DARL Example Notebook: Data Analytics Fun with the Iris Dataset in R

Data Analytics Research Lab (DARL) (Fall 2017)

John S. Erickson (editor)

Overview

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code. Execute chunks by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

The content of this tutorial is based in part on *Computing and Visualizing PCA in R* by Thiago G. Martins. We also use example code from this StackExchange article.

Introduction

- This notebook is a multi-faceted tutorial focusing on methods for analyzing the classic `iris` dataset. The data contain four continuous variables which correspond to physical measures of flowers, and a categorical variable describing the flowers' species.
- In later sections of this notebook we've modified the standard data to demonstrate the need for scaling before applying principle component analysis (PCA)
- The second half of this tutorial demonstrates how to apply and visualize PCA in R. There are many packages and functions that can apply PCA in R; here, we use the function `prcomp` from the `stats` package.
  - "For completeness"" we begin by showing how to visualize PCA in R using "Base R" graphics.
  - However, our preferred visualization function for PCA is `ggbiplot`, which is implemented by Vince Q. Vu and available through github. Some examples from StackExchange are also included.

Exploring the `iris` Dataset

As noted, this tutorial focuses on the classic `iris` dataset. The data contain four continuous variables which corresponds to physical measures of flowers and a categorical variable describing the flowers' species.

```
# Load data
data(iris)
myIris <- iris
head(myIris, 3)
```

```
##   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
```

Sometimes we need to scale our data...

The following is based on an example from StackExchange:

- The `iris` data set is great for illustrating data analysis techniques including PCA. That said, the first four columns describing length and width of sepals and petals as given are not an example of strongly skewed data. Therefore, *log-transforming the standard data does not change the results much, since the resulting rotation of the principal components is erlatively unchanged by log-transformation.* In other situations log-transformation can be a good choice.
- We perform PCA to gain insight into the general structure of a data set. We need to center, scale and sometimes log-transform our data to filter off some trivial effects which could dominate our PCA. The algorithm of a PCA will in turn find the rotation of each PC to minimize the squared residuals, namely the sum of squared perpendicular distances from any sample to the PCs. Large values tend to have high leverage.

Strategy for adding outliers to the `iris` data

- *Our plan is to inject two new samples into the iris data:*
  - One flower (*setosa gigantica*) with 430 cm petal length
  - A second flower (*virginica brevis*) with petal length of 0.0043 cm.
- Both of these flowers are very abnormal, being 100 times larger and 1000 times smaller than "average"" examples.
- The leverage of the first flower is huge, such that the first PCs will mostly describe the differences between the large flower and any other flower. This means that *clustering of species will not be possible due to that one outlier.*
- If we log-transform our data, the absolute value should now describe the relative variation. The small flower becomes the most abnormal one, but it will still be possible to contain all samples in one figure and to provide a "fair"" clustering of the species.

Code for adding samples to the `iris` data

We'll use the following code to modify the `iris` data:

```
#add two new observations from two new species to iris data
levels(myIris[,5]) = c(levels(myIris[,5]),"setosa_gigantica","virginica_brevis")
myIris[151,] = list(6, 3, 430, 1.5, "setosa_gigantica") # a big flower
myIris[152,] = list(6, 3, .0043, 1.5, "virginica_brevis") # a small flower

summary(myIris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length       Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :  0.0043   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:  1.5750   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :  4.3500   Median :1.300
##  Mean   :5.845   Mean   :3.057   Mean   :  6.5375   Mean   :1.203
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:  5.1000   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :430.0000   Max.   :2.500
##              Species
##  setosa          :50
##  versicolor      :50
##  virginica       :50
```

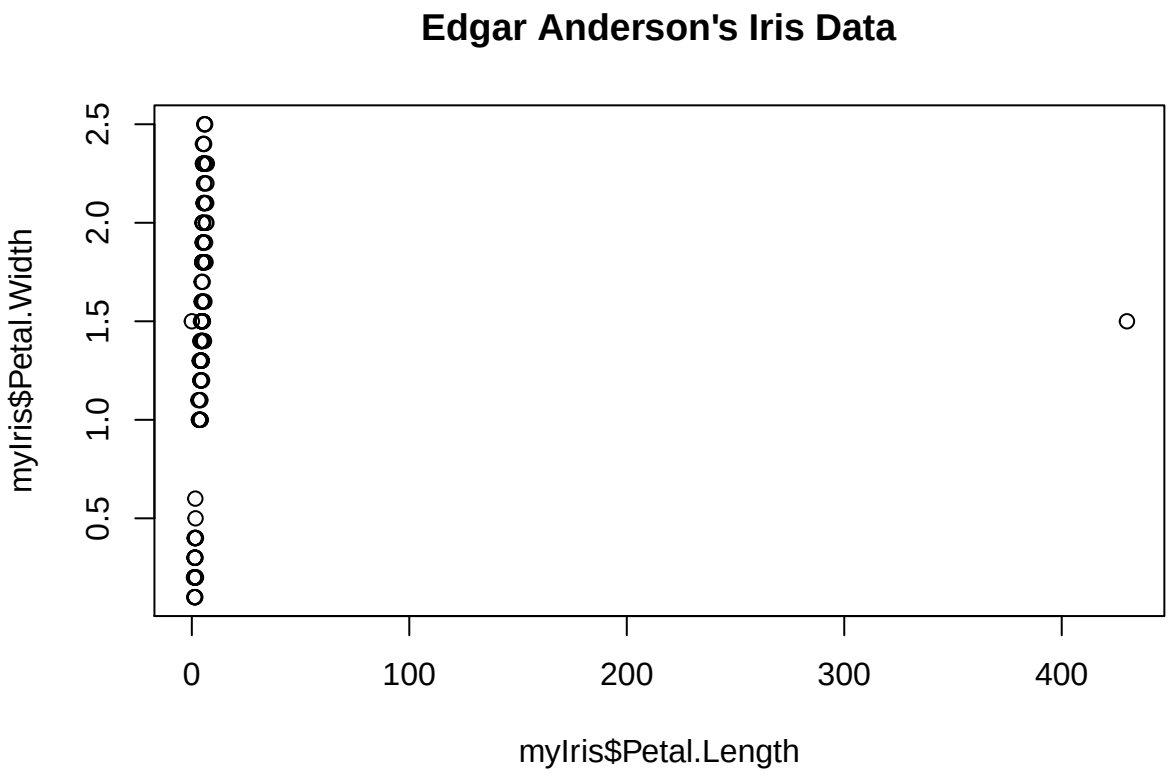
```
##  setosa_gigantica: 1
##  virginica_brevis: 1
##
```

Notice the new Min and Max values for `Petal.Length`.

Simple data exploration with the modifed data

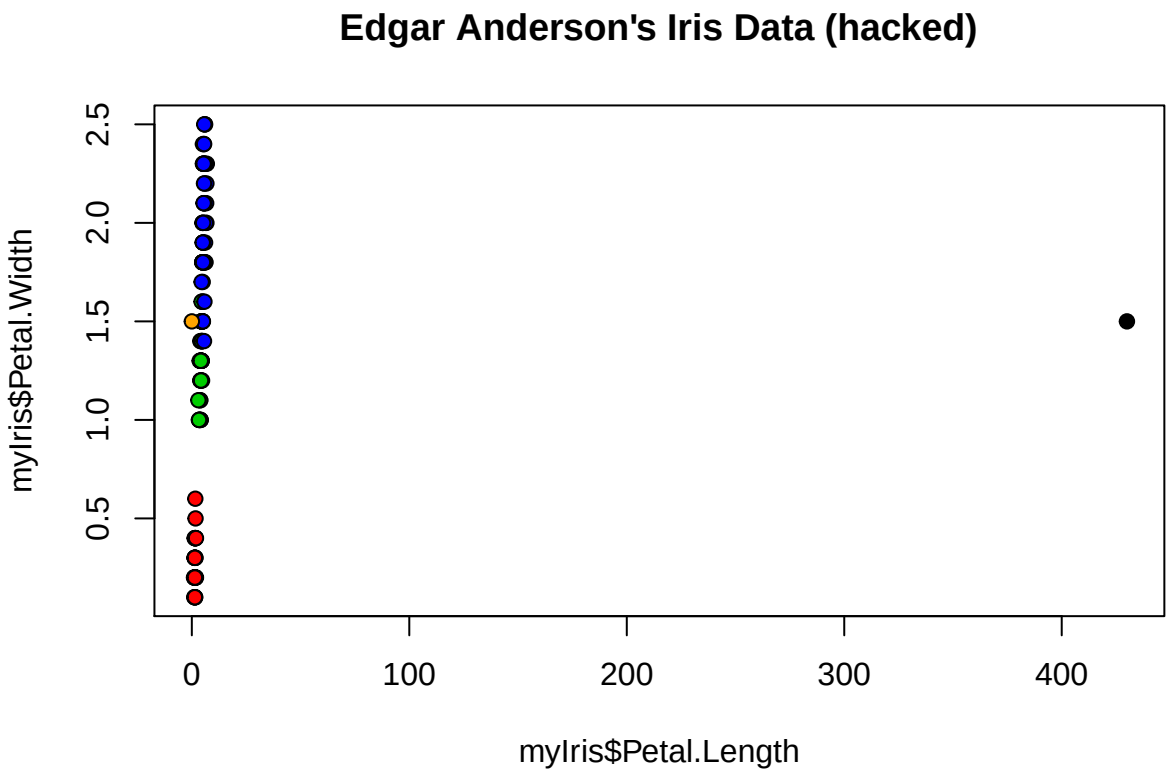
Let's first generate a simple *scatter plot*, plotting `Petal.Length` vs. `Petal.Width`:

```
plot(myIris$Petal.Length, myIris$Petal.Width, main="Edgar Anderson's Iris Data")
```



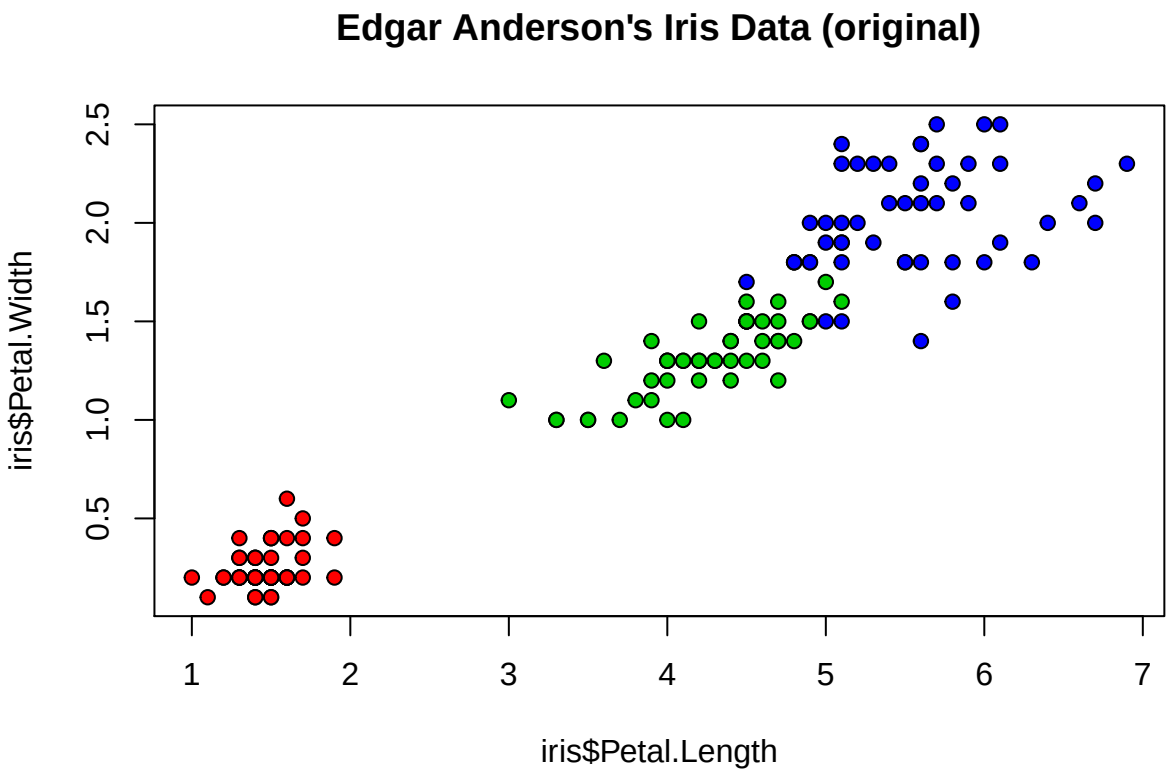
That's pretty ugly! Let's generated the same plot, but coloring by species:

```
plot(myIris$Petal.Length, myIris$Petal.Width, pch=21, bg=c("red","green3","blue","black","orange")[uncla
```



To see the effects of our exceptional flowers, let's look at the unmodified iris data:

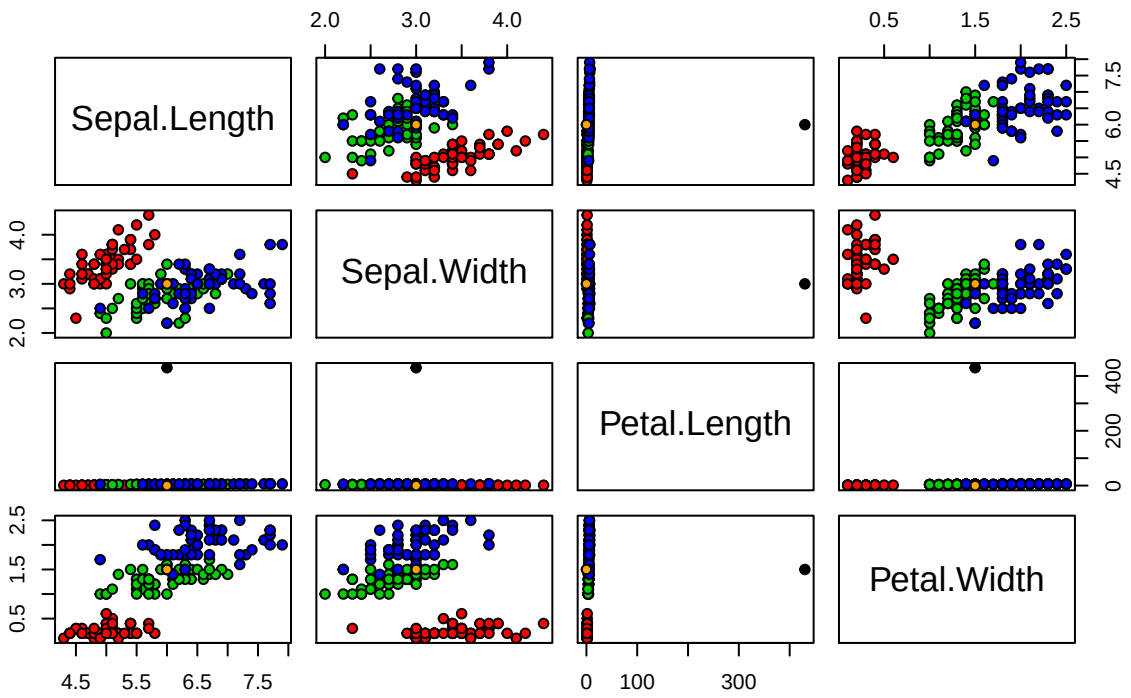
```
plot(iris$Petal.Length, iris$Petal.Width, pch=21, bg=c("red","green3","blue")[unclass(iris$Species)], ma
```



Pairs Scatter Plots

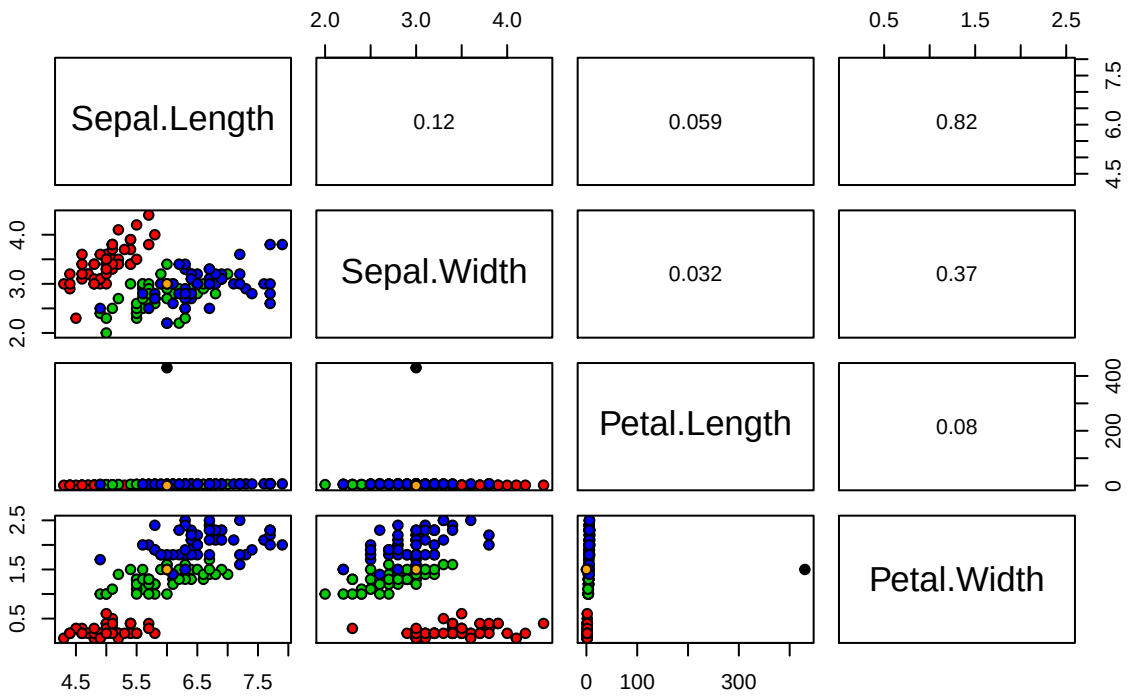
How do the variables behave in relation to each other? We could generate each plot individually, but there is quicker way, using the `pairs` command on the first four columns:

```
pairs(myIris[1:4], main = "Edgar Anderson's Iris Data (hacked)", pch = 21, bg = c("red", "green3", "blu
```



Notice that the panels are mirrored around the diagonal axis. We can create a custom R function `panel.pearson` to draw something else for the upper panels, such as the *Pearson's Correlation*:

```
panel.pearson <- function(x, y, ...) {
  horizontal <- (par("usr")[1] + par("usr")[2]) / 2;
  vertical <- (par("usr")[3] + par("usr")[4]) / 2;
  text(horizontal, vertical, format(abs(cor(x,y)), digits=2))
}

pairs(myIris[1:4], main="Edgar Anderson's Iris Data (hacked)", pch = 21, bg = c("red", "green3", "blue"
```



Let's calculate a simple *linear regression* based on this skewed data:

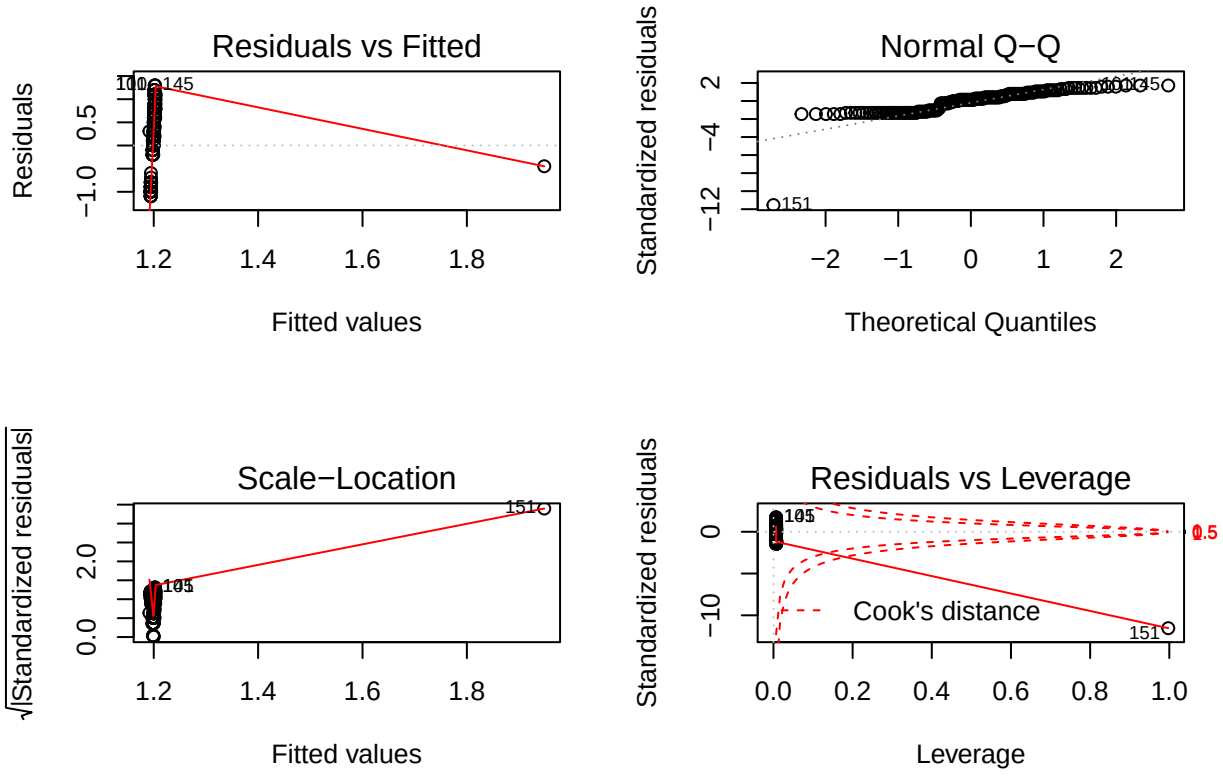
```
fit <- lm(Petal.Width ~ Petal.Length, data=myIris)

# show results
summary(fit)
```

```
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length, data = myIris)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0944 -0.8942  0.1012  0.5996  1.2982
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.191788   0.062577  19.045   <2e-16 ***
## Petal.Length 0.001759   0.001782   0.987    0.325
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.758 on 150 degrees of freedom
## Multiple R-squared:  0.006457,   Adjusted R-squared:  -0.0001664
## F-statistic: 0.9749 on 1 and 150 DF,  p-value: 0.3251
```

Hmmm, that looks pretty ugly... let's generate some "diagnostic plots" to better understand our model:

```
par(mfrow=c(2,2))
plot(fit)
```

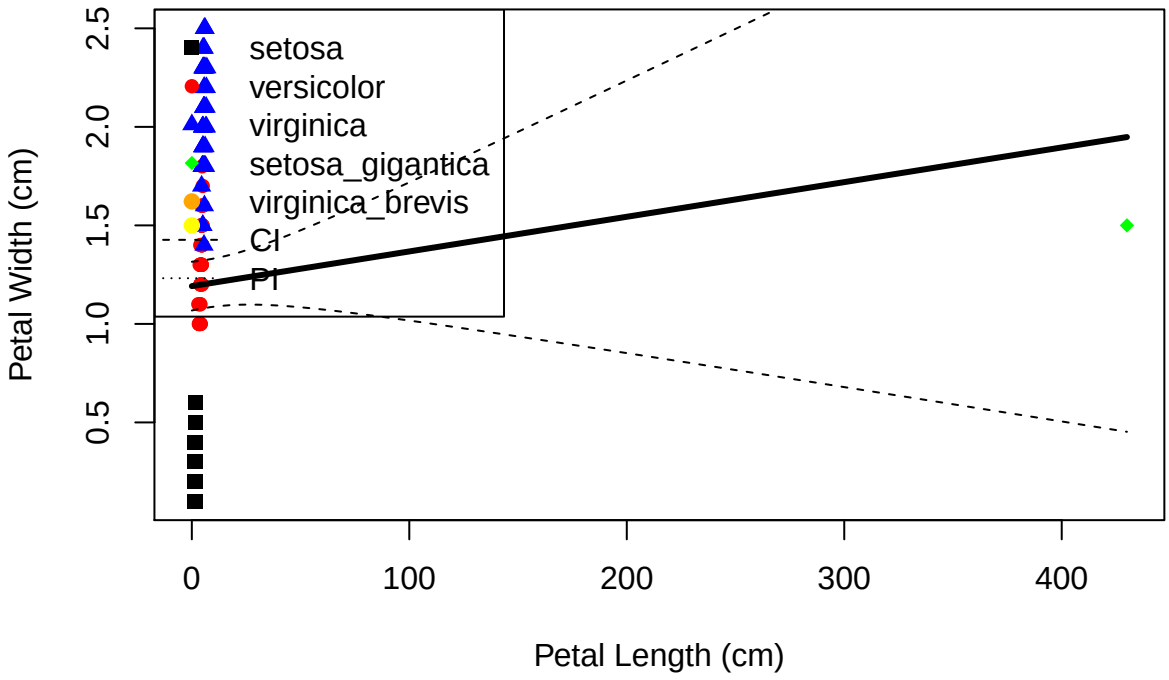
```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```



We can see the adverse effects of our outliers.

Now let's plot a linear regression, including confidence and prediction intervals. Notice how we layer on a key.

```
plot(Petal.Width ~ Petal.Length, col=c("black", "red", "blue", "green", "yellow")[Species], pch=(15:19)
newx <- data.frame(Petal.Length=seq(min(myIris$Petal.Length), max(myIris$Petal.Length), length.out=100)
conf.interval <- predict(fit, newdata=newx, interval="confidence")
pred.interval <- predict(fit, newdata=newx, interval="prediction")
lines(conf.interval[, "fit"] ~ newx[, 1], lty=1, lw=3)
lines(conf.interval[, "lwr"] ~ newx[, 1], lty=2)
lines(conf.interval[, "upr"] ~ newx[, 1], lty=2)
lines(pred.interval[, "lwr"] ~ newx[, 1], lty=3)
lines(pred.interval[, "upr"] ~ newx[, 1], lty=3)
legend("topleft", legend=c(levels(myIris$Species), "CI", "PI"), col=c("black", "red", "blue", "green",
```



This is not very promising; we can see that it is difficult to create a linear model without dropping our "exceptional" flowers. But we'll press on and perform multiple regression anyways...

```
fit2 <- lm(Petal.Width ~ Petal.Length + Sepal.Length + Sepal.Width, data=myIris)

# show our results as a table
summary(fit2)
```

```
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length + Sepal.Length + Sepal.Width,
##     data = myIris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72316 -0.29716 -0.05712  0.18774  1.10761
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5617994  0.3379335  -4.622 8.22e-06 ***
## Petal.Length  0.0005477  0.0009133   0.600     0.55
## Sepal.Length  0.7224446  0.0386814  18.677  < 2e-16 ***
## Sepal.Width  -0.4781376  0.0734052  -6.514 1.08e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3877 on 148 degrees of freedom
```

```
## Multiple R-squared:  0.7435, Adjusted R-squared:  0.7383
## F-statistic:   143 on 3 and 148 DF,  p-value: < 2.2e-16
```

Now we'll check the relationship between our models using `ANOVA`:

```
anova(fit, fit2)
```

```
## Analysis of Variance Table
##
## Model 1: Petal.Width ~ Petal.Length
## Model 2: Petal.Width ~ Petal.Length + Sepal.Length + Sepal.Width
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    150 86.188
## 2    148 22.251  2    63.937 212.63 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can now calculate interaction terms:

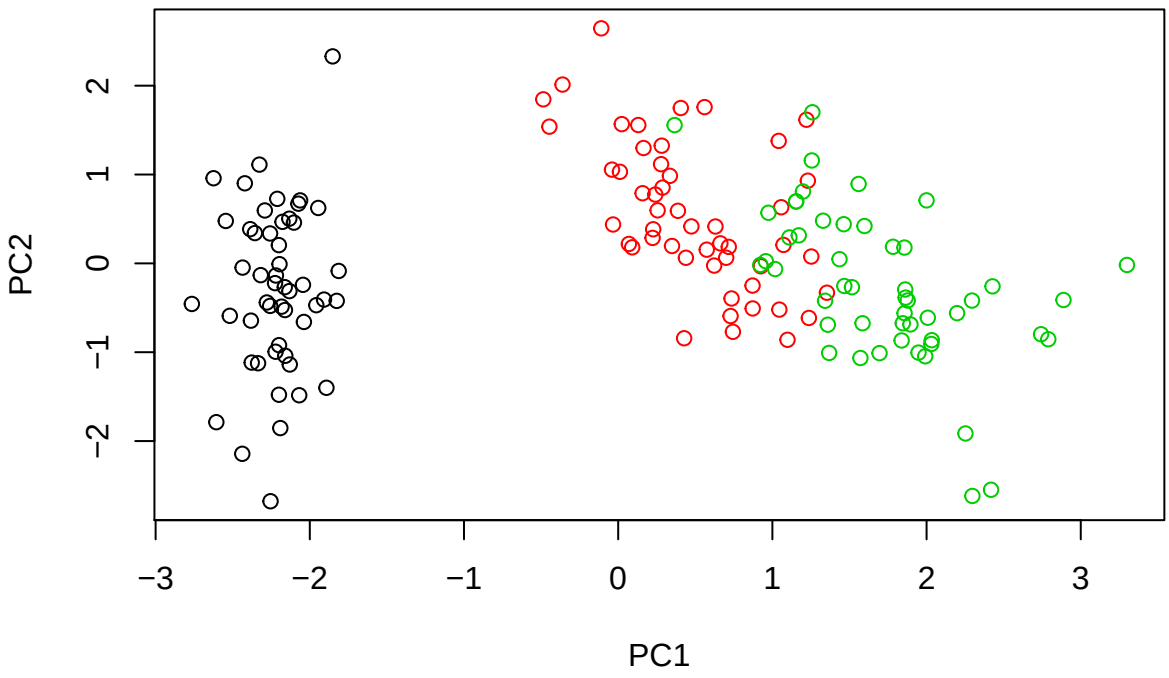
```
fit2int <- lm(Petal.Width ~ Petal.Length + Sepal.Length + Sepal.Width + Petal.Length:Sepal.Length, data=
anova(fit2, fit2int)
```

```
## Analysis of Variance Table
##
## Model 1: Petal.Width ~ Petal.Length + Sepal.Length + Sepal.Width
## Model 2: Petal.Width ~ Petal.Length + Sepal.Length + Sepal.Width + Petal.Length:Sepal.Length
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    148 22.251
## 2    147 19.377  1    2.8738 21.801 6.759e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Computing the Principal Components

Our overall objective is to apply PCA to the four continuous variables and use the categorical variable `Species` to visualize the principal components later. Prior to log scaling, let's examine our PCA on the un-scaled data:

```
#Plotting scores of PC1 and PC" without log transformation
plot(prcomp(iris[,-5],cen=T,sca=T)$x[,1:2],col=iris$Spec)
```



In the following code we apply a log transformation to the continuous variables as suggested by [1] and set center and scale equal to `TRUE` in the call to `prcomp` to standardize the variables prior to the application of PCA:

```
# log transform
log.ir <- log(iris[, 1:4])
ir.species <- iris[, 5]

# apply PCA - scale. = TRUE is highly
# advisable, but default is FALSE.
ir.pca <- prcomp(log.ir,
                 center = TRUE,
                 scale. = TRUE)
```

Since "skewness" and the magnitude of the variables influence the resulting princpal components, it is good practice to apply a *skewness transformation* to center and scale the variables prior to the application of PCA. In our example above, we applied a log transformation to the variables but we could have been more general and applied a **Box-Cox transformation** [2]. At the end of this tutorial we show how to perform all of these transformations and then apply PCA with a single one call to the `preProcess` function of the `caret` package.

Analyzing the Results

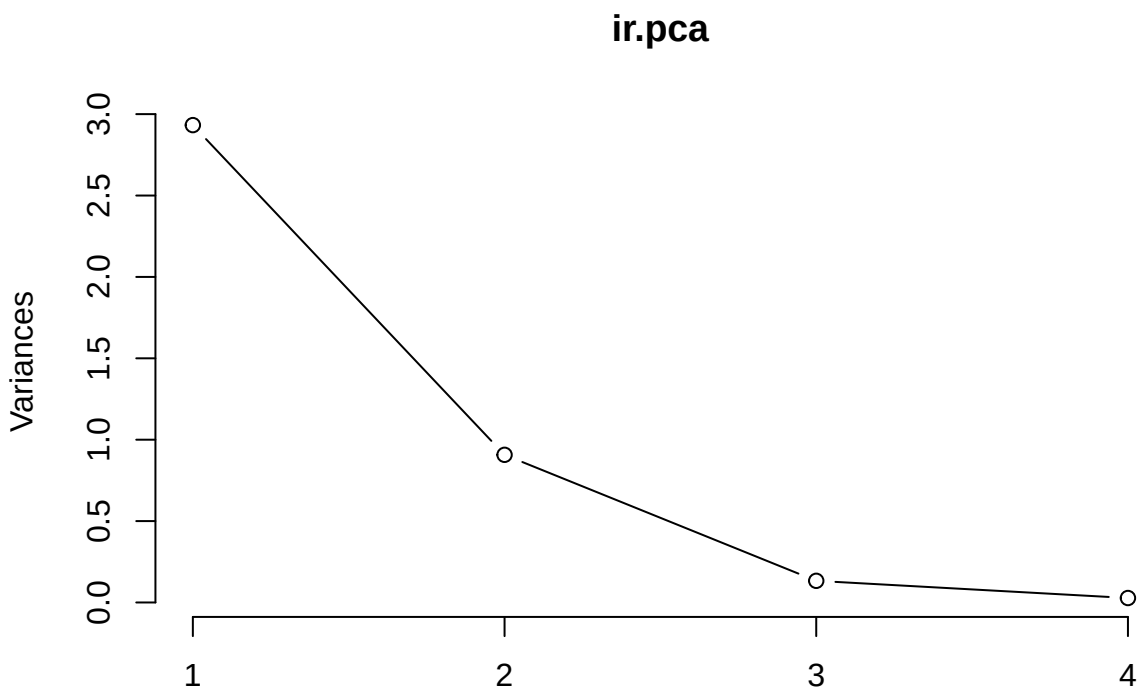
The `prcomp` function returns an object of class `prcomp`, which has some methods available to help interpret our results. The `print` method returns the standard deviation of each of the four principal components and their rotation (or "loadings"), which are the coefficients of the linear combinations of the continuous variables.

```
# print method
print(ir.pca)
```

```
## Standard deviations (1, .., p=4):
## [1] 1.7124583 0.9523797 0.3647029 0.1656840
##
## Rotation (n x k) = (4 x 4):
##                     PC1         PC2        PC3         PC4
## Sepal.Length  0.5038236 -0.45499872  0.7088547  0.19147575
## Sepal.Width  -0.3023682 -0.88914419 -0.3311628 -0.09125405
## Petal.Length  0.5767881 -0.03378802 -0.2192793 -0.78618732
## Petal.Width   0.5674952 -0.03545628 -0.5829003  0.58044745
```

The `plot` method for the `prcomp` class returns a *scree plot* showing the variances (y-axis) associated with the PCs (x-axis). The figure below helps us decide how many PCs to retain for further analysis. In this simple case with only four PCs this is not hard; we can clearly see that the first two PCs explain most of the variability in the data.

```
# plot method
plot(ir.pca, type = "l")
```



The `summary` method describes the importance of the PCs. * The first row describes the *standard deviation* associated with each PC. * The second row shows the *proportion of the variance* in the data explained by each component. * The third row described the *cumulative proportion* of explained variance. We can see there that the first two PCs account for more than {95%} of the variance of the data.

```
# summary method
summary(ir.pca)
```

```
## Importance of components%s:
##                             PC1    PC2     PC3     PC4
## Standard deviation     1.7125 0.9524 0.36470 0.16568
## Proportion of Variance 0.7331 0.2268 0.03325 0.00686
## Cumulative Proportion  0.7331 0.9599 0.99314 1.00000
```

We can node use the `predict` function to predict PCs from new data. For example, let's pretend the last two rows of the `iris` data has just arrived and we want to evaluate their principal component values:

```
# Predict PCs
predict(ir.pca,  newdata=tail(log.ir, 2))
```

```
##           PC1         PC2        PC3         PC4
## 149 1.0809930 -1.01155751 -0.7082289 -0.06811063
## 150 0.9712116 -0.06158655 -0.5008674 -0.12411524
```

The code block below generates a biplot of our PCA using the `ggbiplot` function from the `ggbiplot` package, available through github .

```
#require(devtools)
#install_git("git://github.com/vqv/ggbiplot")

require(ggbiplot)
```

```
## Loading required package: ggbiplot
```

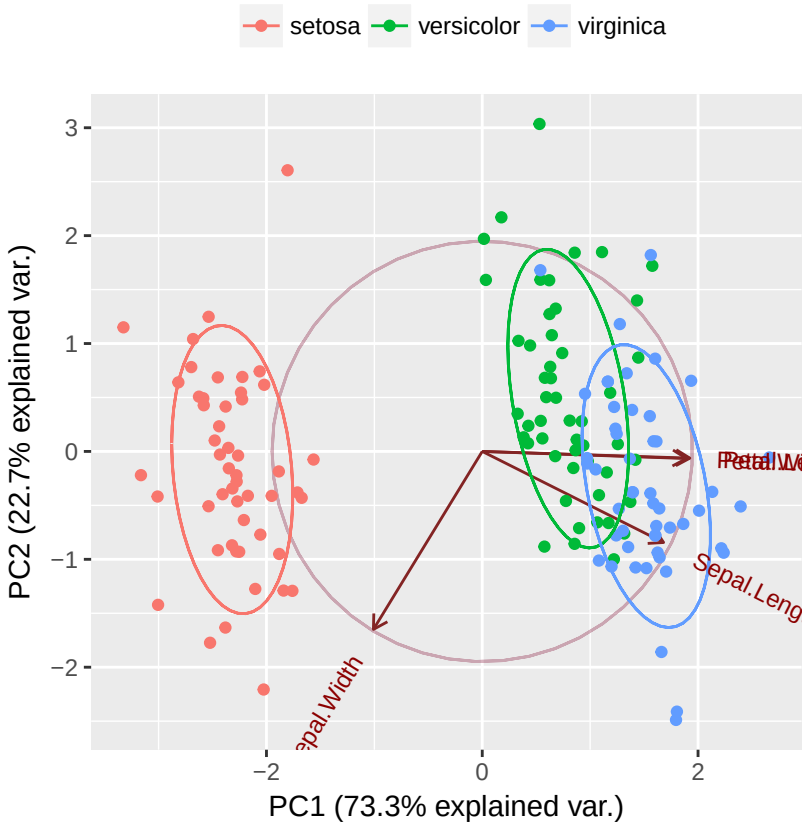
```
## Loading required package: ggplot2
```

```
## Loading required package: plyr
```

```
## Loading required package: scales
```

```
## Loading required package: grid
```

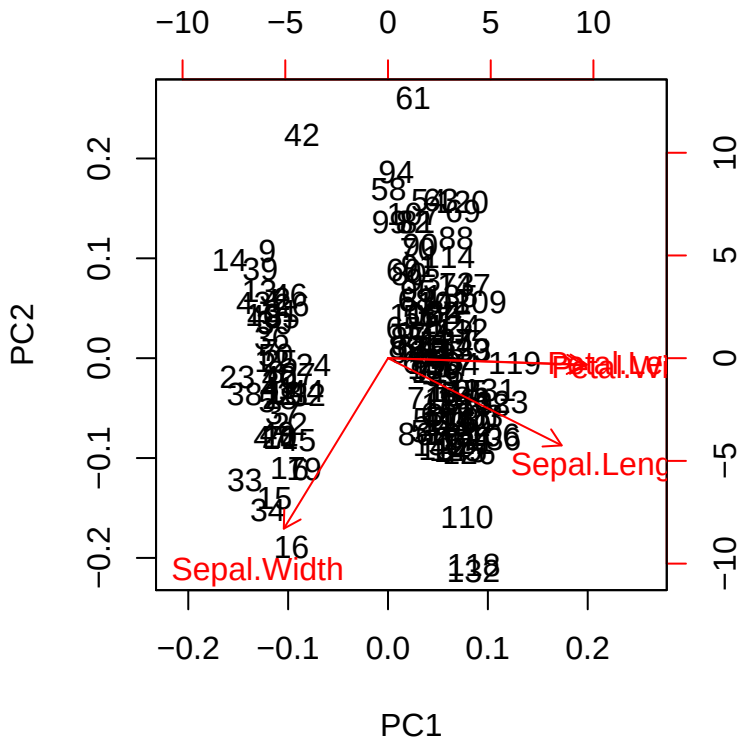
```
g <- ggbiplot(ir.pca, obs.scale = 1, var.scale = 1,
              groups = ir.species, ellipse = TRUE,
              circle = TRUE)
g <- g + scale_color_discrete(name = '')
g <- g + theme(legend.direction = 'horizontal',
               legend.position = 'top')
print(g)
```



This plot projects the data on the first two PCs. Other PCs may be chosen through the arguments to the function. The biplot colors each point according to the flowers' species and draws a Normal contour line with `ellipse.prob` probability (default to {68%}) for each group.

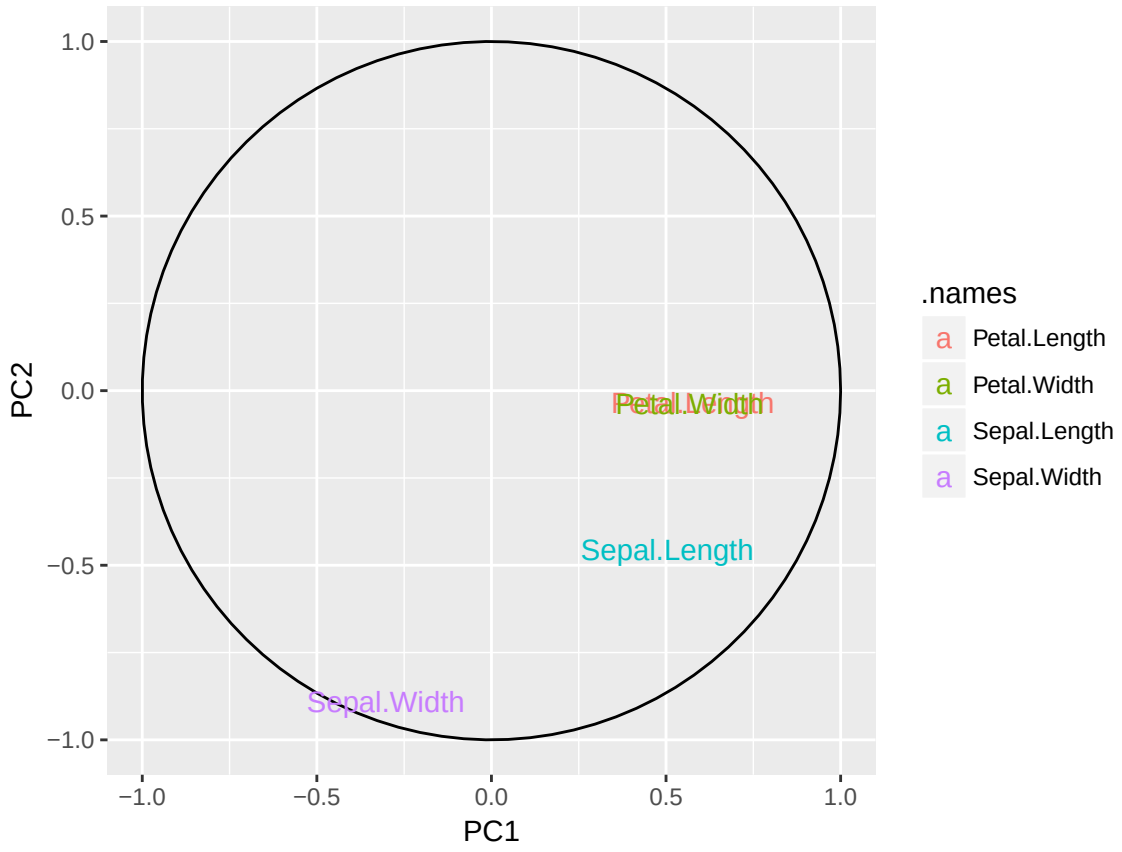
Much more info about `ggbiplot` may be obtained by the usual `?ggbiplot` within RStudio. We think you' wi'll agree that the plot produced by ggbiplot is much better than the one produced by the built-in function:

```
# Plotting PCA using the built-in plot function
biplot(ir.pca)
```



Sometimes it is helpful to plot each variable's coefficients inside a unit circle to help interpret a PCA. Such a figure can be generated by this code, available through this github gist:

```
require(ggplot2)

theta <- seq(0,2*pi,length.out = 100)
circle <- data.frame(x = cos(theta), y = sin(theta))
p <- ggplot(circle,aes(x,y)) + geom_path()

loadings <- data.frame(ir.pca$rotation,
                       .names = row.names(ir.pca$rotation))
p + geom_text(data=loadings,
              mapping=aes(x = PC1, y = PC2, label = .names, colour = .names)) +
  coord_fixed(ratio=1) +
  labs(x = "PC1", y = "PC2")
```



PCA using the caret Package

As noted above, we can first apply a *Box-Cox transformation* to correct for skewness to center and scale each variable and then apply PCA with a single call to the `preProcess` function from the `caret` package.

```
require(caret)
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
trans = preProcess(iris[,1:4], method=c("BoxCox", "center", "scale", "pca"))
PC = predict(trans, iris[,1:4])
```

By default, `preProcess` only keeps the PCs that are necessary to explain at least *95%* of the variability in the data, but this can be changed through the argument `thresh`.

```
# Retained PCs
head(PC, 3)
```

```
##         PC1        PC2
## 1 -2.303540 -0.4748260
## 2 -2.151310  0.6482903
## 3 -2.461341  0.3463921
```

See Unsupervised data pre-processing for predictive modeling for an introduction of the `preProcess` function.

References

[1] Venables, W. N., Brian D. R. Modern applied statistics with S-PLUS. Springer-verlag. (Section 11.1)

[2] Box, G. and Cox, D. (1964). An analysis of transformations. Journal of the Royal Statistical Society. Series B (Methodological) 211-252