Introduction to programming in R: Day 3

Kanhu Charan Moharana

November 27, 2019

Learning objectives

- Functions
  - User defined functions
  - using anonymous function in *apply()*
- Introduction to Bioconductor
  - Bioconductor in genomic research
  - Common packages
- Data wrangling
  - aggregate()
  - other functions()
- Plotting using ggplot
- From here to where?

User defined functions

- One of the great strengths of R is the user's ability to add functions.
- many of the built-in functions in R are actually functions of functions.

```
myfunction <- function(arg1, arg2, ... ){

  statement 1
  statement 2
  statements ...

  return(object)
}
```

- Objects in the function are local to the function
- can return only one object
    - object returned can be any data type.

Example: to calculate area of the circle

- Create a function that accepts radius of a circle and returns area of the circle.

```r
# area of a circle =  pi * r^2
area_circle <- function(r){
  area <- pi * r^2;
  return(area);
}

area_circle(1)
```

```
## [1] 3.141593
```

contd...

Example

```r
## Using the function with a vector
x <- c(1,0.5,12)
area_circle(x)
```

```
## [1]   3.1415927   0.7853982 452.3893421
```

```r
## using with sapply()
sapply( x, area_circle )
```

```
## [1]   3.1415927   0.7853982 452.3893421
```

Example: function for calculating area of a traingle

- Create a function that accepts height, width and base of a triangle and returns area.

```r
# area of a traingle =  height * base * width
area_triangle <- function(h ,b ,w){
  area <- h*b*w;
  return(area);
}

area_triangle(12,4,9)
```

```
## [1] 432
```

contd..

Example: function for calculating area of a traingle

```r
## equal length vectors can be used as input
h <- c(3,4,10,9)
b <- c(5,6,10,19)
w <- c(1,10,12,8)

area_triangle(h=h, b=b, w=w)
```

```
## [1]   15  240 1200 1368
```

User defined functions: return()

- the `return()` state ment returns the final processed value
  - multiple values can be packages as
  1. Named vector
  2. Named List
  3. Data frame

Return different values: a named vector

```
## return as a named vector
circle <- function(r){
  area <- pi * r^2; # pi x r^2
  circumference <- 2 * pi * r; # 2 x pi x r
  result <- c(area=area, circumference=circumference )
  return(result);
}
##Use the function to calculate for r=1
circle(r=1)
```

```
##          area circumference
##      3.141593      6.283185
```

```
##Use the function to calculate for r= a vector
circle(r=c(1,0.5))
```

```
##          area1          area2 circumference1 circumference2
##      3.1415927      0.7853982      6.2831853      3.1415927
```

Return different values:as a named list

```r
## return as a named vector
circle <- function(r){
  area <- pi * r^2; # pi x r^2
  circumference <- 2 * pi * r; # 2 x pi x r
  result <- list(
    area=area,
    circumference=circumference
    )
  return(result);
}
##Use the function to calculate for r=1
circle(r=1);
```

```
## $area
## [1] 3.141593
##
## $circumference
```

Return different values: As a named data frame

```
## return as a named vector
circle <- function(r){
  area <- pi * r^2; # pi x r^2
  circumference <- 2 * pi * r; # 2 x pi x r
  result <- data.frame(
    area=area,
    circumference=circumference
    )
  return(result);
}
# Use the function
circle(r=1);
```

```
##       area circumference
## 1 3.141593      6.283185
```

```
circle(c(1,0.5))
```

Anonymous functions

- anonymous functions are not saved with a name.
- we do not want to reuse the code
- Usually written inline and used with *apply* functions

```
marks <- matrix(sample(1:100, 20,replace=T), nrow=4)
colnames(marks) <- c('Physics','Chemistry','Math','Geography',
rownames(marks) <- c('foo','bar','baz','qux')
marks <- as.data.frame(marks)
head(marks)
```

```
##     Physics Chemistry Math Geography Literature
## foo      40        89   63        52         14
## bar       7        70   12        50         71
## baz      34        53   78        37         62
## qux       7         4   45        87         14
```

Anonymous functions

- using with apply() to calulate percentage

```
marks$Percentage <- apply(marks, 1,
      function(x){
       return(100*sum(x)/500)
       }
      )
head(marks)
```

```
##     Physics Chemistry Math Geography Literature Percentage
## foo      40        89   63        52         14       51.6
## bar       7        70   12        50         71       42.0
## baz      34        53   78        37         62       52.8
## qux       7         4   45        87         14       31.4
```

Anonymous functions using `paste()`

```r
genes <- c('AT04G1245','AT01G9987','AT01G9861',
           'AT01G8761','ATCh12301')

## Add a Prefix 'Ath|' to all genes
pre_genes <- sapply(genes, function(x){
   return(paste('Ath',x,sep='|') )
  } )



pre_genes
```

```
##       AT04G1245       AT01G9987       AT01G9861       AT01C
## "Ath|AT04G1245" "Ath|AT01G9987" "Ath|AT01G9861" "Ath|AT01G8
##       ATCh12301
## "Ath|ATCh12301"
```

Anonymous functions using `strsplit()`

```
## Split Ath from the pre_genes elements

sapply(pre_genes, function(x){
   return(strsplit(x,split="|", fixed=TRUE)[[1]][2] )
  } )
```

```
##    AT04G1245   AT01G9987   AT01G9861   AT01G8761   ATCh1230
## "AT04G1245" "AT01G9987" "AT01G9861" "AT01G8761" "ATCh12301
```

Data Wrangling

- Manipulating raw data tables in to more structured format.

1. Subletting tables
2. Sorting tables
3. Merging two tables
4. Aggregating Data
5. Reshaping tables

Sorting tables

- To sort a vector/dataframe in R, use the *order()* function.
  - returns the vector indexes
  - Default: sorting is ASCENDING.
  - With minus sign (-) Pre-fixed to a variable indicates DESCENDING order.

```r
x <- c(10,8,6,8,14)
x
```

```
## [1] 10  8  6  8 14
```

```r
# Returns increasing indexes
order(x)
```

```
## [1] 3 2 4 1 5
```

```r
x[order(x)]
```

```
## [1]  6  8  8 10 14
```

Sorting tables

```r
df <- data.frame(
  A=c(10,2,18,50),
  B=c(4,2,12,10),
  C=c(1,3,9,5)
)
df
```

```
##    A  B C
## 1 10  4 1
## 2  2  2 3
## 3 18 12 9
## 4 50 10 5
```

Sorting tables

```
df[order(df$A, decreasing=TRUE), ]
```

```
##     A  B C
## 4 50 10 5
## 3 18 12 9
## 1 10  4 1
## 2  2  2 3
```

Compute Summary Statistics of Data Subsets

- *aggregate()* : computes summary statistics for each category

```
women_data <- read.csv('Data/women.tsv', sep="\t")
# head(women_data, 10)

### Calculate mean Heights for each Age category

aggregate(
  x=women_data[, c('Height','Weight')] ,
  by=list(Women_age=women_data$Age
        ),
  mean)
```

```
##   Women_age   Height    Weight
## 1     adult 63.49032 165.84553
## 2     child 53.90876  92.57694
```

Merging Data tables

- *merge ()*: merge two data frames (datasets) horizontally by one or more common key variables (i.e., an inner join).

```
## Gene table
exprs <- data.frame(Gene=c('FLC', 'FT', 'AG', 'LFY', 'CO', 'AP
                    TPM= c(0.5,2.3,12,9.1,21.03,14)
                    )
head(exprs)
```

```
##   Gene   TPM
## 1  FLC  0.50
## 2   FT  2.30
## 3   AG 12.00
## 4  LFY  9.10
## 5   CO 21.03
## 6  AP1 14.00
```

Merging Data tables

```r
## Gene descriptions
desc <- data.frame(Gene=c('FLC', 'FT', 'AG', 'LFY',
                          'CO', 'AP1','EIN2','ABI1', 'ABI3'),
                   AA_len= c(143,144,51,291,
                             101,50,33,90,105),
                   Description=c('Repressor of flowers',
                     'Mobile floral stimulus protein',
                     'Regulator of floral identity',
                      'Regulator of flowering',
                      'Regulator of flowering',
                      'Regulator of floral organ identity',
                      'Ethylene receptor',
                      'Abscisic acid signaling repressor',
                      'Abscisic acid response regulator')
                   )
head(desc)
```

Merging Data tables

Merged table

```r
head(merge(exprs,desc, by="Gene"))
```

```
##   Gene   TPM AA_len                    Description
## 1   AG 12.00     51       Regulator of floral identity
## 2  AP1 14.00     50 Regulator of floral organ identity
## 3   CO 21.03    101            Regulator of flowering
## 4  FLC  0.50    143              Repressor of flowers
## 5   FT  2.30    144      Mobile floral stimulus protein
## 6  LFY  9.10    291            Regulator of flowering
```

Subsetting using %in% operator

- %in% is a very handy operator to match two vectors.
  - x %in% y: compares each elements of x in its left vector y
  - returns a logical vector if a value from x is found in y

```
x <- c('a','c','b','p','o','x')
y <- c('a','b','x','c','d','e')
x %in% y
```

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

```
# y %in% x  ## try
## Subset a named dataframe
exprs[exprs$Gene %in% c('AP1','FT','LFY'), ]
```

```
##   Gene  TPM
## 2   FT  2.3
## 4  LFY  9.1
## 6  AP1 14.0
```

Data arrangements

Wide and Two-Dimensional: The Spreadsheet

```
df <- data.frame(
  Genes=c('Gene1','Gene2','Gene3','Gene4','Gene5'),
  Sample1=sample(seq(0,10, by=1.083), size = 5, replace = T),
  Sample2=sample(seq(0,10, by=2.1), size = 5, replace = T),
  Sample3=sample(seq(0,10, by=3.2), size = 5, replace = T)
)


df
```

```
##   Genes Sample1 Sample2 Sample3
## 1 Gene1   6.498     4.2     0.0
## 2 Gene2   1.083     8.4     3.2
## 3 Gene3   7.581     2.1     0.0
## 4 Gene4   9.747     4.2     9.6
## 5 Gene5   9.747     4.2     6.4
```

Data arrangements

Long and Skinny: The Database

```
## Using Genes as id variables
```

```
head(df2)
```

```
##   Genes  Sample Expression
## 1 Gene1 Sample1      6.498
## 2 Gene2 Sample1      1.083
## 3 Gene3 Sample1      7.581
## 4 Gene4 Sample1      9.747
## 5 Gene5 Sample1      9.747
## 6 Gene1 Sample2      4.200
```

Data arrangements

- Wide format is more human readable and easy to interpret
- Long format is best for data analysis
  - suitable for ggplot2 plots

`https://eagereyes.org/basics/spreadsheet-thinking-vs-database-thinking`

Package: reshape2

- to convert Spreadsheet tables to long Database format and vice versa.
  - melt() : wide => long
  - cast() : long => wide

```
install.packages('reshape2')
```

Melting and Casting DFs

melt()

```r
library(reshape2)
## use melt function
melt(df, id="Genes")
```

```
##    Genes variable value
## 1  Gene1  Sample1 6.498
## 2  Gene2  Sample1 1.083
## 3  Gene3  Sample1 7.581
## 4  Gene4  Sample1 9.747
## 5  Gene5  Sample1 9.747
## 6  Gene1  Sample2 4.200
## 7  Gene2  Sample2 8.400
## 8  Gene3  Sample2 2.100
## 9  Gene4  Sample2 4.200
## 10 Gene5  Sample2 4.200
## 11 Gene1  Sample3 0.000
```

Melting and Casting DFs

cast()

```
## Genes ~ Sample: Describe Genes vs Sample
dcast(df2, formula=Genes ~ Sample)
```

```
## Using Expression as value column: use value.var to overrid
##   Genes Sample1 Sample2 Sample3
## 1 Gene1   6.498     4.2     0.0
## 2 Gene2   1.083     8.4     3.2
## 3 Gene3   7.581     2.1     0.0
## 4 Gene4   9.747     4.2     9.6
## 5 Gene5   9.747     4.2     6.4
```

Data visualization using ‘ggplot2’

- *de facto* standard for data visualization.
- Design publication quality plots with less typing .
- Gives fine control over graphical elements using functions (‘grammar’) to build final image
- Create images as layers; easy to reuse previous codes.

The graphical grammar and its components

Every graphical plot has following components
- Data
The raw data table you want to plot
- Layer
The plots (points, text, lines)
- Scale
Maps data to the graphical output.
- Coordinates
Visualization perspective (rotation, angle)
- Faceting
Sub grouping the plots by categories.
- Themes
Display details (Font, background color, font size)

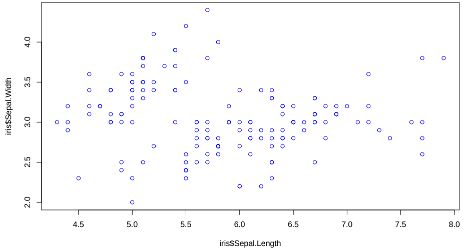
Working with ggplo2 grammar

- **Data:** Raw data in Database format.
    - required
- **Aesthetics:** Mapping your data to the visualization
    - required
- **Layers:** Creates the real visualization using the data and aesthetics provided.
    - at least one layer is required.
    - usually have a pre-fix **geom_<layer name>**

Create a simple XY scatter plot

Scatter plot using base R

```
plot(
  x=iris$Sepal.Length,
  y=iris$Sepal.Width,
  col='blue'
  )
```

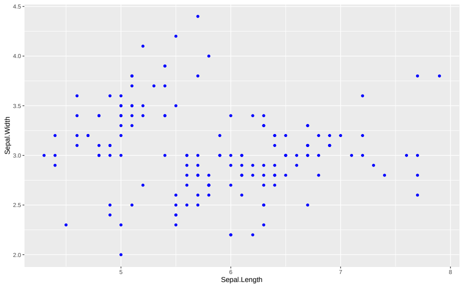


Create a simple XY scatter plot

Scatter plot using ggplot2 lib

```r
# import library
library(ggplot2)

# Data and define aesthetics
p <- ggplot(data=iris,
            aes(x=Sepal.Length, y=Sepal.Width )
            )

# add (+) scatterplot layer to visualize
p + geom_point(color='blue')
```

Create a simple XY scatter plot



Apply categorical Colors

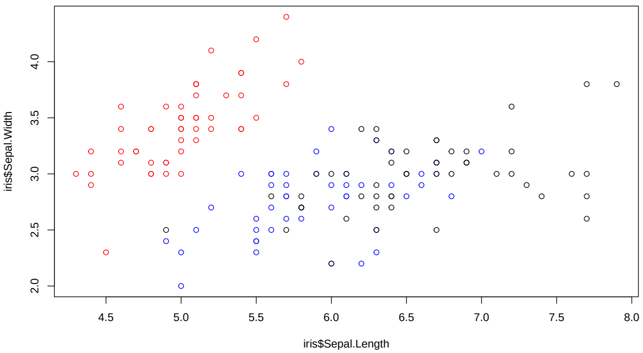
Base R plot

```
# color to each species
levels(iris$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

```
color_point <- function(x){
  ifelse(x=="setosa",'red',
         ifelse(x=="versicolor",'blue','black')
  )
  }
plot(
  x=iris$Sepal.Length,
  y=iris$Sepal.Width,
  col= sapply(iris$Species,color_point )
  )
```

Apply categorical Colors

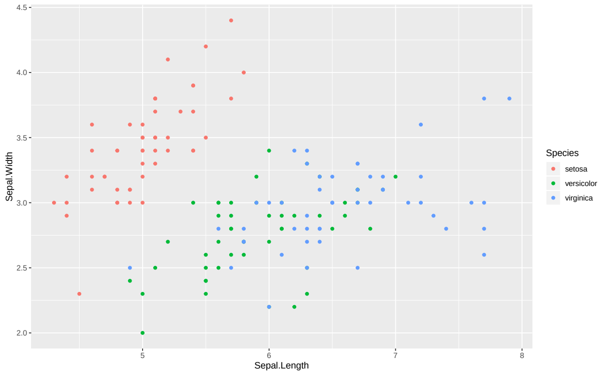


Apply categorical Colors

Using ggplot2

```r
# import library
library(ggplot2)

# Data and define aesthetics (x, y, color)
p <- ggplot(data=iris,
            aes(x=Sepal.Length, y=Sepal.Width, color=Species)
            )

# add (+) scatterplot layer to visualize
p + geom_point()
```

Apply categorical Colors



Add user-defined colors

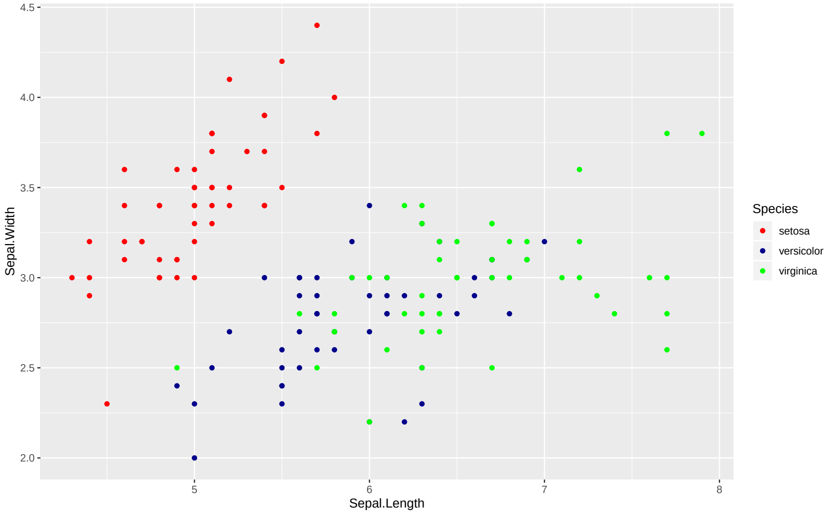
```r
# define colors for each category as a vector
species_color <- c('red','green','darkblue')
## Define category names to each color
names(species_color) <- c("setosa","virginica","versicolor")
species_color
```

```
##     setosa  virginica versicolor
##      "red"    "green" "darkblue"
```

- use this vector to fill color

```r
# Use this mappings to scale_color_manual() layer
p + geom_point() +
  scale_color_manual(values =species_color)
```

Add user-defined colors

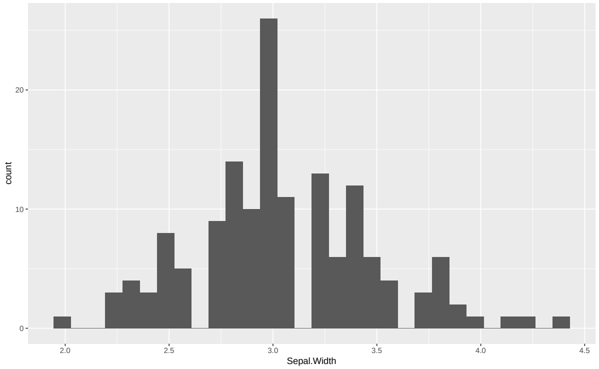


Plot histogram geom_histogram()

```r
# import library
library(ggplot2)

# Data and define aesthetics
p <- ggplot(data=iris,
            aes(x=Sepal.Width)
            )

# add (+) histogram layer to visualize
p + geom_histogram()
```

Plot histogram geom_histogram()

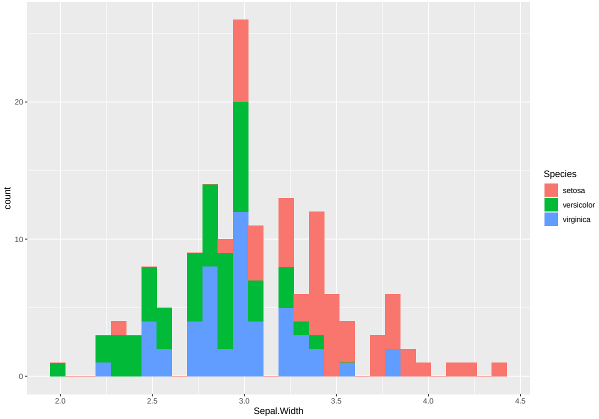
```
## `stat_bin()` using `bins = 30`. Pick better value with `bin
```



Apply color by category

```r
# add (+) histogram layer define color category
p + geom_histogram(aes(fill=Species))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `bir
```

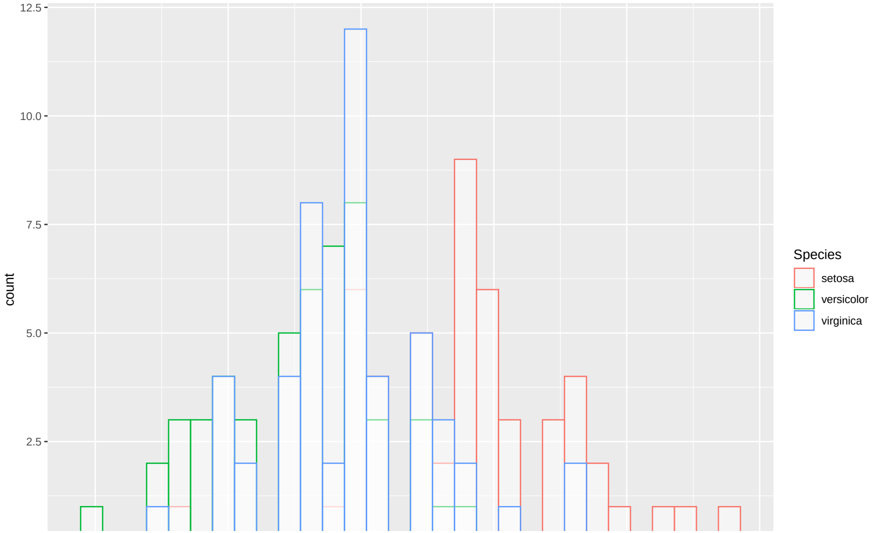


Apply color by category

```r
# add (+) histogram layer define color category
p + geom_histogram(aes(color=Species),
                   fill="white",
                   alpha=0.5,
                   position="identity"
                   )
```

Apply color by category

```
## `stat_bin()` using `bins = 30`. Pick better value with `bin
```



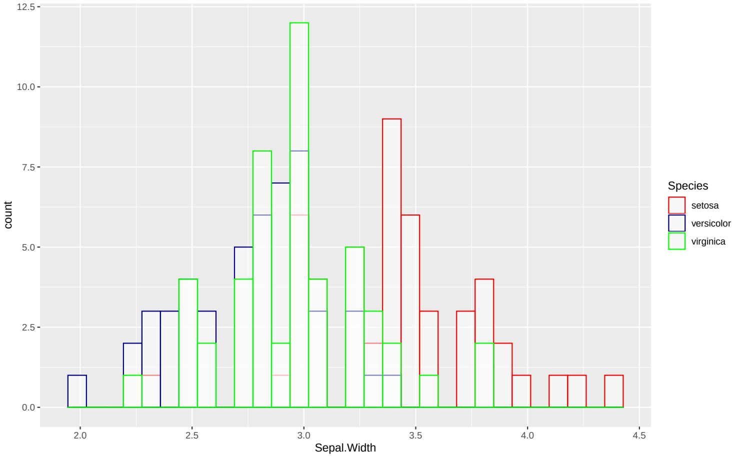
Apply color by category

```
##     setosa  virginica versicolor
##      "red"    "green" "darkblue"
```

```
# add (+) histogram layer + user defied color scale + theme
p + geom_histogram(
  aes(color=Species),
  fill='white',
  alpha=0.5,
  position="identity"
  ) +
  scale_color_manual(values =species_color)
```

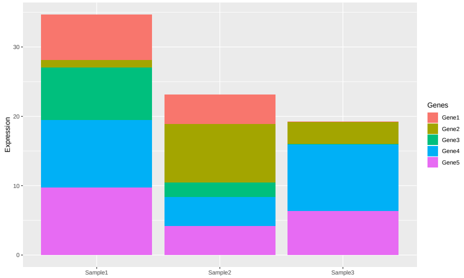
Apply color by category

```
## `stat_bin()` using `bins = 30`. Pick better value with `bin
```



Multiple category bar plots

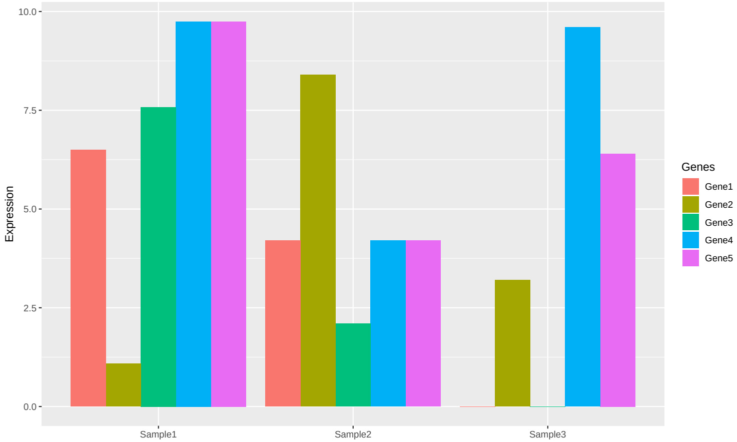
```
library(ggplot2)
# df2 from previous reshape lib slide
p <- ggplot(data=df2,
            aes(x=Sample, y=Expression, fill=Genes)
            )

# add (+) geom_bar
p + geom_bar(stat="identity")
```



Multiple category bar plots

- with deafult options ggplot created a `stacked bar`.

```
# Transform the position of the categories.
p + geom_bar(stat="identity", position="dodge")
```

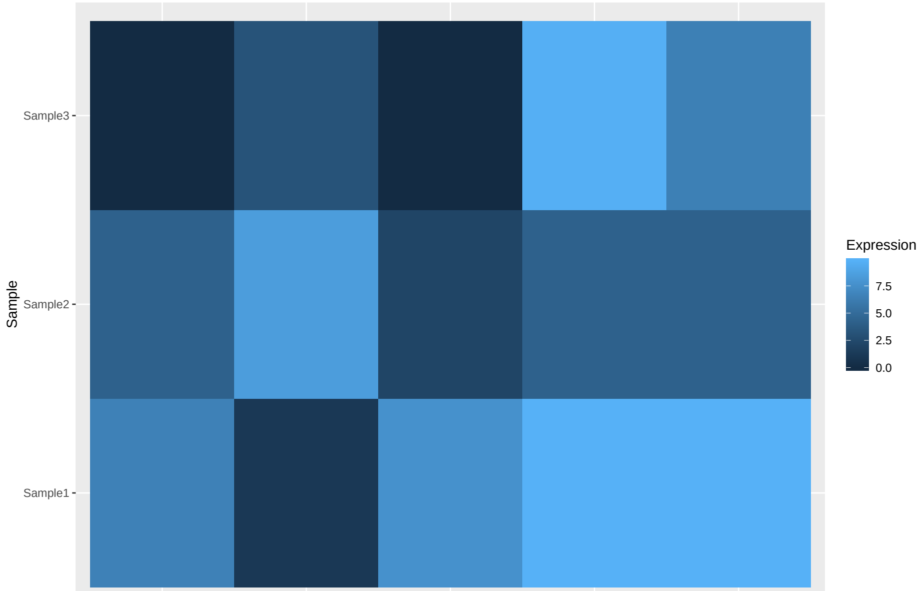


Heatmaps using geom_tile()

- Use the values as **Fill** asthestics

```r
library(ggplot2)

names(df2) = c('Genes','Sample','Expression')
# Data and define aesthetics
p <- ggplot(data=df2,
            aes(x=Genes, y=Sample, fill=Expression)
            )
```

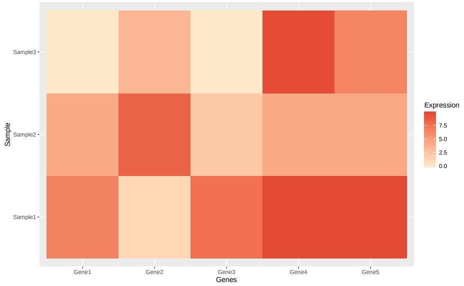
Heatmaps using geom_tile()



Heatmaps using geom_tile()

- Use the values as **Fill** asthestics
- Use specific color range

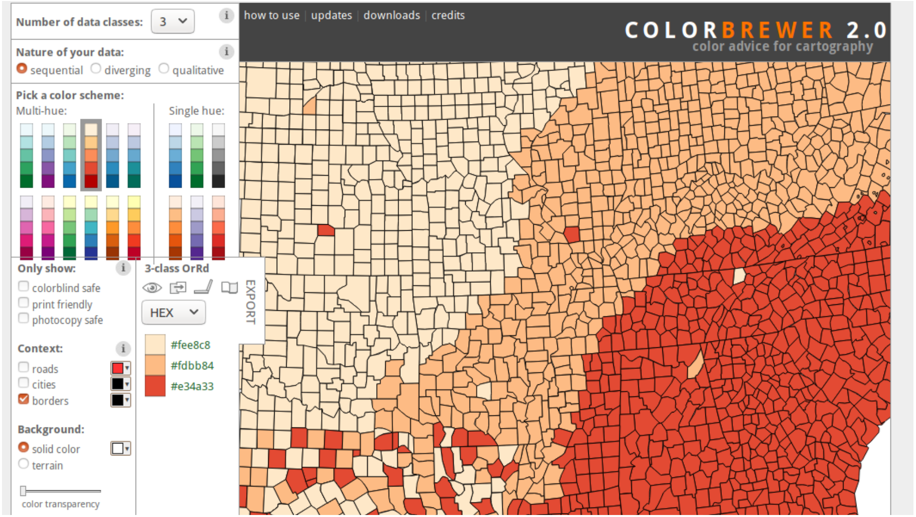
```
## Add a color gradient scale
p + geom_tile() +
  scale_fill_gradient(low = "#fee8c8", high = "#e34a33")
```



Choosing Color Palettes

Online: ColorBrewer2.0

URL: http://colorbrewer2.org/


how to use | updates | downloads | credits
COLORBREWER 2.0
color advice for cartography
Pick a color scheme:
Multi-hue:
Single hue:
EXPORT

Number of data classes: 3

Nature of your data: [✓ sequential] diverging qualitative

Only show: colorblind safe, print friendly, photocopy safe

3-class OrRd

HEX

🟧 #fee8c8
🟧 #fdbb84
🟥 #e34a33

Context: roads, cities, [✓ borders]

Background: [✓ solid color] terrain

color transparency

Choosing Color Palettes

Online: ColorBrewer2.0

Select color from three group

1. **Sequential:** Light colors for low data, dark for high data
2. **Diverging :** Light colors for mid-range data, low and high contrasting dark colors
3. **Qualitative :** Co lours designed to give maximum visual difference between classes

Choosing Color Palettes

R Packages

RColorBrewer

- R packages that uses the scheme from ColorBrewer2.0

```r
# load package
library(RColorBrewer)

# get the colors as a vector
brewer.pal(n = 3,name = 'YlOrRd')
```

```
## [1] "#FFEDA0" "#FEB24C" "#F03B20"
```

```r
brewer.pal(n = 5, name = 'Blues')
```

```
## [1] "#EFF3FF" "#BDD7E7" "#6BAED6" "#3182BD" "#08519C"
```

Choosing Color Palettes :RColorBrewer

Example of Pallates

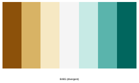
```
brewer.pal.info
```

	maxcolors	category	colorblind
BrBG	11	div	TRUE
RdBu	11	div	TRUE
Accent	8	qual	FALSE
Set2	8	qual	TRUE
Greens	9	seq	TRUE
YlOrBr	9	seq	TRUE

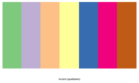
Choosing Color Palettes :RColorBrewer

To view the color palette

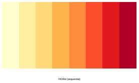
```r
## display a divergent palette
display.brewer.pal(7,"BrBG")
```



```r
## display a qualitative palette
display.brewer.pal(7,"Accent")
```



```r
## display a  Sequential palette
display.brewer.pal(name="YlOrRd", n=8)
```



Choosing Color Palettes

R Packages

ggsci

- color palettes inspired by plots in scientific journals, data visualization libraries, science fiction movies, and TV shows.
- URL: https://cran.r-project.org/web/packages/ggsci/vignettes/ggsci.html

```
# Install package
install.packages('ggsci')
```

```
## load library
library(ggsci)
# NPG: Nature Publishing Group
pal_npg()(5)
```

```
## [1] "#E64B35FF" "#4DBBD5FF" "#00A087FF" "#3C5488FF" "#F39B7
```

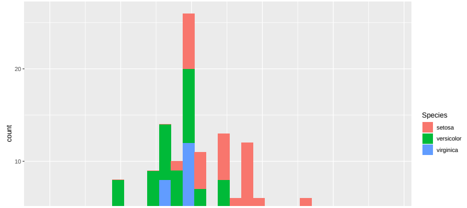
```
# NEJM: he New England Journal of Medicine
```

Choosing Color Palettes

Default ggplot2 color scheme

```
# import library
library(ggplot2)
# Data and define aesthetics
p <- ggplot(data=iris,
            aes(x=Sepal.Width, fill=Species)
            )
# add (+) histogram layer to visualize
p + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `bi
```

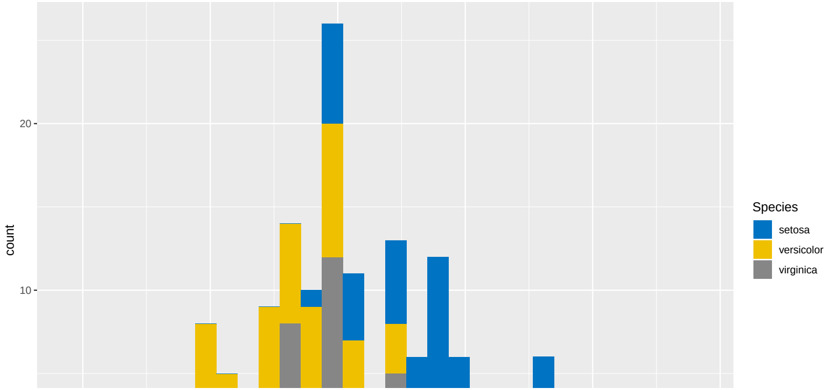


Choosing Color Palettes

ggsci: JCO color scheme

```r
# Journal of Clinical Oncology
p + geom_histogram()+
  scale_fill_jco()
```

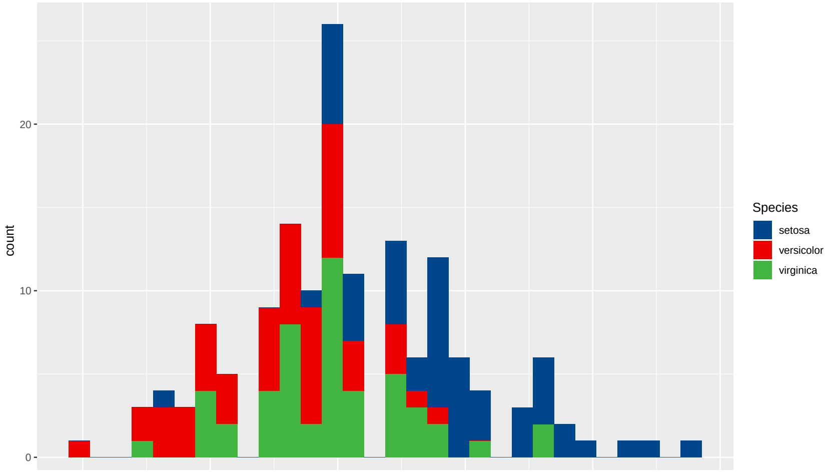
```
## `stat_bin()` using `bins = 30`. Pick better value with `bi
```



ggsci: Lancet color scheme

```r
# Lancet Oncology
p + geom_histogram()+
  scale_fill_lancet()
```

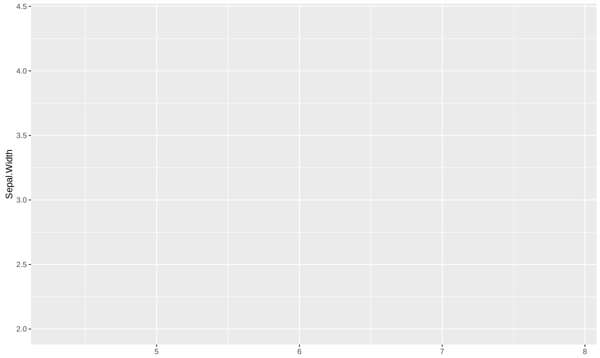
```
## `stat_bin()` using `bins = 30`. Pick better value with `bin
```



Apply themes: ready made themes

Default theme

```
# Data and define aesthetics (x, y, color)
p <- ggplot(data=iris,
            aes(x=Sepal.Length, y=Sepal.Width, color=Species)
            )
p
```

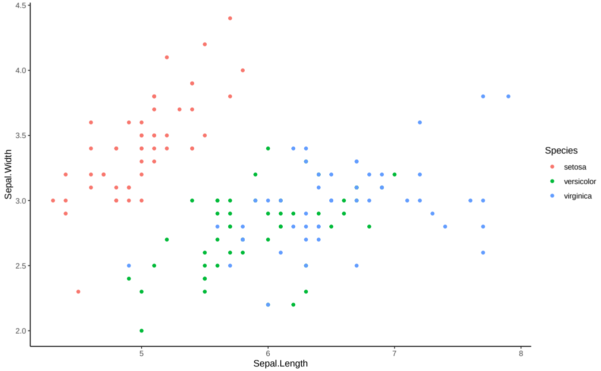


Apply themes: ready made themes

theme_classic()

A classic-looking theme, with x and y axis lines and no gridlines.

```r
# add (+) scatterplot layer + black-white theme
p + geom_point() + theme_classic()
```

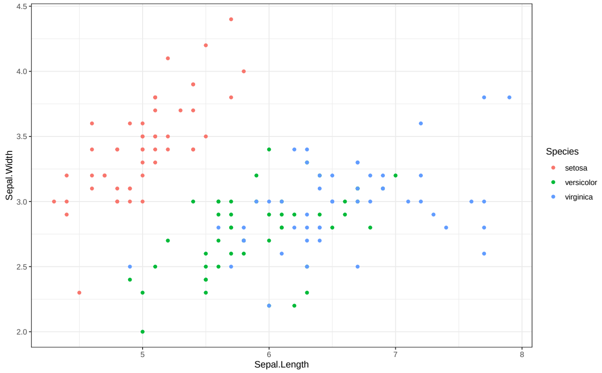


Apply themes: ready made themes

theme_bw()

The classic dark-on-light ggplot2 theme.

```r
# add (+) scatterplot layer + black-white theme
p + geom_point() + theme_bw()
```

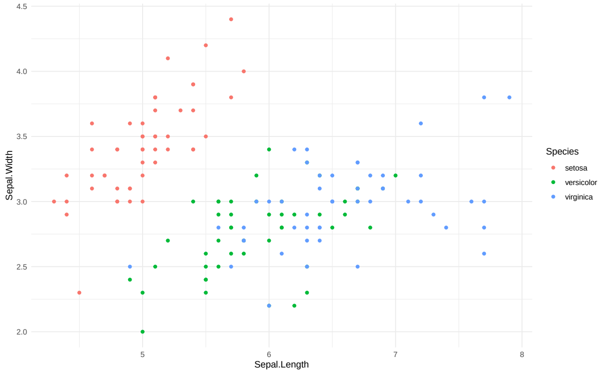


Apply themes: ready made themes

theme_minimal():

A minimalistic theme with no background annotations.

```r
# add (+) scatterplot layer + minimalistic theme
p + geom_point() + theme_minimal()
```



Modify axis, legend, and plot labels

```
p + geom_point()+
  labs(
    title="Title: Sepal.Length Vs Sepal.Width",

       subtitle="Subtitle: iris dataset",

       tag="A",

       caption="Figure caption: Scatterplot comparing Sepal le
       ) +
  ylab("Sepal width (in cm) ")+
  xlab("Sepal length (in cm) ")
```

Modify axis, legend, and plot labels

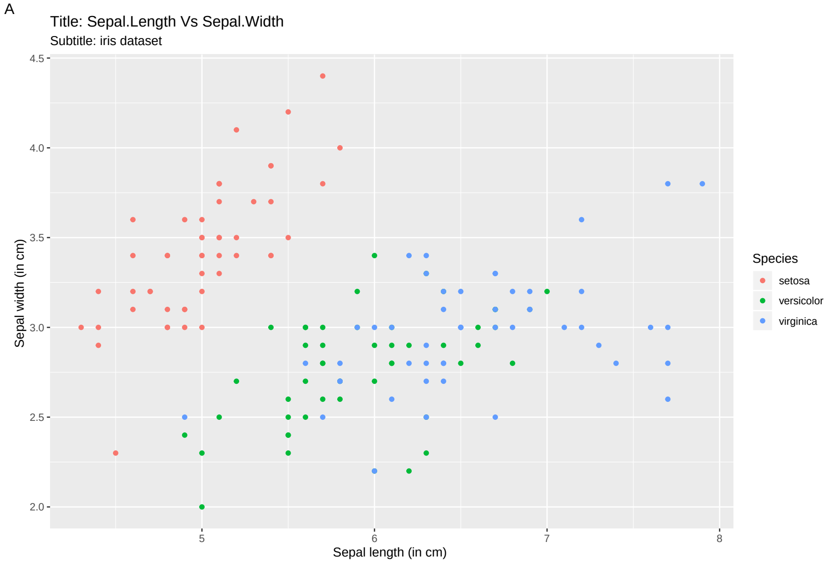
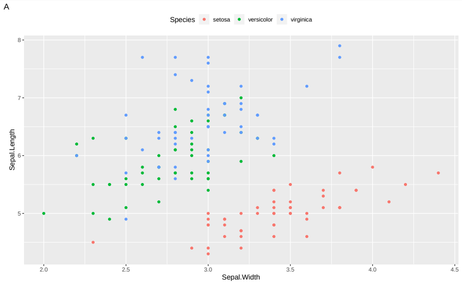


Figure caption: Scatterplot comparing Sepal length and width in three species.

Multiple plots

Create figures and save in a variable

```
library(ggplot2)

p1 <- ggplot(data=iris, aes(x=Sepal.Width,y=Sepal.Length, colo
  geom_point()+theme(legend.position="top")+
  labs(tag='A')
p1
```

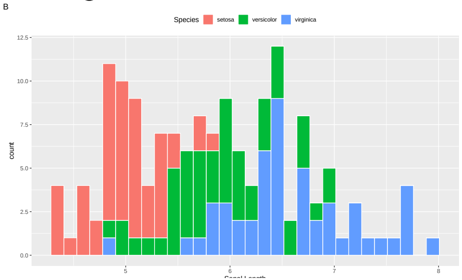


Multiple plots

Create figures and save in a variable

```
p2 <- ggplot(data=iris, aes(x=Sepal.Length,fill=Species)) +
  geom_histogram(color='white')+
  theme(legend.position="top")+
  labs(tag='B')

p2
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `bi
```

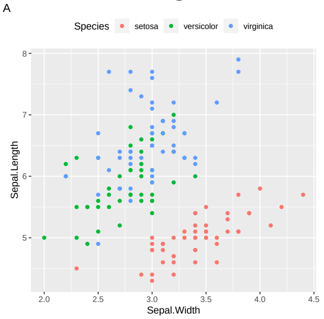


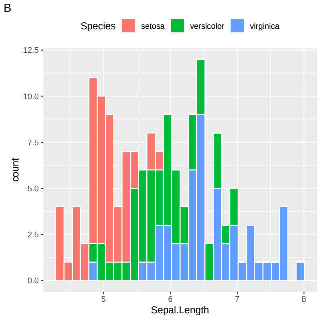
Multiple plots

Create joint figures

```
library(gridExtra)
grid.arrange(p1,p2, nrow=1, respect=TRUE, clip=TRUE)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `bi
```





Add tables with figures

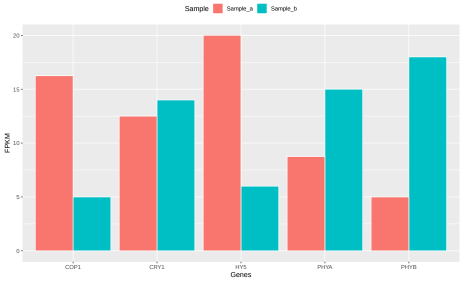
```r
## Table as a data frame
df <- data.frame(Genes=c('PHYB', 'PHYA', 'CRY1','COP1','HY5'),
                 Sample_a=seq(5,20, length=5),
                 Sample_b=sample(seq(1,20), 5 )
                 )
row.names(df) <- df$Genes
exprs <- melt(df)
```

```
## Using Genes as id variables
```

```r
names(exprs) <- c('Genes','Sample', 'FPKM')
```

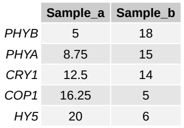
Add tables with figures

```r
## Graphical object
p1 <- ggplot(data=exprs, aes(x=Genes, y=FPKM, fill=Sample))+
  geom_bar(stat="identity", position="dodge", color='white')+
  theme(legend.position="top")
p1
```

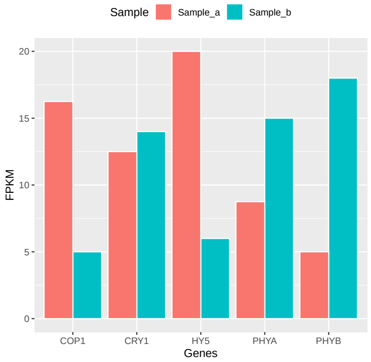


Add tables with figures

```
grid.arrange(tableGrob(head(df[,2:3])), p1,
             nrow=1,
             respect=TRUE )
```



	Sample_a	Sample_b
PHYB	5	18
PHYA	8.75	15
CRY1	12.5	14
COP1	16.25	5
HY5	20	6

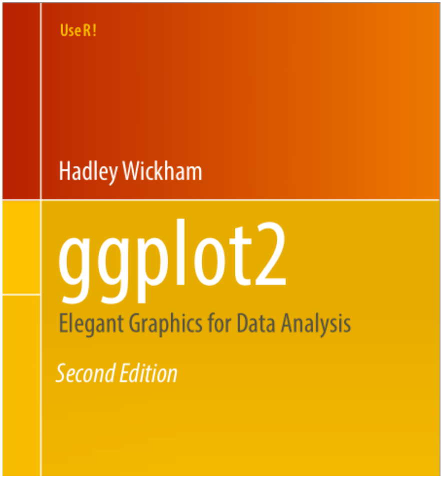


Save plot to file

```r
## Save the 'current ggplot' to file
ggsave("ggPlot_figure.pdf", width = 4, height = 4)

## If p1 or p2 are graphical objects
ggsave("ggPlot_figure.p1.pdf", width = 4, height = 4,
       plot=p1)
ggsave("ggPlot_figure.p2.pdf", width = 4, height = 8,
       plot=p2)
```

More on ggplot2

ggplot2: Elegant Graphics for Data Analysis
Hadley Wickham
Free e-book: https://ggplot2-book.org/



Use R!

Hadley Wickham

ggplot2

Elegant Graphics for Data Analysis

Second Edition

Bio-conductor: Bioinformatics repository

- Bioconductor (www.bioconductor.org)is a project within R.
- Bioconductor works with a high throughput genomic data from DNA sequence, micro-array, proteomics, imaging and a number of other data types.
- The current available version of Bioconductor (Version 3.6) consists of 1477 software packages.

Why

- dependencies: It makes very easy to resolve library dependencies.
- compatibility : Correct and compatible versions of all the dependencies are installed.
- Reliability : Packages in Bioconductor are (reasonably) reliable. Compiled every day by someone around the world. So any error will be reported.

Bio-conductor: Bioinformatics repository

Install core packages

```
install.packages("BiocManager")
```

Install a package from Bioconductor

```
BiocManager::install('ape')
```

Update all packages

```
BiocManager::install()
```

Where to go from here?

- Take your data, import in R
- Search Internet
    - How to do . . . use keyword **R**
    - Stack overflow, R-blogger
- Follow YouTube channels, blog sites
    - twitter hash tags:
    - #rstats
    - one R tip per day (@RLangTip)
- Print a cheatsheet
- Each day practice a little.

Thank you for participating

- Email - kcm.eid@gmail.com
- Twitter - @kc_moharana