Introduction to ggplot2, plotly and shiny

Lecture 1b

732A98

Ggplot2, plotly and shiny

- **Ggplot2**: modern package for static graphics in R
- **Ggplot2**: based on the *layered grammar of graphics*

- **Plotly**: modern package for interactive graphics and animation
- **Plotly**: generates independent HTML pages with JavaScript

- **Shiny**: modern package for interactive graphics
- **Shiny**: allows for low-level access to interactivity
- **Shiny**: apps need an R server running in background

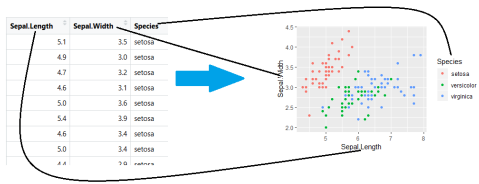
Layered grammar of graphics: why?

Grammar of graphics allows:

- Understand the composition of complicated graphics
- Reveal connections between seemingly different graphics
- Helps to understand how a well-formed graphics look like

Layered grammar of graphics: basic idea

- Graph is a mapping $f(D) \rightarrow A$ of data D to aesthetics $(x, y, color)$



Key ingridients

- Aesthetics (aes)
  - Coordinates, size, shape, color
  - Data variables are mapped into aes
  - Sometimes aes can be constant
- Geometry (geom)
- Statistics (transformation)
- Scale
- Coordinate System (coord)
  - Can be cartesian, polar,..
- Faceting

Key ingridients

A **plot** is

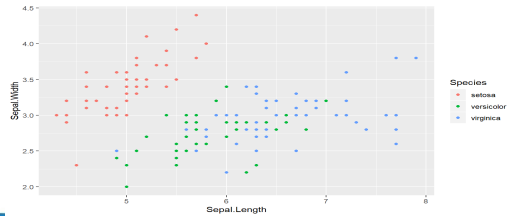
- Default dataset and mapping to aesthetics
- **Layers**: (Geometry, Statistics, [PositionAdjustment], [ExtraData], [ExtraAesthetics])
- One scale per aesthetics
- Coordinate system
- Facet specification

Example

```
library(ggplot2)
p<-ggplot()+
  layer(
    data=irisData,
    mapping=aes(x=Sepal.Length, y=Sepal.Width, color=Species),
    geom="point", stat="identity", position="identity"
  )+
  scale_y_continuous() +
  scale_x_continuous()+
  coord_cartesian()
print(p)
```


Example

- Note: '+' symbol is used to combine plot elements: layers, axis, scales…



Ggplot2: Smart defaults

- Some settings are typical:
  - Cartesian coordinates are often a default choice
  - Scales can be defined from the type of variable and aesthetics
  - Normally 1 layer used
  - …

```
p<-ggplot(irisData,aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point()
```


Ggplot2: Geometries

A plenty of different geometries available

- `geom_bar()` bar charts
- `geom_boxplot()`box plots
- `geom_points()` points
- `geom_smooth()` add a smoother
- `geom_map` plot a map
- …
- Sometimes geometries need a *transformed data* -> see **Statistical Transformation**

Statistical Transformation

Used to summarize data in some way

- `stat_count()` counts number of cases in each group
- `stat_quantile()` fits quantile regression and extracts quantiles
- `stat_density()`computes density estimates
- `stat_function()` computes a function for each x value
- …

See GGplot2 reference

Scales

Various scales and their parameters can be set

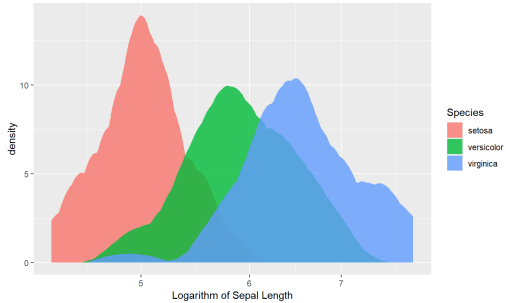
- `labs()` `xlab()` `ylab()` `ggtitle()`Modify axis, legend, and plot labels
- `lims()` `xlim()` `ylim()` Set scale limits
- `scale_alpha()` transparency scale
- `scale_colour_continuous()`Continuous colour scale
- `scale_y_log10()` logarithmic y scale
- …

Example

- Statistics, scale and label adjusted

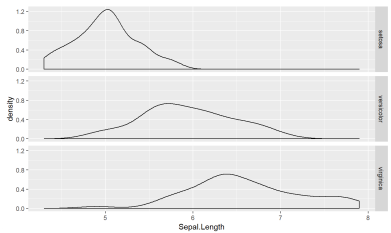
```
ggplot(irisData, aes(x=Sepal.Length,fill=Species))+
 stat_density(alpha=0.8, kernel = "epanechnikov", position="identity")+
 scale_x_log10()+
 xlab("Logarithm of Sepal Length")
```


Example



Example -facets

```
ggplot(irisData, aes(x=Sepal.Length))+
 geom_density()+ facet_grid(vars(Species))
```



Ggplot2 - a lot more to learn!

- Hadley's book
- GGplot2 reference
- Ggplot2 cheat sheet

Plotly

- A Javascript library for interactive and dynamic plotting
- Includes interfaces from R, Python,...
- Produces a standalone HTML
- Some interactivity build-in in each plot
- Ggplot2 graphs can easily be made interactive
- Can be directly published on web
- Some functions are commercial
- Developed by a commercial company

From ggplot2 to plotly

- Simple converting is simple

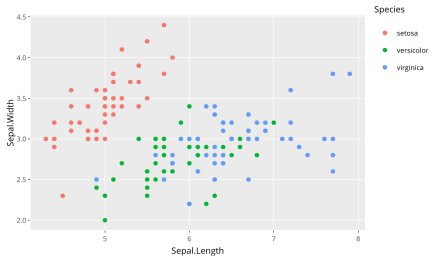
```
p<-ggplot(irisData,aes(x=Sepal.Length, y=Sepal.Width,color=Species)) +
  geom_point()

library(plotly)

ggplotly(p)
```


From ggplot2 to plotly

- Check the buttons of the toolbox

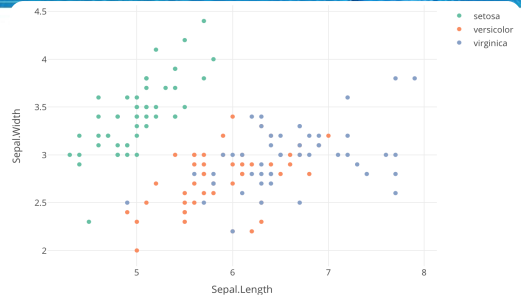


Without ggplot2

- Some plotly graphs can also be constructed directly from data

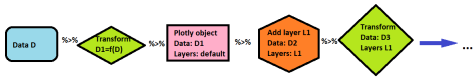
```
plot_ly(irisData, x=~Sepal.Length, y=~Sepal.Width,
        color = ~Species)%>%add_markers()
```

Without ggplot2



Data-plot pipeline in plotly

- Process of creating a graph is represented by a pipeline
- Plotly transfers graphics and data in the pipeline
- Pipeline operator `%>%`



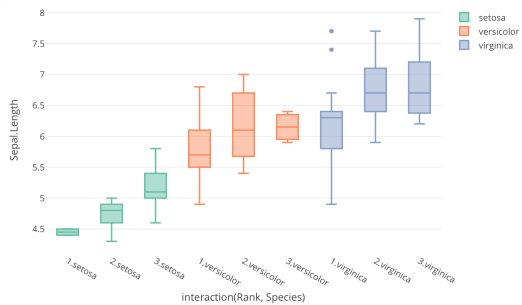
- Transformations to the data may be done with **dplyr** functions or with internal graphical transformations

Example: data-plot pipeline

```
library(dplyr)
myPlot <- irisData %>% mutate( Rank=ntile(Sepal.Width,3))%>%
  group_by(Species) %>%
  plot_ly(x=~interaction(Rank, Species), y=~Sepal.Length,
          color=~Species) %>%
  add_boxplot()
myPlot
```

- `mutate()` adds new variables
- `ntile()` splits continuous variable into classes

Example: data-plot pipeline



Data in the pipeline

- Current data can be seen with `plotly_data()`

```
plotly_data(myPlot)[1:5,]
```

```
## # A tibble: 5 x 4
## # Groups:   Species [1]
##   Sepal.Length Sepal.Width Species  Rank
##          <dbl>       <dbl> <fct>   <int>
## 1          5.1         3.5 setosa      3
## 2          4.9         3   setosa      2
## 3          4.7         3.2 setosa      2
## 4          4.6         3.1 setosa      2
## 5          5           3.6 setosa      3
```

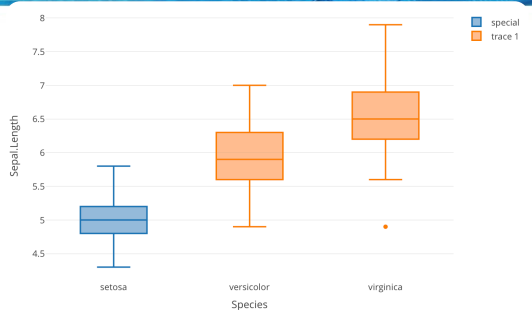

Pipeline within a pipeline

- When data is transformed, original data is not saved
- Solution: Creating pipeline within pipeline `add_function()`

```
myPlot <- irisData %>% plot_ly(x=~Species, y=~Sepal.Length) %>%
  add_fun(function(plot) {
    plot %>% filter(Species == "setosa") %>%
      add_boxplot( name="special")
    }) %>%
  filter(Species != "setosa") %>%
  add_boxplot(hoverinfo="none")

myPlot
```


Pipeline within a pipeline

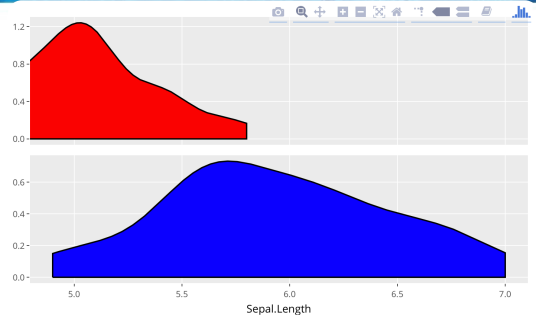


Arranging multiple views

- Use `subplot()` function
- It can combine both plotly plots and ggplot2 plots

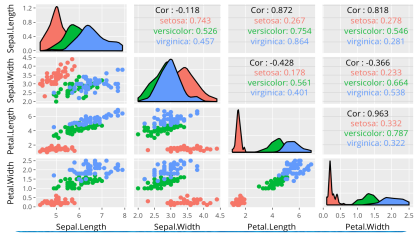
```
p1<-ggplot(irisData %>%  filter(Species=="setosa"),
           aes(x=Sepal.Length))+
  geom_density(fill="red")
p2<-ggplot(irisData %>%  filter(Species=="versicolor"),
           aes(x=Sepal.Length))+
  geom_density(fill="blue")
subplot(p1, p2, nrows=2,shareX=TRUE)
```

Arranging multiple views



Other plotly features

- Plots can be linked and animated
- Sliders and other controls can be added



Plotly - read more

- Plotly book
- Plotly R website
- Plotly cheat sheet
- Plotly R reference

Shiny

- Shiny is an R package that allows to build interactive web apps straight from R
  - Interactive plots is one of the options
- Created HTML pages need a running R server to work
  - If creating plots for yourself, your laptop = server
- Created webpages can be published on Shiny server

Shiny architecture

Shiny and Plotly

- Plotly: HTML pages are standalone, everything is done in Javascript
- Shiny: Without running R session, page is not working
- Shiny: allows for low-level interaction coding
- Shiny: allows to make complicated computations at the server side -> often faster than same functionality in Plotly
- Shiny: Plotly graphs can be embedded into Shiny

Coding in Shiny

Define UI and server function

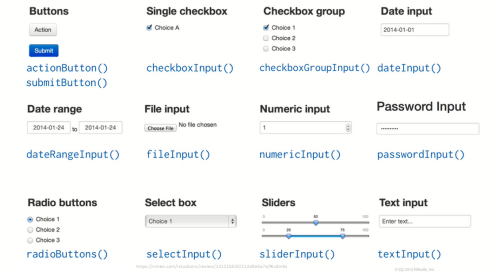
```
library(shiny)
# Define UI for application
ui <- fluidPage(
   ...
)

# Define server actions
server <- function(input, output) {
  # Use input
...
  # Save updates to output
}
```


Shiny-example

Open app1.R

Shiny- UI elements



Shiny- UI elements

UI Outputs

Functions for creating user interface elements that, in conjunction with rendering functions, display different kinds of output from your application.

`htmlOutput (uiOutput)`	Create an HTML output element
`plotOutput (imageOutput)`	Create an plot or image output element
`outputOptions`	Set options for an output object.
`tableOutput (dataTableOutput)`	Create a table output element
`textOutput`	Create a text output element
`verbatimTextOutput`	Create a verbatim text output element
`downloadButton (downloadLink)`	Create a download button or link
`Progress`	Reporting progress (object-oriented API)
`withProgress`	Reporting progress (functional API)

Reactivity

- Shiny listens to changes in `input$...` variables
- When change is detected, `server()` is called
- Server should return `renderXXX()` functions into `output`
- UI uses elements in `output$...` to render information.
- `input$...` variables are called **reactive** variables.

Rendering functions

Rendering functions

Functions that you use in your application's server side code, assigning them to outputs that appear in your user interface.

`renderPlot`	Plot Output
`renderText`	Text Output
`renderPrint`	Printable Output
`renderDataTable`	Table output with the JavaScript library DataTables
`renderImage`	Image file output
`renderTable`	Table Output
`renderUI`	UI Output

Secondary reactive variables

- Depend on the original reactive variables
- Changes in the new variables causes changes in the output
- Useful for ex. when the action should depend on **whole history** of actions
- Use `reactiveValues()` and `observeEvent()` to work with those
- Example: app2.R

Shiny Gadgets

- Useful for own analysis of data rather than publication
- Written as one function
- Returns results into R directly
- Use package `miniUI`

- Example: app3.R

More about shiny

- Basics of Shiny Apps
- Shiny video tutorial 1
- Interactive graphics video
- Shiny articles

Read at home

- Paper 'Layered grammar of Graphics' by Wickham (2010)
- Hadley's book Chapters 1-7, selectively
- Plotly book Chapters 1, 2.1, 3.1,3.2
- Basics of Shiny Apps