Outline

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
  - Cookbook for R 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 `theme()`.
- Preferred input data object
  - `qgplot`: `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

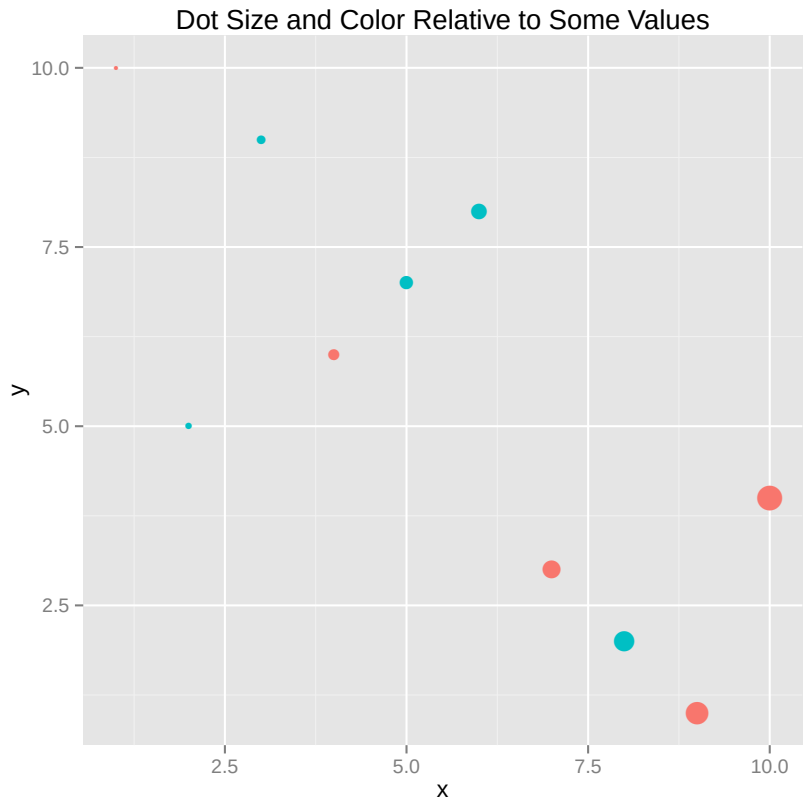
```
> qplot(x, y, geom="point", size=x, color=cat) +
+       theme(legend.position = "none")
```

Plot different shapes

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

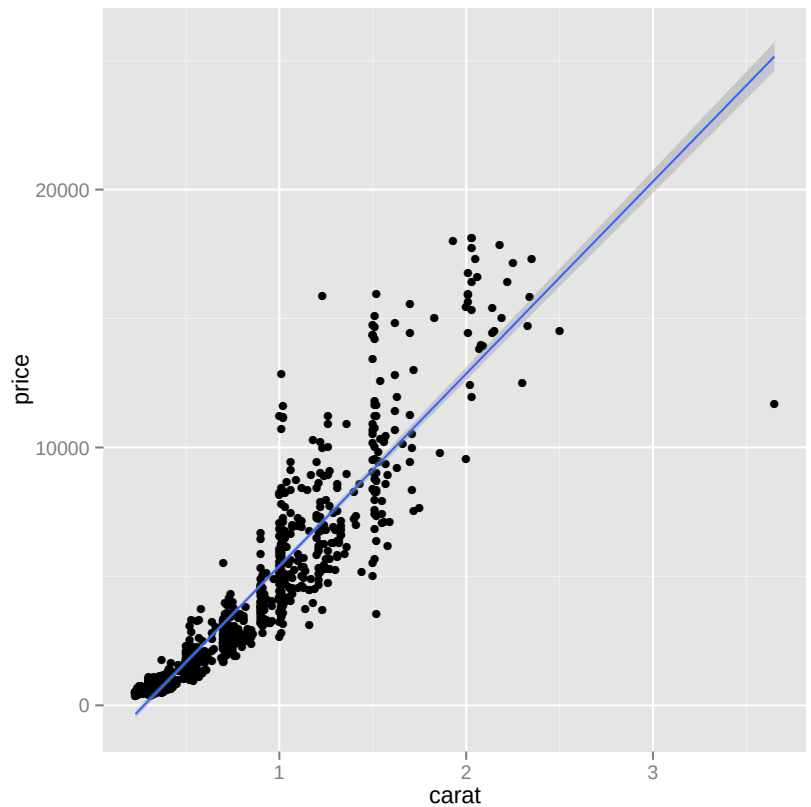
qplot: Scatter Plot with qplot

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



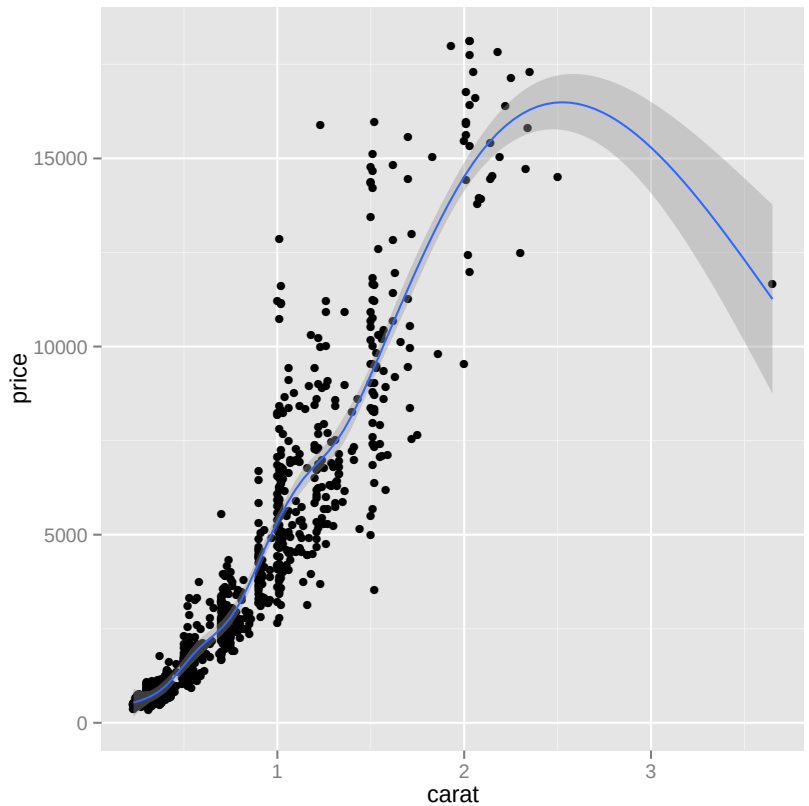
qplot: Scatter Plot with Regression Line

```
> 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 `theme()`
- Some examples
  - Change background color to white

```
...  + theme(panel.background=element_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

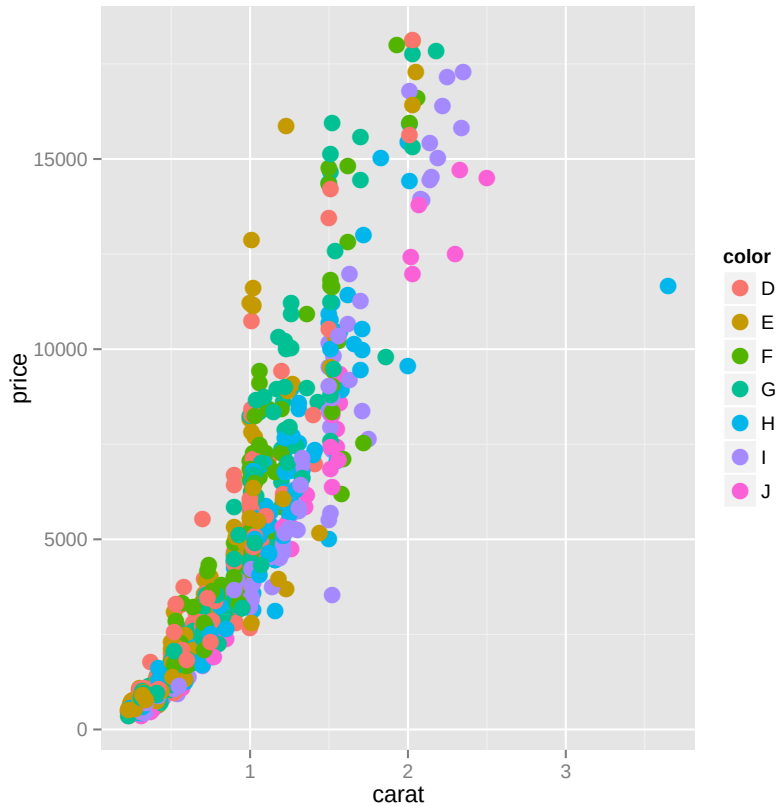
```
> p %+% diamonds[sample(nrow(diamonds), 100),]
```

Saves plot stored in variable p to file

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

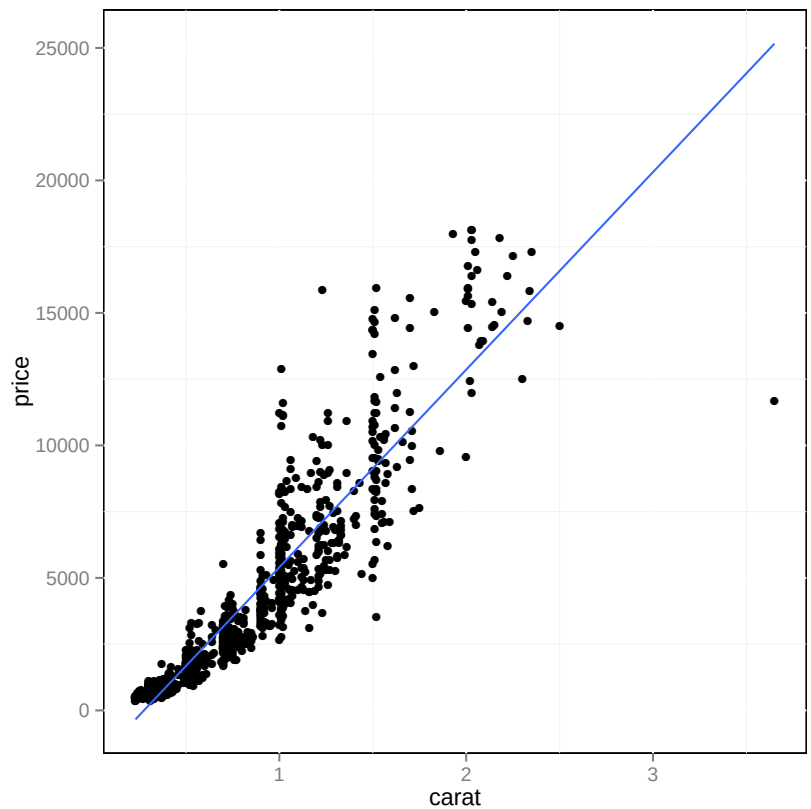
ggplot: Scatter Plot

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



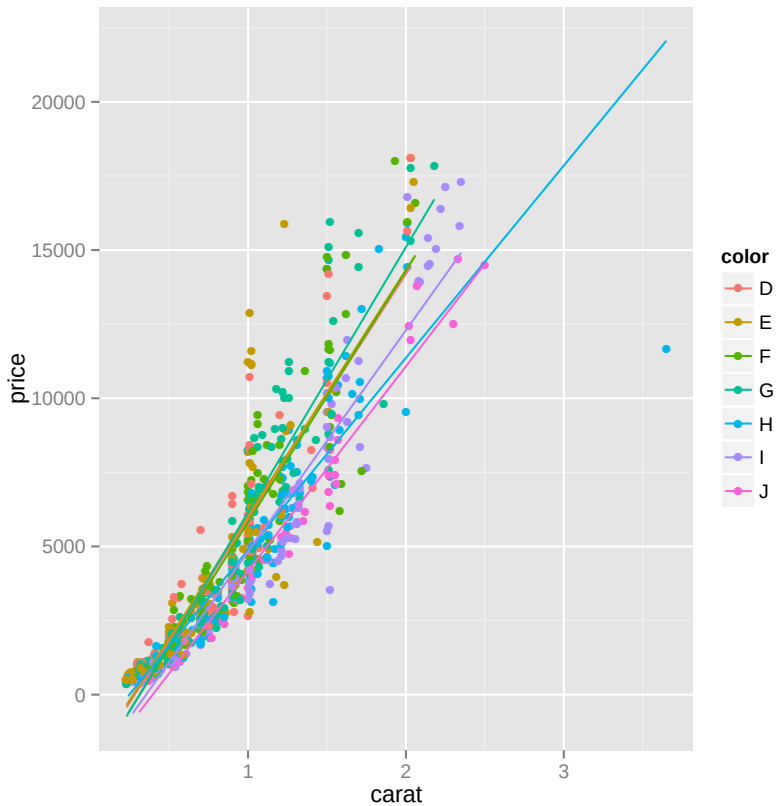
ggplot: Scatter Plot with Regression Line

```
> p <- ggplot(dsmall, aes(carat, price)) + geom_point() +
+             geom_smooth(method="lm", se=FALSE) +
+             theme(panel.background=element_rect(fill = "white", colour = "blac
> print(p)
```



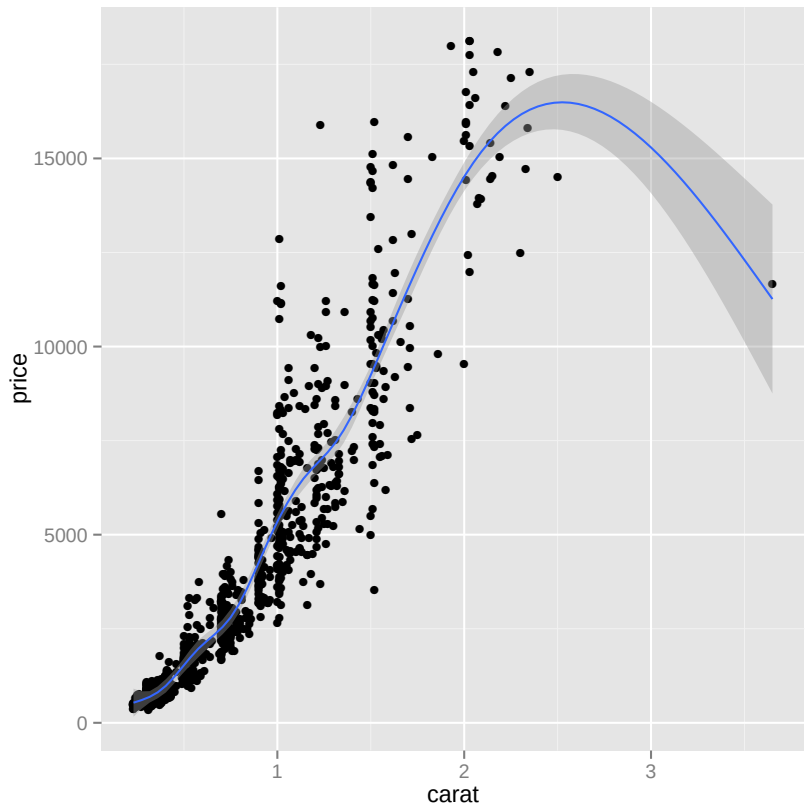
ggplot: Scatter Plot with Several Regression Lines

```
> 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)
```



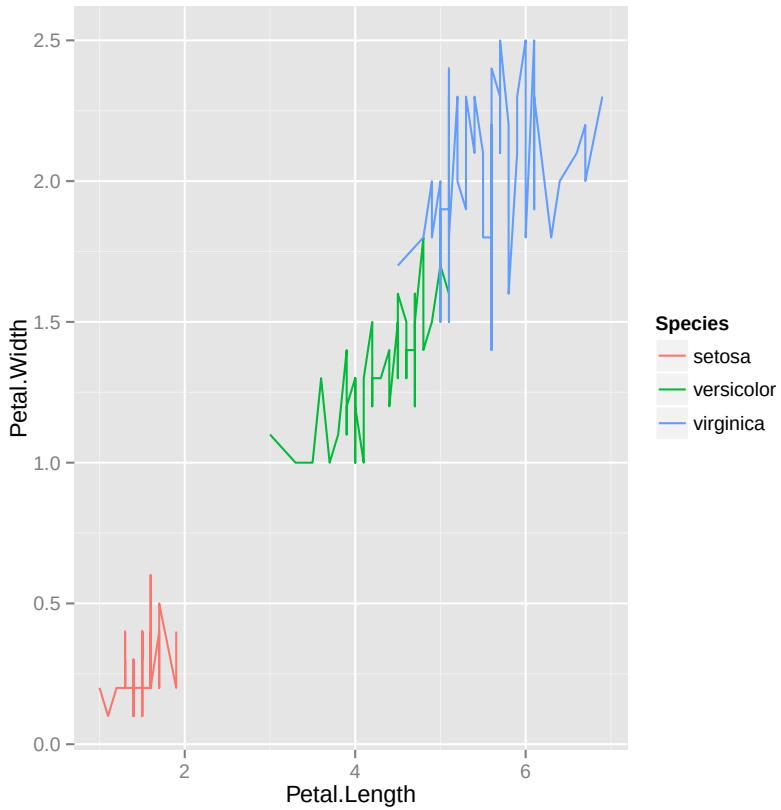
ggplot: Scatter Plot with Local Regression Curve (loess)

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



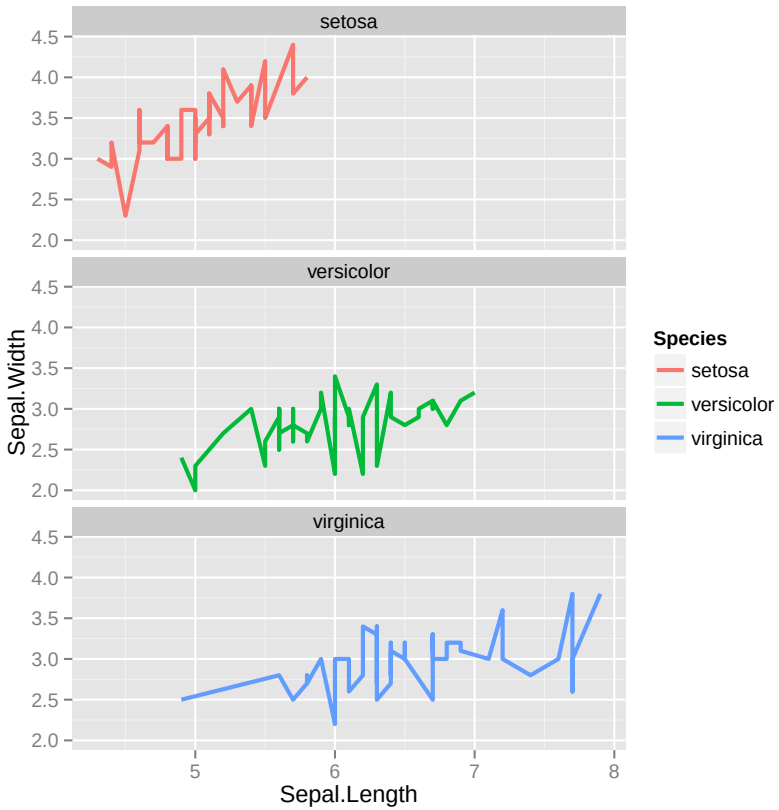
ggplot: Line Plot

```
> 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` functionss 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 saparate 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)
```

Convert iris_mean with melt

```
> library(reshape2) # Defines melt function
> df_mean <- melt(iris_mean, id.vars=c("Species"), variable.name = "Samples", value.
```

Convert iris_sd with melt

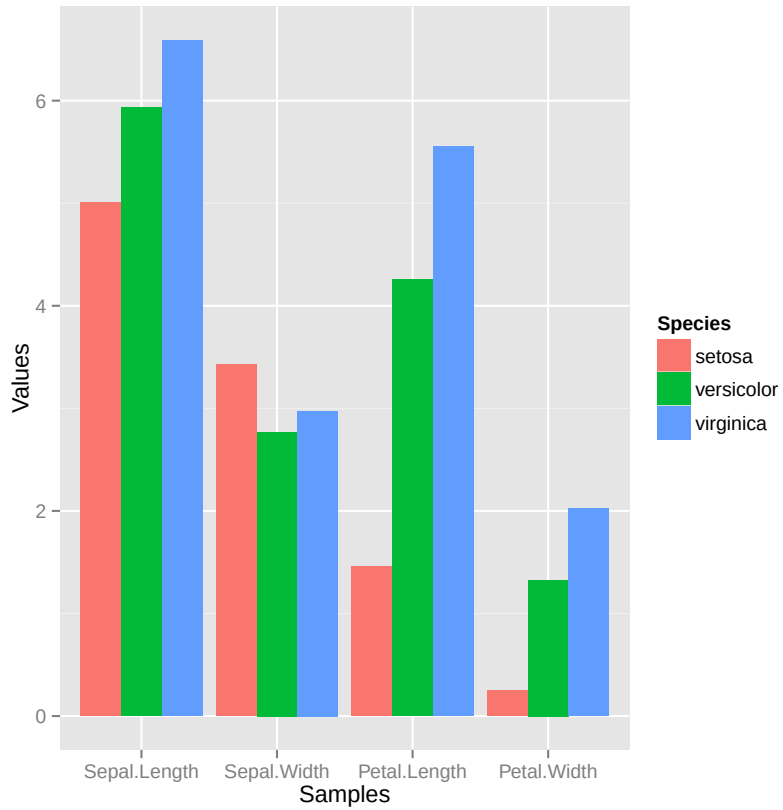
```
> df_sd <- melt(iris_sd, id.vars=c("Species"), variable.name = "Samples", value.name
```

Define standard deviation limits

```
> limits <- aes(ymax = df_mean[,"Values"] + df_sd[,"Values"], ymin=df_mean[,"Values"
```

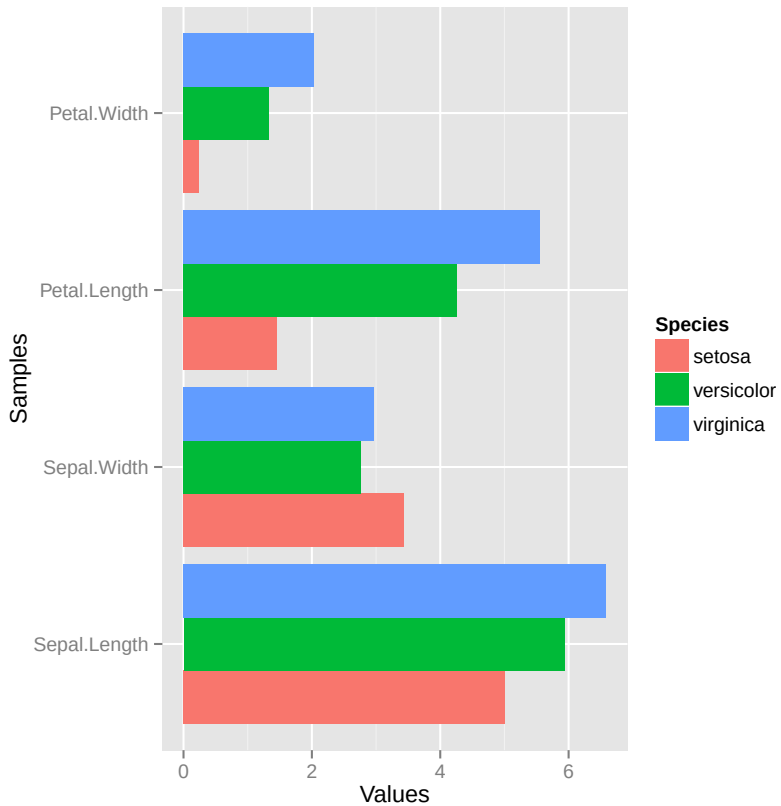
ggplot: Bar Plot

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



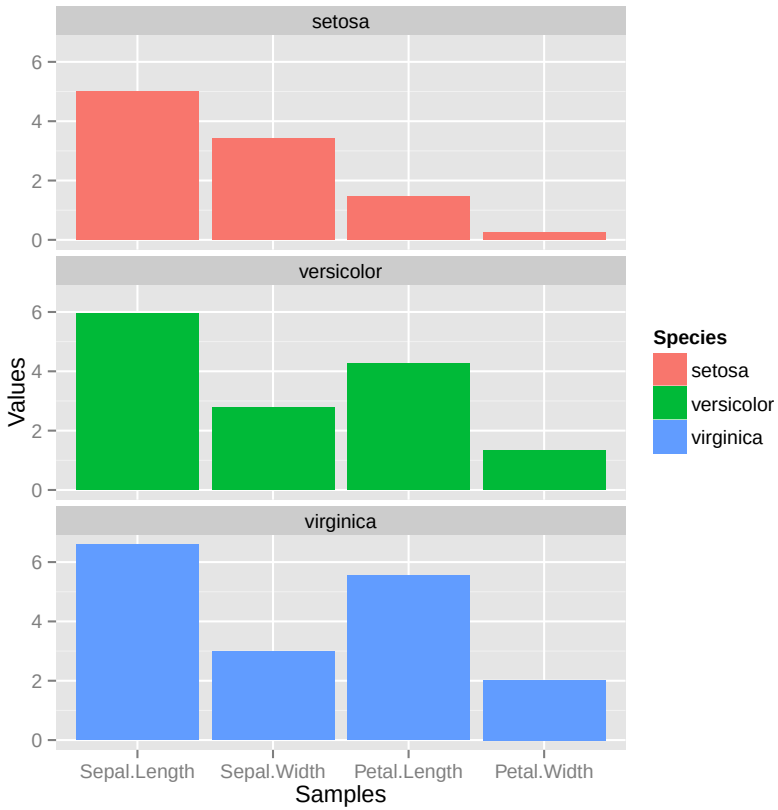
ggplot: Bar Plot Sideways

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



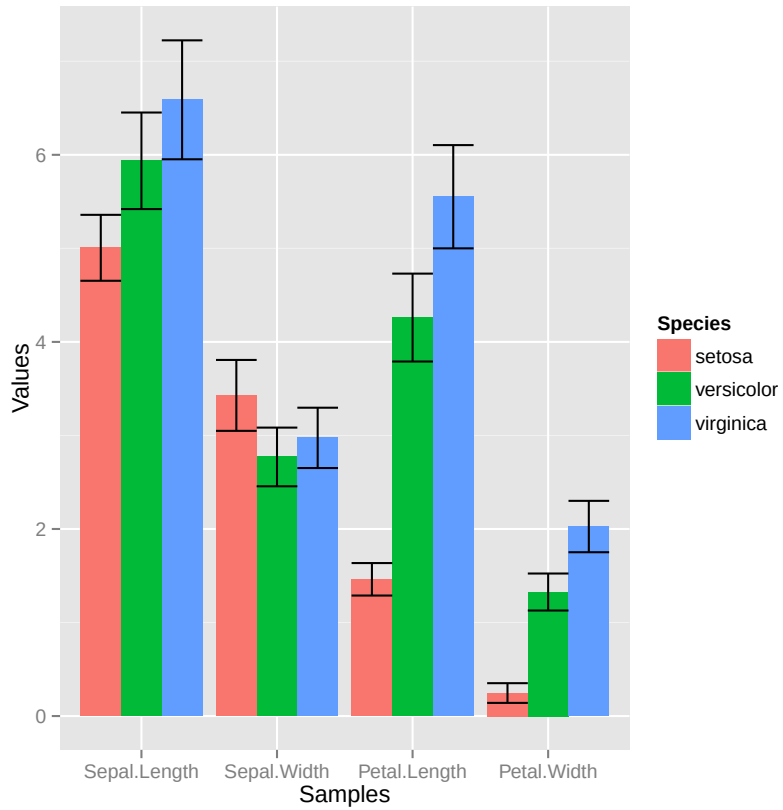
ggplot: Bar Plot with Faceting

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



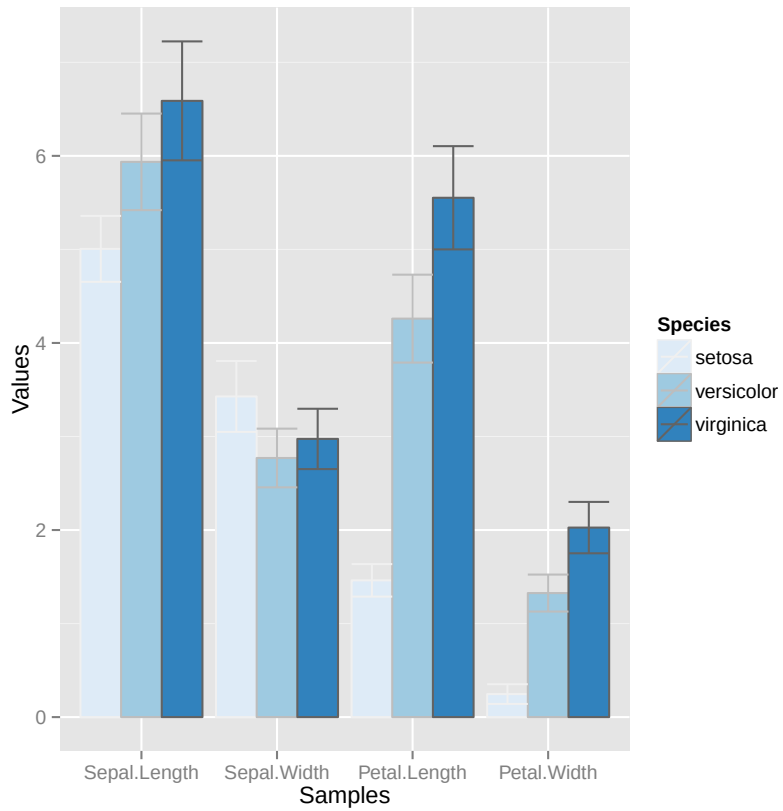
ggplot: Bar Plot with Error Bars

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



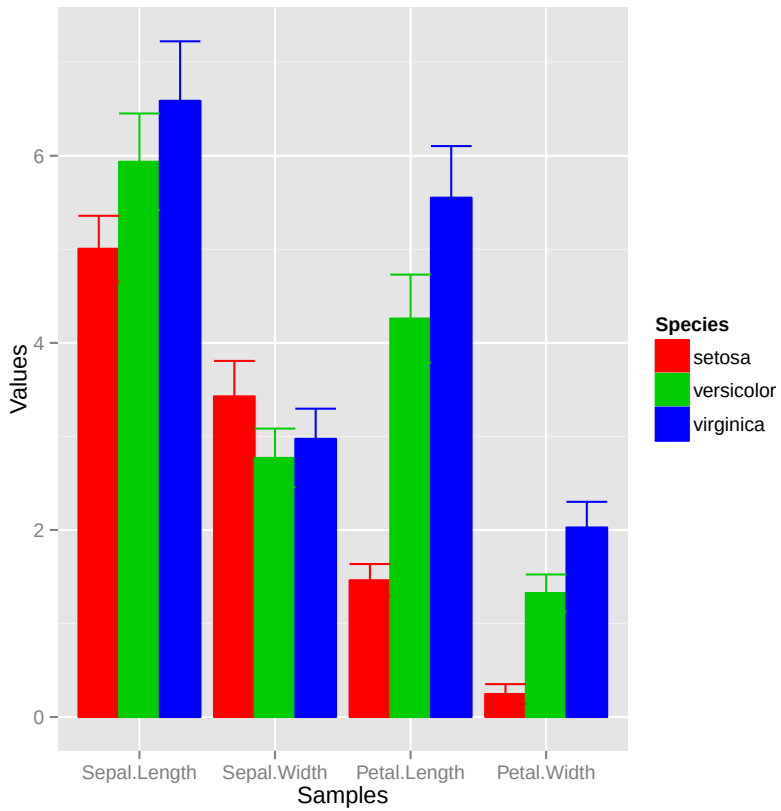
ggplot: Changing Color Settings

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



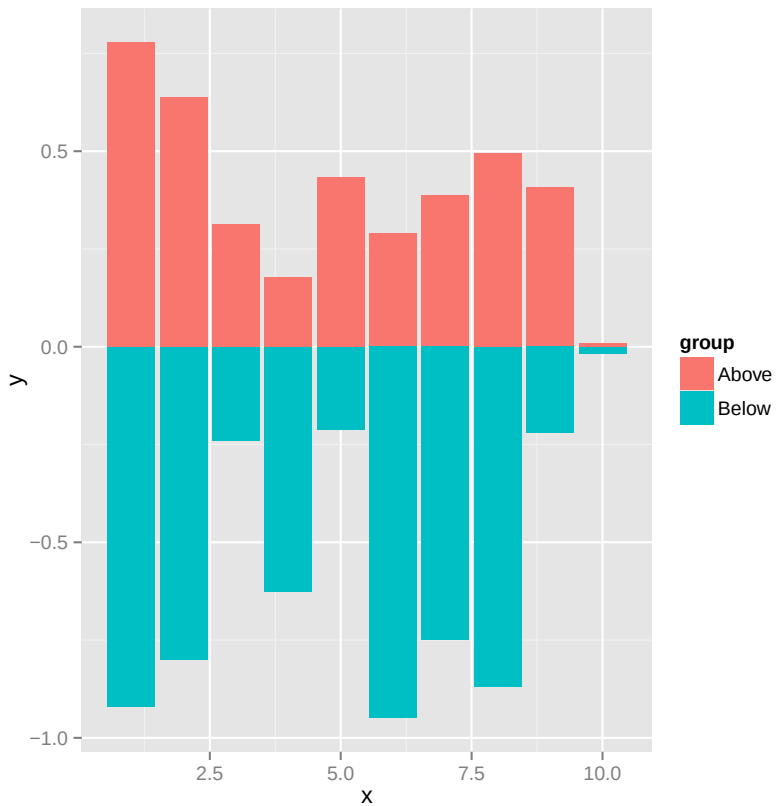
ggplot: Using Standard Colors

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



ggplot: Mirrored Bar Plots

```
> df <- data.frame(group = rep(c("Above", "Below"), each=10), x = rep(1:10, 2), y = c(runif(10, 0, 1), runi
> p <- ggplot(df, aes(x=x, y=y, fill=group)) +
+             geom_bar(stat="identity", position="identity")
> 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 `melt` function from the *reshape2* package to bring the results into the expected format for `ggplot`.

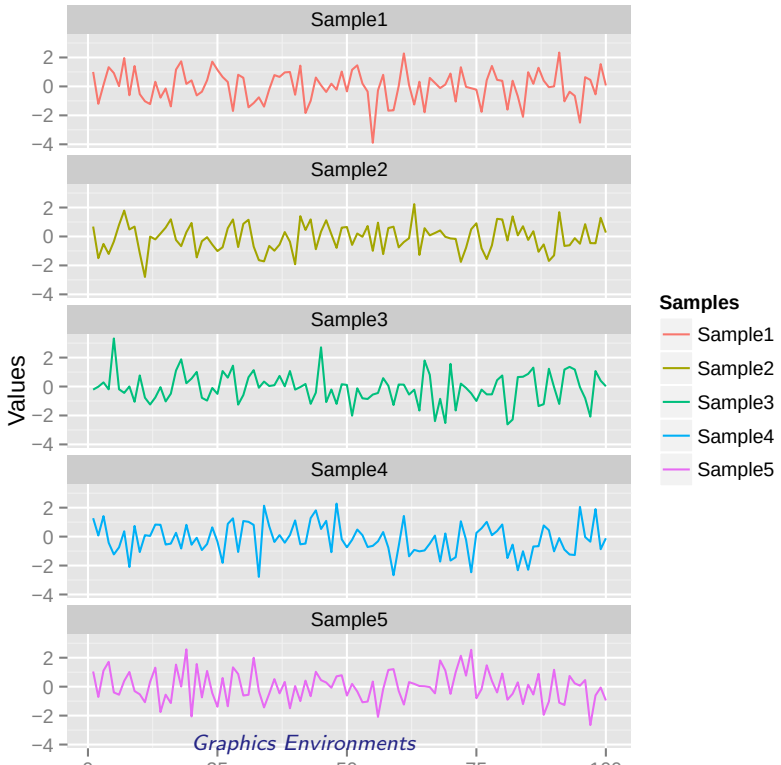
Task 2 Generate two bar plots: one with stacked bars and one with horizontally arranged bars.

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
```

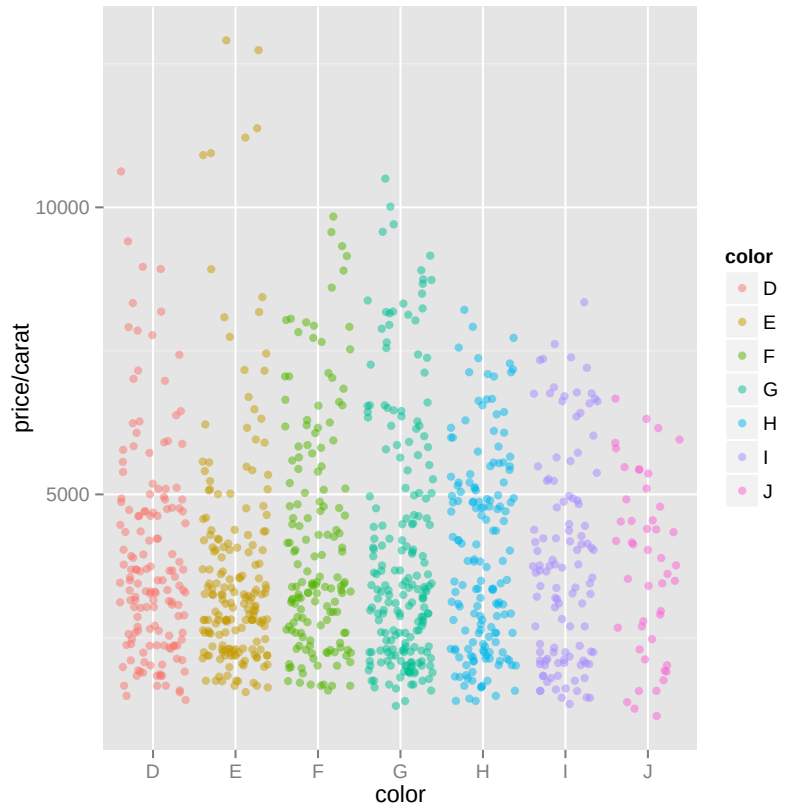
ggplot: Data Reformatting Example for Line Plot

```
> y <- matrix(rnorm(500), 100, 5, dimnames=list(paste("g", 1:100, sep=""), paste("Sample", 1:5, sep="")))
> y <- data.frame(Position=1:length(y[,1]), y)
> y[1:4, ] # First rows of input format expected by melt()
   Position    Sample1    Sample2     Sample3     Sample4    Sample5
g1        1  1.0002088  0.6850199 -0.21324932  1.27195056  1.0479301
g2        2 -1.2024596 -1.5004962 -0.01111579  0.07584497 -0.7100662
g3        3  0.1023678 -0.5153367  0.28564390  1.41522878  1.1084695
g4        4  1.3294248 -1.2084007 -0.19581898 -0.42361768  1.7139697
> df <- melt(y, id.vars=c("Position"), variable.name = "Samples", value.name="Values")
> 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()
```



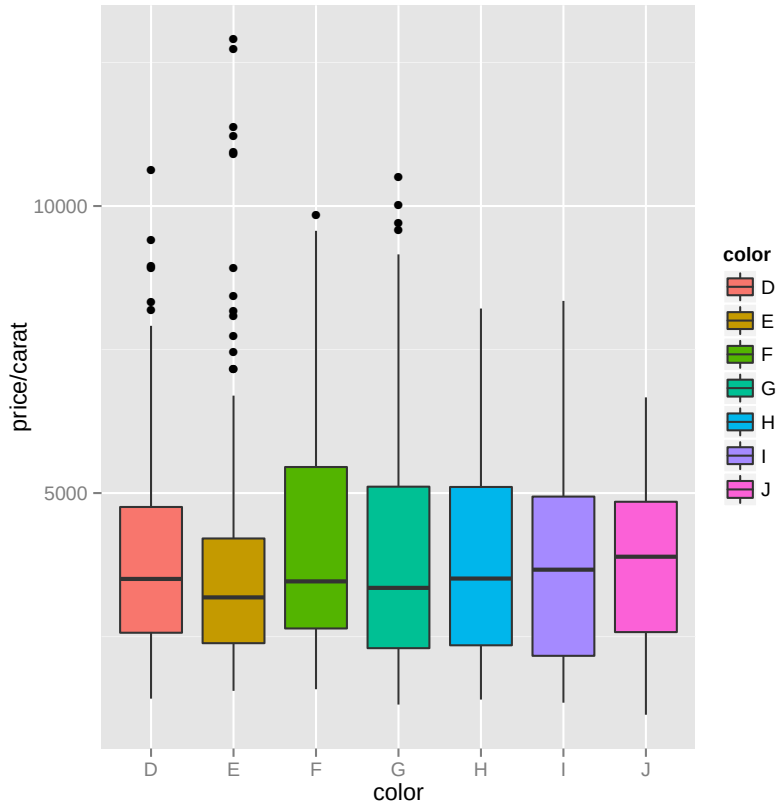
ggplot: Jitter Plots

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



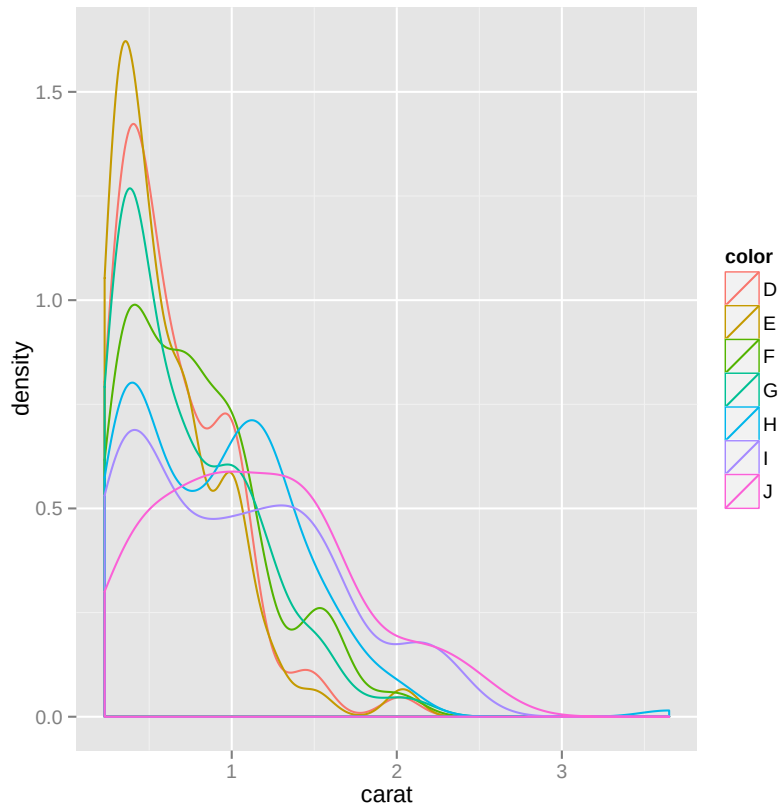
ggplot: Box Plots

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



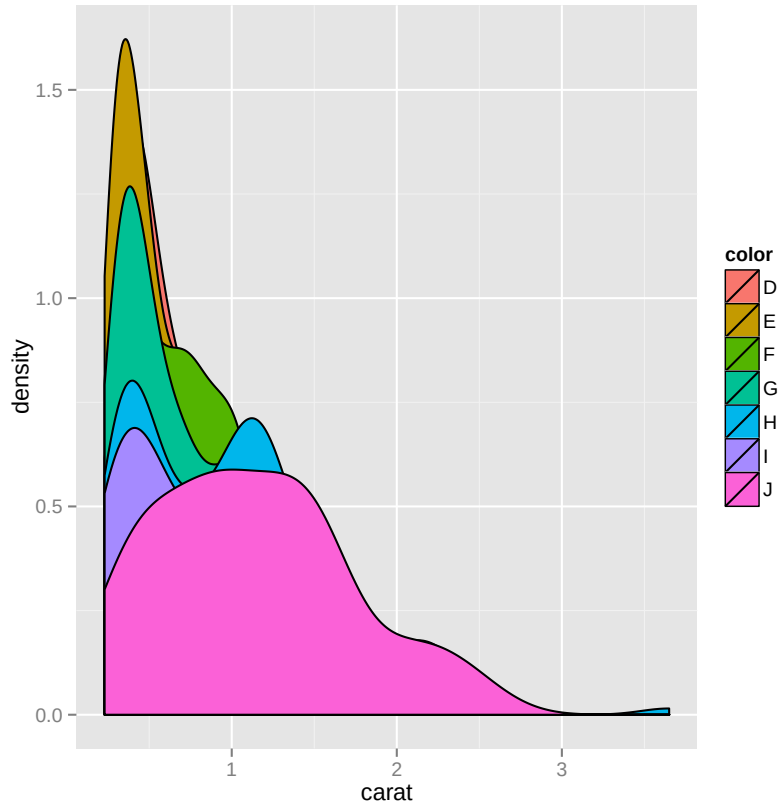
ggplot: Density Plot with Line Coloring

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



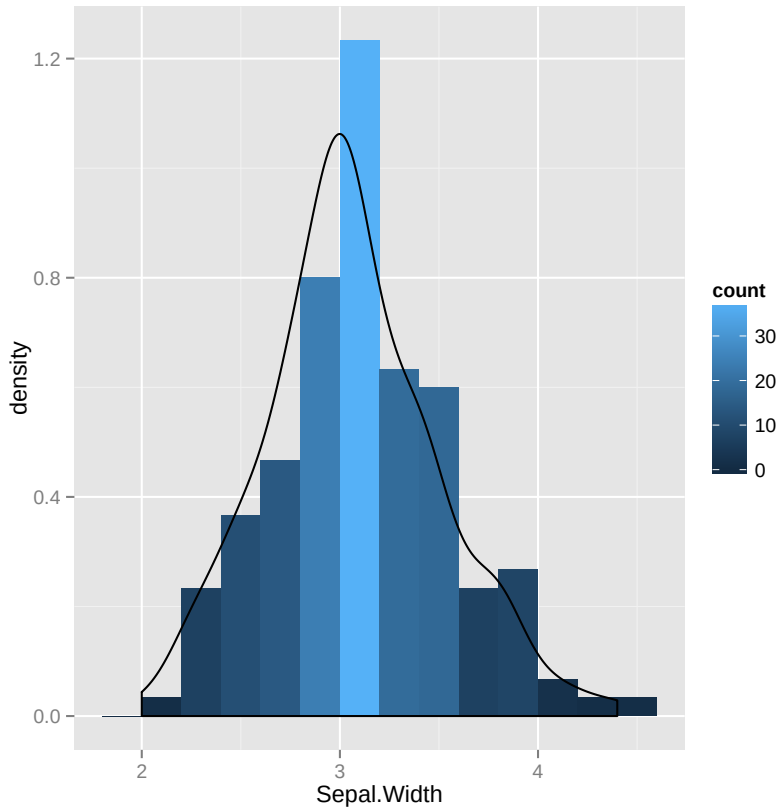
ggplot: Density Plot with Area Coloring

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



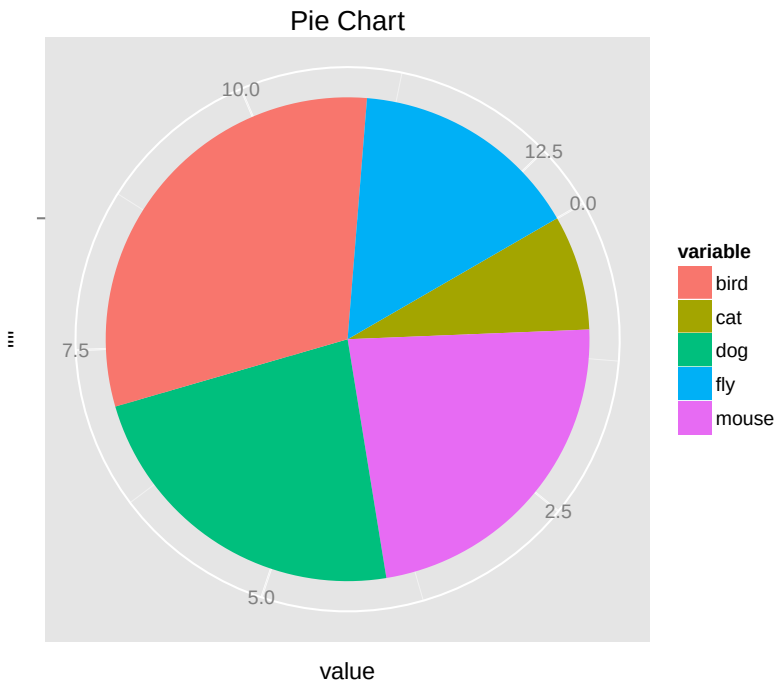
ggplot: Histograms

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



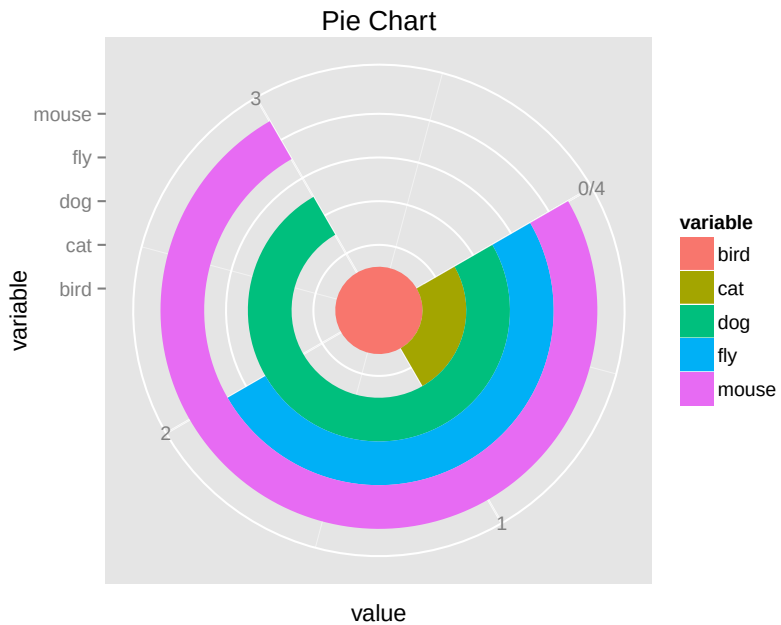
ggplot: Pie Chart

```
> 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, stat="identity") +
+             coord_polar("y", start=pi / 3) + ggtitle("Pie Chart")
> print(p)
```



ggplot: Wind Rose Pie Chart

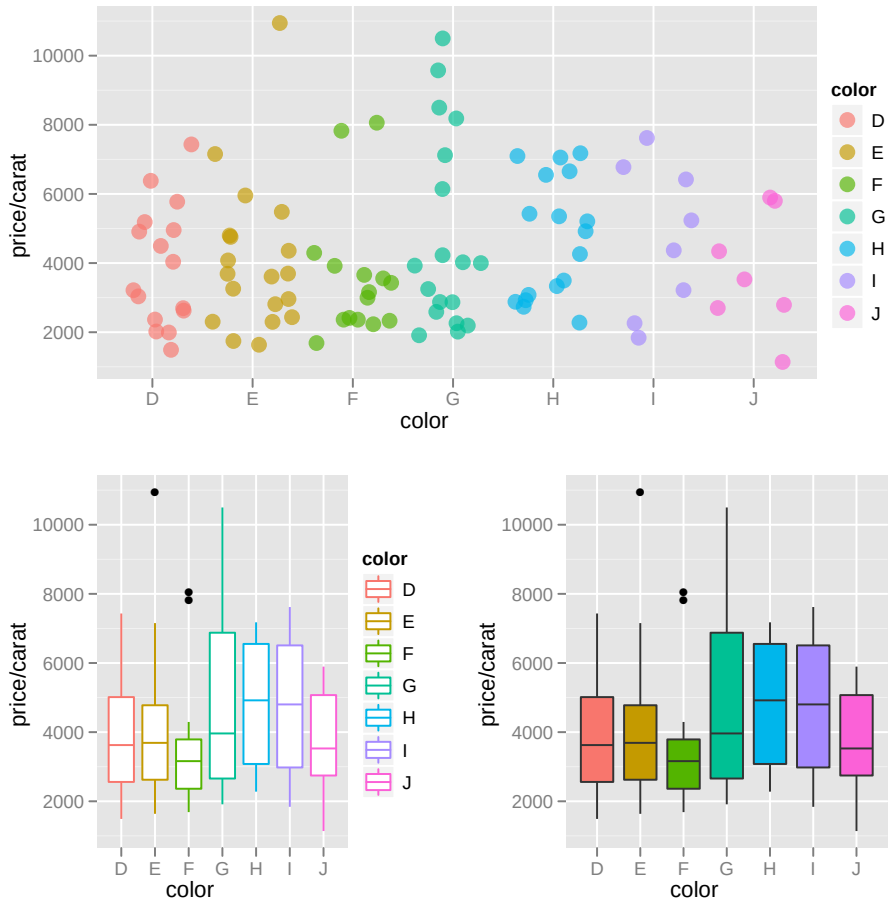
```
> p <- ggplot(df, aes(x = variable, y = value, fill = variable)) +
+         geom_bar(width = 1, stat="identity") + coord_polar("y", start=pi / 3) +
+         ggtitle("Pie Chart")
> print(p)
```



ggplot: Arranging Graphics on One Page

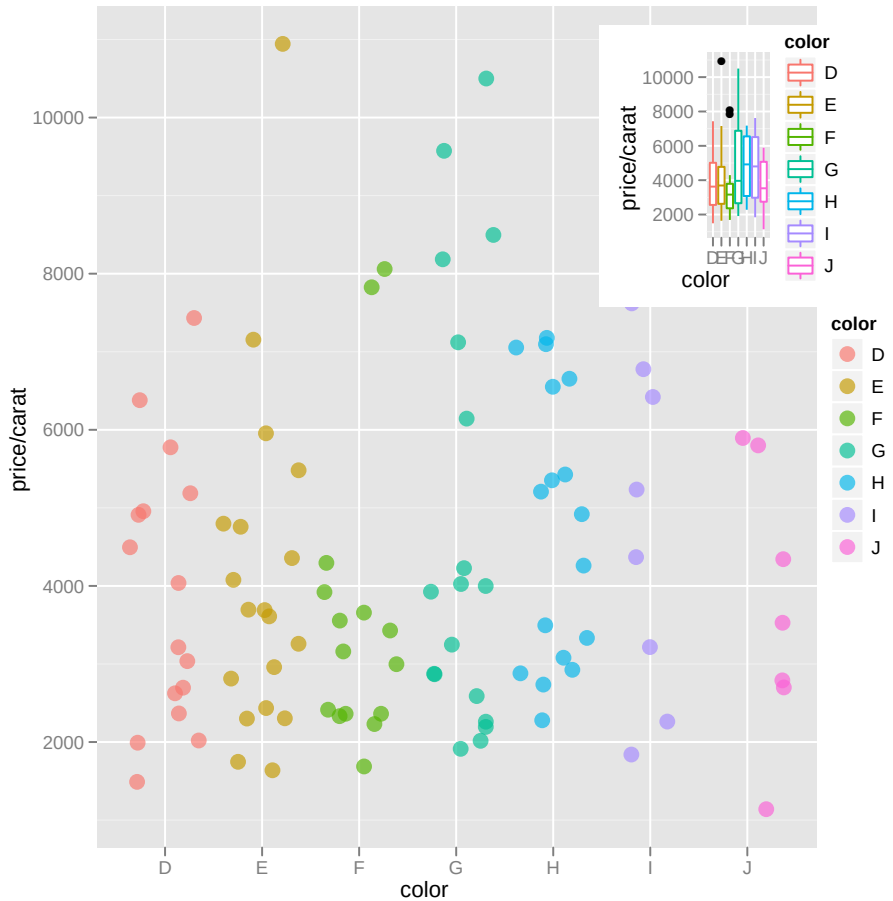
```
> library(grid)
> a <- ggplot(dsmall, aes(color, price/carat)) + geom_jitter(size=4, alpha = I(1 / 1.5), aes(color=color))
> b <- ggplot(dsmall, aes(color, price/carat, color=color)) + geom_boxplot()
> c <- ggplot(dsmall, aes(color, price/carat, fill=color)) + geom_boxplot() + theme(legend.position = "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

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, 2
> 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)

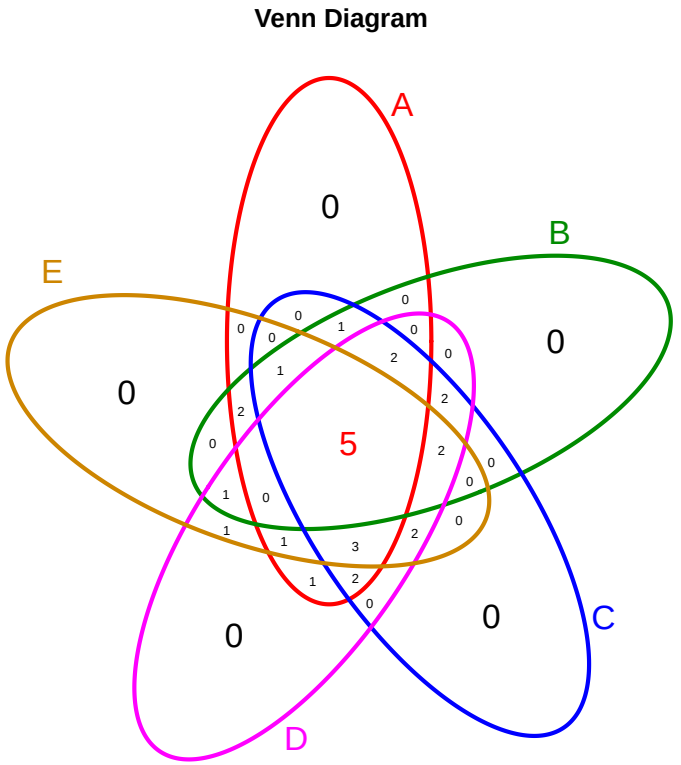
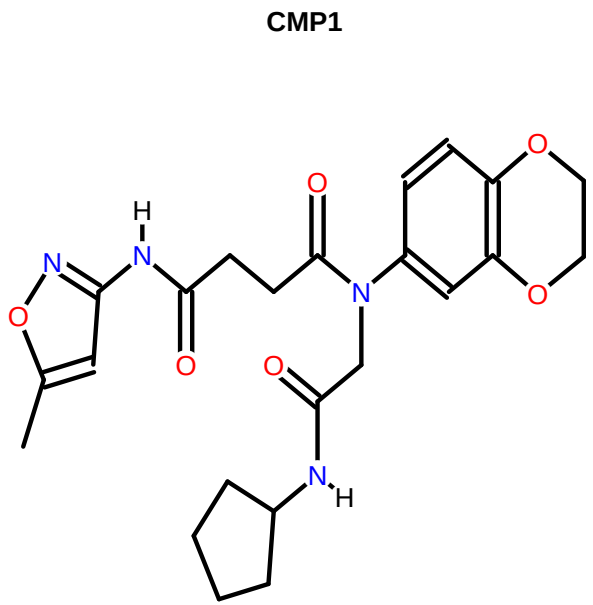


Figure: Venn Diagram

Compound Depictions with ChemmineR

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



Outline

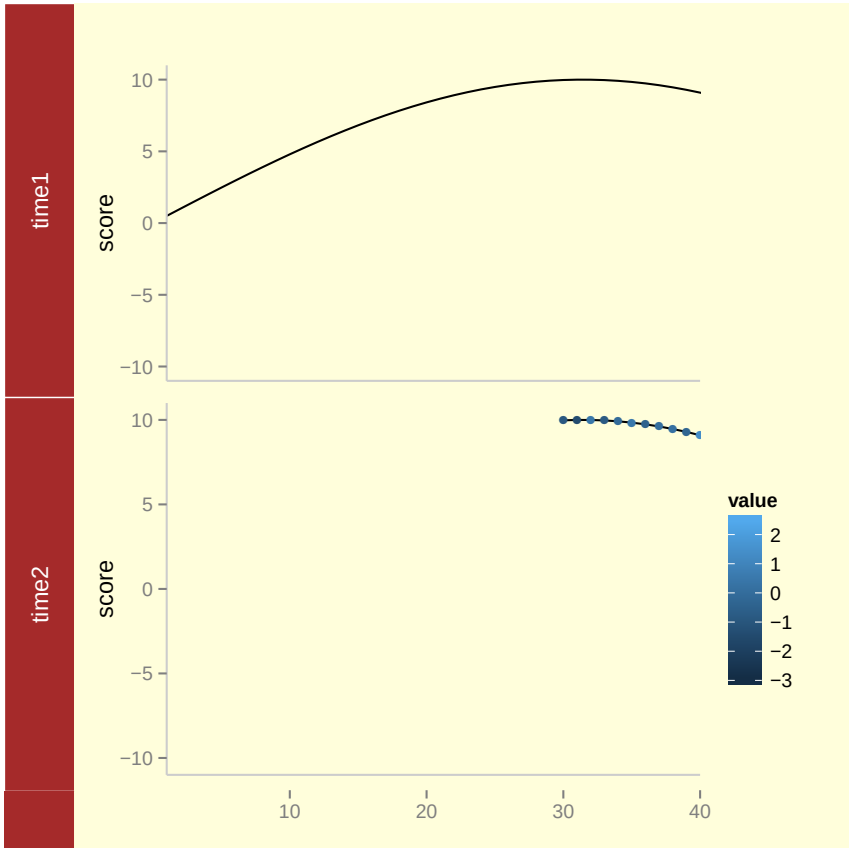
Outline

ggbio: A Programmable Genome Browser

- A genome browser is a visulalization tool for plotting different types of genomic data in separate tracks along chromosomes.
- The *ggbio* package (Yin et al., 2012) facilitates plotting of complex genome data objects, such as read alignments (SAM/BAM), genomic context/annotation information (gff/txdb), variant calls (VCF/BCF), and more. To easily compare these data sets, it extends the faceting facility of *ggplot2* to genome browser-like tracks.
- Most of the core object types for handling genomic data with R/Bioconductor are supported: `GRanges`, `GAlignments`, `VCF`, etc. For more details, see Table 1.1 of the *ggbio* vignette Link.
- *ggbio*'s convenience plotting function is `autoplot`. For more customizable plots, one can use the generic `ggplot` function.
- Apart from the standard `ggplot2` plotting components, `ggbio` defines serval new components useful for genomic data visualization. A detailed list is given in Table 1.2 of the vignette Link.
- Useful web sites:
  - *ggbio* manual Link
  - *ggbio* functions Link
  - `autoplot` demo Link

Tracks: Aligning Plots Along Chromosomes

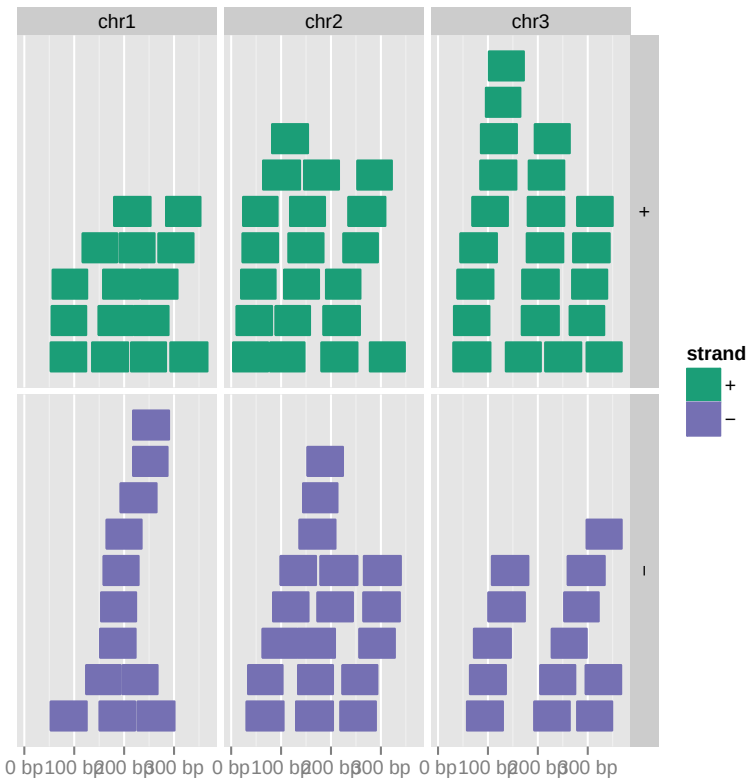
```
> library(ggbio)
> df1 <- data.frame(time = 1:100, score = sin((1:100)/20)*10)
> p1 <- qplot(data = df1, x = time, y = score, geom = "line")
> df2 <- data.frame(time = 30:120, score = sin((30:120)/20)*10, value = rnorm(120-30 +1))
> p2 <- ggplot(data = df2, aes(x = time, y = score)) + geom_line() + geom_point(size = 2, aes(color = value
> tracks(time1 = p1, time2 = p2) + xlim(1, 40) + theme_tracks_sunset()
```



Plotting Genomic Ranges

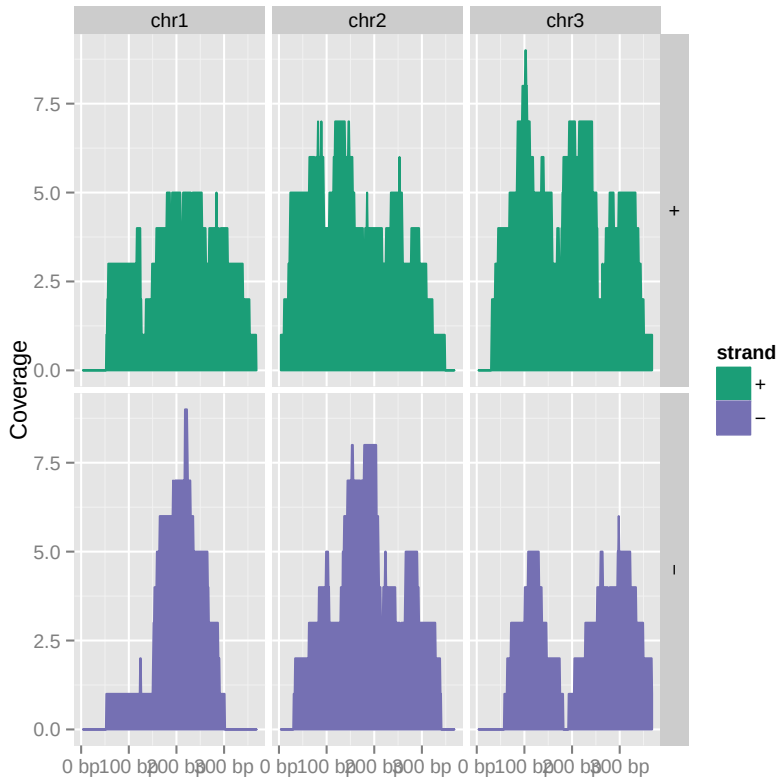
GRanges objects are essential for storing alignment or annotation ranges in R/Bioconductor. The following creates a sample GRanges object and plots its content.

```
> library(GenomicRanges)
> set.seed(1); N <- 100; gr <- GRanges(seqnames = sample(c("chr1", "chr2", "chr3"), size = N, replace = TRU
> autoplot(gr, aes(color = strand, fill = strand), facets = strand ~ seqnames)
```



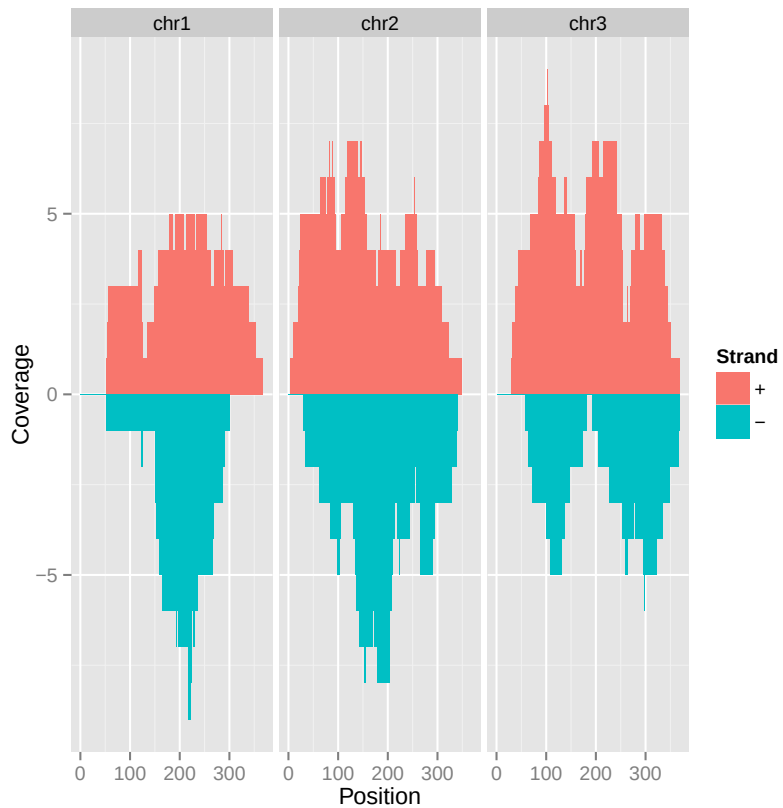
Plotting Coverage Instead of Ranges

```
> autoplot(gr, aes(color = strand, fill = strand), facets = strand ~ seqnames, stat = "coverage")
```



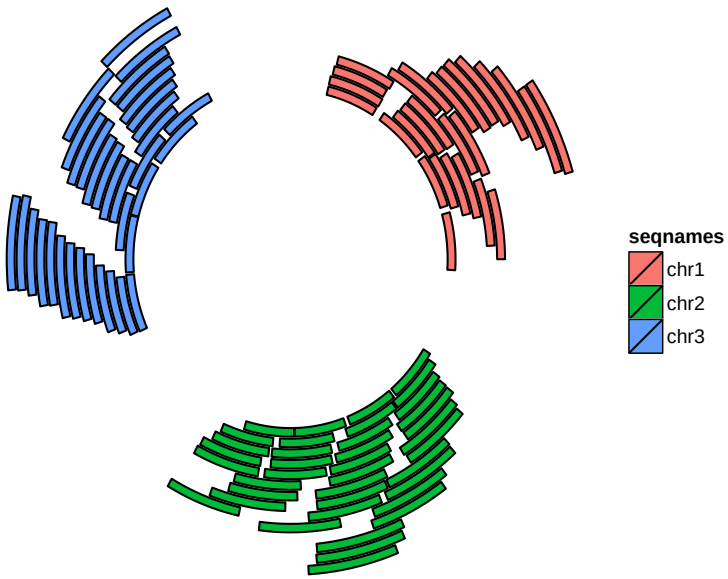
Mirrored Coverage Plot

```
> pos <- sapply(coverage(gr[strand(gr)=="+"]), as.numeric)
> pos <- data.frame(Chr=rep(names(pos), sapply(pos, length)), Strand=rep("+", length(unlist(pos))), Positio
> neg <- sapply(coverage(gr[strand(gr)=="-"]), as.numeric)
> neg <- data.frame(Chr=rep(names(neg), sapply(neg, length)), Strand=rep("-", length(unlist(neg))), Positio
> covdf <- rbind(pos, neg)
> p <- ggplot(covdf, aes(Position, Coverage, fill=Strand)) +
+             geom_bar(stat="identity", position="identity") + facet_wrap(~Chr)
> p
```



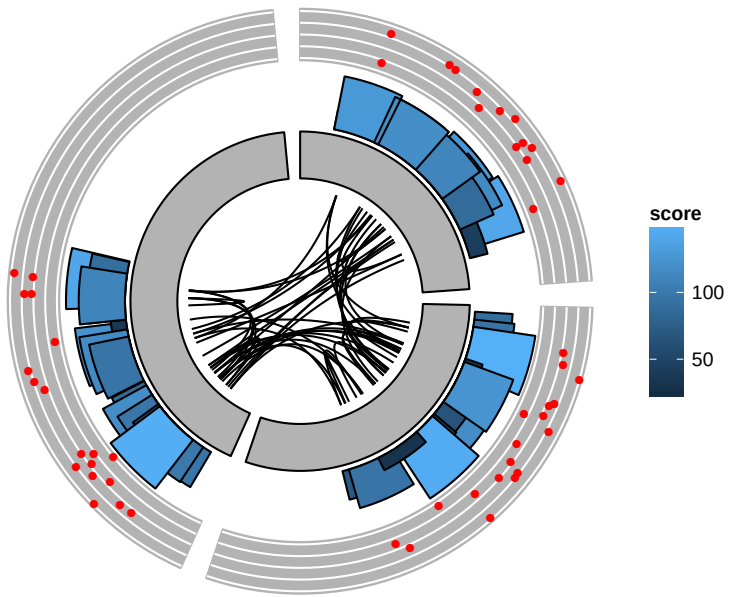
Circular Layout

```
> autoplot(gr, layout = "circle", aes(fill = seqnames))
```



More Complex Circular Example

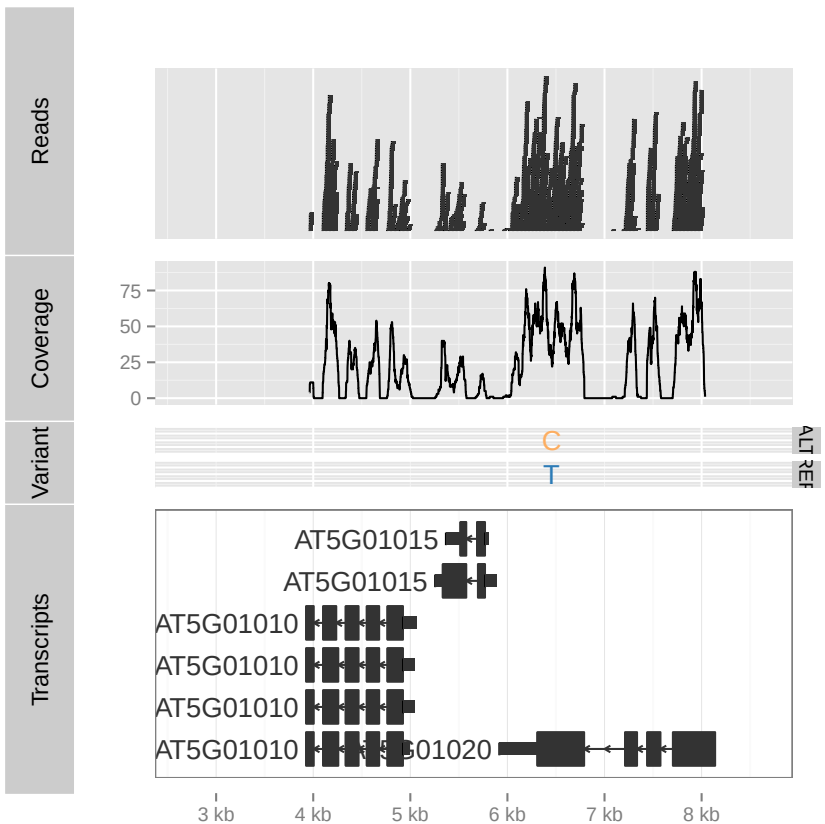
```
> seqlengths(gr) <- c(400, 500, 700)
> values(gr)$to.gr <- gr[sample(1:length(gr), size = length(gr))]
> idx <- sample(1:length(gr), size = 50)
> gr <- gr[idx]
> ggplot() + layout_circle(gr, geom = "ideo", fill = "gray70", radius = 7, trackWidth = 3) +
+   layout_circle(gr, geom = "bar", radius = 10, trackWidth = 4,
+                 aes(fill = score, y = score)) +
+   layout_circle(gr, geom = "point", color = "red", radius = 14,
+                 trackWidth = 3, grid = TRUE, aes(y = score)) +
+   layout_circle(gr, geom = "link", linked.to = "to.gr", radius = 6, trackWidth = 1)
```



Viewing Alignments and Variants

To make the following example work, please download and unpack this data archive Link containing GFF, BAM and VCF sample files.

```
> library(rtracklayer); library(GenomicFeatures); library(Rsamtools); library(VariantAnnotation)
> ga <- readGAlignmentsFromBam("./data/SRR064167.fastq.bam", use.names=TRUE, param=ScanBamParam(which=GRang
> p1 <- autoplot(ga, geom = "rect")
> p2 <- autoplot(ga, geom = "line", stat = "coverage")
> vcf <- readVcf(file="data/varianttools_gnsap.vcf", genome="ATH1")
> p3 <- autoplot(vcf[seqnames(vcf)=="Chr5"], type = "fixed") + xlim(4000, 8000) + theme(legend.position = "
> txdb <- makeTranscriptDbFromGFF(file="./data/TAIR10_GFF3_trunc.gff", format="gff3")
> p4 <- autoplot(txdb, which=GRanges("Chr5", IRanges(4000, 8000)), names.expr = "gene_id")
> tracks(Reads=p1, Coverage=p2, Variant=p3, Transcripts=p4, heights = c(0.3, 0.2, 0.1, 0.35)) + ylab("")
```



Additional Sample Plots

- `autoplot` demo Link

Outline

Additional Packages for Visualizing Genome Data

- Gviz Link
- RCircos (Zhang et al., 2013) Link
- Genome Graphs Link
- genoPlotR Link

Outline

Outline

What is Clustering?

- Clustering is the classification/partitioning of data objects into similarity groups (clusters) according to a defined distance measure.
- It is used in many fields, such as machine learning, data mining, pattern recognition, image analysis, genomics, systems biology, etc.
- Machine learning typically regards data clustering as a form of unsupervised learning.

Why Clustering and Data Mining in R?

- Efficient data structures and functions for clustering.
- Efficient environment for algorithm prototyping and benchmarking.
- Comprehensive set of clustering and machine learning libraries.
- Standard for data analysis in many areas.
- Overview: Clustering Task View on CRAN

Data Transformations

Choice depends on data set!

- Center & standardize
  1. Center: subtract vector mean from each value
  2. Standardize: devide by standard deviation

  $$\Rightarrow Mean = 0 \text{ and } STDEV = 1$$
- Center & scale with the `scale()` fuction
  1. Center: subtract vector mean from each value
  2. Scale: divide centered vector by their root mean square (rms)

  $$x_{rms} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} {x_i}^2}$$

  $$\Rightarrow Mean = 0 \text{ and } STDEV = 1$$
- Log transformation
- Rank transformation: replace measured values by ranks
- No transformation

Distance Methods

List of most common ones!

- Euclidean distance for two profiles X and Y

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Disadvantages: not scale invariant, not for negative correlations

- Maximum, Manhattan, Canberra, binary, Minowski, ...
- Correlation-based distance: $1 - r$
  - Pearson correlation coefficient (PCC)

$$r = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{(\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2)(\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2)}}$$

Disadvantage: outlier sensitive

  - Spearman correlation coefficient (SCC)

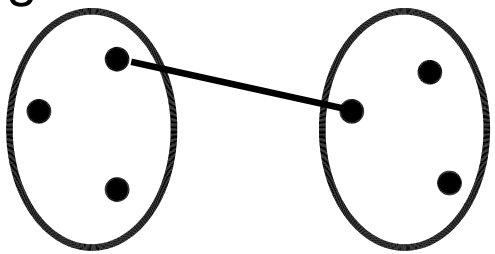
Same calculation as PCC but with ranked values!

There Are Many more Distance Measures

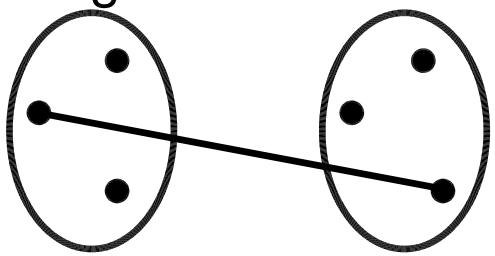
- If the distances among items are quantifiable, then clustering is possible.
- Choose the most accurate and meaningful distance measure for a given field of application.
- If uncertain then choose several distance measures and compare the results.

Cluster Linkage

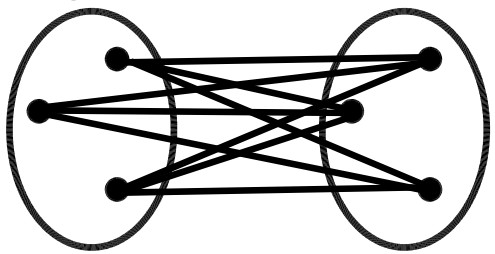
Single Linkage



Complete Linkage



Average Linkage



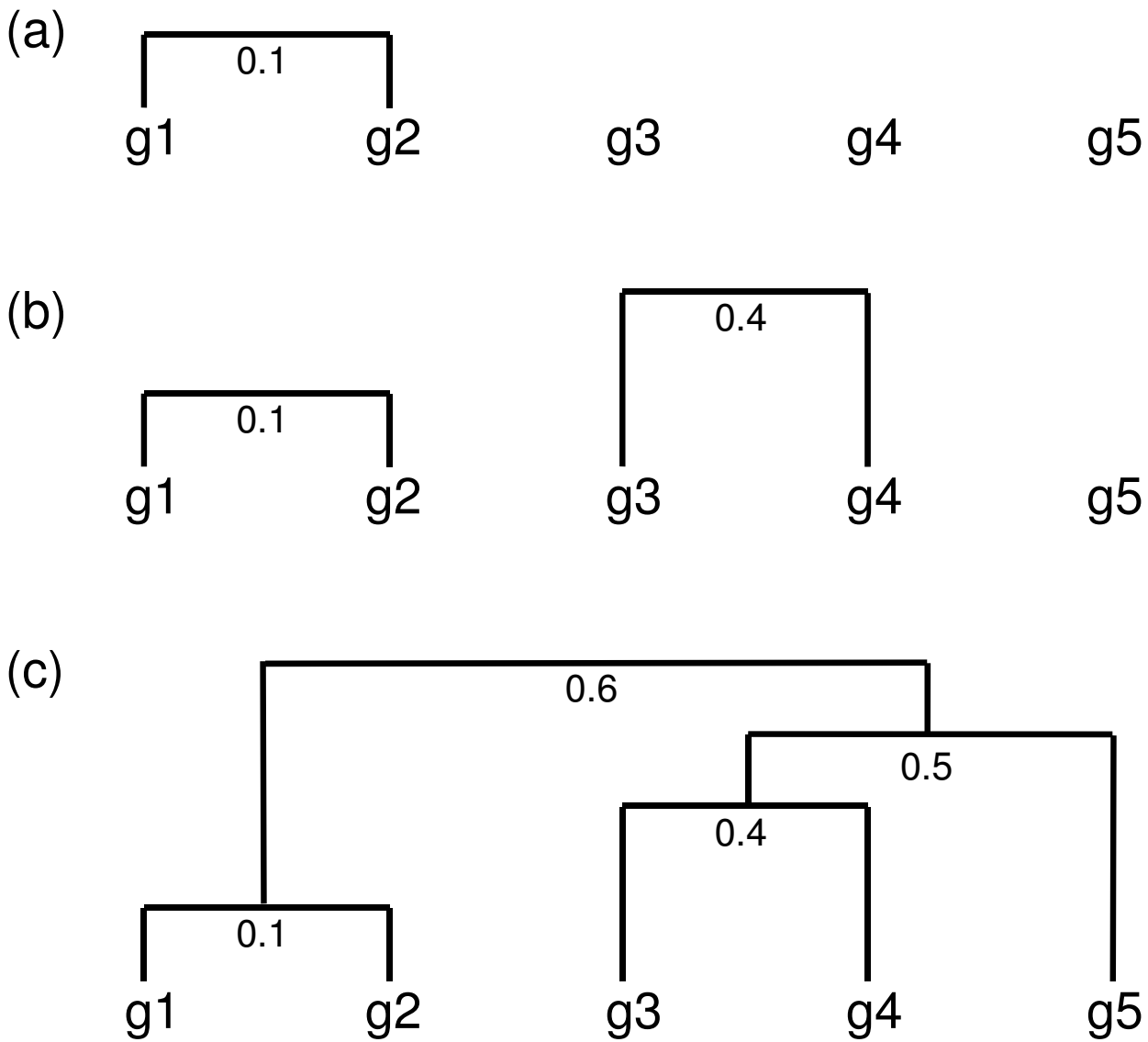
Outline

Hierarchical Clustering Steps

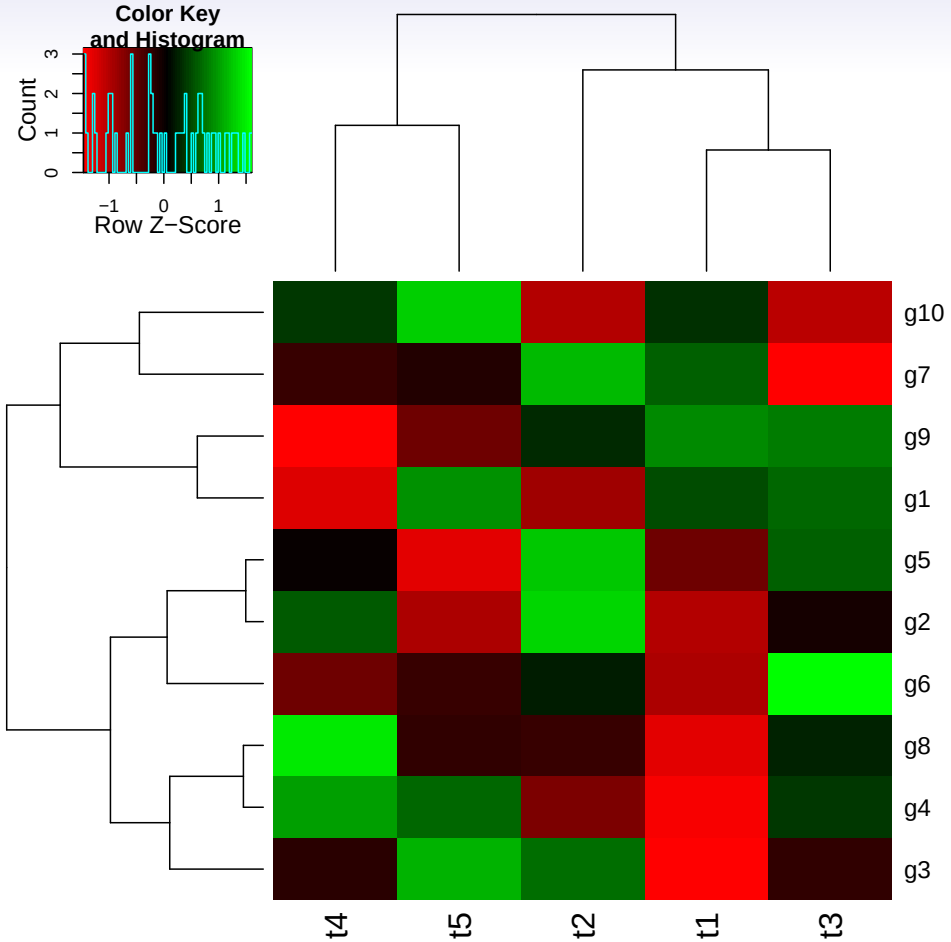
1. Identify clusters (items) with closest distance
2. Join them to new clusters
3. Compute distance between clusters (items)
4. Return to step 1

Hierarchical Clustering

Agglomerative Approach



Hierarchical Clustering Result with Heatmap



- A heatmap is a color coded table. To visually identify patterns, the rows and columns of a heatmap are often sorted by hierarchical clustering trees.
- In case of gene expression data, the row tree usually represents the genes, the column tree the treatments and the colors in the heat table represent the intensities or ratios of the underlying gene expression data set.

Hierarchical Clustering Approaches in R

1. Agglomerative approach (bottom-up)

   `hclust()` and `agnes()`

2. Divisive approach (top-down)

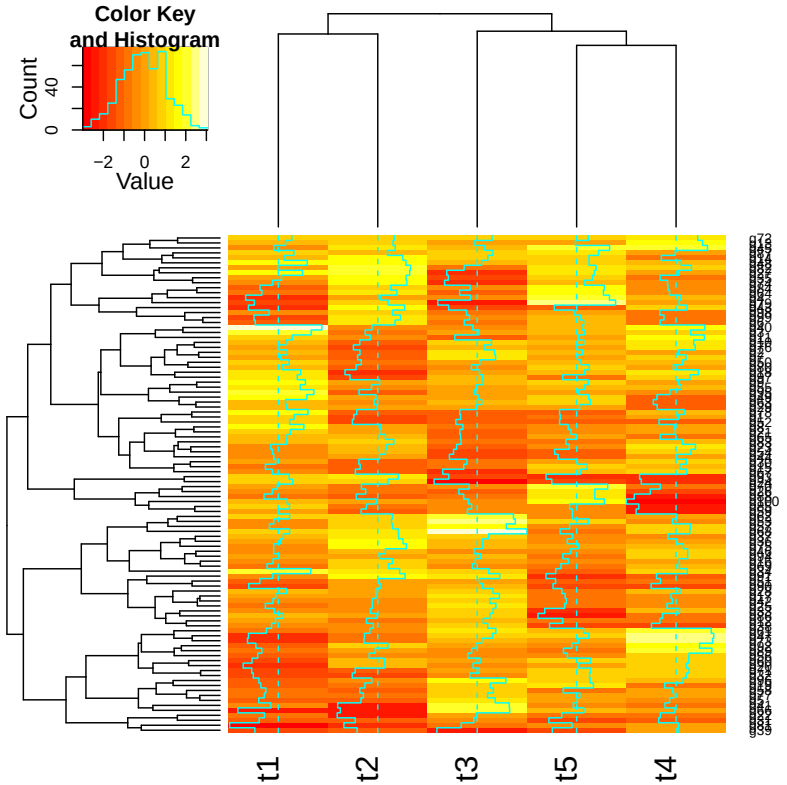
   `diana()`

Tree Cutting to Obtain Discrete Clusters

1. Node height in tree
2. Number of clusters
3. Search tree nodes by distance cutoff

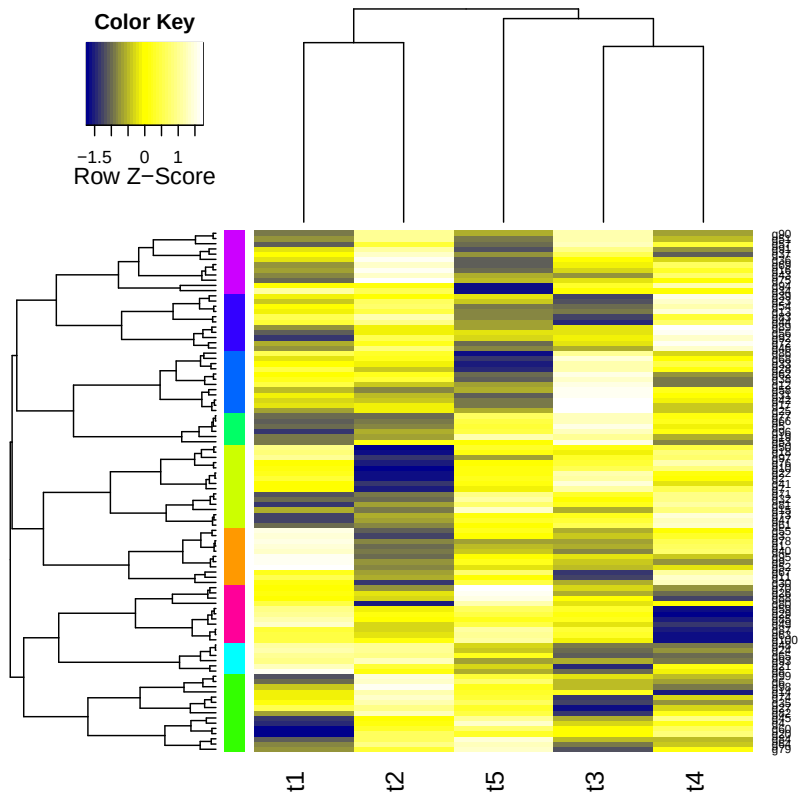
Example: hclust and heatmap.2

```
> library(gplots)
> y <- matrix(rnorm(500), 100, 5, dimnames=list(paste("g", 1:100, sep=""), paste("t", 1:5, sep="")))
> heatmap.2(y) # Shortcut to final result
```



Example: Stepwise Approach with Tree Cutting

```
> ## Row- and column-wise clustering
> hr <- hclust(as.dist(1-cor(t(y), method="pearson")), method="complete")
> hc <- hclust(as.dist(1-cor(y, method="spearman")), method="complete")
> ## Tree cutting
> mycl <- cutree(hr, h=max(hr$height)/1.5); mycolhc <- rainbow(length(unique(mycl)), start=0.1, end=0.9); mycolh
> ## Plot heatmap
> mycol <- colorpanel(40, "darkblue", "yellow", "white") # or try redgreen(75)
> heatmap.2(y, Rowv=as.dendrogram(hr), Colv=as.dendrogram(hc), col=mycol, scale="row", density.info="none", trac
```

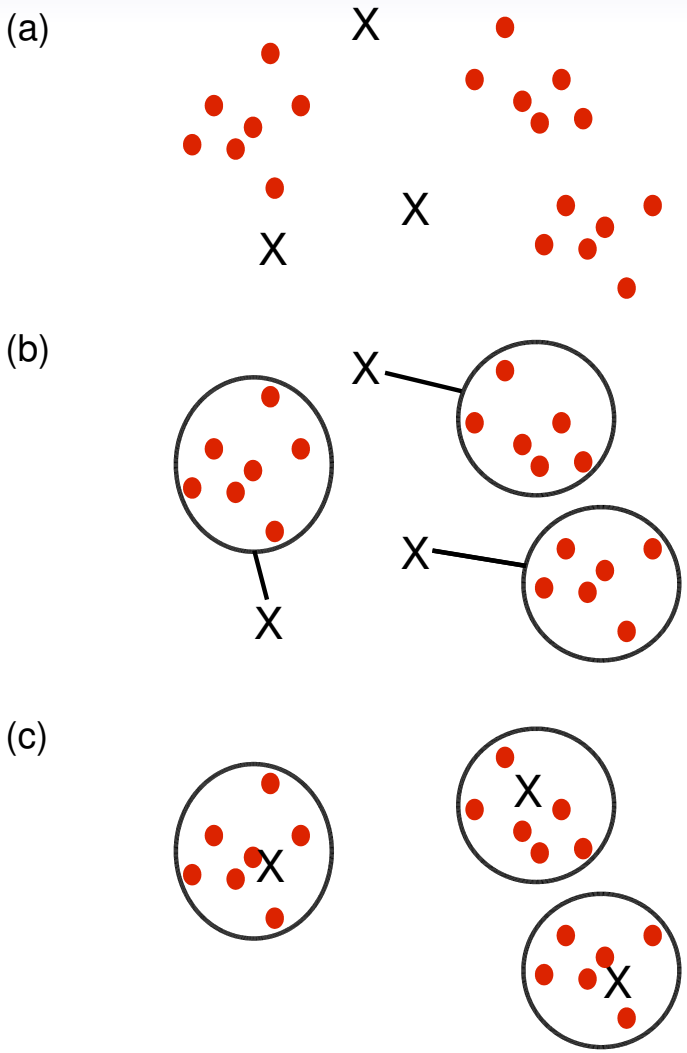


Outline

K-Means Clustering

1. Choose the number of k clusters
2. Randomly assign items to the k clusters
3. Calculate new centroid for each of the k clusters
4. Calculate the distance of all items to the k centroids
5. Assign items to closest centroid
6. Repeat until clusters assignments are stable

K-Means



Example: Clustering with `kmeans` Function

```
> km <- kmeans(t(scale(t(y))), 3)
> km$cluster
```

```
  g1   g2   g3   g4   g5   g6   g7   g8   g9  g10  g11  g12  g13  g14  g15  g16  g1
   2    1    2    3    1    3    1    2    2    1    3    1    3    2    3    3
 g45  g46  g47  g48  g49  g50  g51  g52  g53  g54  g55  g56  g57  g58  g59  g60  g6
   3    3    3    2    2    2    1    2    1    3    2    3    1    1    2    3
 g89  g90  g91  g92  g93  g94  g95  g96  g97  g98  g99 g100
   3    1    1    3    2    1    2    1    1    3    3    2
```

Fuzzy C-Means Clustering

1. In contrast to strict (hard) clustering approaches, fuzzy (soft) clustering methods allow multiple cluster memberships of the clustered items.
2. This is commonly achieved by assigning to each item a weight of belonging to each cluster.
3. Thus, items on the edge of a cluster, may be in the cluster to a lesser degree than items in the center of a cluster.
4. Typically, each item has as many coefficients (weights) as there are clusters that sum up for each item to one.

Example: Fuzzy Clustering with `fanny`

```
> library(cluster) # Loads the cluster library.
> fannyy <- fanny(y, k=4, metric = "euclidean", memb.exp = 1.2)
> round(fannyy$membership, 2)[1:4,]
```

```
   [,1] [,2] [,3] [,4]
g1 0.82 0.04 0.10 0.05
g2 0.82 0.05 0.12 0.01
g3 0.98 0.01 0.01 0.01
g4 0.03 0.82 0.03 0.12
```

```
> fannyy$clustering
```

```
  g1   g2   g3   g4   g5   g6   g7   g8   g9  g10  g11  g12  g13  g14  g15  g16  g1
   1    1    1    2    3    4    1    1    1    1    1    3    4    4    2    4
 g45  g46  g47  g48  g49  g50  g51  g52  g53  g54  g55  g56  g57  g58  g59  g60  g6
   2    4    2    4    1    1    3    1    3    4    1    2    3    3    3    2
 g89  g90  g91  g92  g93  g94  g95  g96  g97  g98  g99 g100
   2    3    3    2    4    3    1    2    1    4    4    4
```

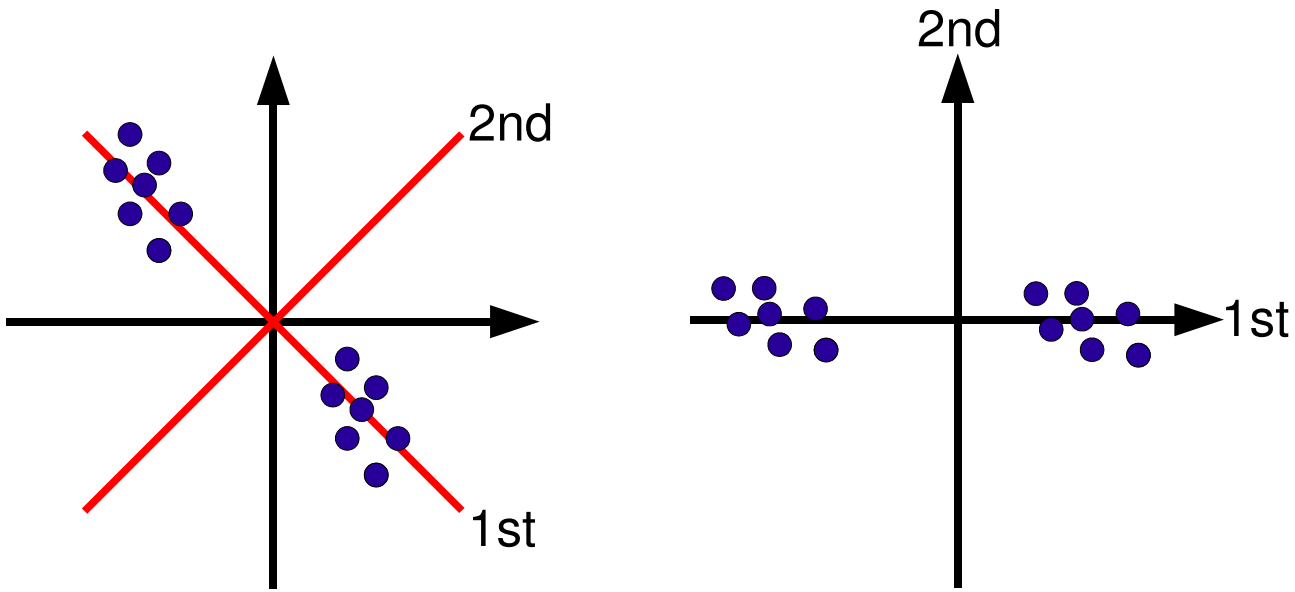
Principal Component Analysis (PCA)

Principal components analysis (PCA) is a data reduction technique that allows to simplify multidimensional data sets to 2 or 3 dimensions for plotting purposes and visual variance analysis.

Basic PCA Steps

- Center (and standardize) data
- First principal component axis
  - Accross centroid of data cloud
  - Distance of each point to that line is minimized, so that it crosses the maximum variation of the data cloud
- Second principal component axis
  - Orthogonal to first principal component
  - Along maximum variation in the data
- 1^{st} PCA axis becomes x-axis and 2^{nd} PCA axis y-axis
- Continue process until the necessary number of principal components is obtained

PCA on Two-Dimensional Data Set



Identifies the Amount of Variability between Components

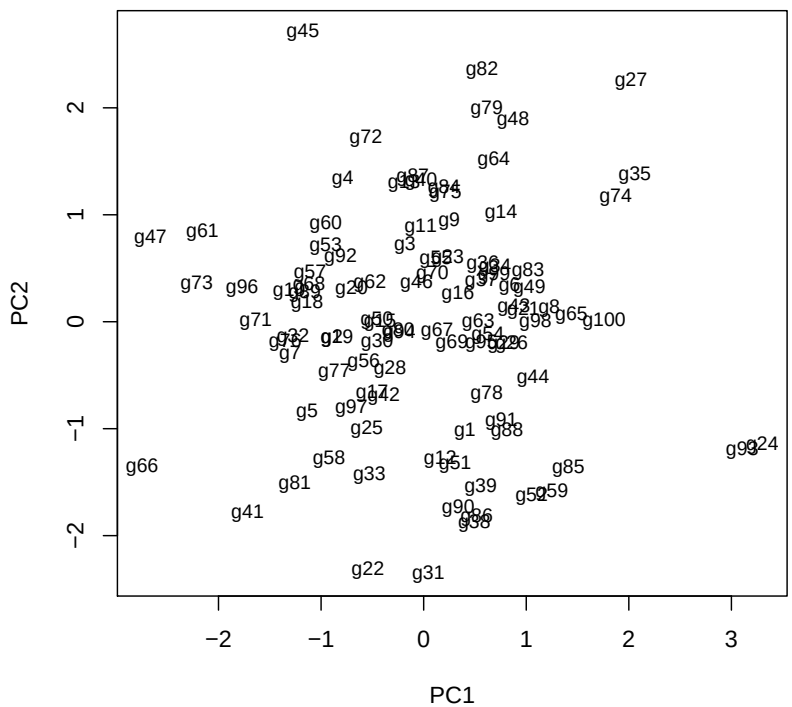
Example

Principal Component	**1st**	**2nd**	**3rd**	**Other**
Proportion of Variance	62%	34%	3%	rest

1st and 2nd principal components explain 96% of variance.

Example: PCA

```
> pca <- prcomp(y, scale=T)
> summary(pca) # Prints variance summary for all principal components
Importance of components:
                          PC1    PC2    PC3    PC4    PC5
Standard deviation     1.0996 1.0505 1.0247 0.9219 0.8873
Proportion of Variance 0.2418 0.2207 0.2100 0.1700 0.1575
Cumulative Proportion  0.2418 0.4626 0.6725 0.8425 1.0000
> plot(pca$x, pch=20, col="blue", type="n") # To plot dots, drop type="n"
> text(pca$x, rownames(pca$x), cex=0.8)
```



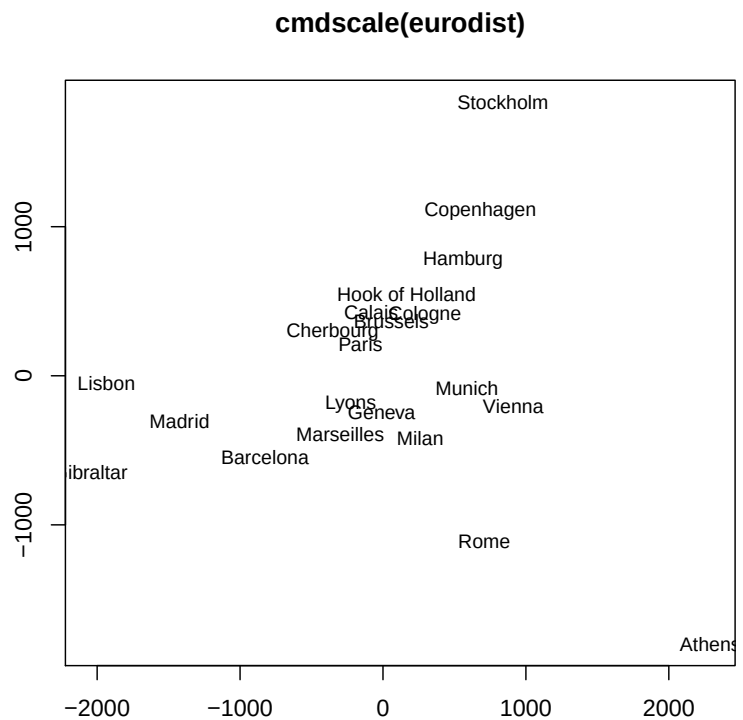
Multidimensional Scaling (MDS)

- Alternative dimensionality reduction approach
- Represents distances in 2D or 3D space
- Starts from distance matrix (PCA uses data points)

Example: MDS with `cmdscale`

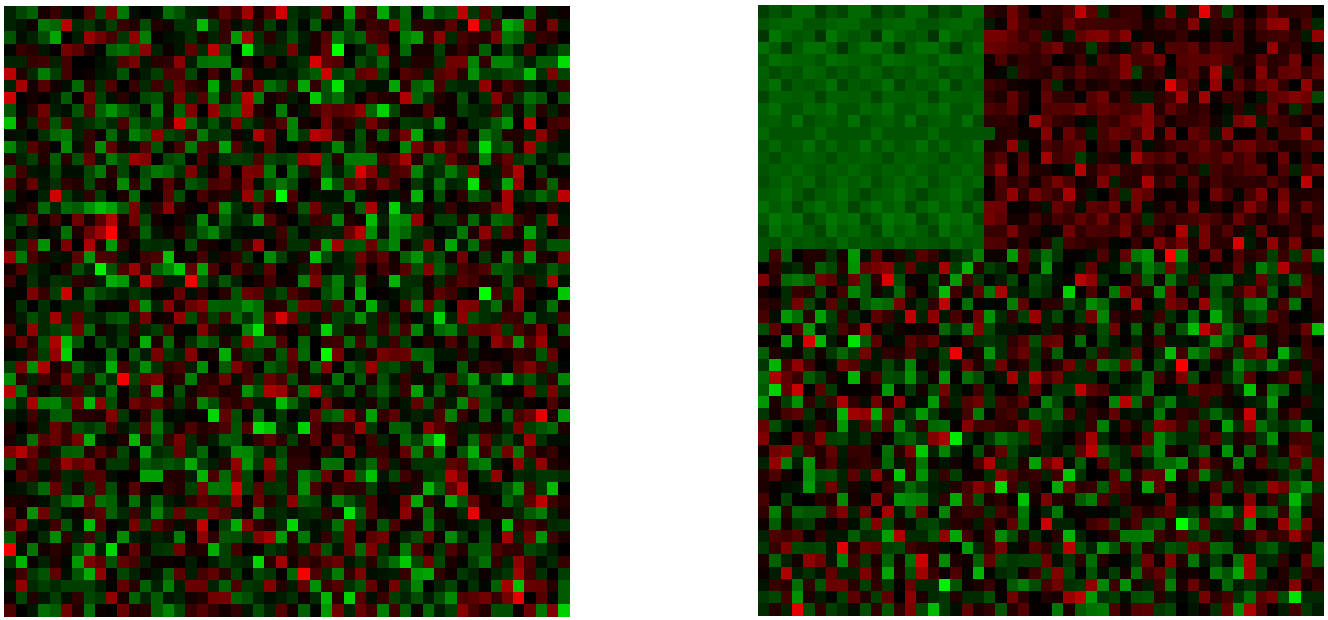
The following example performs MDS analysis on the geographic distances among European cities.

```
> loc <- cmdscale(eurodist)
> plot(loc[,1], -loc[,2], type="n", xlab="", ylab="", main="cmdscale(eurodist)")
> text(loc[,1], -loc[,2], rownames(loc), cex=0.8)
```



Biclustering

Finds in matrix subgroups of rows and columns which are as similar as possible to each other and as different as possible to the remaining data points.



Unclustered $\Rightarrow$ Clustered

Bibliography I

Yin, T., Cook, D., Lawrence, M., Aug 2012. ggbio: an R package for extending the grammar of graphics for genomic data. Genome Biol 13 (8).
URL http://www.hubmed.org/display.cgi?uids=22937822

Zhang, H., Meltzer, P., Davis, S., 2013. RCircos: an R package for Circos 2D track plots. BMC Bioinformatics 14, 244–244.
URL http://www.hubmed.org/display.cgi?uids=23937229