Lecture 16: Analysing and Visualizing Data I

In [1]:
```
%load_ext rmagic
```

Today's lecture topics:

- **Bivariate Analysis:** visualizing **population clusters** and **variable relations**
- **Multivariate Analysis:** Visualization
- **Multivariate analysis:** Clustering

1. Bivariate Analysis: visualizing population clusters and variable relations

Consider a **population** for which one has only **two variables**.

As example, we will simulate grade data for a student population, for which one has the midterm grade and the final grade.

The maximal score should be 100.

To simulate continous variable, one can use the function

```
rnorm(n, mean, sd)
```

returning a numeric vector of real numbers

- which is **normally distributed** around `mean`
- has **standard deviation** `std`

In [2]:
```
%%R

set.seed(1)
```

Since the numbers generated by `rnorm` may fall outside our allowed range (0 to 100), we need take care of these bad values. We can correct these bad values using the **bracket operator** and **logical indexing**:

```
X[X < some_value] = some_other_value
```

Since we may reuse this code more than one time, let's package it into a function:

In [3]:
```
%%R

simulate.grades = function(n, score.mean, score.sd, score.max){
        grades = rnorm(n, mean=score.mean, sd=score.sd)
        grades[grades < 0] = 0
        grades[grades > score.max] = score.max
        return(grades)
    }
```

We can now simulate our grades, and check their values by plotting the variable **histograms**, using the command

```
hist(X, main, xlim)
```

which takes

- a **continous variable** `X`
- an argument `main` specifying the histogram title
- an argument `xlim` containing the axis upper and lower values

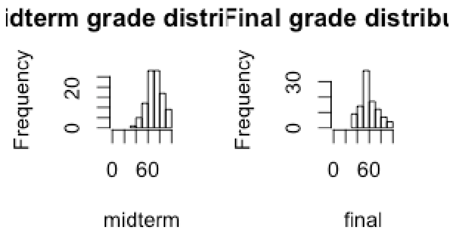
The function

```
par(mfrow=c(n,m))
```

allows us to diplay our plots on a $n \times m$ grid of plots (similar to `subplots` function in `matplotlib`).

In [4]:
```
%%R -r 86 -w 300 -h 200
n = 100

midterm = simulate.grades(n, score.mean=70, score.sd=15, score.max=100)
final   = simulate.grades(n, score.mean=60, score.sd=15, score.max=100)
```

```
par(mfrow=c(1,2))

hist(midterm, main='Midterm grade distribution', xlim=c(0,100))
hist(final, main='Final grade distribution', xlim=c(0,100))
```



Row and column analysis

At this point, we have roughly **two possible types of analysis:**

> **Population (or row) Analysis:**
>
> We can investigate whether our variables split our population in natural group called **clusters**. This is an analysis of the rows.

Example: Are the student grades splitting the class into students of different strenght? We may use these cluster to establish letter grades for instance.

> **Variable (or column) Analysis:**
>
> We can investigate whether our **variables** are **related** to each other, or wether they are **indenpendent.** This is an analysis of the columns.

Example: Is the final score strongly dependent on the midterm score? Can we find a functional relationship between these two scores?

Scatter plots

A **scatter plot** of our variables will give us indications for these two types of analysis:

- possible **clusters in the rows** (population)
- possible **relations between the columns** (variables)

> **Definition:**
>
> A **scatter plot** of **two numeric vectors** $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ with the same number n of elements is a graph with **two axes** representing the two **variable ranges** and displaying n **points** with coordinates $(x_1, y_1), \ldots, (x_n, y_n)$.

Suppose we have

- a **population** Ω we wish to study
- **two continous variables** $X, Y : \Omega \longrightarrow \mathbb{R}$ of interest for our study
- a **population sample** $S = \{s_1, \ldots, s_m\} \subset \Omega$

The **values of our variables on the population sample** gives us two vectors:

$$X = (x_1, \ldots, x_m), \quad \text{where} \quad x_i = X(s_i)$$
$$Y = (y_1, \ldots, y_m), \quad \text{where} \quad y_i = Y(s_i)$$

In a **scatter plot** each **sample individual** $s_i \in S$ is represented as a **point in** $\mathbb{R}^2$ **with coordinates** (x_i, y_i).

Looking at the **geometry** of the **cloud of points** plotted **in the scatter plot** in the xy-plane:

$$(x_1, y_1),\ (x_2, y_2), \ldots,\ (x_m, y_m) \in \mathbb{R}^2$$

will give us **indications on**

- the **variable relations**
- the **population clusters**

Important remark: Of course, we will only obtain **indications** this way. This is particularly true if we don't have values for the whole populations but only for a sample of the population. **It may be the case that the clusters and relations we see for a given sample disappear after add more individual to our sample!!!**

In R, the function that allows us to display **scatter plots** of two numeric vectors is the [plot function (http://www.r-tutor.com/elementary-statistics/quantitative-data/scatter-plot)](http://www.r-tutor.com/elementary-statistics/quantitative-data/scatter-plot):

```
plot(X, Y, xlim, ylim, main, xlab, ylab)
```

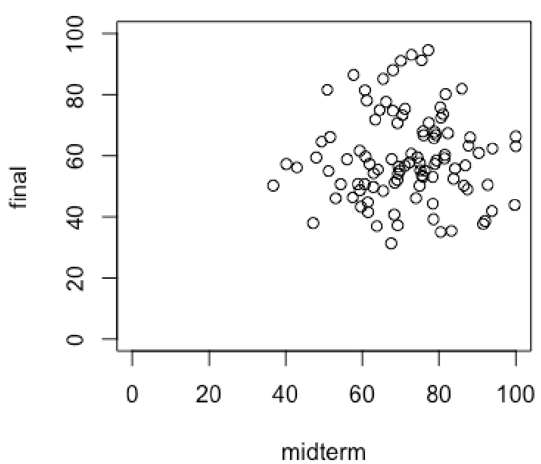
that takes

- two numeric vectors `X` and `Y`
- two numeric vectors `xlim` and `ylim` containing the **axis upper-and-lower bounds**
- **legend strings** `main` for the title, `xlab, ylab` for the x-and-y axis legends

Remark: The appearance or not of clusters in the scatter plot may depends on the scale chosen for the x-and-y axis!

Let us display a scatter plot of our two grade variables:

In [5]:
```
%%R -r 96 -w 400 -h 400

plot(midterm, final, xlim=c(0, 100), ylim=c(0, 100))
```



From this scatter plots, we see a **single cluster of points** centered on

```
(mean(midterm), mean(final))
```

but **no particular relationship between the two exam scores** (i.e. a student may have performed well at the midterm and badly at the final in a completely independent way).

This type of **cloud** is characteristic of **normally distributed independent variables**. So not much to report.

Simulating variable relationships

A better model for grade simulation can be achieved by considering as before that

- **the two exam scores are normally distributed around two different means**

but now we will add the following **assumptions to our model**:

- **students work harder for the final exam**
- **final grades are generally lower than midterm grades**
- **relative student performances should be somewhat the same** for both exams

Mathematically, we can choose a **linear model** to represent the assumption aboves:

$$F = \alpha M - \beta + \epsilon$$

where

- α is a **positive proportionality factor** representing the relative amount of work put into the final in comparison with the midterm
- β is a **positive** number modelling the fact that the **final is harder than the midterm**
- ϵ is a **normally distributed error term** accounting for the fact that **particular students may not follows perfectly our model**

Let us simulate grades according to this model. We will do that in three steps:

- simulate the midterm grades as before
- use our model $F = \alpha \times M - \beta$ to simulate the final grades
- add an normally distributed error term ϵ to the final grades

Good practice wants us to package our linear simulation in a function:

In [6]:

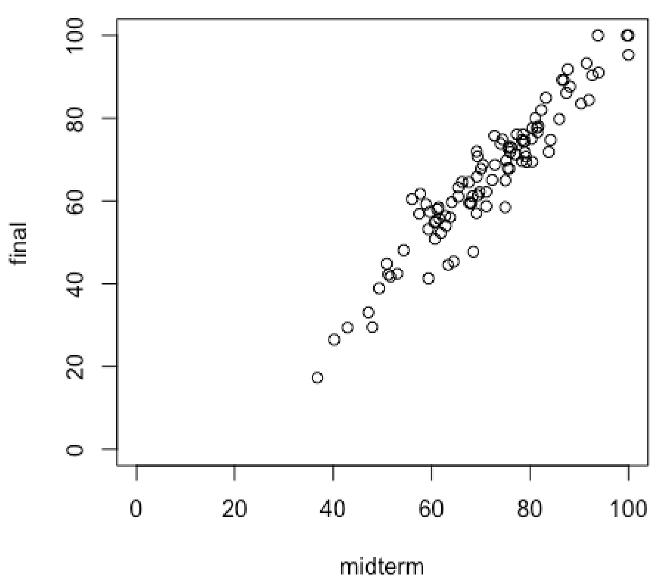
```
%%R

clean.grades = function(g, score.min, score.max){
        g[g < score.min] = score.min
        g[g > score.max] = score.max
        return(g)
    }

simulate.final = function(midterm, alpha, beta, error, score.max){

        epsilon = error * rnorm(length(midterm))
        final_sim = clean.grades(alpha*midterm - beta + epsilon, 0, score.max)
        return(final_sim)
    }
```

Without adding the **error term** ϵ, a **scatter plot** makes the **variable relation cristal clear**.

Increasing the error term tends to mask the functional realtionship between our two variables, but it is still visible:

In [7]:

```
%%R -r 96

n = 100; alpha = 1.2; beta = 20; error = 5; score.max = 100

final = simulate.final(midterm, alpha, beta, error, score.max)

plot(midterm, final, xlim=c(0, 100), ylim=c(0, 100))
```



Simulating population clusters

Now suppose, we want to add to our grade model the fact that student from different majors may perform differently in the class exams.

Suppose we have two majors:

- LITT for litterature majors
- DOUB for double majors in CS and statistics

For a "Computing with Data" class, one may expect the DOUB major to perform the better and the LITT major.

Let's try to built this feature into our simulated grade data.

First, we simulate midterm grades with different means and standard deviations for the two groups:

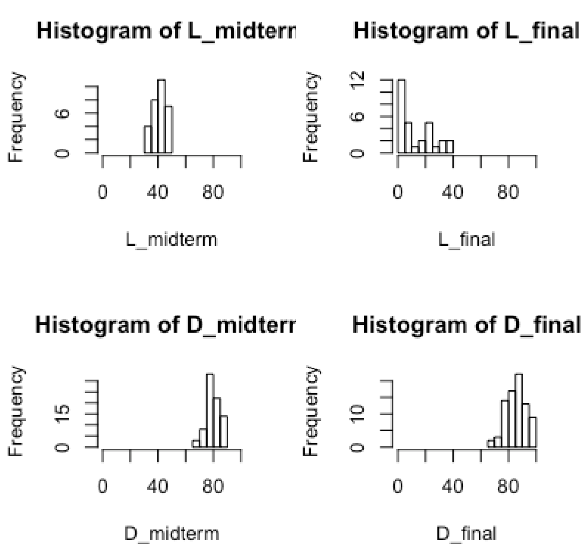
In [8]:
```
%%R
max_score = 100

L_number = 30; L_mean = 40; L_sd = 5
D_number = 80; D_mean = 80; D_sd = 5

L_midterm = simulate.grades(L_number, L_mean, L_sd, max_score)
D_midterm = simulate.grades(D_number, D_mean, D_sd, max_score)
```

Second, we use our linear model with different parameters for the two groups:

In [9]:
```
%%R
L_alpha = 0.5; L_beta = 10; L_error = 15
D_alpha = 1.2; D_beta = 10; D_error = 5

L_final   = simulate.final(L_midterm, L_alpha, L_beta, L_error, max_score)
D_final   = simulate.final(D_midterm, D_alpha, D_beta, D_error, max_score)
```

The **historgram plots** for the exam scores for each section already reveal a certain **clustering** in the data:

In [10]:
```
%%R -r 96 -w 400 -h 400
par(mfrow=c(2,2))
hist(L_midterm, xlim=c(0,100)); hist(L_final, xlim=c(0,100))
hist(D_midterm,   xlim=c(0,100)); hist(D_final, xlim=c(0,100))
```



Third, we create a **data frame for each major** with a **column for the section names**, and we **concatenate the two data frames** into a single one using the **row bind** function:

In [11]:
```
%%R

L_grades  = data.frame(Final=L_final, Midterm=L_midterm, Section=rep('LITT', L_number))
D_grades  = data.frame(Final=D_final, Midterm=D_midterm, Section=rep('DOUB', D_number))
grades    = rbind(L_grades, D_grades)

print(head(grades))
```

```
      Final  Midterm Section
1  6.411672 44.46837    LITT
2 17.169801 34.76351    LITT
3 31.799930 49.85669    LITT
4  0.000000 38.08184    LITT
```

```
5  6.514070 48.27073    LITT
6 21.634841 47.56106    LITT
```

Invoked with an integer n, the function

```
sample(n)
```

return a vector with the first n integers **randomly permuted entries**.

Let us use this trick to shuffle the LITT and DOUB sections in our data frame grade.

Remark: The function `nrow(df)` and `ncol(df)` return respectively the number of rows and the number of columns of the data frame `df`.

In [12]:

```
%%R

m = nrow(grades)

grades = data.frame(grades[sample(m), ], Student=paste('Name', 1:m, sep=''), row.names=4)

grades$Section = factor(grades$Section)

print(tail(grades))
```

```
            Final  Midterm Section
Name105  85.93962 80.64428    DOUB
Name106 100.00000 85.55716    DOUB
Name107  84.84895 84.73293    DOUB
Name108  90.74680 74.16169    DOUB
Name109  87.89609 81.53302    DOUB
Name110  21.25706 47.48521    LITT
```

We can now display the **scatter plot** of our two variables.

To verify that the clusters correspond to the majors, we'd like to plot our points in two different colors depending on the section.

The function **plot** takes a **character vector**

```
col=color_vector
```

containing the **colors with which each point is to be plotted.**

To construct this **color vector**, we can first contruct a vector with the

In [13]:

```
%%R

my_colors = c('blue', 'purple')
```

Since the data frame column corresponding to the section is a **factor** with two possible integer values:

- 1 for 'LITT'
- 2 for 'DOUB'

we can construct our **color vector** using the **bracket operator** on the vector containing our two colors:

In [14]:

```
%%R

point_colors = my_colors[grades$Section]

print(point_colors)
```

```
  [1] "blue"   "blue"   "purple" "purple" "blue"   "purple" "purple" "purple"
  [9] "purple" "purple" "blue"   "purple" "purple" "purple" "purple" "blue"  
 [17] "purple" "purple" "purple" "blue"   "blue"   "purple" "purple" "blue"  
 [25] "purple" "purple" "blue"   "blue"   "purple" "purple" "purple" "blue"  
 [33] "purple" "purple" "purple" "purple" "purple" "blue"   "purple" "purple"
 [41] "blue"   "purple" "purple" "purple" "purple" "purple" "purple" "purple"
 [49] "purple" "purple" "blue"   "purple" "purple" "purple" "purple" "purple"
 [57] "purple" "blue"   "purple" "purple" "purple" "blue"   "purple" "blue"  
 [65] "blue"   "purple" "purple" "blue"   "purple" "purple" "blue"   "blue"  
 [73] "purple" "purple" "purple" "purple" "purple" "blue"   "purple" "purple"
 [81] "blue"   "purple" "purple" "blue"   "purple" "purple" "purple" "purple"
 [89] "purple" "purple" "purple" "purple" "purple" "blue"   "blue"   "purple"
 [97] "purple" "blue"   "purple" "purple" "blue"   "purple" "purple" "blue"  
[105] "purple" "purple" "purple" "purple" "purple" "blue"  
```

Let us pass this **color vector** as the `col` argument to plot and **add a legend** using the function:

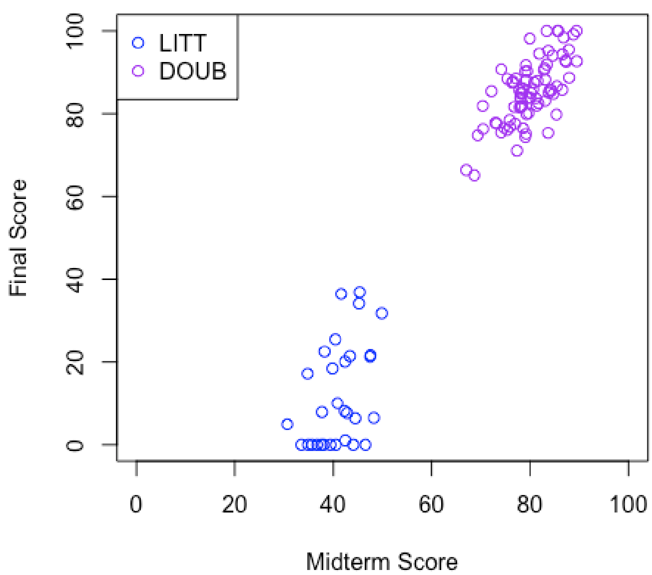
Let us pass this **color vector** as the `col` argument to plot and **add a legend** using the function:

```
legend(location, legend=levels(grades$Section), col=my_colors, pch=1)
```

where

- the `location` argument can be `'topleft', 'toprigth', 'bottomleft', 'bottomrigth'`
- the `legend` corresponds to the **category names**
- the `col` is a character vector containing the **category colors**
- the `pch=1` is a necessary argument that you should not worry about now

In [15]:
```
%%R -r 96

plot(grades$Midterm, grades$Final, xlim=c(0, 100), ylim=c(0, 100), #
     col=point_colors, xlab='Midterm Score', ylab='Final Score')

legend('topleft', legend=levels(grades$Section), col=my_colors, pch=1)
```



Conclusion:

The **scatter plot** shows us what we put into the data. Namely:

- **two student clusters** corresponding to two sections "LITT" and "DOUB"
- a seemingly **linear relationship** between the final grade and the midterm grade

The **linear variable relation** is more pronounced on the "DOUB" clusters. This is due to the fact that we cranked up the **error** term for the "LITT" student (L_ERROR = 15 while D_ERROR = 5).

EXERCISE: Try to see how the **population clusters** and the **variable relation** survive modification of the various parameters we entered into the model. This will give you sense of the care with with you should draw conclusion with real data.

2. Multivariate Analysis: Visualization

With **two variables**, it's possible to visualize **population clusters** and **variable relations** on a **single scatter plot**, since it involves to plot points on the **variable plane**.

With **three variables** we can still use a scatter plot, although we now need to plot our points in a **3 dimensional space**, which renders the interpretation slightly more challenging.

With **more than three variables** the only way to visualize the **scatter plot** (which involves plotting in a n-dimensional space, where n is the number of variables) is to **plot 2-dimensional projections of the n-dimensional scatter plot**. This means that we can plot all the **scatter plots for each pair of variables**. The visual situation is however more challenging to interpret, and we will need to use other tools for that, as we will see.

To get started, let's add a variable to our simulated grade data and plot the 3D scatter plot.

The variable we will add is an homework grade. To keep simulation simple, we will generate a homework grade for the whole class, without making any distinction between the two sections.

In [16]:
```
%%R

homework = simulate.grades(L_number + D_number, score.mean=80, score.sd=10, max_score)
grades$Homework = homework
grades = grades[,c('Final', 'Midterm', 'Homework', 'Section')]

print(tail(grades))
```

```
             Final  Midterm Homework Section
Name105  85.93962 80.64428 70.01079    DOUB
Name106 100.00000 85.55716 90.77850    DOUB
Name107  84.84895 84.73293 68.01026    DOUB
Name108  90.74680 74.16169 82.16637    DOUB
Name109  87.89609 81.53302 81.43087    DOUB
Name110  21.25706 47.48521 69.34250    LITT
```

The `lattice` library and the `cloud` function

The **lattice library** offers advance plotting capabilities in R. To load it, type in

```
library(lattice)
```

The display **3D scatter plots** of 3 continous variables, say X, Y, and Z, stored in a data frame `myFrame`, we will use the lattice function

```
cloud(Z ~ X + Y, data=myFrame, group=cat, auto.key=T)
```

- The **first argument** is an **R formula**. For now, you only need to understand that

  ```
  Z ~ X + Y
  ```

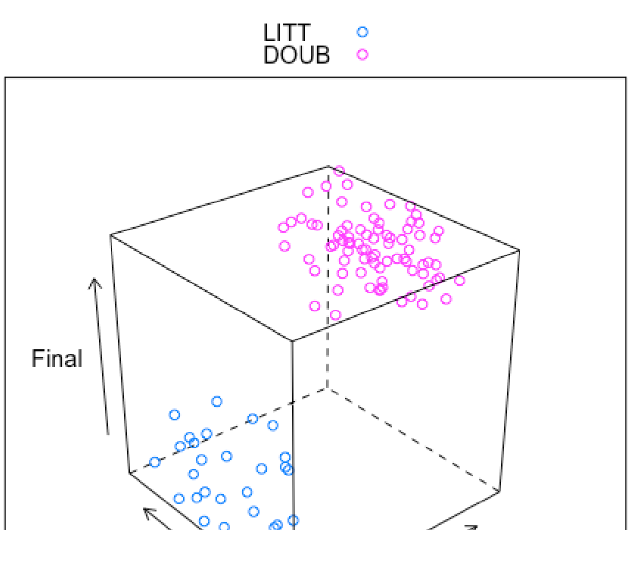
means that Z will be plotted on the z-axis, while X and Y will be plotted on the xy-plane.

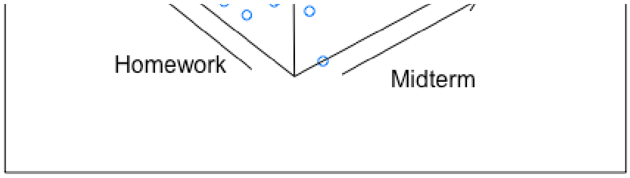
- The **second argument** `data` is the data frame from which the **quantitative variables** X, Y, and Z will be retrieved from.
- The **third argument** `group` is a **categorical variable** from the data frame `myFrame` indicating that the points from different categories should be displayed with different colors.
- The **last argument** `auto.key` is a Boolean argument indicating a legend describing which color corresponds to which category should be plotted.

Let us display a 3D scatter plot of our variables `Final`, `Midterm` and `Homework` and plot them with different colors depending on the `Section` variable.

Remark: In the notebook, one needs to store the output of `cloud` into a variable, which we will need to print in order for the plot to appears.

In [17]:
```
%%R -r 96 -w 700 -h 500

library(lattice)

grade_cloud = cloud(Final ~ Midterm + Homework, data=grades, group=Section, auto.key=T)

print(grade_cloud)
```





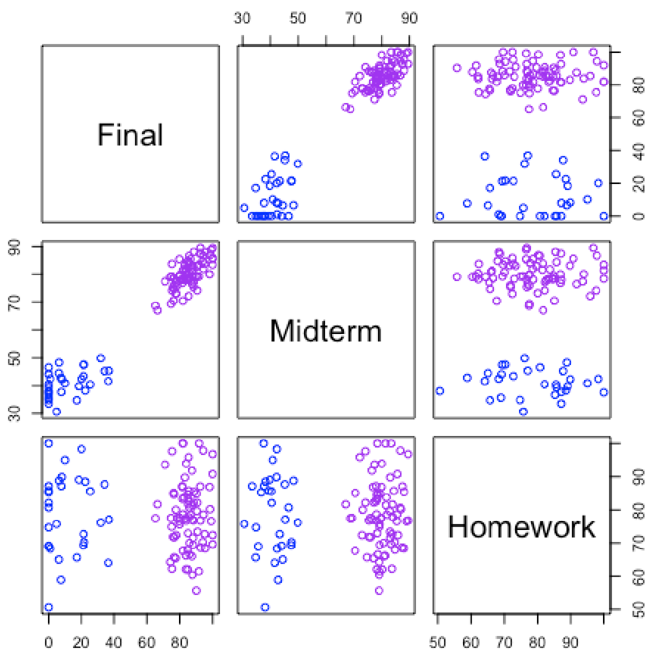
One still clearly seet the two clusters of LITT and DOUB majors. The linear relationship between the variables is however not clearly perceptible on the plot.

The scatter plot matrix

Another way to display **variable relations** and **population clusters** for **more than 2 variables** is to plot a **scatter plot matrix** containing a scatter plot of **each pair** of variables.

One can again pass our **color vector** to the `col` argument.

In [18]:
```
%%R -r 96 -w 500 -h 500

plot(grades[,c(1,2,3)], col=point_colors)
```



The two **section clusters are very visible** in each of the scatter plots.

At contrast with our 3D scatter plot, **one can still see the linear relation ship between the midterm score and the final score** that we build into our data.

Although still very useful for three variables, we begin to get a sense that **more sophisticated methods will be needed as the number of variable increases**... That's what we are going to learn next.

3. Multivariate analysis: Clustering

Let us add a fourth variable to our simulated grade data:

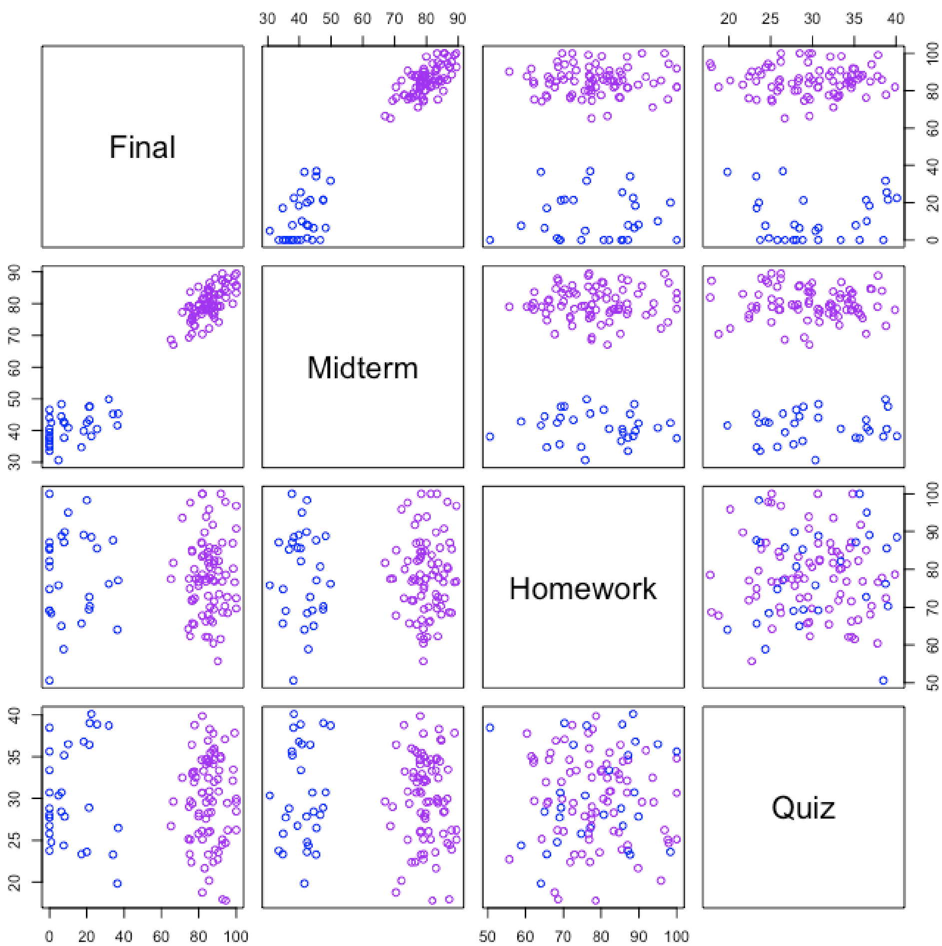
In [19]:
```
%%R

quiz = simulate.grades(L_number + D_number, score.mean=30, score.sd=5, max_score)
grades$Quiz = quiz
grades = grades[,c('Final', 'Midterm', 'Homework', 'Quiz', 'Section')]
print(head(grades))
```

```
         Final  Midterm  Homework      Quiz Section

Name1  8.167408 42.28068  89.86770 27.85688    LITT
Name2  0.000000 39.39495  85.75620 26.71910    LITT
```

```
Name3 91.891964 83.39170 100.00000 34.79697     DOUB
Name4 87.820700 79.03601  60.37647 37.78026     DOUB
Name5  1.018811 42.47156  68.35079 24.79602     LITT
Name6 83.712879 81.23105  66.23481 34.65286     DOUB
```

We can still display a **scatter plot matrix**:

In [20]:
```
%%R -r 96 -w 700 -h 700

plot(grades[,c(1,2,3,4)], col=point_colors)
```



We still see some clustering appearing for some variables, but for others (Quiz and Homework) the clustering does not seem to be present.

As the number of variable grows, we need to use other methods to retrieve clusters from the data.

The distance matrix

Consider n **variables** $X_1, \dots, X_n$ for a **population sample** of m **individual**.

All the **variable values** for all the **sample individuals** can be arranged into a matrix

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & & & \vdots \\ x_{m1} & x_{2m} & \cdots & x_{mn} \end{pmatrix} \qquad \text{where} \qquad x_{ij} = X_i(s_j) \quad (\text{value of the variable} \quad X_i \quad \text{for the individual} \quad s_j\ )$$

In other word, this matrix is the **data frame stripped from its row and column labels**.

The **points** $v_1, \dots, v_m$ represented in a **scatter plots** are just the **row vectors** of this value matrix:

$$
\begin{aligned}
v_1 &= (x_{11}, x_{12}, \ldots, x_{1n}) \\
v_2 &= (x_{21}, x_{22}, \ldots, x_{2n}) \\
&\vdots \\
v_m &= (x_{m1}, x_{m2} \ldots, x_{mn})
\end{aligned}
$$

In R, a **matrix** is a represented by a **vector** with the **hidden dimension vector** set up with the corresponding number of rows and columns, in the same way a **class** is a **list** with the **hidden class string** set up to the **class name**.

This **hidden dimension vector** is accessed and set up through the function

```
dim(x)
```

Changing the dimension of a vector makes R interpret the **resulting object as an instance of the class** `matrix`.

In [21]:
```
%%R

x = c(1,2,3,4,5,6,7,8,9)
print(class(x))

dim(x) = c(3,3)

print(class(x)); cat('\n')
print(x)
```

```
[1] "numeric"
[1] "matrix"

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

One can extract the **value matrix** from a data frame using the **conversion funtion**

```
as.matrix(x)
```

In [22]:
```
%%R

scores = as.matrix(grades[,c(1,2,3,4)])
print(class(scores))
```

```
[1] "matrix"
```

The **distance matrix** of a matrix $A = (x_{ij})$ is the matrix containing all the **distances between** the matrix **row vectors** $v_1, v_2, \ldots, v_m$:

$$
\mathrm{dist}(A) \quad = \quad \begin{pmatrix} d(v_1, v_1) & d(v_1, v_2) & \cdots & d(v_1, v_n) \\ d(v_2, v_1) & d(v_2, v_2) & \cdots & d(v_2, v_n) \\ \vdots & & & \vdots \\ d(v_m, v_1) & d(v_m, v_2) & \cdots & d(v_m, v_n) \end{pmatrix}
$$

where

$$
d(v_i, v_j) = \sum_{k=1}^{m} \sqrt{(x_{ik}^2 - x_{jk}^2)}
$$

is the **euclidean distance** between the **row vectors** v_i and v_j.

In R, one computes **the distance matrix** using the function

```
dist(x)
```

The **distance matrix** contains all the necessary information to perform **clustering**.

In [23]:
```
%%R

scores_dist = dist(scores)
```

Hierachical Clustering Analysis

The function

The function

```
hclust(distance_matrix)
```

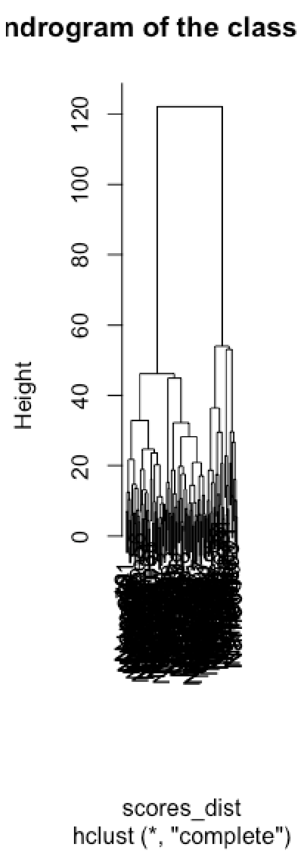
takes a **distance matrix** and returns an object of the class `hclust`, which contains a **hierachical groups of the population**.

This **hierachical groups** are computed as follows by `hclust`:

- it starts by **groupping indiduals (i.e. rows) two by two by smallest distance**
- then it computes a distance between this pairs of two and **group these pairs again two by two by smallest distance**
- this algorithm continues until there is **no group is left for groupping**

The **hierachy that results** can be further plotted as a **dendrogram** using the `plot` funciton.

In [24]:
```
%%R -r 96 -w 200 -h 600

hclusters = hclust(scores_dist)

plot(hclusters, main='Dendrogram of the class scores')
```



We see from the dendrogram that we can obtain a different number of cluster by **cutting the dendogram at a given height**.

In our example,

- **cutting** at a **distance between 60 and 120**, then **two cluster emerges** that most probably correspond to the two majors.
- **cutting** at a **distance of 40**, will produce **several additional clusters** revealing more structure in the data.

Of course, if the distance between the two cluster is **small** it may indicate that the **cluster structure** is not very strong. In our example, only the two main clusters seems to be real.

To **retrieve a given number of cluster**, one uses the function

```
cutree(hcluster_object, number_of_clusters)
```

which returns a **integer vector** where each element is an **integer** telling **which cluster the corresponding individual in our population belongs to**. This **labels** of this vectors are the **individual names**.

In [25]:
```
%%R
```

```
clusters = cutree(hclusters, 2)
print(clusters)
```

```
  Name1   Name2   Name3   Name4   Name5   Name6   Name7   Name8   Name9  Name10 
      1       1       2       2       1       2       2       2       2       2 
 Name11  Name12  Name13  Name14  Name15  Name16  Name17  Name18  Name19  Name20 
      1       2       2       2       2       1       2       2       2       1 
 Name21  Name22  Name23  Name24  Name25  Name26  Name27  Name28  Name29  Name30 
      1       2       2       1       2       2       1       1       2       2 
 Name31  Name32  Name33  Name34  Name35  Name36  Name37  Name38  Name39  Name40 
      2       1       2       2       2       2       2       1       2       2 
 Name41  Name42  Name43  Name44  Name45  Name46  Name47  Name48  Name49  Name50 
      1       2       2       2       2       2       2       2       2       2 
 Name51  Name52  Name53  Name54  Name55  Name56  Name57  Name58  Name59  Name60 
      1       2       2       2       2       2       2       1       2       2 
 Name61  Name62  Name63  Name64  Name65  Name66  Name67  Name68  Name69  Name70 
      2       1       2       1       1       2       2       1       2       2 
 Name71  Name72  Name73  Name74  Name75  Name76  Name77  Name78  Name79  Name80 
      1       1       2       2       2       2       2       1       2       2 
 Name81  Name82  Name83  Name84  Name85  Name86  Name87  Name88  Name89  Name90 
      1       2       2       1       2       2       2       2       2       2 
 Name91  Name92  Name93  Name94  Name95  Name96  Name97  Name98  Name99 Name100 
      2       2       2       1       1       2       2       1       2       2 
Name101 Name102 Name103 Name104 Name105 Name106 Name107 Name108 Name109 Name110 
      1       2       2       1       2       2       2       2       2       1 
```

We can now separate the rows or our initial data frame into two categories, corresponding to the returned clusters and check that they represent the two majors:

In [26]:

```
%%R

cluster1.names = names(clusters[clusters == 1])
students1 = grades[cluster1.names,]
print(students1)
```

```
            Final  Midterm  Homework     Quiz Section
Name1    8.167408 42.28068  89.86770 27.85688    LITT
Name2    0.000000 39.39495  85.75620 26.71910    LITT
Name5    1.018811 42.47156  68.35079 24.79602    LITT
Name11   6.514070 48.27073  88.78673 30.72553    LITT
Name16  36.858106 45.34581  77.06705 26.46216    LITT
Name20   7.650583 42.83610  58.85665 24.38272    LITT
Name21  34.135075 45.21806  87.68278 23.27935    LITT
Name24  18.456039 39.82039  89.04705 36.80462    LITT
Name27  10.040498 40.85245  95.02425 36.48378    LITT
Name28   0.000000 36.72109  85.28308 28.81202    LITT
Name32   6.411672 44.46837  65.03373 28.43104    LITT
Name38  31.799930 49.85669  76.14792 38.71780    LITT
Name41   4.950563 30.65106  75.76314 30.35330    LITT
Name51   0.000000 35.67982  68.99259 27.74343    LITT
Name58  21.393468 43.39615  72.68805 36.41388    LITT
Name62   7.888644 37.72932  87.15932 35.16833    LITT
Name64  21.634841 47.56106  70.26098 39.00863    LITT
Name65  25.473728 40.41483  85.59218 38.85771    LITT
Name68   0.000000 33.52930  87.13033 23.74760    LITT
Name71  36.444332 41.61503  64.06714 19.82498    LITT
Name72  22.496625 38.23300  88.47793 40.10673    LITT
Name78  20.099845 42.41015  98.29730 23.59381    LITT
Name81   0.000000 38.08184  50.60226 38.46387    LITT
Name84   0.000000 44.07351  69.15280 30.70273    LITT
Name94   0.000000 46.53951  80.71281 28.06864    LITT
Name95   0.000000 37.58013 100.00000 35.61793    LITT
Name98   0.000000 34.87726  74.72632 25.78762    LITT
Name101 17.169801 34.76351  65.66622 23.32371    LITT
Name104  0.000000 40.49539  82.10907 33.37872    LITT
Name110 21.257056 47.48521  69.34250 28.89553    LITT
```

In [26]:

In [26]:

In [26]:

In [26]:

In [ ]: