1. Introduction
This article gives an exposition of how to use the R statistical software for multivariate analyses, with a focus on principal components analysis (PCA) and cluster analysis. We will use the Iris data set of Fisher for all purposes unless otherwise specified.

2. Reading Data into R
There are a number of ways of reading data into R. For our purpose, we will use read.table() function to read the data into R.

> iris=read.table("d:/iris.txt", header=T)

Here, the data is loaded in R as iris. After reading the data in R, attach the data so that variables can be directly accessed.

> attach(data)

3. Plotting Multivariate Data
Once you have read a multivariate data set into R, the next step is usually to make a plot of the data.

A matrix scatter plot
One common way of plotting multivariate data is to make a “matrix scatterplot”, showing each pair of variables plotted against each other. We can use the “scatterplotMatrix()” function from the “car” R package to do this. To use this function, first install the “car” R package. Once installed the “car” R package, load the “car” R package by using:

> library("car")

Now, you can then use the “scatterplotMatrix()” function to plot the multivariate data. Alternative, one can use pairs() function.

> pairs(iris[,1:4])

The output is like
4. Calculating Summary Statistics for Multivariate Data

One may be interested to calculate summary statistics such as the mean and standard deviation for each of the variables in the multivariate data set.

This can be done using the "mean()", "sd()", "cv()", "var()" functions in R. For example, to calculate the mean and standard deviations of each of four variables in the iris data set

```
> mean(Sepal_length)
5.843333

> var(Sepal_length)
0.6856935
```

Similarly use for other variables.

This tells that the mean of variable Sepal_length is 5.843333 and the variance is 0.6856935.
Summary() gives some commonly used descriptive statistics of the variables in the data set. For example,
> summary(iris)
gives the following output.

<table>
<thead>
<tr>
<th></th>
<th>Sepal_length</th>
<th>Sepal_width</th>
<th>Petal_length</th>
<th>Petal_width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>4.300</td>
<td>2.000</td>
<td>1.000</td>
<td>0.100</td>
<td>I. setosa :50</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>5.100</td>
<td>2.800</td>
<td>1.600</td>
<td>0.300</td>
<td>I. versicolor:50</td>
</tr>
<tr>
<td>Median</td>
<td>5.800</td>
<td>3.000</td>
<td>4.350</td>
<td>1.300</td>
<td>I. virginica :50</td>
</tr>
<tr>
<td>Mean</td>
<td>5.843</td>
<td>3.057</td>
<td>3.758</td>
<td>1.199</td>
<td></td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>6.400</td>
<td>3.300</td>
<td>5.100</td>
<td>1.800</td>
<td></td>
</tr>
<tr>
<td>Max.</td>
<td>7.900</td>
<td>4.400</td>
<td>6.900</td>
<td>2.500</td>
<td></td>
</tr>
</tbody>
</table>

5. Principal Component Analysis
The purpose of principal component analysis is to find the best low-dimensional representation of the variation in a multivariate data set. To carry out a principal component analysis (PCA) on a multivariate data set, first standardize the variables under study using the “scale()” function. This is necessary if the input variables have very different variances.

Once you have standardized the variables, carry out a principal component analysis using the “prcomp()” function in R.

For example, to carry out a principal components analysis on the standardized variables, use:

>pc1=prcomp(iris[,1:4],scale=T)

Here, scale option in the function prcomp scales all the variable in the data set such that variables have zero mean and variance 1.

Standard deviations:

[1] 1.7083611 0.9560494 0.3830886 0.1439265

Rotation:

<table>
<thead>
<tr>
<th></th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sepal_length</td>
<td>0.5210659</td>
<td>-0.37741762</td>
<td>0.7195664</td>
<td>0.2612863</td>
</tr>
<tr>
<td>Sepal_width</td>
<td>-0.2693474</td>
<td>-0.92329566</td>
<td>-0.2443818</td>
<td>-0.1235096</td>
</tr>
<tr>
<td>Petal_length</td>
<td>0.5804131</td>
<td>-0.02449161</td>
<td>-0.1421264</td>
<td>-0.8014492</td>
</tr>
<tr>
<td>Petal_width</td>
<td>0.5648565</td>
<td>-0.06694199</td>
<td>-0.6342727</td>
<td>0.5235971</td>
</tr>
</tbody>
</table>
A summary of the principal component analysis results can be obtained using the “summary()” function on the output of “prcomp()”.

```r
> summary(pc1)

Importance of components:

PC1      PC2      PC3      PC4
Standard deviation    1.7084  0.9560  0.38309  0.14393
Proportion of Variance 0.7296  0.2285  0.03669  0.00518
Cumulative Proportion 0.7296  0.9581  0.99482  1.00000
```

This gives us the standard deviation of each component, and the proportion of variance explained by each component. The standard deviation of the components is stored in a named element called “sdev” of the output variable made by “prcomp”:

```r
> pc1$sdev
  1.7083611 0.9560494 0.3830886 0.1439265
```

In order to decide how many principal components should be retained, it is common to summarize the results of a principal components analysis by making a scree plot, which can be done in R using the “screeplot()” function:

```r
> screeplot(pc1, type="lines")
```

![Scree plot](image-url)
From above, it may be seen that either first 2 or 3 components should be retained.

Another way to decide how many principal components to retain is to decide to keep the number of components required to explain at least some minimum amount of the total variance. For example, if it is important to explain at least 90% of the variance, we would retain the first two principal components, which can be seen from the output of “summary(pc1)” that the first two principal components explain 95.81% of the variance.

**Loadings of the principal components**
The loadings for the principal components are stored in a named element “rotation” of the variable returned by “prcomp()”. This contains a matrix with the loadings of each principal component, where the first column in the matrix contains the loadings for the first principal component, the second column contains the loadings for the second principal component, and so on.

Therefore, to obtain the loadings for the first principal component, use:

```r
> pc1$rotation
```

<table>
<thead>
<tr>
<th></th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sepal_length</td>
<td>0.5210659</td>
<td>-0.37741762</td>
<td>0.7195664</td>
<td>0.2612863</td>
</tr>
<tr>
<td>Sepal_width</td>
<td>-0.2693474</td>
<td>-0.92329566</td>
<td>-0.2443818</td>
<td>-0.1235096</td>
</tr>
<tr>
<td>Petal_length</td>
<td>0.5804131</td>
<td>-0.02449161</td>
<td>-0.1421264</td>
<td>-0.8014492</td>
</tr>
<tr>
<td>Petal_width</td>
<td>0.5648565</td>
<td>-0.06694199</td>
<td>-0.6342727</td>
<td>0.5235971</td>
</tr>
</tbody>
</table>

This means that the first principal component is a linear combination of the variables: 0.52Z1 –0.27Z2 + 0.58Z3+0.56Z4 where Z1, Z2, Z3...Z4 are the standardized versions of the variables in the data set (that each have mean of 0 and variance of 1).

Note that the square of the loadings sum to 1, as this is a constraint used in calculating the loadings:

```r
> sum((pc1$rotation[,1])^2)
[1] 1
```

6. Cluster Analysis

R has a number of functions for cluster analysis. In this section, k-means and hierarchical agglomerative clustering using R are discussed.

**Data Preparation**

Prior to clustering data, you may want to remove or estimate missing data and rescale variables for comparability.

```r
# Prepare Data
>iris = na.omit(iris) # listwise deletion of missing
>iris1=iris[,1:4]
>iris1= scale(iris1) # standardize variables
```
**k-means clustering**

K-means clustering is the most popular partitioning method. It requires the analyst to specify the number of clusters to extract. Use following functions to perform a K-means clustering.

```r
> fit = kmeans(iris1, 5) # 5 cluster solution
```

To get the cluster means, use

```r
> aggregate(iris1, by=list(fit$cluster), FUN=mean)
```

<table>
<thead>
<tr>
<th>Group.1</th>
<th>Sepal_length</th>
<th>Sepal_width</th>
<th>Petal_length</th>
<th>Petal_width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.1924784</td>
<td>0.4015443</td>
<td>-1.3090939</td>
<td>-1.26089024</td>
</tr>
<tr>
<td>2</td>
<td>-0.3516137</td>
<td>-1.3285553</td>
<td>0.1026061</td>
<td>0.01228268</td>
</tr>
<tr>
<td>3</td>
<td>-0.6259564</td>
<td>1.8042613</td>
<td>-1.2826445</td>
<td>-1.22905673</td>
</tr>
<tr>
<td>4</td>
<td>1.3926646</td>
<td>0.2323817</td>
<td>1.1567451</td>
<td>1.21327591</td>
</tr>
<tr>
<td>5</td>
<td>0.3804044</td>
<td>-0.3896455</td>
<td>0.6067908</td>
<td>0.56390985</td>
</tr>
</tbody>
</table>

# append cluster assignment
```r
> iris1 <- data.frame(iris1, fit$cluster)
```

**Hierarchical Agglomerative**

There are a wide range of hierarchical clustering approaches. To perform hierarchical clustering the function `hclust()` function can be used with a number of options. For example, Ward's method of clustering using R can be implemented as

```r
# Ward Hierarchical Clustering
> d = dist(iris1, method = "euclidean") # distance matrix
> fit = hclust(d, method="ward")
> plot(fit) # display dendogram
```

![Cluster Dendrogram](image)
> groups = cutree(fit, k=5)  # cut tree into 5 clusters
# draw dendogram with red borders around the 5 clusters
> rect.hclust(fit, k=5, border="blue")

From the above dendogram, five clusters separated by blue boundary lines can be identified.