

Graphics and Data Visualization in R

Overview

Thomas Girke

May 25, 2012

Overview

Graphics Environments

- Base Graphics

- Grid Graphics

- lattice

- ggplot2

Specialty Graphics

Outline

Overview

Graphics Environments

- Base Graphics

- Grid Graphics

- lattice

- ggplot2

Specialty Graphics

Graphics in R

- Powerful environment for visualizing scientific data
- Integrated graphics and statistics infrastructure
- Publication quality graphics
- Fully programmable
- Highly reproducible
- Full \LaTeX [Link](#) & Sweave [Link](#) support
- Vast number of R packages with graphics utilities

Documentation on Graphics in R

General

- Graphics Task Page [Link](#)
- R Graph Gallery [Link](#)
- R Graphical Manual [Link](#)
- Paul Murrell's book R (Grid) Graphics [Link](#)

Interactive graphics

- rggobi (GGobi) [Link](#)
- iplots [Link](#)
- Open GL (rgl) [Link](#)

Graphics Environments

Viewing and saving graphics in R

- On-screen graphics
- postscript, pdf, svg
- jpeg/png/wmf/tiff/...

Four major graphic environments

- Low-level infrastructure
 - R Base Graphics (low- and high-level)
 - *grid*: Manual [Link](#), Book [Link](#)
- High-level infrastructure
 - *lattice*: Manual [Link](#), Intro [Link](#), Book [Link](#)
 - *ggplot2*: Manual [Link](#), Intro [Link](#), Book [Link](#)

Outline

Overview

Graphics Environments

- Base Graphics

- Grid Graphics

- lattice

- ggplot2

Specialty Graphics

Outline

Overview

Graphics Environments

- Base Graphics

- Grid Graphics

- lattice

- ggplot2

Specialty Graphics

Base Graphics: Overview

Important high-level plotting functions

- `plot`: generic x-y plotting
- `barplot`: bar plots
- `boxplot`: box-and-whisker plot
- `hist`: histograms
- `pie`: pie charts
- `dotchart`: cleveland dot plots
- `image`, `heatmap`, `contour`, `persp`: functions to generate image-like plots
- `qqnorm`, `qqline`, `qqplot`: distribution comparison plots
- `pairs`, `coplot`: display of multivariant data

Help on these functions

- `?myfct`
- `?plot`
- `?par`

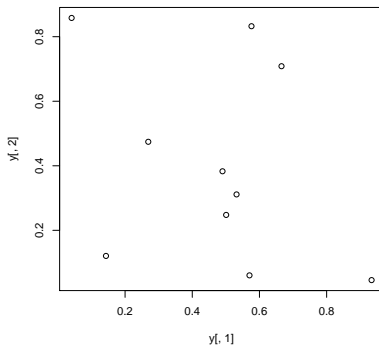
Base Graphics: Preferred Input Data Objects

- Matrices and data frames
- Vectors
- Named vectors

Scatter Plot: very basic

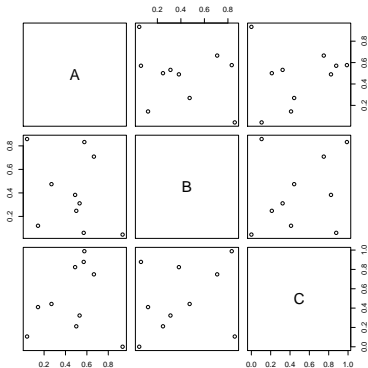
Sample data set for subsequent plots

```
> set.seed(1410)
> y <- matrix(runif(30), ncol=3, dimnames=list(letters[1:10], LETTERS[1:3]))
> plot(y[,1], y[,2])
```



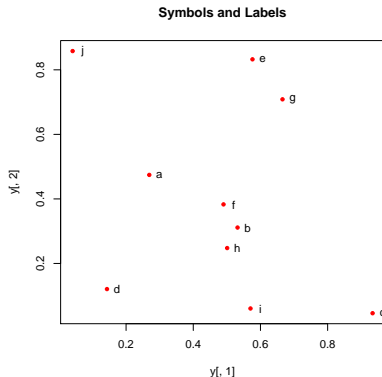
Scatter Plot: all pairs

```
> pairs(y)
```



Scatter Plot: with labels

```
> plot(y[,1], y[,2], pch=20, col="red", main="Symbols and Labels")  
> text(y[,1]+0.03, y[,2], rownames(y))
```



Scatter Plots: more examples

Print instead of symbols the row names

```
> plot(y[,1], y[,2], type="n", main="Plot of Labels")  
> text(y[,1], y[,2], rownames(y))
```

Usage of important plotting parameters

```
> grid(5, 5, lwd = 2)  
> op <- par(mar=c(8,8,8,8), bg="lightblue")  
> plot(y[,1], y[,2], type="p", col="red", cex.lab=1.2, cex.axis=1.2,  
+       cex.main=1.2, cex.sub=1, lwd=4, pch=20, xlab="x label",  
+       ylab="y label", main="My Main", sub="My Sub")  
> par(op)
```

Important arguments

- `mar`: specifies the margin sizes around the plotting area in order: `c(bottom, left, top, right)`
- `col`: color of symbols
- `pch`: type of symbols, samples: `example(points)`
- `lwd`: size of symbols
- `cex.*`: control font sizes
- For details see `?par`

Scatter Plots: more examples

Add a regression line to a plot

```
> plot(y[,1], y[,2])  
> myline <- lm(y[,2]~y[,1]); abline(myline, lwd=2)  
> summary(myline)
```

Same plot as above, but on log scale

```
> plot(y[,1], y[,2], log="xy")
```

Add a mathematical expression to a plot

```
> plot(y[,1], y[,2]); text(y[1,1], y[1,2],  
>      expression(sum(frac(1,sqrt(x^2*pi)))), cex=1.3)
```

Exercise 1: Scatter Plots

Task 1 Generate scatter plot for first two columns in iris data frame and color dots by its Species column.

Task 2 Use the xlim/ylim arguments to set limits on the x- and y-axes so that all data points are restricted to the left bottom quadrant of the plot.

Structure of iris data set:

```
> class(iris)
```

```
[1] "data.frame"
```

```
> iris[1:4,]
```

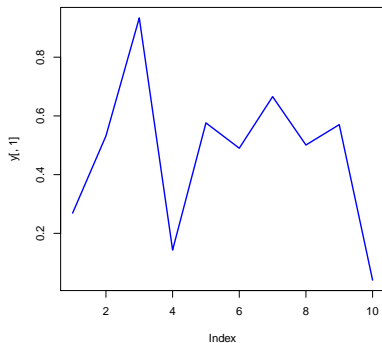
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa

```
> table(iris$Species)
```

setosa	versicolor	virginica
50	50	50

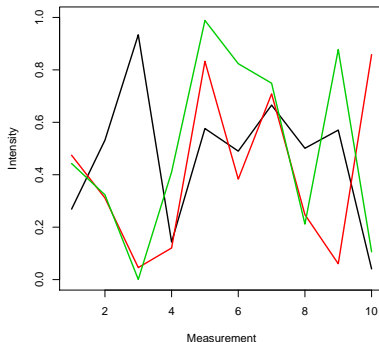
Line Plot: Single Data Set

```
> plot(y[,1], type="l", lwd=2, col="blue")
```



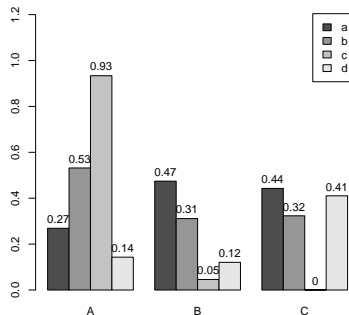
Line Plots: Many Data Sets

```
> split.screen(c(1,1));  
[1] 1  
> plot(y[,1], ylim=c(0,1), xlab="Measurement", ylab="Intensity", type="l", lwd=2, col=1)  
> for(i in 2:length(y[,])) {  
+   screen(1, new=FALSE)  
+   plot(y[,i], ylim=c(0,1), type="l", lwd=2, col=i, xaxt="n", yaxt="n", ylab="",  
+       xlab="", main="", bty="n")  
+ }  
> close.screen(all=TRUE)
```



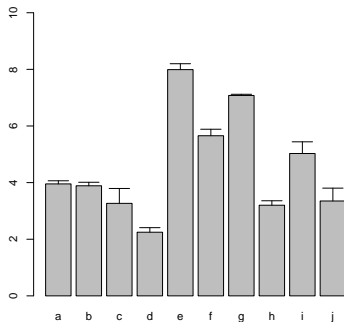
Bar Plot Basics

```
> barplot(y[1:4,], ylim=c(0, max(y[1:4,])+0.3), beside=TRUE,  
+         legend=letters[1:4])  
> text(labels=round(as.vector(as.matrix(y[1:4,])),2), x=seq(1.5, 13, by=1)  
+       +sort(rep(c(0,1,2), 4)), y=as.vector(as.matrix(y[1:4,]))+0.04)
```



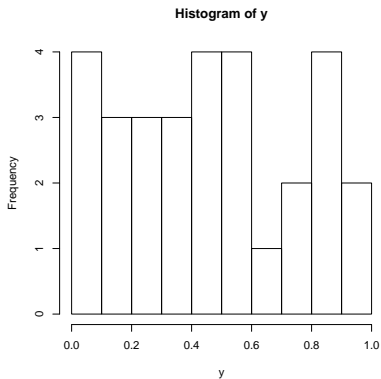
Bar Plots with Error Bars

```
> bar <- barplot(m <- rowMeans(y) * 10, ylim=c(0, 10))  
> stdev <- sd(t(y))  
> arrows(bar, m, bar, m + stdev, length=0.15, angle = 90)
```



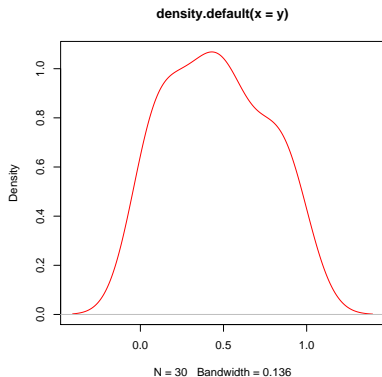
Histograms

```
> hist(y, freq=TRUE, breaks=10)
```



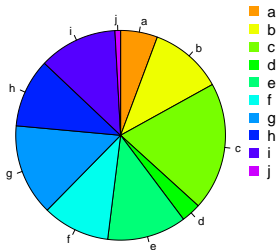
Density Plots

```
> plot(density(y), col="red")
```



Pie Charts

```
> pie(y[,1], col=rainbow(length(y[,1])), start=0.1, end=0.8), clockwise=TRUE)
> legend("topright", legend=row.names(y), cex=1.3, bty="n", pch=15, pt.cex=1.8,
+ col=rainbow(length(y[,1])), start=0.1, end=0.8), ncol=1)
```



Color Selection Utilities

Default color palette and how to change it

```
> palette()
```

```
[1] "black"    "red"      "green3"   "blue"     "cyan"     "magenta"  "yellow"   "gray"
```

```
> palette(rainbow(5, start=0.1, end=0.2))
```

```
> palette()
```

```
[1] "#FF9900" "#FFBF00" "#FFE600" "#F2FF00" "#CCFF00"
```

```
> palette("default")
```

The `gray` function allows to select any type of gray shades by providing values from 0 to 1

```
> gray(seq(0.1, 1, by= 0.2))
```

```
[1] "#1A1A1A" "#4D4D4D" "#808080" "#B3B3B3" "#E6E6E6"
```

Color gradients with `colorpanel` function from *gplots* library

```
> library(gplots)
```

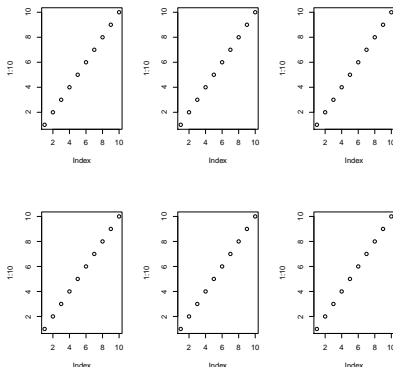
```
> colorpanel(5, "darkblue", "yellow", "white")
```

Much more on colors in R see Earl Glynn's color chart [Link](#)

Arranging Several Plots on Single Page

With `par(mfrow=c(nrow,ncol))` one can define how several plots are arranged next to each other.

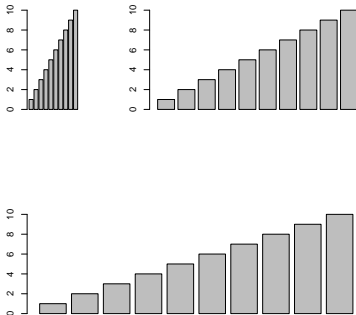
```
> par(mfrow=c(2,3)); for(i in 1:6) { plot(1:10) }
```



Arranging Plots with Variable Width

The `layout` function allows to divide the plotting device into variable numbers of rows and columns with the column-widths and the row-heights specified in the respective arguments.

```
> nf <- layout(matrix(c(1,2,3,3), 2, 2, byrow=TRUE), c(3,7), c(5,5),  
+               respect=TRUE)  
> # layout.show(nf)  
> for(i in 1:3) { barplot(1:10) }
```



Saving Graphics to Files

After the `pdf()` command all graphs are redirected to file `test.pdf`. Works for all common formats similarly: `jpeg`, `png`, `ps`, `tiff`, ...

```
> pdf("test.pdf"); plot(1:10, 1:10); dev.off()
```

Generates Scalable Vector Graphics (SVG) files that can be edited in vector graphics programs, such as Inkscape.

```
> library("RSvgDevice"); devSVG("test.svg"); plot(1:10, 1:10); dev.off()
```

Exercise 2: Bar Plots

Task 1 Calculate the mean values for the Species components of the first four columns in the iris data set. Organize the results in a matrix where the row names are the unique values from the iris Species column and the column names are the same as in the first four iris columns.

Task 2 Generate two bar plots: one with stacked bars and one with horizontally arranged bars.

Structure of iris data set:

```
> class(iris)

[1] "data.frame"

> iris[1:4,]

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1          3.5          1.4          0.2  setosa
2          4.9          3.0          1.4          0.2  setosa
3          4.7          3.2          1.3          0.2  setosa
4          4.6          3.1          1.5          0.2  setosa

> table(iris$Species)

  setosa versicolor virginica 
    50         50         50
```

Outline

Overview

Graphics Environments

Base Graphics

Grid Graphics

lattice

ggplot2

Specialty Graphics

grid Graphics Environment

- What is *grid*?
 - Low-level graphics system
 - Highly flexible and controllable system
 - Does not provide high-level functions
 - Intended as development environment for custom plotting functions
 - Pre-installed on new R distributions
- Documentation and Help
 - Manual [Link](#)
 - Book [Link](#)

Outline

Overview

Graphics Environments

Base Graphics

Grid Graphics

lattice

ggplot2

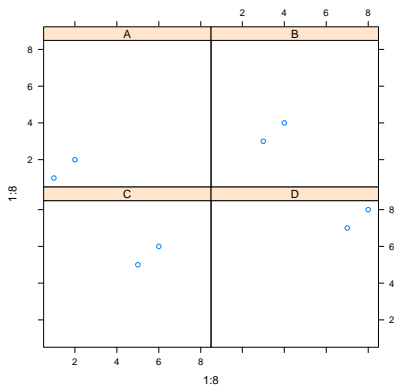
Specialty Graphics

lattice Environment

- What is *lattice*?
 - High-level graphics system
 - Developed by Deepayan Sarkar
 - Implements Trellis graphics system from S-Plus
 - Simplifies high-level plotting tasks: arranging complex graphical features
 - Syntax similar to R's base graphics
- Documentation and Help
 - Manual [Link](#)
 - Intro [Link](#)
 - Book [Link](#)
 - `library(help=lattice)` opens a list of all functions available in the lattice package
 - Accessing and changing global parameters:
`?lattice.options` and `?trellis.device`

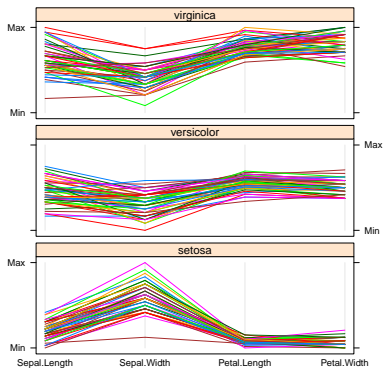
Scatter Plot Sample

```
> library(lattice)
> p1 <- xyplot(1:8 ~ 1:8 | rep(LETTERS[1:4], each=2), as.table=TRUE)
> plot(p1)
```



Line Plot Sample

```
> library(lattice)
> p2 <- parallel(~iris[1:4] | Species, iris, horizontal.axis = FALSE,
+               layout = c(1, 3, 1))
> plot(p2)
```



Outline

Overview

Graphics Environments

- Base Graphics

- Grid Graphics

- lattice

- ggplot2**

Specialty Graphics

ggplot2 Environment

- What is *ggplot2*?
 - High-level graphics system
 - Implements grammar of graphics from Leland Wilkinson [Link](#)
 - Streamlines many graphics workflows for complex plots
 - Syntax centered around main `ggplot` function
 - Simpler `qplot` function provides many shortcuts
- Documentation and Help
 - Manual [Link](#)
 - Intro [Link](#)
 - Book [Link](#)

ggplot2 Usage

- `ggplot` function accepts two arguments
 - Data set to be plotted
 - Aesthetic mappings provided by `aes` function
- Additional parameters such as geometric objects (e.g. points, lines, bars) are passed on by appending them with `+` as separator.
- List of available `geom_*` functions: [Link](#)
- Settings of plotting theme can be accessed with the command `theme_get()` and its settings can be changed with `opts()`.
- Preferred input data object
 - `ggplot: data.frame` (support for vector, matrix, ...)
 - `ggplot: data.frame`
- Packages with convenience utilities to create expected inputs
 - *plyr*
 - *reshape*

qplot Function

- qplot syntax is similar to R's basic plot function
- Arguments:
 - x: x-coordinates (e.g. col1)
 - y: y-coordinates (e.g. col2)
 - data: data frame with corresponding column names
 - xlim, ylim: e.g. xlim=c(0,10)
 - log: e.g. log="x" or log="xy"
 - main: main title; see ?plotmath for mathematical formula
 - xlab, ylab: labels for the x- and y-axes
 - color, shape, size
 - ...: many arguments accepted by plot function

qplot: Scatter Plots

Create sample data

```
> library(ggplot2)
> x <- sample(1:10, 10); y <- sample(1:10, 10); cat <- rep(c("A", "B"), 5)
```

Simple scatter plot

```
> qplot(x, y, geom="point")
```

Prints dots with different sizes and colors

```
> qplot(x, y, geom="point", size=x, color=cat,
+       main="Dot Size and Color Relative to Some Values")
```

Drops legend

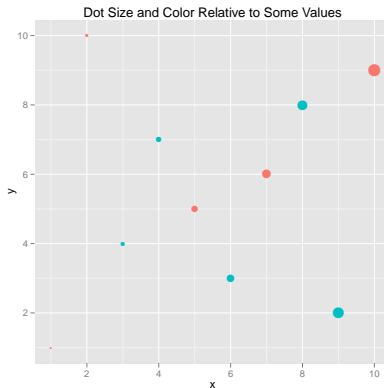
```
> qplot(x, y, geom="point", size=x, color=cat) +
+       opts(legend.position = "none")
```

Plot different shapes

```
> qplot(x, y, geom="point", size=5, shape=cat)
```

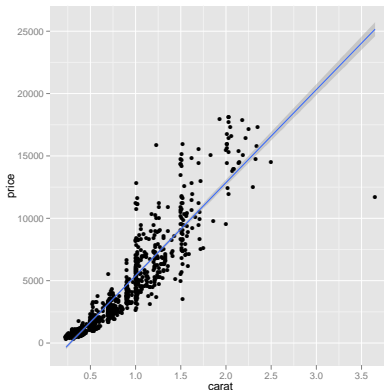
qplot: Scatter Plot with qplot

```
> p <- qplot(x, y, geom="point", size=x, color=cat,  
+           main="Dot Size and Color Relative to Some Values") +  
+           opts(legend.position = "none")  
> print(p)
```



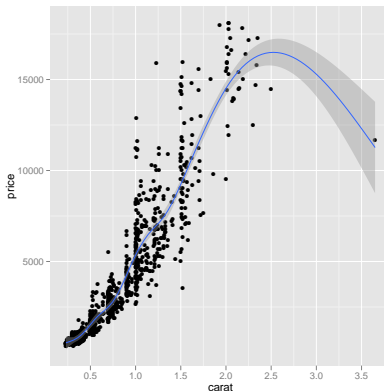
qplot: Scatter Plot with Regression Line

```
> set.seed(1410)
> dsmall <- diamonds[sample(nrow(diamonds), 1000), ]
> p <- qplot(carat, price, data = dsmall, geom = c("point", "smooth"),
+           method = "lm")
> print(p)
```



qplot: Scatter Plot with Local Regression Curve (loess)

```
> p <- qplot(carat, price, data=dsmall, geom=c("point", "smooth"), span=0.4)  
> print(p) # Setting 'se=FALSE' removes error shade
```



ggplot Function

- More important than `qplot` to access full functionality of *ggplot2*
- Main arguments
 - data set, usually a `data.frame`
 - aesthetic mappings provided by `aes` function
- General `ggplot` syntax
 - `ggplot(data, aes(...)) + geom_*() + ... + stat_*() + ...`
- Layer specifications
 - `geom_*(mapping, data, ..., geom, position)`
 - `stat_*(mapping, data, ..., stat, position)`
- Additional components
 - `scales`
 - `coordinates`
 - `facet`
- `aes()` mappings can be passed on to all components (`ggplot`, `geom_*`, etc.). Effects are global when passed on to `ggplot()` and local for other components.
 - `x`, `y`
 - `color`: grouping vector (factor)
 - `group`: grouping vector (factor)

Changing Plotting Themes with ggplot

- Theme settings can be accessed with `theme_get()`
- Their settings can be changed with `opts()`
- Some examples
 - Change background color to white
... + `opts(panel.background=theme_rect(fill = "white", colour = "black"))`

Storing ggplot Specifications

Plots and layers can be stored in variables

```
> p <- ggplot(dsmall, aes(carat, price)) + geom_point()  
> p # or print(p)
```

Returns information about data and aesthetic mappings followed by each layer

```
> summary(p)
```

Prints dots with different sizes and colors

```
> bestfit <- geom_smooth(methodw = "lm", se = F, color = alpha("steelblue", 0.5))  
> p + bestfit # Plot with custom regression line
```

Syntax to pass on other data sets

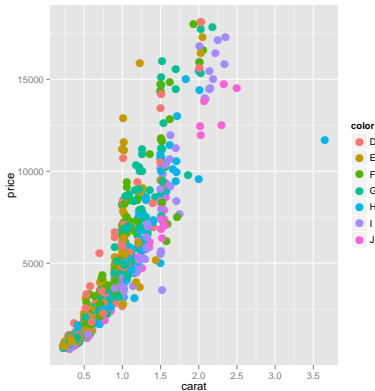
```
> p %>% diamonds[sample(nrow(diamonds), 100),]
```

Saves plot stored in variable p to file

```
> ggsave(p, file="myplot.pdf")
```

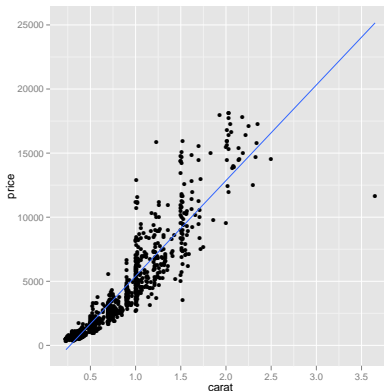
ggplot: Scatter Plot

```
> p <- ggplot(dsmall, aes(carat, price, color=color)) +  
+   geom_point(size=4)  
> print(p)
```



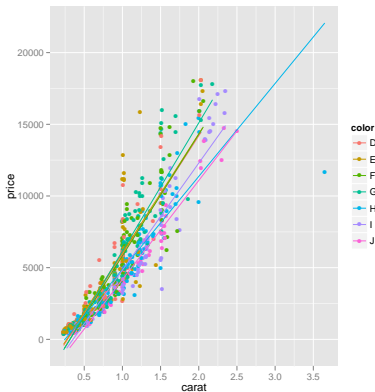
ggplot: Scatter Plot with Regression Line

```
> p <- ggplot(dsmall, aes(carat, price)) + geom_point() +  
+       geom_smooth(method="lm", se=FALSE)  
> print(p)
```



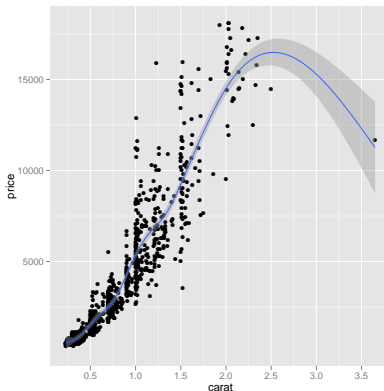
ggplot: Scatter Plot with Several Regression Lines

```
> p <- ggplot(dsmall, aes(carat, price, group=color)) +  
+   geom_point(aes(color=color), size=2) +  
+   geom_smooth(aes(color=color), method = "lm", se=FALSE)  
> print(p)
```



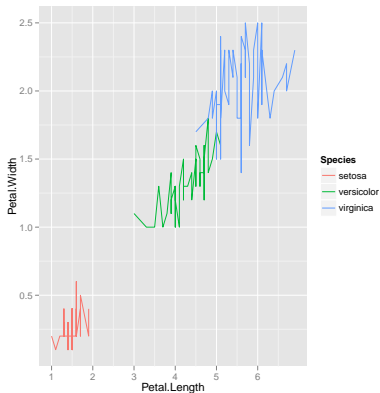
ggplot: Scatter Plot with Local Regression Curve (loess)

```
> p <- ggplot(dsmall, aes(carat, price)) + geom_point() + geom_smooth()  
> print(p) # Setting 'se=FALSE' removes error shade
```



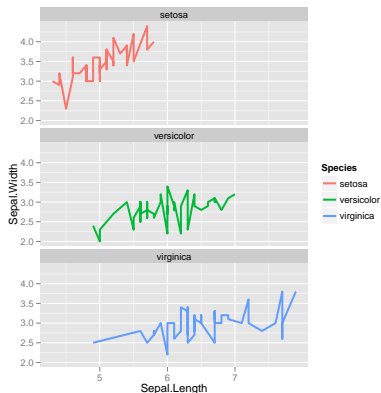
ggplot: Line Plot

```
> p <- ggplot(iris, aes(Petal.Length, Petal.Width, group=Species,  
+                       color=Species)) + geom_line()  
> print(p)
```



ggplot: Faceting

```
> p <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) +  
+   geom_line(aes(color=Species), size=1) +  
+   facet_wrap(~Species, ncol=1)  
> print(p)
```



Exercise 3: Scatter Plots

- Task 1** Generate scatter plot for first two columns in `iris` data frame and color dots by its `Species` column.
- Task 2** Use the `xlim`, `ylim` functions to set limits on the x- and y-axes so that all data points are restricted to the left bottom quadrant of the plot.
- Task 3** Generate corresponding line plot with faceting show individual data sets in separate plots.

Structure of `iris` data set:

```
> class(iris)
```

```
[1] "data.frame"
```

```
> iris[1:4,]
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa

```
> table(iris$Species)
```

setosa	versicolor	virginica
50	50	50

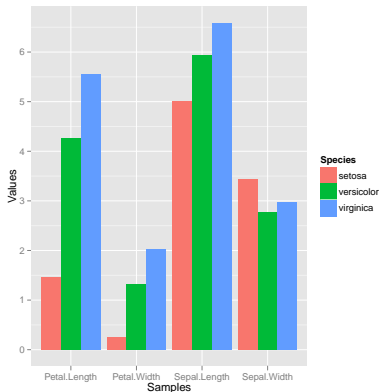
ggplot: Bar Plots

Sample Set: the following transforms the iris data set into a ggplot2-friendly format

```
> ## Calculate mean values for aggregates given by Species column
> ## in iris data set
> iris_mean <- aggregate(iris[,1:4], by=list(Species=iris$Species), FUN=mean)
> ## Calculate standard deviations for aggregates given by Species
> ## column in iris data set
> iris_sd <- aggregate(iris[,1:4], by=list(Species=iris$Species), FUN=sd)
> ## Define function to convert data frames into ggplot2-friendly format.
> convertDF <- function(df=df, mycolnames=c("Species", "Values", "Samples")) {
+   myfactor <- rep(colnames(df)[-1], each=length(df[,1]))
+   mydata <- as.vector(as.matrix(df[, -1]))
+   df <- data.frame(df[,1], mydata, myfactor)
+   colnames(df) <- mycolnames; return(df)
+ }
> ## Convert iris_mean
> df_mean <- convertDF(iris_mean, mycolnames=c("Species", "Values", "Samples"))
> ## Convert iris_sd
> df_sd <- convertDF(iris_sd, mycolnames=c("Species", "Values", "Samples"))
> ## Define standard deviation limits
> limits <- aes(ymax = df_mean[,2] + df_sd[,2], ymin=df_mean[,2] - df_sd[,2])
```

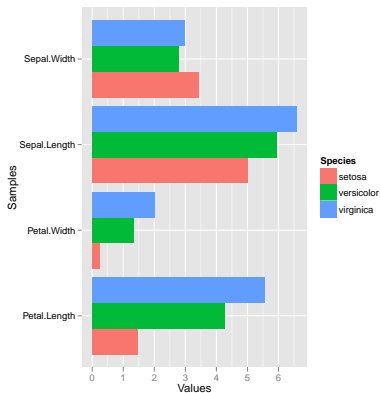
ggplot: Bar Plot

```
> p <- ggplot(df_mean, aes(Samples, Values, fill = Species)) +  
+   geom_bar(position="dodge")  
> print(p)
```



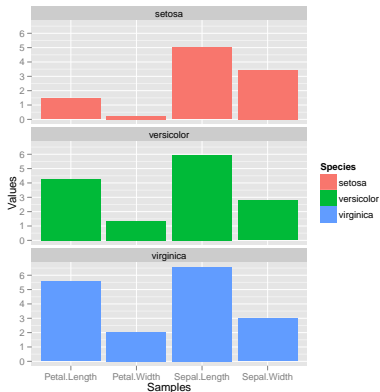
ggplot: Bar Plot Sideways

```
> p <- ggplot(df_mean, aes(Samples, Values, fill = Species)) +  
+   geom_bar(position="dodge") + coord_flip() +  
+   opts(axis.text.y=theme_text(angle=0, hjust=1))  
> print(p)
```



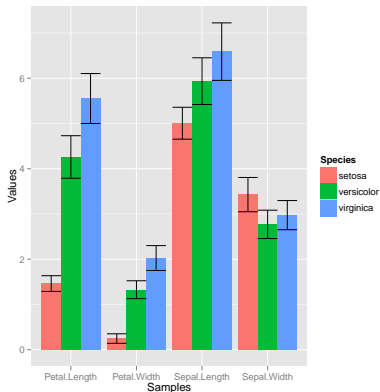
ggplot: Bar Plot with Faceting

```
> p <- ggplot(df_mean, aes(Samples, Values)) + geom_bar(aes(fill = Species)) +  
+           facet_wrap(~Species, ncol=1)  
> print(p)
```



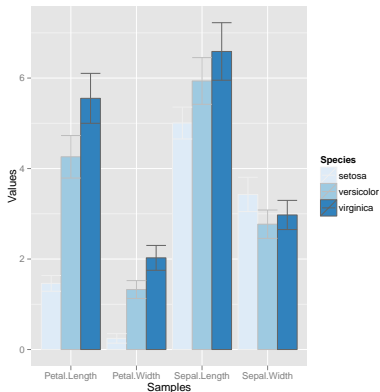
ggplot: Bar Plot with Error Bars

```
> p <- ggplot(df_mean, aes(Samples, Values, fill = Species)) +  
+   geom_bar(position="dodge") + geom_errorbar(limits, position="dodge")  
> print(p)
```



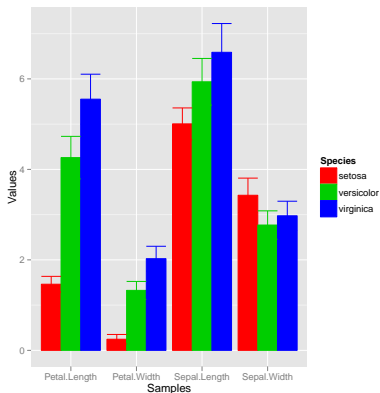
ggplot: Changing Color Settings

```
> library(RColorBrewer)
> # display.brewer.all()
> p <- ggplot(df_mean, aes(Samples, Values, fill=Species, color=Species)) +
+   geom_bar(position="dodge") + geom_errorbar(limits, position="dodge") +
+   scale_fill_brewer(palette="Blues") + scale_color_brewer(palette = "Greys")
> print(p)
```



ggplot: Using Standard Colors

```
> p <- ggplot(df_mean, aes(Samples, Values, fill=Species, color=Species)) +  
+   geom_bar(position="dodge") + geom_errorbar(limits, position="dodge") +  
+   scale_fill_manual(values=c("red", "green3", "blue")) +  
+   scale_color_manual(values=c("red", "green3", "blue"))  
> print(p)
```



Exercise 4: Bar Plots

Task 1 Calculate the mean values for the Species components of the first four columns in the iris data set. Use the `convertDF` function from one of the previous slides to bring the results into the expected format for `ggplot`.

Task 2 Generate two bar plots: one with stacked bars and one with horizontally arranged bars.

Structure of iris data set:

```
> class(iris)
```

```
[1] "data.frame"
```

```
> iris[1:4,]
```

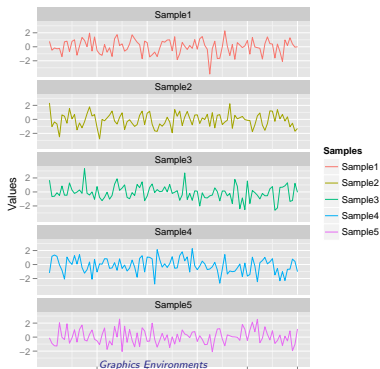
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa

```
> table(iris$Species)
```

setosa	versicolor	virginica
50	50	50

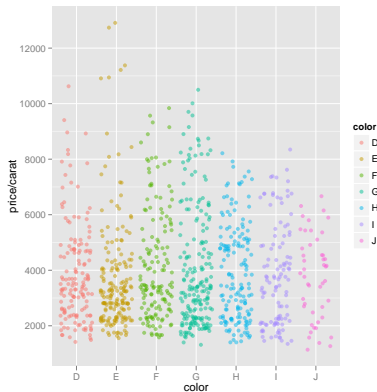
ggplot: Data Reformatting Example for Line Plot

```
> y <- matrix(rnorm(500), 100, 5, dimnames=list(paste("g", 1:100, sep=""), paste("Sample", 1:5, sep="")))
> y <- data.frame(count=1:length(y[,1]), y)
> y[1:4, ] # First rows of input format expected by convertDF()
  count Sample1 Sample2 Sample3 Sample4 Sample5
g1     1  0.7674733 2.3429578 1.6815086 -1.202619 -0.1053735
g2     2 -0.4875971 -1.0326720 -0.6556324  1.174138 -0.8991985
g3     3 -0.1699970 -0.3695197 -0.6096398  1.357860 -1.2370991
g4     4 -0.2866302 -0.6544649 -0.1169518  1.173512 -1.2724114
> df <- convertDF(y, mycolnames=c("Position", "Values", "Samples"))
> p <- ggplot(df, aes(Position, Values)) + geom_line(aes(color=Samples)) + facet_wrap(~Samples, ncol=1)
> print(p)
> ## Represent same data in box plot
> ## ggplot(df, aes(Samples, Values, fill=Samples)) + geom_boxplot()
```



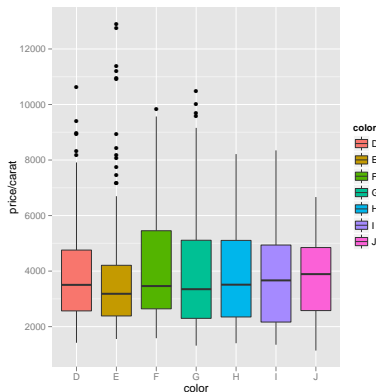
ggplot: Jitter Plots

```
> p <- ggplot(dsmall, aes(color, price/carat)) +  
+       geom_jitter(alpha = I(1 / 2), aes(color=color))  
> print(p)
```



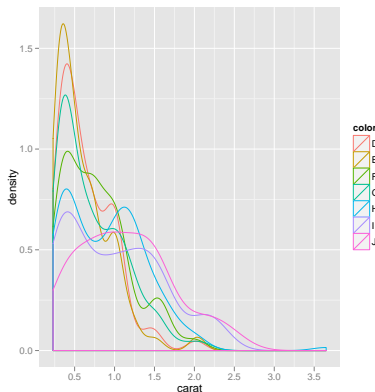
ggplot: Box Plots

```
> p <- ggplot(dsmall, aes(color, price/carat, fill=color)) + geom_boxplot()  
> print(p)
```



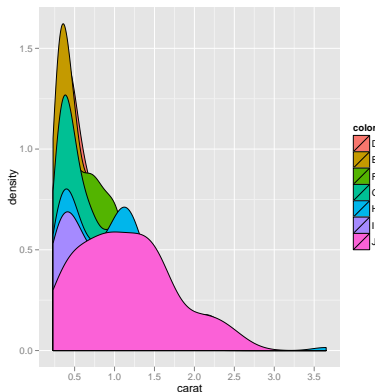
ggplot: Density Plot with Line Coloring

```
> p <- ggplot(dsmall, aes(carat)) + geom_density(aes(color = color))  
> print(p)
```



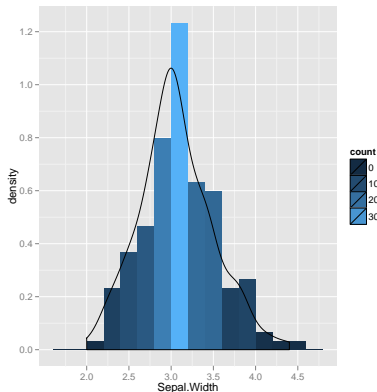
ggplot: Density Plot with Area Coloring

```
> p <- ggplot(dsmall, aes(carat)) + geom_density(aes(fill = color))  
> print(p)
```



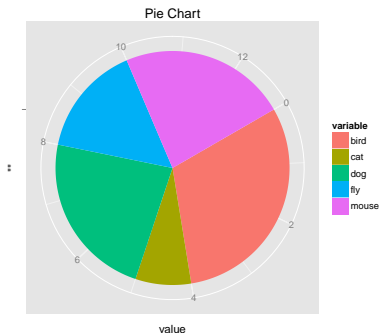
ggplot: Histograms

```
> p <- ggplot(iris, aes(x=Sepal.Width)) + geom_histogram(aes(y = ..density..,  
+ fill = ..count..), binwidth=0.2) + geom_density()  
> print(p)
```



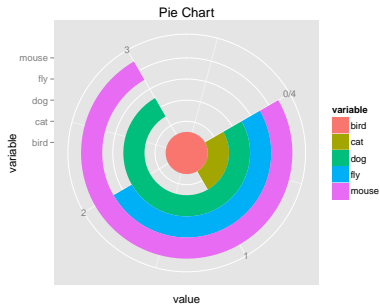
ggplot: Pie Chart

```
> df <- data.frame(variable=rep(c("cat", "mouse", "dog", "bird", "fly")),  
+                   value=c(1,3,3,4,2))  
> p <- ggplot(df, aes(x = "", y = value, fill = variable)) +  
+     geom_bar(width = 1) +  
+     coord_polar("y", start=pi / 3) + opts(title = "Pie Chart")  
> print(p)
```



ggplot: Wind Rose Pie Chart

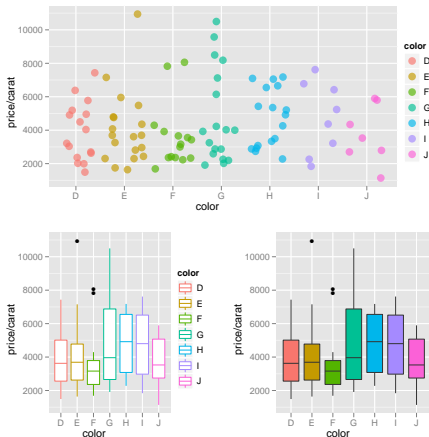
```
> p <- ggplot(df, aes(x = variable, y = value, fill = variable)) +  
+   geom_bar(width = 1) + coord_polar("y", start=pi / 3) +  
+   opts(title = "Pie Chart")  
> print(p)
```



ggplot: Arranging Graphics on One Page

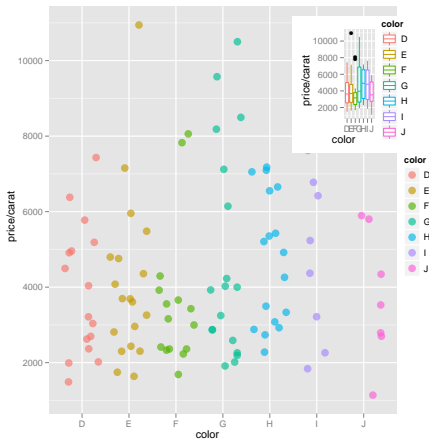
```
> library(grid)
> a <- ggplot(dsmall, aes(color, price/carat)) + geom_jitter(size=4, alpha = I(1 / 1.5), aes(color=color))
> b <- ggplot(dsmall, aes(color, price/carat, color=color)) + geom_boxplot()
> c <- ggplot(dsmall, aes(color, price/carat, fill=color)) + geom_boxplot() + opts(legend.position = "none")
> grid.newpage() # Open a new page on grid device
> pushViewport(viewport(layout = grid.layout(2, 2))) # Assign to device viewport with 2 by 2 grid layout
> print(a, vp = viewport(layout.pos.row = 1, layout.pos.col = 1:2))
> print(b, vp = viewport(layout.pos.row = 2, layout.pos.col = 1))
> print(c, vp = viewport(layout.pos.row = 2, layout.pos.col = 2, width=0.3, height=0.3, x=0.8, y=0.8))
```

ggplot: Arranging Graphics on One Page



ggplot: Inserting Graphics into Plots

```
> # pdf("insert.pdf")
> print(a)
> print(b, vp=viewport(width=0.3, height=0.3, x=0.8, y=0.8))
> # dev.off()
```



Outline

Overview

Graphics Environments

- Base Graphics

- Grid Graphics

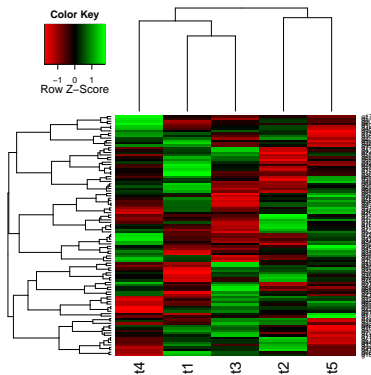
- lattice

- ggplot2

Specialty Graphics

Trees and Heatmaps

```
> library(gplots)
> y <- matrix(rnorm(500), 100, 5, dimnames=list(paste("g", 1:100, sep=""),
+ paste("t", 1:5, sep="")))
> hr <- hclust(as.dist(1-cor(t(y), method="pearson")), method="complete")
> hc <- hclust(as.dist(1-cor(y, method="spearman")), method="complete")
> heatmap.2(y, Rowv=as.dendrogram(hr), Colv=as.dendrogram(hc), col=redgreen(75), scale="row",
+ density.info="none", trace="none")
```

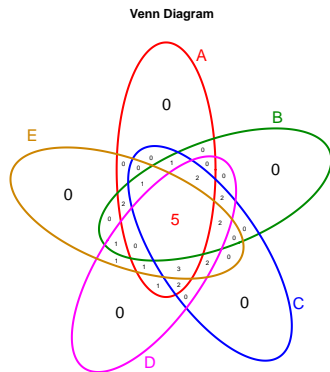


Venn Diagrams (Code)

```
> source("http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/My_R_Scripts/overLapper.R")

> setlist5 <- list(A=sample(letters, 18), B=sample(letters, 16), C=sample(letters, 20), D=sample(letters, 20))
> OLlist5 <- overLapper(setlist=setlist5, sep="_", type="vennsets")
> counts <- sapply(OLlist5$Venn_List, length)
> # pdf("venn.pdf")
> vennPlot(counts=counts, ccol=c(rep(1,30),2), lcex=1.5, ccex=c(rep(1.5,5), rep(0.6,25),1.5))
> # dev.off()
```

Venn Diagram (Plot)



Unique objects: All = 26; S1 = 18; S2 = 16; S3 = 20; S4 = 22; S5 = 18

Figure: Venn Diagram

Compound Depictions with ChemmineR

```
> library(ChemmineR)
> data(sdfsamplle)
> plot(sdfsamplle[1], print=FALSE)
```

CMP1

