

Volpe R Course: Session 4, Communicating Results

Instructors: Don Fisher, Dan Flynn, Jessie Yang
Course webpage: <http://bit.ly/volpeR>

9/26/2017

Overview

Last session:

- ▶ Data exploration
 - ▶ Aggregation
 - ▶ Merging and Matching

Today: Communicating results

- ▶ Regular expression review
- ▶ Basic graphics
- ▶ ggplot2 package
- ▶ Basic analysis

Regular expressions

Quick digression to review material from last session!

The use of `sub` above is an example of *regular expression*. These are very flexible and form the basis for all search engines. In R, there are number of ways to use regular expressions, with `grep` and `sub` being two of the most common ones. `regexpr` also provides additional options for those familiar with Perl or Python.

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

```
## [1] "Testing"          "matching"          " and substitutions"
```

```
## [1] 3
```

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

Homework review

Will review example solutions to the homework assignments here

Make summary tables using `aggregate` and other functions

Make a subset using `grep`

Use `merge`

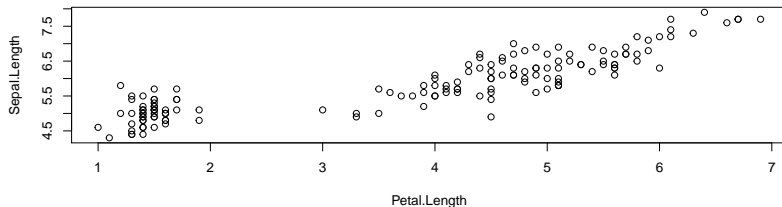
Basic graphics

You have already seen the power of `plot()` in examples. We have used it to make scatterplots, but also to make diagnostic plots of linear models. When in doubt, you can always try to `plot()` an object and R will make a guess about what kind of plot you want.

Example:

```
data(iris) # see ?iris

# Formula (y ~ x) versus (x, y) specification
plot(Sepal.Length ~ Petal.Length, data = iris)
```



```
plot(iris$Petal.Length, iris$Sepal.Length) # identical
```

Basic graphics (plus some analysis)

You frequently want to analyze what the trend of the data is statistically, not just graphically. We can use a simple linear model and plot the result using `lm()`:

```
plot(Sepal.Length ~ Petal.Length, data = iris)
```

```
summary(linmod <- lm(Sepal.Length ~ Petal.Length, data = iris))
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.24675 -0.29657 -0.01515  0.27676  1.00269
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.30660    0.07839   54.94  <2e-16 ***
## Petal.Length   0.40892    0.01889   21.65  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4071 on 148 degrees of freedom
## Multiple R-squared:  0.76, Adjusted R-squared:  0.7583
## F-statistic: 468.6 on 1 and 148 DF, p-value: < 2.2e-16
```

Basic graphics

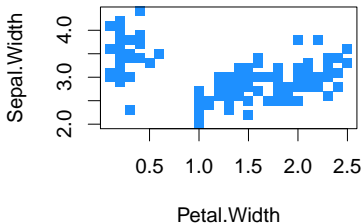
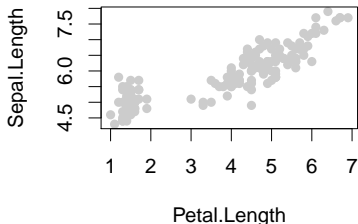
Back to graphics! The power of the base graphics package in R is control of each element of the plot. The challenge is knowing how to exercise that control.

`par()` sets graphics parameters and contains much of what you want to do. Highlights from `par`:

- `mfrow` - matrix format. Set to `c(1, 2)` for two panel figure, adjacent
- `pch` - plotting character.
- `lty` - line type. Try `lty = 2` or `lty = 3`
- `cex` - character expansion. Very useful for making symbols more visible
- `mar` - margins. Set to `mar = rep(1, 4)` for narrow margins all around.
- `col` - colors, see `colors()` for full list or www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf

Basic graphics

```
par(mfrow = c(1, 2),  
    pch = 16,  
    cex = 1.5) # makes everything bigger  
  
plot(Sepal.Length ~ Petal.Length,  
     col = "grey80",  
     data = iris)  
  
plot(Sepal.Width ~ Petal.Width,  
     col = "dodgerblue",  
     pch = 15, # can override par settings  
     data = iris)
```



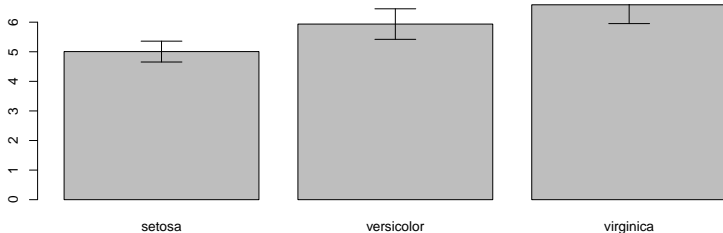
Basic graphics

`barplot()` can be used to create a bar chart, but it is a bit clunky to add error bars.
Basic workflow:

```
means <- tapply(iris$Sepal.Length, iris$Species, mean)
sds <- tapply(iris$Sepal.Length, iris$Species, sd)

# save the location of the bars as 'bp'
bp <- barplot(means)

arrows(bp, means+sds, # see ?arrows
       bp, means-sds,
       angle = 90,
       code = 3)
```

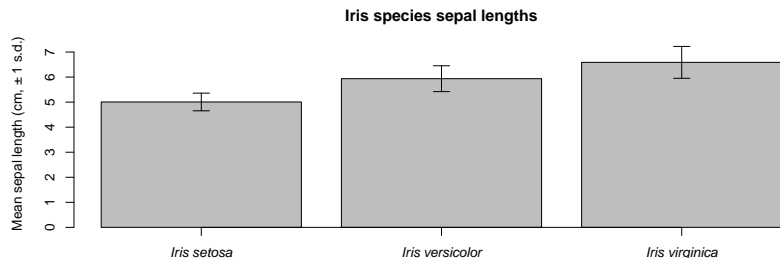


Basic graphics

Improve a graphic by ensuring that there are clear labels (arrows code not shown).

```
par(mar = c(3, 4, 3, 1)) # Narrowing margins
bp <- barplot(means,
              ylab = "Mean sepal length (cm, ? 1 s.d.)",
              main = "Iris species sepal lengths",
              xaxt = "n", # suppress axis plotting here
              ylim = c(0, 7.5))

axis(side = 1, at = bp,
      labels = paste("Iris", c("setosa", "versicolor", "virginica")),
      font = 3)
```



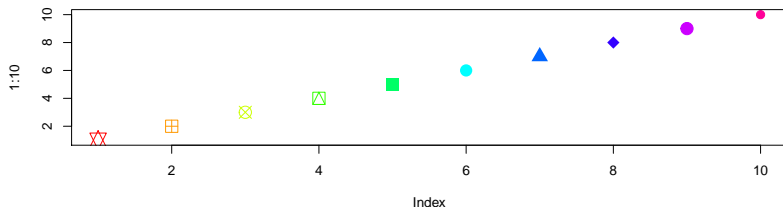
Basic graphics

Save your graphics typically as pdf or jpeg. You can do this by putting `pdf()` and `dev.off()` around your code, or by using `dev.print()`.

```
pdf("Test graphic.pdf", width = 4, height = 6)
plot(1:10, pch = 1:10, col = 1:10, cex = 2)
dev.off()
```

```
## pdf
## 2
```

```
plot(1:10, pch = 11:20, col = rainbow(10), cex = 2)
```

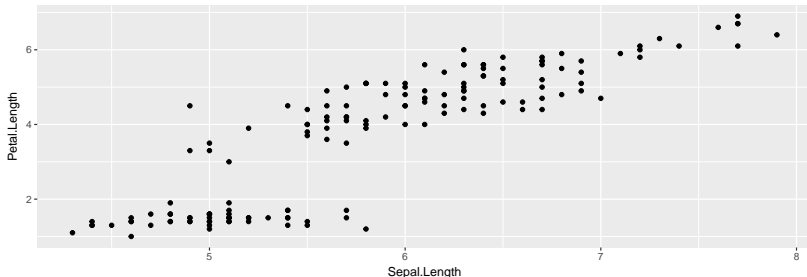


```
dev.print(device = jpeg, file = "Test graphic.jpg", height = 500, width = 800)
```

ggplot graphics

A different model of graphics has recently become popular. ggplots use a different 'grammar' of graphics, and can be great for quickly making visually appealing graphics. Mastering the control of each element can take time.

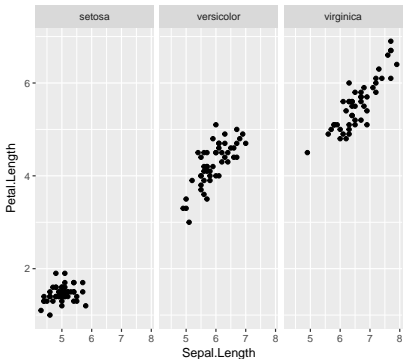
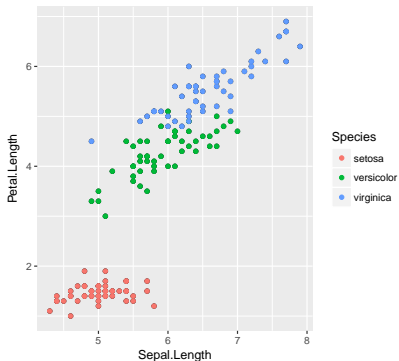
```
library(ggplot2) # install.packages('ggplot2')  
#      Dataset      x      y      type of plot  
gp1 <- ggplot(iris) + aes(Sepal.Length, Petal.Length) + geom_point()  
gp1
```



ggplot graphics

You can use one basic graph data set and do different versions of it. Two ways of visualizing a factor grouping:

```
gp1 + geom_point(aes(color = Species))  
gp1 + facet_wrap(~ Species)
```



Interactive

These can be made interactive in an HTML document using `plot.ly`. See <https://plot.ly/r/> if interested in trying this! You will need to download several packages, so this code will not work for you immediately:

```
library(plotly)  
ggplotly(ex1)
```

Tables in R

Carrying out any analysis in R generates a lot of tables. A few options:

- ▶ Copy and paste
- ▶ Sink
- ▶ Save to text file

Simplest is copy and paste from the console window into Excel, then use text-to-columns. But not repeatable!

Tables in R

You can 'sink' your output to a text file. First, set up all your analysis, then put `sink()` before and after your block of code.

```
sink("Test Sink.txt")
with(iris, cor.test(Sepal.Width, Petal.Width))
table(iris$Species)
sink()
```

Or output a data frame to a text file directly:

```
write.csv(iris[1:10,], file = "Iris Test.csv", row.names = F)
```


R Markdown

A better way for writing a report is to take advantage of the relatively recent tool, R Markdown. A few resources:

<http://rmarkdown.rstudio.com/>

http://rmarkdown.rstudio.com/authoring_quick_tour.html

Go to File > New File ... > R Markdown

Select HTML or Word as the default output, and then try clicking 'knit'.

You may need to `install.packages("rmarkdown")`

Homework review: Outliers

Time permitting, we will go back to the Hubway data and examine the question of outliers

We will review tripduration outliers from the Hubway data.