Introduction to Computing for Economics and Management

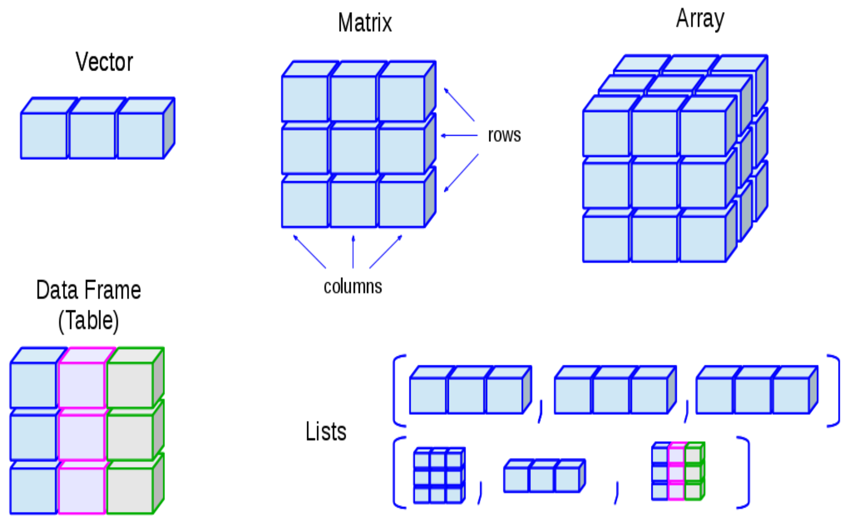
Final Summary

Final Exam Contents

- Vectors
- Matrices
- Lists
- Data frames
- For-loop
- While-loop
- Repeat-loop
- Break and next statements
- If-else statement
- Nested loops
- Scatter plot
- Histogram
- Barplot
- Figure arrays
- Stripchart
- Pie chart
- Read from the keyboard
- Import files
- Accessing data from the Internet
- Writing data to files
- Factors
- Correlation and regression

Data structure types

- Vectors: one-dimensional arrays
  - Numeric vectors
  - Complex vectors
  - Logical vectors
  - Character vectors or text strings
- Matrices: two-dimensional
- Arrays: multi-dimensional
- Factors: vectors of categorical varial
- Lists: ordered collection of objects
- Data frames: generalization of matrices, different columns can store different mode data
- Functions: objects that make specific operations



Short Summary Vectors

Integer mode

```
> person.weight <- c(65, 66, 61)
```

Numeric (floating-point number)

```
> person.height <- c(1.70, 1.75, 1.62)
```

Character (string)

```
> person.name <- c("Can", "Cem", "Hande")
```

Logical (Boolean)

```
> person.female <- c(FALSE, FALSE, TRUE)
```

Complex

```
> complex.numbers <- c(1+2i, -1+0i)
```

Short Summary Vectors

Short Summary Vectors

Filtering

```
> person.height[person.height > 1.65]
 Can   Cem
1.72 1.75
```

Recycling

```
> c(1, 2, 3) + c(1, 2, 3, 4)
[1] 2 4 6 5
```

Vector operations

```
> person.weight / person.height^2
  C(1,2,3)              c(2,4)
> person.weight / person.height * person.height
     Can       Cem     Hande
21.97134 21.55102 23.24341
```

Short Summary Matrices

Creation

```
> y <- matrix(c(1,2,3,4),nrow=2,ncol=2)

> cbind(c(1,2), c(3,4))
```

Matrix operations

Transposition `t(y)`
Element by element product `y * y`
Matrix multiplication `y %*% y`
Matrix scalar multiplication `3 * y`
Matrix addition `y + y`

Indexing, e.g. select first and second row

```
> z[c(1,2),]
```

Short Summary Matrices

Assign new values to submatrices

```
> z[c(1:2), c(2:3)] <- matrix(c(20,21,22,23), nrow=2)
```

Filtering, e.g. obtain those rows of matrix `z` having elements in the second column which are at least equal to 5

```
> z[z[,2] >= 5,]
```

Short Summary Lists

Creation

```
> joe <- list(name="Joe", salary=55000, staff=T)
```

Indexing

```
> joe$salary
> joe[["salary"]]
> joe[[2]]
```

Vectors as list components

```
> my.list <- list(vec1 = c(1,2), vec2 = c(3,4),
vec3 = 5:7)
```

Nested list

Short Summary Data frames

Creation

```
> person <- data.frame(height=person.height,
weight=person.weight)
```

Indexing

```
> person[[1]]
> person[["height"]]
> person$height
> person[c(1,2),]
> person[-3,]
```

Filtering

```
> person[person$height >= 1.7,]
```

Short Summary Data frames

Data import

```
> person.data <- read.table(header=TRUE,
"height_weight_data.txt", sep=",")
```

Data modifications

```
> person.data$BMI <- person.data$Weight /
person.data$Height^2
```

Summary

```
> summary(person.data)
```

Merging

```
> merge(person.data, person.data2)
```

Factors example 1

```
patientId <- c(1,2,3,4)
age <- c(25,34,28,52)
diabetes <- c("Type1","Type2","Type1","Type1")
diabetes <- factor(diabetes)
status <- c("Poor","Improved","Excellent","Poor")
status <- factor(status,order=TRUE)
patientdata <- data.frame(patientId,age,diabetes,status)
summary(patientdata)
```

```
  patientId         age          diabetes       status
 Min.   :1.00   Min.   :25.00   Type1:3   Excellent:1
 1st Qu.:1.75   1st Qu.:27.25   Type2:1   Improved :1
 Median :2.50   Median :31.00             Poor     :2
 Mean   :2.50   Mean   :34.75
 3rd Qu.:3.25   3rd Qu.:38.50
 Max.   :4.00   Max.   :52.00
```

Looping

The most frequently used looping construct is

```
x <-c(1,2,3)
for(i in x) {expression}
```

The for-loop iterates through all elements of the vector `vec`

For each element of the vector `vec` there will be one iteration of the loop and `expression` is executed

At each iteration, the variable `x` takes the value of the current element of `vec`

- First iteration: `x = vec[1]`
- Second iteration: `x = vec[2]`

```
l <- list(c(1,2),c(3,4,5))
for (i in 1:length(l)){
  if (i==1)
    next
  for (y in l[[i]])
    print(y)
}
```

Print variable when iterating

Let's print out the value of variable `x` when iterating through the vector `vec`

```
> vec <- c(1:5)

> vec
[1] 1 2 3 4 5

> for(x in vec) {print (x)}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Print variable when iterating

- The for-loops works with other modes beside numeric as well

Example: like before we print the value of the variable when iterating through a vector of strings

```
> word.vector <- c("a", "text", "consists",
"of")

> for(word in word.vector) {print (word)}
[1] "a"
[1] "text"
[1] "consists"
[1] "of"
```

Print variable when iterating

As an alternative we can create a new vector which ranges from 1 until the length of the vector, iterate through this vector and access the original vector via indexing

```
> vector.indices <- 1:length(word.vector)

> vector.indices
[1] 1 2 3 4

> for(i in vector.indices) {print(word.vector[i])}
[1] "a"
[1] "text"
[1] "consists"
[1] "of"
```

Print variable when iterating

We can write the alternative way in one line

```
> for(i in word.vector) {print (i)}

> for(i in 1:length(word.vector)) {print
(word.vector[i])}

[1] "a"
[1] "text"
[1] "consists"
[1] "of"
```

Compute length of a vector

Write our own function for computing the length of a vector

```
## function to compute length of vector vec
vec.length <- function(vec)
{
  # initialize counter
  counter <- 0

  # iterate through vec and increase counter
  for(x in vec) {counter <- counter + 1}

  # return counter
  return(counter)
}
```

Compute Euclidean norm of a vector

```
## compute Euclidean norm of a vector vec
Euclid.norm <- function(vec)
{
  # initialize norm
  norm <- 0

  # compute sum of squared vector elements
  for(x in vec) {norm <- norm + x^2}

  # sqrt of sum
  norm <- sqrt(norm)

  return(norm)
}
```

Square elements of a vector

We can change the elements of the input vector and return a new vector, e.g. square the elements of a vector

```
## square elements of vector vec
square.vec <- function(vec)
{
  # initialize output vector vec.res
  vec.res <- c()

  # fill vec.res with squared elements of vec
  for(x in vec) {vec.res <- c(vec.res, x^2)}

  return(vec.res)
}
```

Nested loops

In nested loops, an inner loop is placed inside of another outer loop

```
for(i in 1:2)
{
  for(j in 1:3)
  {
    print(paste("outer", i, "inner", j))
  }
}
[1] "outer 1 inner 1"
[1] "outer 1 inner 2"
[1] "outer 1 inner 3"
[1] "outer 2 inner 1"
[1] "outer 2 inner 2"
[1] "outer 2 inner 3"
```

While loop

A frequently used looping construct is

`while(condition) {expression}`

- As long as the condition is satisfied, the expression is executed

Example:

```
> i <- 1
> while(i<5) {i <- i+1}
> i
[1] 5
```

- In the example we observe that the while loop is executed 4 times

Break

We can control when to exit the while-loop by using `break` in combination with an `if` statement

```
i <- 1
while(TRUE)
  {
    i <- i + 1
    if(i >= 10) {break}
  }
i
[1] 10
```

Repeat loop

Another looping construct is

- `repeat {expression}`

- Expression is executed until the loop is terminated with `break`

- In comparison to the while-loop there is no longer a condition test

- We can use it whenever we don’t have a condition to test

Repeat loop

Example in which we use `repeat` instead of `while(TRUE)`

```
i <- 1
repeat
  {
    i <- i + 1
    if(i >= 10) {break}
  }
i
[1] 10
```

Next

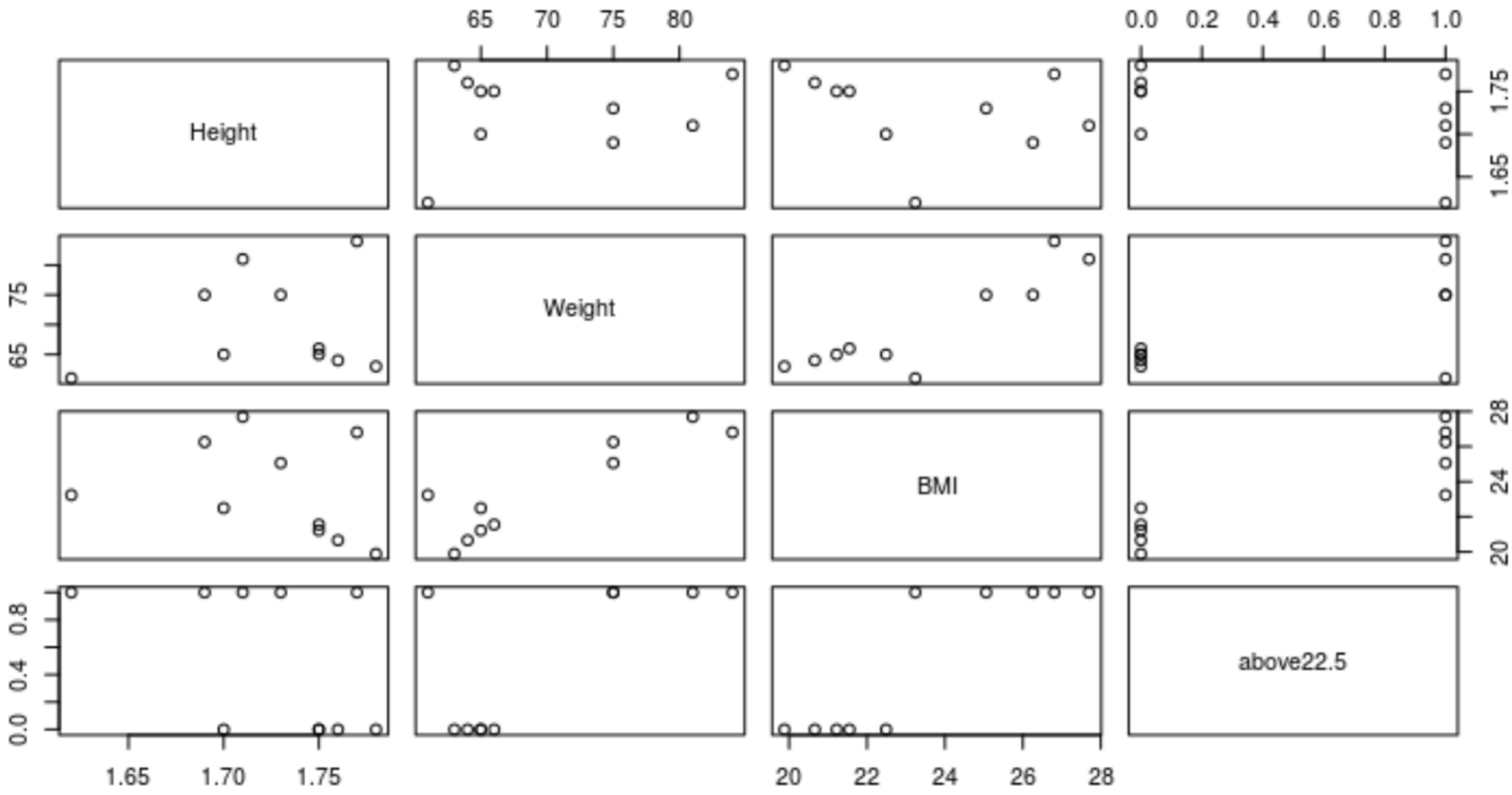
- Another useful statement is `next`, which skips the remainder of the current iteration of the loop and proceed directly to the next iteration

We can use a `next` statement in while-loops, repeat-loops and for-loops as well

```
for(i in 1:3)
{
  print("a")
  next
  print("b")
}
[1] "a"
[1] "a"
[1] "a"
```

Scatter plots of data frames

```
> plot(person.data[,2:5])
```



Histogram

- A histogram is similar to a barplot since we visualize how many observations fall within specified divisions called “bins”
- In R we simply call the `hist` function
- `hist` creates the bins automatically, counts the number of observations that fall within the bins and plots the result

A histogram of normal distributed random numbers with mean 0 is created with

```
> rnorm1000 <- rnorm(1000)
> hist(rnorm1000)
```

Histogram

```
> hist(rnorm1000)
```

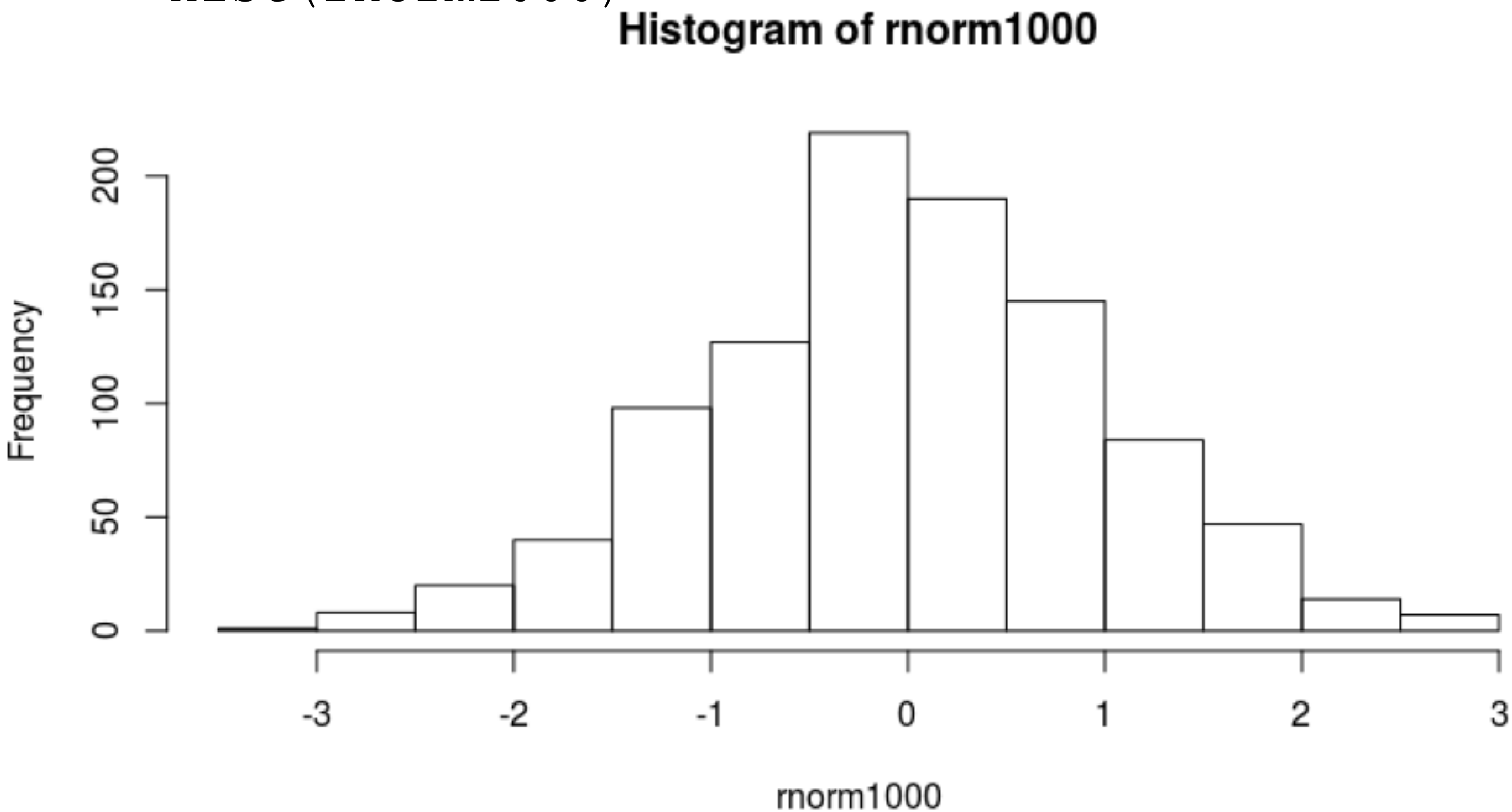


Figure array

- Sometimes we need to place several figures in the same plot
- For example, we would like to have the plot of the normal distributed random numbers and the corresponding histogram in one plot
- We use `par` to specify how several figures will be drawn in an number of rows by number of columns array
  - `par(mfrow=c(nr, nc))` specifies that figures will be drawn in an `nr`-by-`nc` array by rows
  - `par(mfcol=c(nr, nc))` specifies that figures will be drawn in an `nr`-by-`nc` array by columns

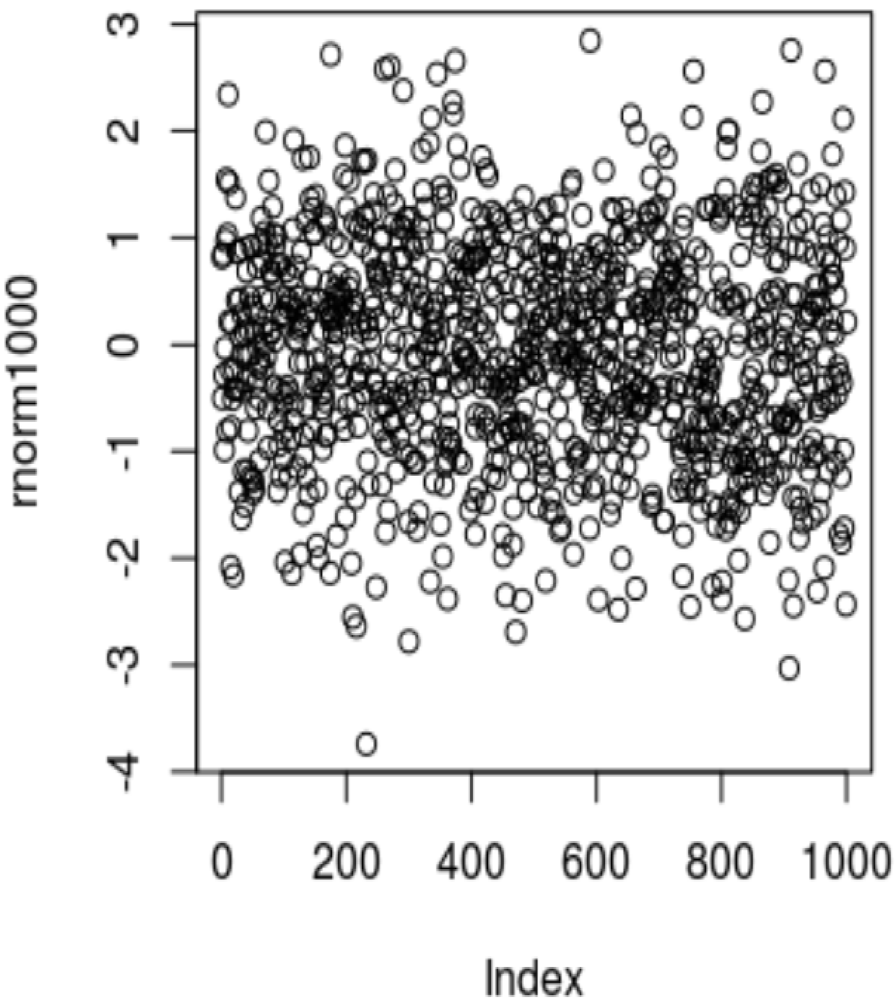
Figure array

We place the plot of the normal distributed random numbers and the corresponding histogram side by side

```
> rnorm1000 <- rnorm(1000)
> par(mfrow=c(1,2))
> plot(rnorm1000)
> hist(rnorm1000)
> par(mfrow=c(1,1))
```

- The last line resets `mfrow` to its default value

Figure array



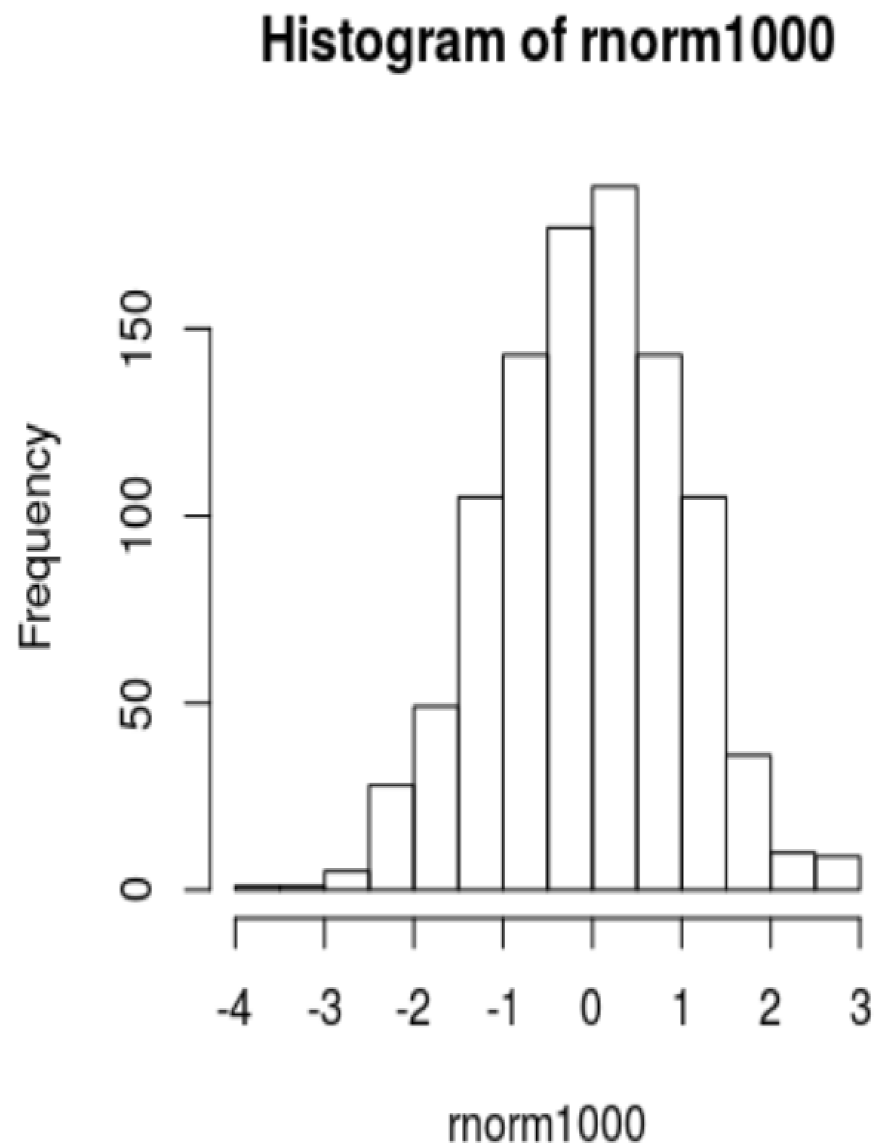
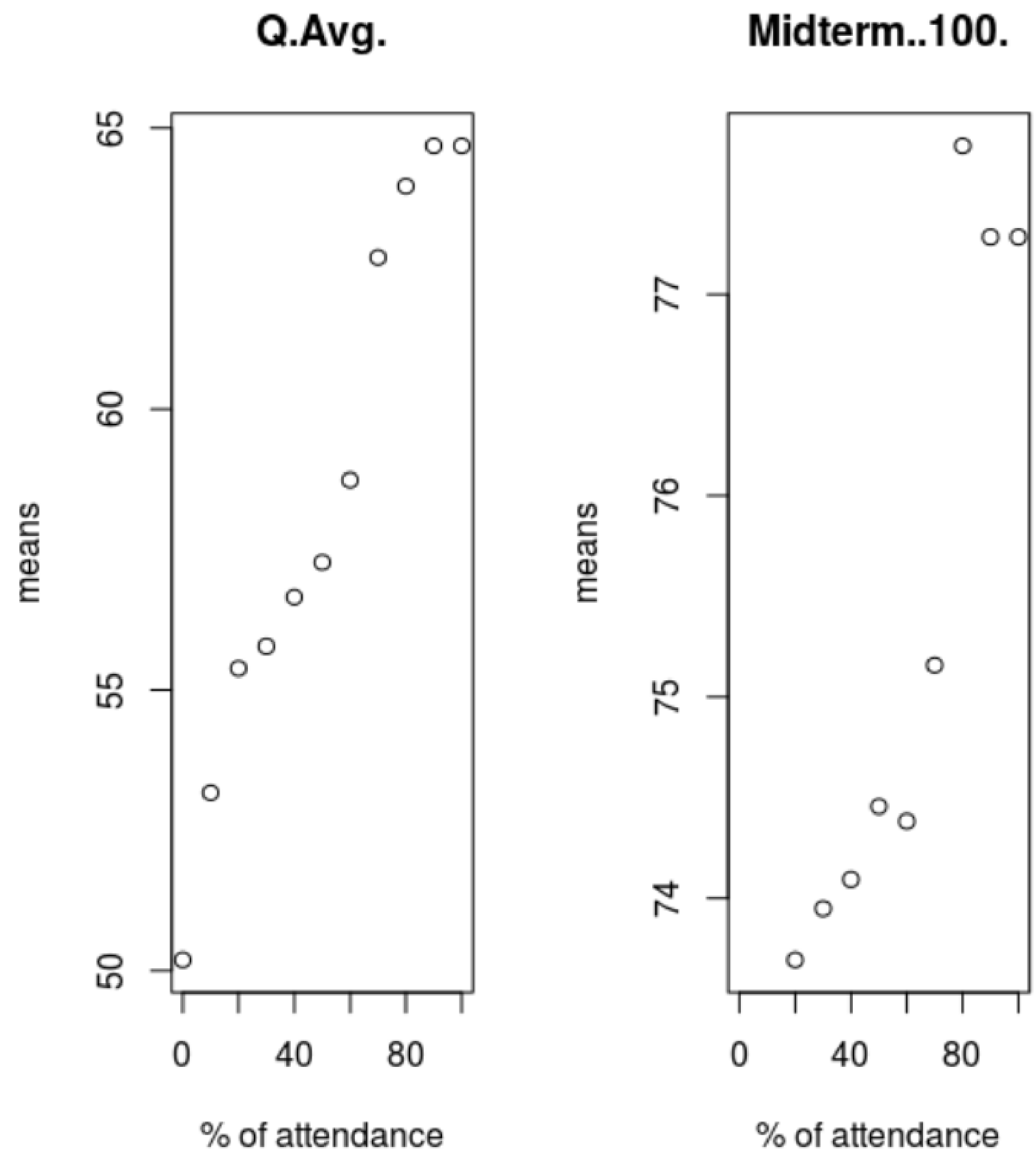


Figure array exercise: mid-semester analysis of grades, grades average bins, v2

```
# for each attendance level, plot mean grade of quiz and
# midterm
par(mfrow=c(1,2))
for (column in c(1:length(grades))){
  cNames=colnames(grades)
  if (cNames[column]=="Q.Avg." ||   # or
      cNames[column]=="Midterm..100." ){
    means<-c()
    indices <- c()
    for (x in seq(0,100,10)){
      m = mean(grades[grades$Attendance>=x,column])
      means <- c(means,m)
      indices <- c(indices,x)
    }
    plot(indices,means,main=colnames(grades)
         [column],xlab="% of attendance")
  }
}
```

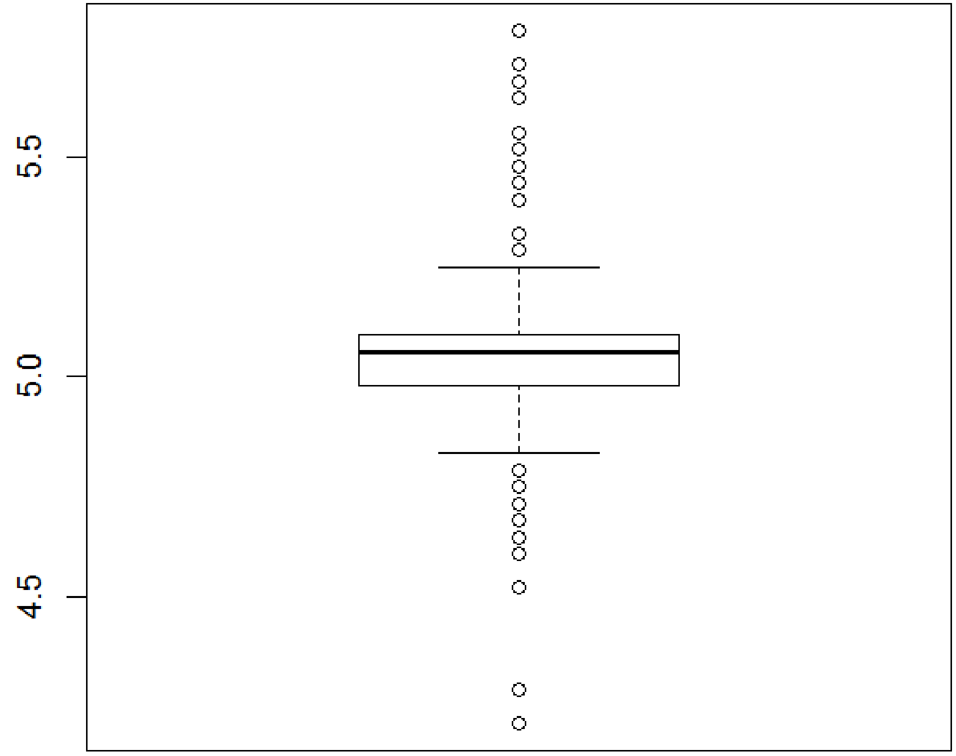
Figure array exercise:



Boxplot

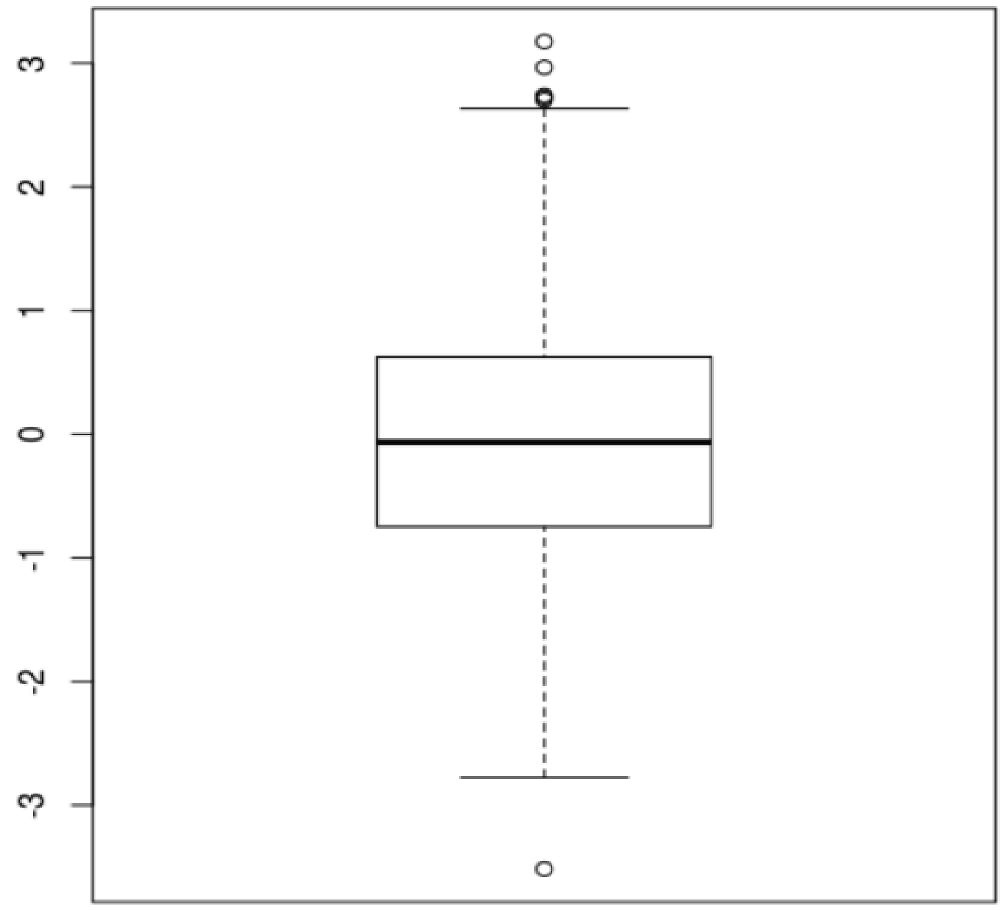
A “boxplot”, also known as “box-and-whiskers plot”, is a graphical summary of a distribution.

We differentiate between
- Box in the middle
- The lines (“whiskers”)
- Additional points



Boxplot

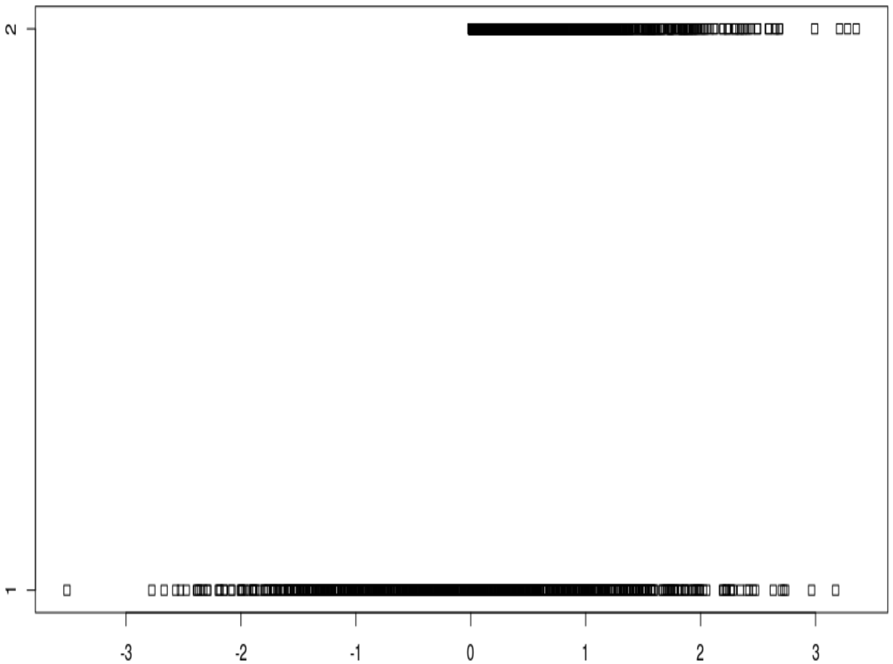
- The box in the middle indicates first quartile, median, and third quartile
- The lines (“whiskers”) show the largest or smallest observation that falls within a distance of 1.5 times the box size from the nearest quartile
- If any observations fall farther away, the additional points are considered “extreme” values and are shown separately

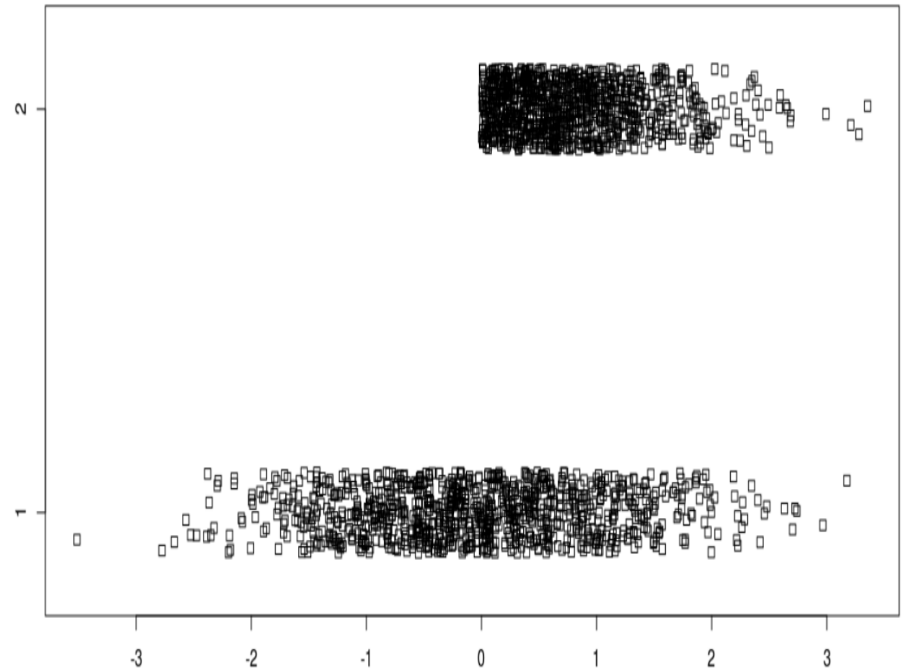


Stripcharts

```
stripchart(list(rnumbers$rnorm1000,
rnumbers$pos.rnorm1000))
```

```
stripchart(list(rnumbers$rnorm100
0, rnumbers$pos.rnorm1000),
method="jitter")
```





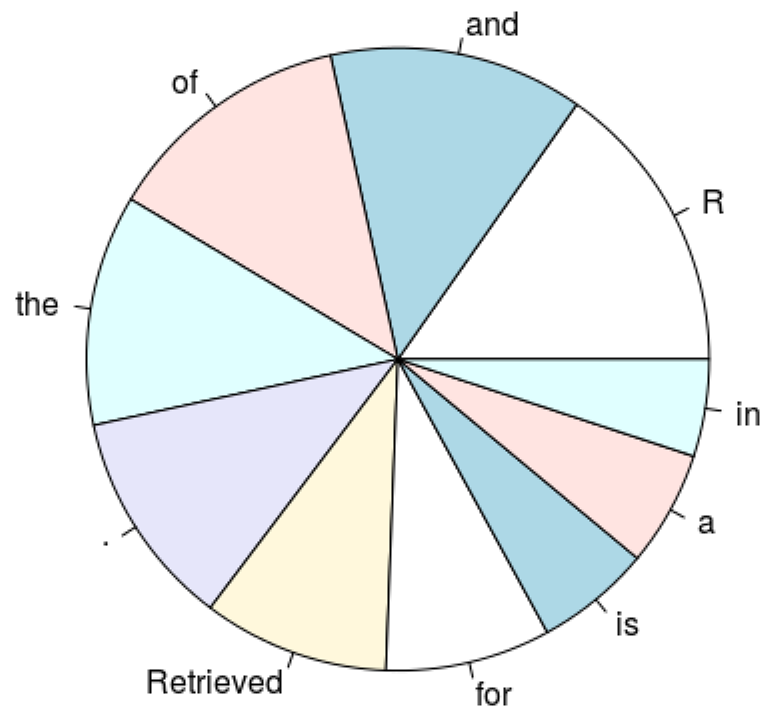
Pie charts

- Pie charts are an alternative to barplots
- We first start with the word list example
- Next we consider another example on caffeine consumption and marital status
- Next we consider an example on sales data

Pie chart of word frequency

We create a pie chart of word frequency with

```
> pie(word.freq[1:10])
```



Read data with scan

We already know the `scan` function from reading text data from a file into a vector

```
> word.vec <- scan("text.txt", what="")
```

- Usually we use `scan` to read the entire content of a file into a vector
- We have learned that we need to be careful when dealing with vectors and mixed modes

Read data with scan

We already know the `scan` function from reading text data from a file into a vector

```
> word.vec <- scan("text.txt", what="")
```

- Usually we use `scan` to read the entire content of a file into a vector
- We have learned in a previous lecture that we need to be careful when dealing with vectors and mixed modes

In the following we learn how `scan` behaves when confronted with mixed modes

Accessing files from the Internet

- Certain input functions, such as `read.table()` and `scan()`, can read data from web URLs as well

- Reading data from the Internet is of particular interest when dealing with real-time data like stock exchange

- In the following we will access files from
  - The UCI Machine Learning Repository
  - Project Gutenberg

Writing to a file

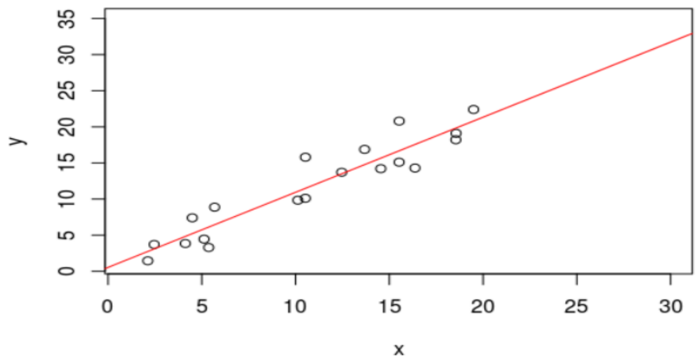
- So far we have imported data, processed the data and visualized the results

- A convenient way to save data to a file is `write.table()`

- It works similar to `read.table()`

- `write.table()` writes a data frame into a file

Linear regression

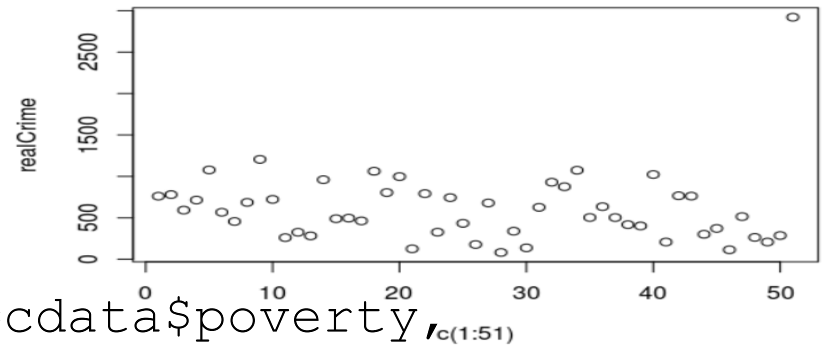



```
> plot(x,y,xlim=c(1,30),ylim=c(1,3
> myModel <- lm(y~x, data=df)
> abline(myModel,col="red")
> str(myModel)
List of 12
 $ coefficients : Named num [1:2] 0.536 1.04
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
 $ residuals    : Named num [1:20] -1.21 4.12 -1.37
-1.45 -2.82 ...
  ..- attr(*, "names")= chr [1:20] "1" "2" "3" "4" ..
# myModel is a list of 12 elements, use the first
element
```

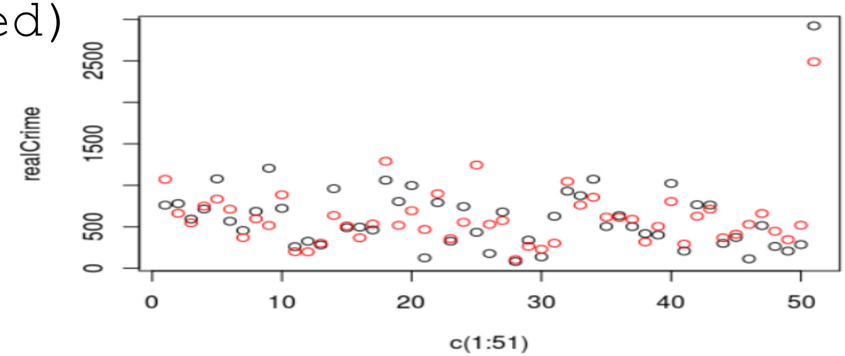
Linear regression, real world

Let us check out how our regression model performs:

```
realCrime <- cdata$crime
crimeModel <- lm(crime ~ poverty + single, data =
cdata)
plot(c(1:51),realCrime)
```




```
testData <- data.frame(poverty=cdata$poverty,
single=cdata$single)
predCrime <- predict(crimeModel, testData)
points(c(1:51),predCrime,col=red)
```



Final Exam

- **CMPE140.01**: Monday 2nd January between 10:00 and 11:30 in the computer lab BM B4
- **CMPE140.02**: Monday 2nd January between 11:30 and 13:00 in the computer lab BM B4

Preparation for Final Exam

- Make sure you did all homework
- Beside the lecture slides, use the slides and R scripts from problem and lab sessions and check the quizzes
- Everything is in the course webpage.
- Have a good exam !