(x=Species, y=mean, fill=variable)) +
  geom_bar(stat="identity", position="dodge")

p1 <- ggplot(iris_final, aes(x=Species, y=mean, fill=variable)) +
  geom_bar(stat="identity", position="dodge", color="black") +
  geom_errorbar(aes(ymin=mean-sd, ymax=mean+sd), width=.2, position=position_dodge(0.9))
```

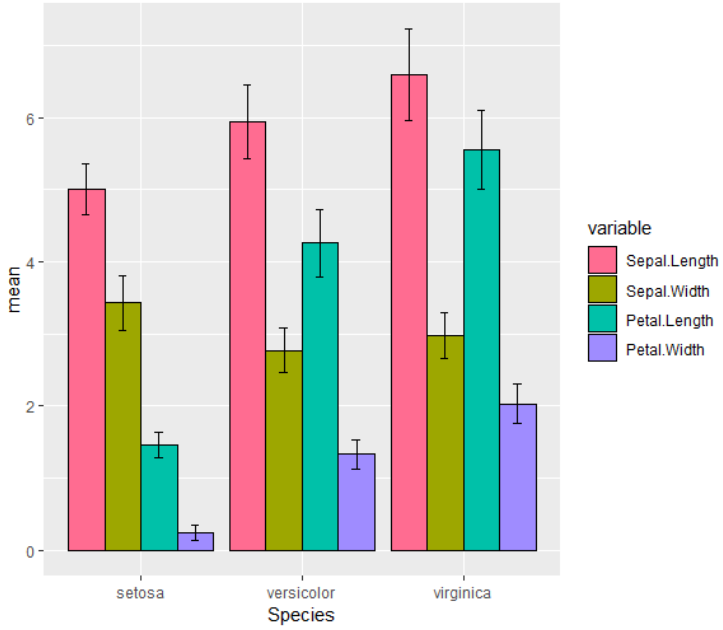


Scale

- aes mapping data to variable. But no detail indications
  - position
  - color and fill
  - size
  - shape
  - line type
- Scales are modified with a series of functions using a scale_<aesthetic>_<type> naming scheme. Try typing scale_<tab> to see a list of scale modification functions.
- Common scale arguments
  - name: the first argument gives the axis or legend title
  - limits: the minimum and maximum of the scale
  - breaks: the points along the scale where labels should appear
  - labels: the labels that appear at each break

Scale

```
p1 + scale_fill_brewer(palette = "Greens")

p1 + scale_fill_hue(h = c(0, 360))

p2 <- p1 + scale_fill_hue(h = c(0, 360)) +
  scale_y_continuous(name="Length/width Mean") +
  scale_x_discrete(name="Iris species")

p2 <- p1 + scale_fill_hue(h = c(0, 360)) +
  ylab("Length/width mean") +
  xlab("Iris species") +
  labs(title = "IRIS Comparison", subtitle="Three types of iris",
caption="Data source: xxx")

p2 <- p1 + scale_fill_hue(h = c(0, 360)) +
  ylab("Length/width mean") +
  xlab("Iris species") +
  labs(title = "IRIS Comparison", subtitle="Three types of iris",
caption="Data source: xxx", fill="")
```



Theme

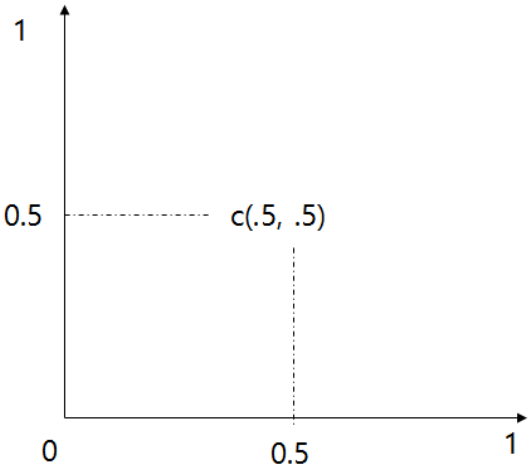
- Axis labels
- Plot background
- Facet label backround
- Legend appearance

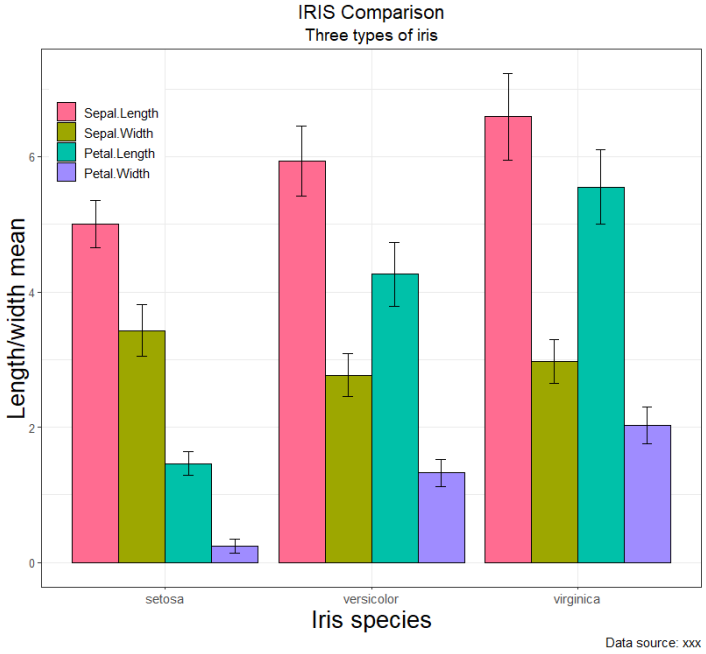
Usage

```
theme(line, rect, text, title, aspect.ratio, axis.title, axis.title.x,
  axis.title.x.top, axis.title.x.bottom, axis.title.y, axis.title.y.left,
  axis.title.y.right, axis.text, axis.text.x, axis.text.x.top,
  axis.text.x.bottom, axis.text.y, axis.text.y.left, axis.text.y.right,
  axis.ticks, axis.ticks.x, axis.ticks.x.top, axis.ticks.x.bottom,
  axis.ticks.y, axis.ticks.y.left, axis.ticks.y.right, axis.ticks.length,
  axis.line, axis.line.x, axis.line.x.top, axis.line.x.bottom, axis.line.y,
  axis.line.y.left, axis.line.y.right, legend.background, legend.margin,
  legend.spacing, legend.spacing.x, legend.spacing.y, legend.key,
  legend.key.size, legend.key.height, legend.key.width, legend.text,
  legend.text.align, legend.title, legend.title.align, legend.position,
  legend.direction, legend.justification, legend.box, legend.box.just,
  legend.box.margin, legend.box.background, legend.box.spacing,
  panel.background, panel.border, panel.spacing, panel.spacing.x,
  panel.spacing.y, panel.grid, panel.grid.major, panel.grid.minor,
  panel.grid.major.x, panel.grid.major.y, panel.grid.minor.x,
  panel.grid.minor.y, panel.ontop, plot.background, plot.title,
  plot.subtitle, plot.caption, plot.tag, plot.tag.position, plot.margin,
  strip.background, strip.background.x, strip.background.y,
  strip.placement, strip.text, strip.text.x, strip.text.y,
  strip.switch.pad.grid, strip.switch.pad.wrap, ..., complete = FALSE,
  validate = TRUE)
```

Theme overriding

```
p2 + theme_bw() +
  theme(
    text=element_text(size=14),
    axis.text.y = element_text(size=10),
    axis.title.y = element_text(size=20),
    axis.title.x = element_text(size=20),
    #legend.position = "bottom",
    legend.position = c(0.1,0.9),
    plot.title=element_text(hjust=0.5),
    plot.subtitle=element_text(hjust=0.5)
    )
```





Dataset for exercise

Experiment conditions

- **Cell types: 1~4**
- **Drug type: 1 (phenol)**
- **Drug concentrations: 11 points**
- **Replications: 4 times**

Dataset

```
setwd("C:\\Rprog\\07")

source("read_plate.R")

design_file_name <- "exp_design2.xlsx"
data_file_names <- c("20171012-phenol-1.xls",
                     "20171012-phenol-2.xls",
                     "20171227-phenol-1.xls",
                     "20171227-phenol-2.xls")

mydata1 <- multiple_plate_excel_reader2(design_file_name, data_file_names[1], sheet4design=1)
mydata2 <- multiple_plate_excel_reader2(design_file_name, data_file_names[2], sheet4design=2)
mydata3 <- multiple_plate_excel_reader2(design_file_name, data_file_names[3], sheet4design=3)
mydata4 <- multiple_plate_excel_reader2(design_file_name, data_file_names[4], sheet4design=4)

mydata <- rbind(mydata1, mydata2, mydata3, mydata4)
```

```
> head(mydata)
  well_names        OD    GFP sample_names replication drugname concentration
1        G02 0.9042823 124002            1           1   phenol         0e+00
2        F02 0.9368631 127999            1           1   phenol         5e-02
3        E02 0.9228352  44070            1           1   phenol         5e-01
4        D02 0.8994368   4280            1           1   phenol         5e+00
5        C02 0.9145258   3928            1           1   phenol         5e+01
6        B02 0.9241626   3882            1           1   phenol         5e+02
> dim(mydata)
[1] 308   7
```

Dataset

Experiment conditions

Cell types: 1~6

Drug type: 1 (phenol)

Drug concentrations: 11 points

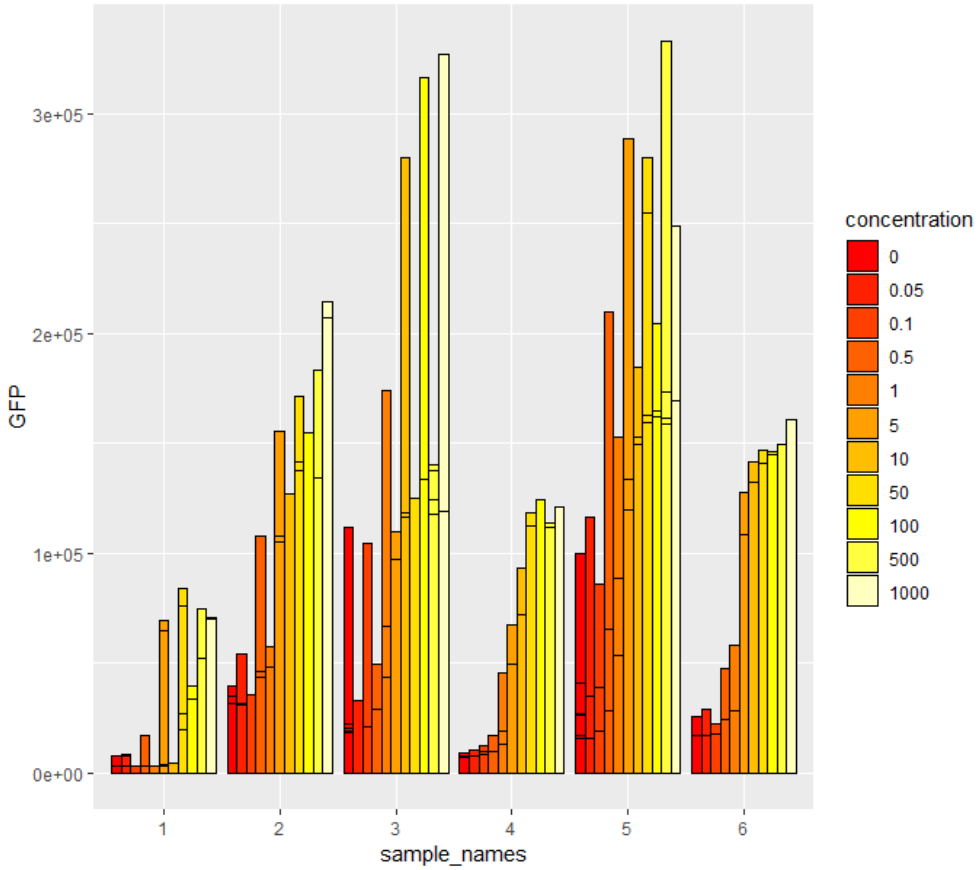
Replications: 4 times

```
> head(mydata)
  well_names        OD    GFP sample_names replication drugname concentration
1        G02 0.9042823 124002            1           1   phenol          0e+00
2        F02 0.9368631 127999            1           1   phenol          5e-02
3        E02 0.9228352  44070            1           1   phenol          5e-01
4        D02 0.8994368   4280            1           1   phenol          5e+00
5        C02 0.9145258   3928            1           1   phenol          5e+01
6        B02 0.9241626   3882            1           1   phenol          5e+02
> dim(mydata)
[1] 308   7
```

```
> str(mydata2)
'data.frame':   308 obs. of  7 variables:
 $ well_names   : chr  "G02" "F02" "E02" "D02" ...
 $ OD           : num  0.904 0.937 0.923 0.899 0.915 ...
 $ GFP          : num  124002 127999 44070 4280 3928 ...
 $ sample_names : Factor w/ 6 levels "1","2","3","4",..: 1 1 1 1 1 1 2 2 2 2 ...
 $ replication  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
 $ drugname     : Factor w/ 1 level "phenol": 1 1 1 1 1 1 1 1 1 1 ...
 $ concentration: Factor w/ 11 levels "0","0.05","0.1",..: 1 2 4 6 8 10 1 2 3 3 ...
```

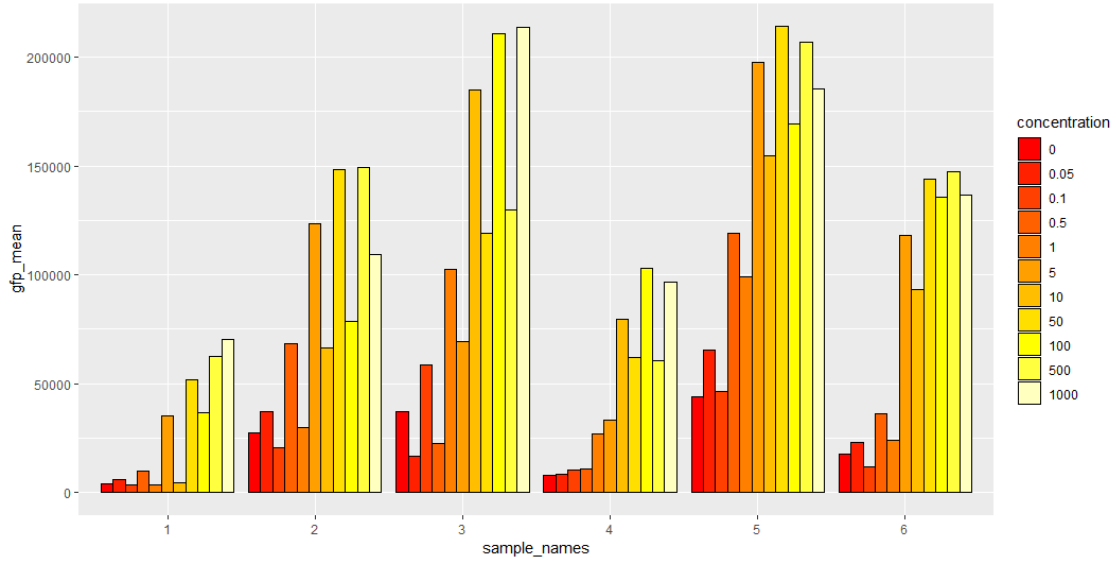
barplot - ggplot

```
mydata2 <- mydata
mydata2$concentration <- as.factor(mydata2$concentration)

ggplot(data=mydata2, aes(x=sample_names, y=GFP, fill=concentration)) +
  geom_bar(stat="identity", position="dodge", color="black") +
  scale_fill_manual(values = heat.colors(11))
```



Plot gfp mean values

```
grouped_data <- group_by(mydata2, sample_names, drugname, concentration)
grouped_data_mean <- summarize(grouped_data, gfp_mean=mean(GFP))
ggplot(grouped_data_mean, aes(x=sample_names, y=gfp_mean, fill=concentration)) +
  geom_bar(stat="identity", position="dodge", color="black") +
  scale_fill_manual(values = heat.colors(11))
```




```
save(mydata2, file="mydata2.Rdata")
load(mydata2)
```

Next

- Let’s finish the barplot of our dataset!
- Install packages
  - shiny
  - Biostrings
  - rentrez