HEADS

PhD PROGRAMME IN HEALTH DATA SCIENCE

Exploring R

Computational Statistics

PhD Programme in Health Data Science

Pedro Pereira Rodrigues

Teresa Henriques

U.PORTO
FMUP FACULDADE DE MEDICINA UNIVERSIDADE DO PORTO

MEDCIDS
DEPARTAMENTO DE MEDICINA DA COMUNIDADE, INFORMAÇÃO E DECISÃO EM SAÚDE

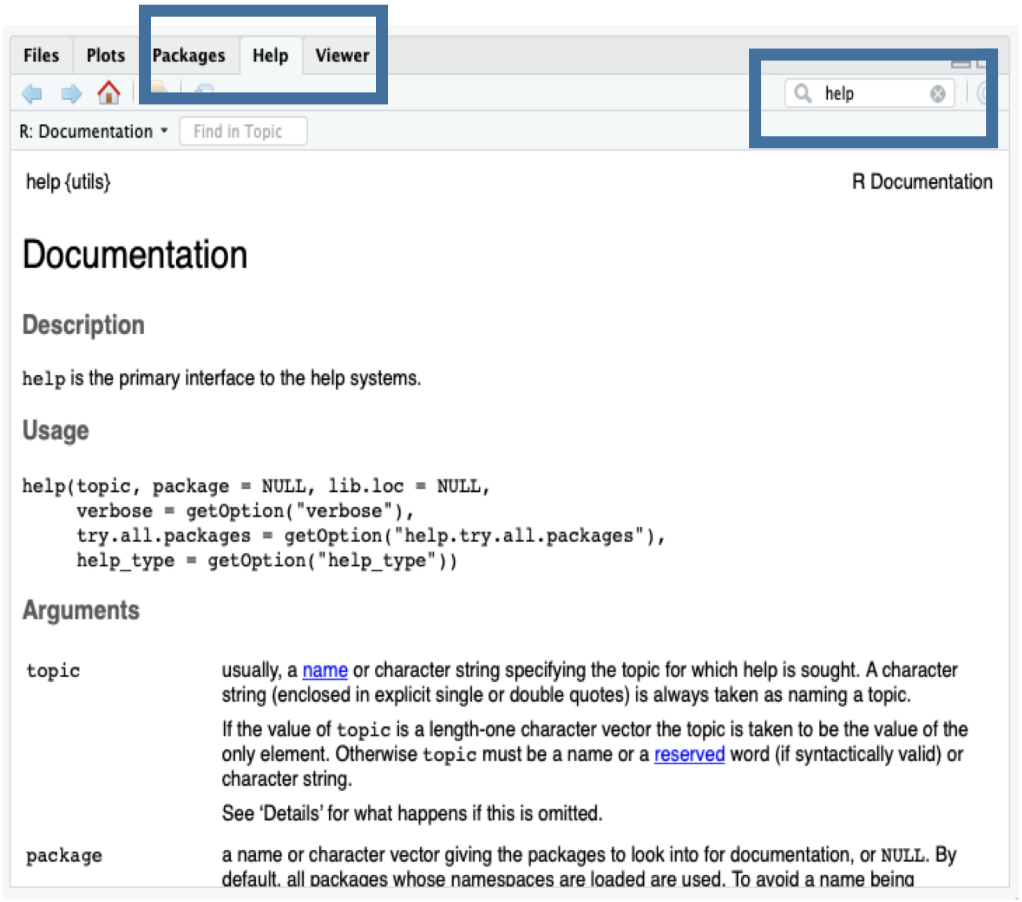
CINTESIS
Health. Research.

Getting Help

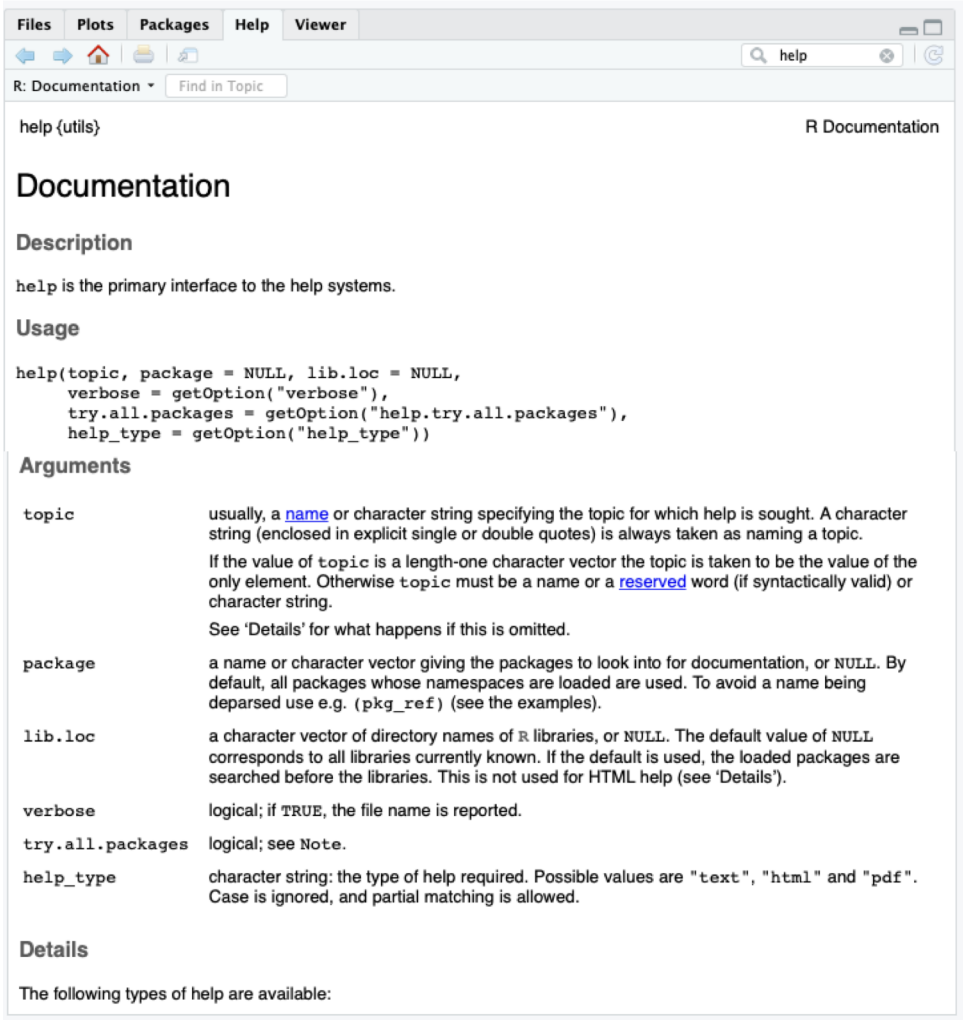
Help:

> help("mean")

> ?"mean"



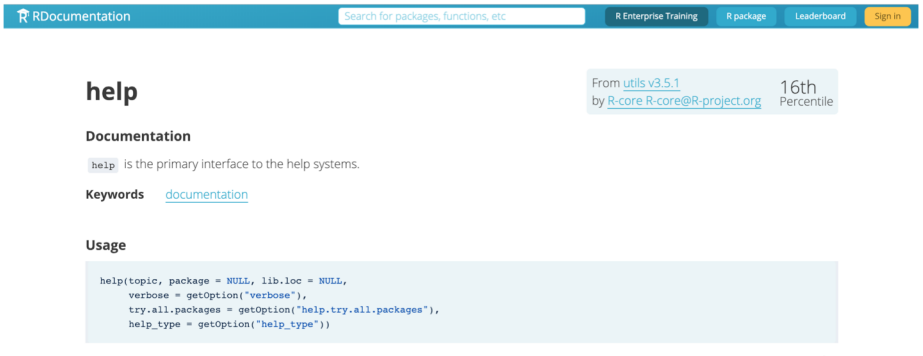
Getting Help

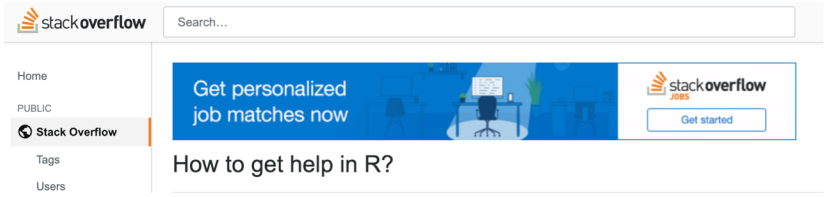


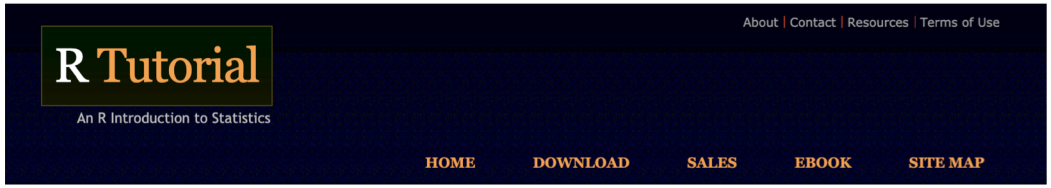
Examples

```
help()
help(help)              # the same

help(lapply)

help("for")             # or ?"for", but quotes/backticks are needed

try({# requires working TeX installation:
 help(dgamma, help_type = "pdf")
 ## -> nicely formatted pdf -- including math formula -- for help(dgamma):
 system2(getOption("pdfviewer"), "dgamma.pdf", wait = FALSE)
})

help(package = "splines") # get help even when package is not loaded

topi <- "women"
help(topi)

try(help("bs", try.all.packages = FALSE)) # reports not found (an error)
help("bs", try.all.packages = TRUE)       # reports can be found
                                          # in package 'splines'

## For programmatic use:
topic <- "family"; pkg_ref <- "stats"
help((topic), (pkg_ref))
```

Getting Help - internet







Arithmetic Operators

- In the Console (command line) :

```
> 10 + 20
[1] 30
> 3 * 10
[1] 30
> 2**3
[1] 8
> 2^3
[1] 8
```

```
> 4/3
[1] 1.333333
> 4%%3
[1] 1
> 5 + 2 * 3
[1] 11
> (5 + 2)* 3
[1] 21
```

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y)
x %/% y	integer division

Relational Operators

- To compare values:

```
> 2 == 2
[1] TRUE
> 2 == 3
[1] FALSE
> 2 != 2
[1] FALSE
> 5 > 4
[1] TRUE
```

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to

Logic Operators

- Boolean operations like AND, OR etc.

```
> !TRUE
[1] FALSE
> !0
[1] TRUE
> TRUE & FALSE
[1] FALSE
> TRUE | FALSE
 [1] TRUE
```

Operator	Description
!	Logical Not
x | y	x OR y
x & y	x AND y
x || y	Logical OR
x && y	Logical AND

Functions

- a set of statements organized to perform a specific task
- ***func_name*** **<- function (*input arguments*) { statement }**

- The arguments can be mandatory or optional.
- The optional arguments usually have a pre-defined default value.
- To enumerate the arguments:

sample(x, size, replace = FALSE, prob = NULL) **<-** sample takes a sample of the specified size from the elements of x using either with or without replacement.

```
# by positional matching.
s1 <-sample(15)
> s1
[1]  7  9 11  3  4  2 13 10  5 12  1 14  8 15  6
```

```
# by name of arguments.
s2 <-sample(15,replace=TRUE)
> s2
[1]  3  7  2  3  5 11  9  3  3  3  9  3  3  9 14
```

If statement

- ***if(cond) expr*** <- returns the value of the expression evaluated

```
> x <- 5
> if (x>0) print(”x is a positive  number”)
??????
```

```
>if (x<0) print("x is a negative number")
??????
```

- ***if() else***

```
> if (x>0) { print("x is a positive number")} else ("x is a non-positive number")
??????
```

```
> if (x<0) { print("x is a negative number")} else ("x is a non-negative number")
??????
```

If statement

- ***if(cond) expr*** <- returns the value of the expression evaluated

```
> x <- 5
> if (x>0) print(”x is a positive  number”)
[1] "x is a positive number”

>if (x<0) print("x is a negative number")
```

- ***if() else***

```
> if (x>0) { print("x is a positive number")} else ("x is a non-positive number")
[1] "x is a positive number"

> if (x<0) { print("x is a negative number")} else ("x is a non-negative number")
[1] "x is non-negative number"
```

if() else

```
> x <- 0

>if (x < 0) { print("x is a negative number")
} else if (x > 0) { print("x is a positive number")
} else print("x is zero")
??????
```

- ***ifelse(test, yes, no)*** <- returns a value with the same shape as test which is filled with elements selected from either yes or no depending on whether the element of test is TRUE or FALSE.

```
> x <- 10
> y <- ifelse(x>=5,yes = "high",no = "low")
> y
??????
> ifelse(x>=5,yes = ifelse(x>10,"high","medium"),no = "low")
??????
```

ifelse

```
> x <- 0

>if (x < 0) { print("x is a negative number")
} else if (x > 0) { print("x is a positive number")
} else print("x is zero")
[1] "x is zero”
```

- ***ifelse(test, yes, no)*** <- returns a value with the same shape as test which is filled with elements selected from either yes or no depending on whether the element of test is TRUE or FALSE.

```
> x <- 10
> y <- ifelse(x>=5,yes = "high",no = "low")
> y
[1] "high"
> ifelse(x>=5,yes = ifelse(x>10,"high","medium"),no = "low")
[1] "medium"
```

For loop

- ***for(var in seq) expr***

```
> a <- 0
> for (j in 1:10){
          if(j%%2==0) a <- a+j
}
> a
```

?????

```
> m <- matrix(1:10,nrow =5,ncol = 2)
> res = NULL
> for (i in 1:dim(m)[1]){
      for (j in i:dim(m)[1]){
           res <- c(res,m[i,1]*m[j,2])
       }
}
> res
```

?????

For loop

- ***for(var in seq) expr***

```
> a <- 0
> for (j in 1:10){
          if(j%%2==0) a <- a+j
}
> a
[1] 30
```

```
> m <- matrix(1:10,nrow =5,ncol = 2)
> res = NULL
> for (i in 1:dim(m)[1]){
      for (j in i:dim(m)[1]){
           res <- c(res,m[i,1]*m[j,2])
       }
}
> res
[1]  6  7  8  9 10 14 16 18 20 24 27 30
36 40 50
```

For loop

- For loop vs Vectorization

- When using for loops:
  - Using functions that don’t take vector arguments

  - Loops where each iteration is dependent on the results of previous iterations

While Loops

- **`while`**`(cond) expr` <- loop until a specific condition is met.

```
> errorLim <- 0.5
> e <- 0.75

> while (e > errorLim){

  e = e^2
}

> e
```

?????

While Loops

- **`while`**`(cond) expr` <- loop until a specific condition is met.

```
> errorLim <- 0.5
> e <- 0.75

> while (e > errorLim){

  e = e^2
}

> e

[1] 0.3164062
```

Next and Break

- **`next`** statement

```
> x<-0
> for (i in 1:10){
  if (i%%2==0) {
   next
  }
   x <- x +i
}
> x
??????
```

- **`break`** statement

```
> x<-0
> for (i in 1:10){
  if (i%%2==0) {
   break
  }
   x <- x +i
}
> x
??????
```

Next and Break

- **`next`** statement

```
> x<-0
> for (i in 1:10){
  if (i%%2==0) {
   next
  }
   x <- x +i
}
> x
[1] 25
```

- **`break`** statement

```
> x<-0
> for (i in 1:10){
  if (i%%2==0) {
   break
  }
   x <- x +i
}
> x
[1] 1
```

Data Structures

DIMENSION	CONTENT TYPE HOMOGENEOUS	CONTENT TYPE HETEROGENEOUS
1D	Atomic vector	List
2D	Matrix	Data frame
ND	Array	

- ***typeof(x)*** – returns the *type* of an R object.

Type	Description
"NULL"	NULL
"symbol"	a variable name
"logical"	a vector containing logical values
"integer"	a vector containing integer values
"double"	a vector containing real values
"complex"	a vector containing complex values
"character"	a vector containing character values
"expression"	an expression object
…	…

Access Structures

```
> data = read.csv(file.choose())
```

- Columns

#by index

```
> data[,5]
```

#by column name

```
> colnames(data)
> data[,"Age"]
```

#by logical

```
> data[,c(T,F,F,T,T,F,F,F,F)]
```

#double square bracket ***(data frames and lists)***

```
> data[["AdvEvent"]]
```

#by $ ***(data frames and Lists)***

```
> data$AdvEvent
```

- Rows

#by index

```
> data[3,]
```

#by rowname

```
> rownames(data)
> data["3",]
```

#by logical

```
> i = c(F,F,T,rep(F,384))
> data[i,]
```

Apply Functions

- ***apply(X, MARGIN, FUN, …)*** <- returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

- ! apply coerces everything into the same type. Numbers can become characters !

- ***lapply(X, FUN, …)***
- ***lapply*** <- returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.

- ***sapply*** <- is a wrapper of lapply by default returning a vector, matrix or array

- ***vapply*** <- is similar to sapply, but has a pre-specified type of return value

Apply Functions

- ***apply(X, MARGIN, FUN, …)*** <- returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

- ***MARGIN*** **–** for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns

- ! apply coerces everything into the same type. Numbers can become characters !

```
> A = data[,c("Age","Diastolic","Systolic","AdvEvent")]
> apply(A,2,mean)
          Age    Diastolic     Systolic     AdvEvent
 67.2739018   60.9307642 132.0470547    0.2816537
```

Apply Functions

```
> i = sapply(data,is.numeric)
> i
```

```
  Subject            Gender               Age       TypeSurgery        SurgStatus          Diastolic
    FALSE             FALSE              TRUE             FALSE             FALSE              TRUE
 Systolic Cross_Clamp_Time          AdvEvent
     TRUE              TRUE              TRUE
```

```
> sapply(data[,i],mean,na.rm = T)
```

```
              Age         Diastolic          Systolic Cross_Clamp_Time          AdvEvent
      67.2739018        60.9307642       132.0470547        72.0052083         0.2816537
```

Apply Functions

```
> lapply(data[,i],mean,na.rm=T)
$Age
[1] 67.2739

$Diastolic
[1] 60.93076

$Systolic
[1] 132.0471

$Cross_Clamp_Time
[1] 72.00521

$AdvEvent
[1] 0.2816537
```

For loop

```
> m <- NULL
> for (j in colnames(data)){
          if (is.numeric(data[,j]))     m <- c(m, mean(data[,j],na.rm=T))
}
> m
[1]  67.2739018  60.9307642 132.0470547  72.0052083   0.2816537
```

Factors

- ***factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)***
- ***levels(x)*** <- provides access to the levels attribute of a variable.

- Nominal

```
> levels(data$Gender)
[1] "Female" "Male"
```

- Ordinal

```
>levels(data$SurgStatus)
[1] "Elective" "Urgent"
```

```
> data$AdvEvent <- factor(data$AdvEvent,levels = c(0,1),labels=c("No","Yes"))
```

More functions - which

- ***which(x, arr.ind = FALSE, useNames = TRUE)***

```
> which(data$Gender=="Male")
```

```
> which(data[,"Age"]>90)
```

```
>length(which(data[,9]==1))
```

More Functions - str

- ***str(object, …)*** <- compactly display the structure of an R object

```
>str(data)
'data.frame':          387 obs. of  9 variables:
 $ Subject         : Factor w/ 387 levels "Sbj001","Sbj002",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Gender          : Factor w/ 2 levels "Female","Male": 2 2 1 2 2 2 2 2 2 1 ...
 $ Age             : int  50 67 71 51 69 78 70 58 80 63 ...
 $ TypeSurgery     : Factor w/ 3 levels "CAB","CAB + Valve",..: 3 1 1 3 1 2 1 3 3 2 ...
 $ SurgStatus      : Factor w/ 2 levels "Elective","Urgent": 1 1 2 1 2 2 1 2 1 2 ...
 $ Diastolic       : num  45.5 52.6 56.7 71.4 58.1 ...
 $ Systolic        : num  112.1 109 98.2 150.8 131.5 ...
 $ Cross_Clamp_Time: int  83 67 83 51 63 114 94 138 42 172 ...
 $ AdvEvent        : int  0 0 0 0 1 0 0 0 0 0 ...
```

More Functions - summary

- ***summary(object, …)*** <- produce result summaries of the results of various model fitting functions.

```
> summary(data)
 Subject       Gender          Age              TypeSurgery     SurgStatus    Diastolic
 Sbj001 :  1   Female:109   Min.   :24.00   CAB         :197   Elective:212   Min.   :26.42
 Sbj002 :  1   Male  :278   1st Qu.:60.00   CAB + Valve: 76   Urgent  :175   1st Qu.:53.28
 Sbj003 :  1                Median :67.00   Valve       :114                  Median :60.53
 Sbj004 :  1                Mean   :67.27                                     Mean   :60.93
 Sbj005 :  1                3rd Qu.:76.00                                     3rd Qu.:67.79
 Sbj006 :  1                Max.   :92.00                                     Max.   :94.35
 (Other):381
    Systolic      Cross_Clamp_Time    AdvEvent
 Min.   : 88.71   Min.   : 13.00   Min.   :0.0000
 1st Qu.:115.64   1st Qu.: 50.75   1st Qu.:0.0000
 Median :130.00   Median : 67.00   Median :0.0000
 Mean   :132.05   Mean   : 72.01   Mean   :0.2817
 3rd Qu.:145.63   3rd Qu.: 85.25   3rd Qu.:1.0000
 Max.   :193.75   Max.   :298.00   Max.   :1.0000
                  NA's   :3
```

More Functions - subset

- ***subset(x, subset, select,…)*** <- return subsets of vectors, matrices or data frames which meet conditions.

```
> data_E <- subset(data, SurgStatus=="Elective")
```

```
> head(data)
```

	Subject	Gender	Age	TypeSurgery	SurgStatus	Diastolic	Systolic	Cross_Clamp_Time	AdvEvent
1	Sbj001	Male	50	Valve	Elective	45.54094	112.1085	83	0
2	Sbj002	Male	67	CAB	Elective	52.59201	109.0264	67	0
4	Sbj004	Male	51	Valve	Elective	71.39448	150.7692	51	0
7	Sbj007	Male	70	CAB	Elective	54.80710	135.9122	94	0
9	Sbj009	Male	80	Valve	Elective	70.75592	123.5638	42	0
11	Sbj011	Male	74	CAB + Valve	Elective	74.83939	153.4632	95	1

More functions - by

- ***by(data, INDICES, FUN, …, simplify = TRUE)*** <- is an object-oriented wrapper for ***tapply*** applied to data frames.

```
> by(data,data$TypeSurgery, function(x){
 mean.age <- mean(x$Age)})
```

More functions - by

- ***by(data, INDICES, FUN, …, simplify = TRUE)*** <- is an object-oriented wrapper for ***tapply*** applied to data frames.

```
> by(data,data$TypeSurgery, function(x){
 mean.age <- mean(x$Age)})
data$TypeSurgery: CAB
[1] 66.28426
------------------------------------------------------------------------------
data$TypeSurgery: CAB + Valve
[1] 71.39474
------------------------------------------------------------------------------
data$TypeSurgery: Valve
[1] 66.23684
```

More functions - aggregate

- ***aggregate()*** <- splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

```
> data_a_g <- aggregate(A,by = list(data$Gender),FUN = mean)
```

```
> data_a_g
  Group.1      Age Diastolic Systolic  AdvEvent
1  Female 70.67890  57.24856 133.4530 0.2844037
2    Male 65.93885  62.37451 131.4958 0.2805755
```

Packages

- Check packages: `installed.packages()`
- Remove: `remove.packages()`
- Update: `update.packages()`
- Load: `library();package::function()`
- Unload: `detach()`
- Help: `package?x` and `help(package = "x").`

Exercises

Solutions

1. Open the database Anesthesia-BD.csv

```
> data = read.csv(file.choose()) ###load data
> str(data) ###see data structure
'data.frame':       387 obs. of  9 variables:
 $ Subject         : Factor w/ 387 levels "Sbj001","Sbj002",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Gender          : Factor w/ 2 levels "Female","Male": 2 2 1 2 2 2 2 2 2 1 ...
 $ Age             : int  50 67 71 51 69 78 70 58 80 63 ...
 $ TypeSurgery     : Factor w/ 3 levels "CAB","CAB + Valve",..: 3 1 1 3 1 2 1 3 3 2 ...
 $ SurgStatus      : Factor w/ 2 levels "Elective","Urgent": 1 1 2 1 2 2 1 2 1 2 ...
 $ Diastolic       : num  45.5 52.6 56.7 71.4 58.1 ...
 $ Systolic        : num  112.1 109 98.2 150.8 131.5 ...
 $ Cross_Clamp_Time: int  83 67 83 51 63 114 94 138 42 172 ...
 $ AdvEvent        : int  0 0 0 0 1 0 0 0 0 0 ...

> data$AdvEvent <- factor(data$AdvEvent, levels = c(0,1),labels=c("No","Yes"))
```

2. For the columns which contain numeric values, create a new summary table in which the rows are the mean, the standard deviation, and the median of each of the numeric columns.

```
> T <- sapply(data[,sapply(data,is.numeric)],function(x){

return(rbind(mean(x,na.rm = T),sd(x,na.rm = T),median(x,na.rm = T)))})

> rownames(T) = c("Mean","SD","Median")

> T

            Age Diastolic  Systolic Cross_Clamp_Time

Mean   67.27390  60.93076 132.04705         72.00521

SD     11.25574  11.35387  21.22864         32.04680

Median 67.00000  60.52591 130.00410         67.00000
```

3. Considering only the subjects who had CAB or Valve surgery, how many had adverse events? How many males and females in the two groups (having or not having adverse events)? (count and percentage)

```
> ind = which(data$TypeSurgery=="CAB" | data$TypeSurgery=="Valve" )

> table(data[ind,"AdvEvent"])

 No Yes
231  80

> prop.table(table(data[ind,"AdvEvent"]))
       No       Yes
0.7427653 0.2572347
```

3. Considering only the subjects who had CAB or Valve surgery, how many had adverse events? How many males and females in the two groups (having or not having adverse events)? (count and percentage)

```
> table(data[ind,"Gender"],data[ind,"AdvEvent"])
         No Yes
  Female  60  21
  Male   171  59

> prop.table(table(data[ind,"Gender"],data[ind,"AdvEvent"]))
                No        Yes
  Female 0.19292605 0.06752412
  Male   0.54983923 0.18971061
```

4.Merge the two datasets (Anesthesia-BD.csv and Anesthesia-BD2.csv) by the subject identification number.

```
> data2 = read.csv(file.choose()) ###load data Anesthesia-BD2.csv
> str(data2)
'data.frame':           363 obs. of  7 variables:
 $ SubjectID         : Factor w/ 363 levels "Sbj001","Sbj002",..: 3 25 30 32 36 39 46 56 60 63 ...
 $ Diabetes          : int  0 0 1 1 0 0 0 1 0 0 ...
 $ RFLastA1cLevel    : num  5.7 5.6 7.1 6.7 6.7 6.3 6 6.5 5.3 4.9 ...
 $ ChronicLungDisease: Factor w/ 4 levels "Mild","Moderate",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Hypertension      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ CHF               : int  0 1 1 1 1 1 1 0 1 1 ...
 $ Euroscore         : Factor w/ 330 levels "#N/A","0.501743134",..: 8 64 4 92 249 36 195 125 214 224 ...
> data2 = read.csv(file.choose(),na.strings = c("NA","#N/A"))
> str(data2)
'data.frame':           363 obs. of  7 variables:
 $ SubjectID         : Factor w/ 363 levels "Sbj001","Sbj002",..: 3 25 30 32 36 39 46 56 60 63 ...
 $ Diabetes          : int  0 0 1 1 0 0 0 1 0 0 ...
 $ RFLastA1cLevel    : num  5.7 5.6 7.1 6.7 6.7 6.3 6 6.5 5.3 4.9 ...
 $ ChronicLungDisease: Factor w/ 4 levels "Mild","Moderate",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Hypertension      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ CHF               : int  0 1 1 1 1 1 1 0 1 1 ...
 $ Euroscore         : num  0.608 1.144 0.558 1.338 3.07 ...
```

4.Merge the two datasets (Anesthesia-BD.csv and Anesthesia-BD2.csv) by the subject identification number.

A) The merge must be such that only the common subjects should be present in the final database (natural join).

```
> Ma = merge(data,data2,by.x = "Subject",by.y="SubjectID",all = F)
> dim(Ma)
[1] 363  15
```

B) The merge must be such that if some subject is not in one of the databases, the subject should be in the database, and missing information must be not available (full outer join).

```
> Mb = merge(data,data2,by.x = "Subject",by.y="SubjectID",all = T)
> dim(Mb)
[1] 387  15
```

Consider the following statement: "For people without diabetes, the normal range for the hemoglobin A1c level is between 4% and 5.6%. Hemoglobin A1c levels between 5.7% and 6.4% mean you have a higher chance of getting diabetes. Levels of 6.5% or higher mean you have diabetes."

5. Create a new variable with three factors (normal, prediabetes, and diabetes), taking into consideration the values the A1c levels. Compare the results obtained with the variable Diabetes (assuming that 1 means to have diabetes and 0 no diabetes).

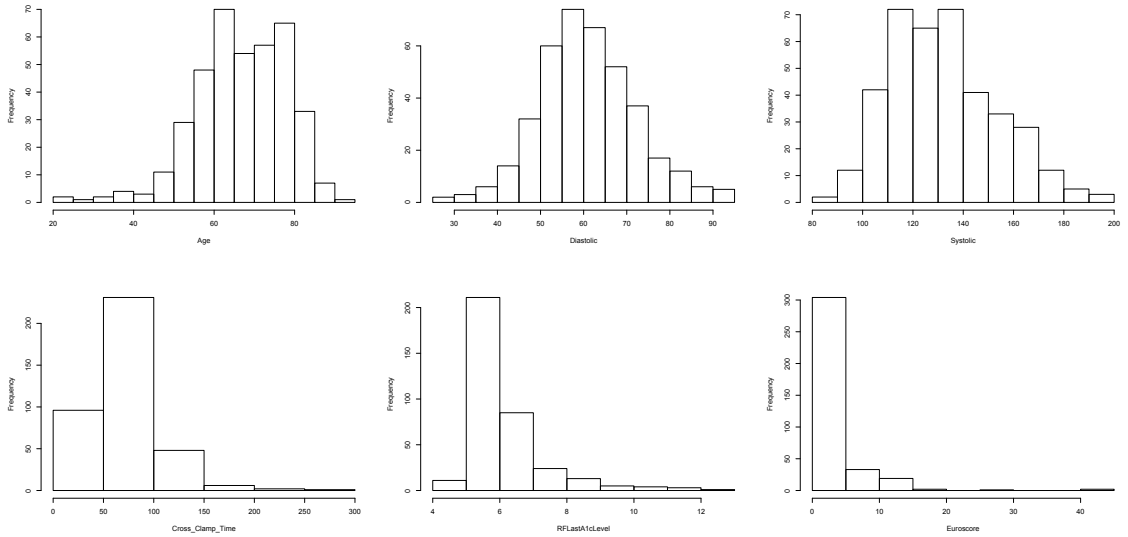
```
> diab <- ifelse(Mb$RFLastA1cLevel<5.7,0,ifelse(Mb$RFLastA1cLevel<6.4,1,2))

> Mb$Diabetes2 <- factor(x = diab,levels = 0:2, labels =
c("Normal","PreDiabetes","Diabetes"))

> Mb$Diabetes <- factor(Mb$Diabetes,levels=0:1,labels=c("Normal","Diabetes"))
> table(Mb$Diabetes,Mb$Diabetes2)

           Normal PreDiabetes Diabetes
  Normal      120          99       11
  Diabetes      7          35       83
```

6. Create a table with the comparison between the groups having or not having adverse events for all the variables available in the combined database. Use the appropriate measures for each type of variable.

```
> str(Mb)
'data.frame':           387 obs. of  16 variables:
 $ Subject           : Factor w/ 387 levels "Sbj001","Sbj002",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Gender            : Factor w/ 2 levels "Female","Male": 2 2 1 2 2 2 2 2 2 1 ...
 $ Age               : int  50 67 71 51 69 78 70 58 80 63 ...
 $ TypeSurgery       : Factor w/ 3 levels "CAB","CAB + Valve",..: 3 1 1 3 1 2 1 3 3 2 ...
 $ SurgStatus        : Factor w/ 2 levels "Elective","Urgent": 1 1 2 1 2 2 1 2 1 2 ...
 $ Diastolic         : num  45.5 52.6 56.7 71.4 58.1 ...
 $ Systolic          : num  112.1 109 98.2 150.8 131.5 ...
 $ Cross_Clamp_Time  : int  83 67 83 51 63 114 94 138 42 172 ...
 $ AdvEvent          : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 1 1 1 1 1 ...
 $ Diabetes          : Factor w/ 2 levels "Normal","Diabetes": 2 1 1 1 1 1 1 1 1 2 ...
 $ RFLastA1cLevel    : num  7.9 5.5 5.7 6 5.8 NA 5.9 6 5.2 11.2 ...
 $ ChronicLungDisease: Factor w/ 4 levels "Mild","Moderate",..: 3 3 1 3 3 3 3 3 3 3 ...
 $ Hypertension      : int  1 1 1 0 1 0 1 1 1 0 ...
 $ CHF               : int  1 0 0 0 0 1 1 1 1 1 ...
 $ Euroscore         : num  0.951 2.383 0.608 4.648 10.599 ...
 $ Diabetes2         : Factor w/ 3 levels "Normal","PreDiabetes",..: 3 1 2 2 2 NA 2 2 1 3 ...
```

6. Create a table with the comparison between the groups having or not having adverse events for all the variables available in the combined database. Use the appropriate measures for each type of variable.

```
> Mb$Hypertension <- factor(Mb$Hypertension,levels=0:1,labels=c("No","Yes"))
> Mb$CHF <- factor(Mb$CHF,levels=0:1,labels=c("No","Yes"))
> Mb$Subject <- as.character(Mb$Subject)

> quartz() #windows()
> par(mfrow=c(2,3))
>for (i in which(sapply(Mb,is.numeric))){
  hist(Mb[,i],xlab=colnames(Mb)[i])
}
```



6. Create a table with the comparison between the groups having or not having adverse events for all the variables available in the combined database. Use the appropriate measures for each type of variable.

```
> table <- by(Mb[,which(sapply(Mb,is.numeric))],INDICES = Mb$AdvEvent,FUN = function(x)
apply(x,2,quantile,na.rm=T))
> table
Mb$AdvEvent: No
     Age Diastolic  Systolic Cross_Clamp_Time RFLastA1cLevel  Euroscore
0%    25  26.42047  88.70903               15            4.7  0.5017431
25%   59  53.30694 115.03977               51            5.6  1.1325681
50%   67  60.75382 129.50300               68            5.8  1.9099247
75%   76  68.64874 143.12238               83            6.4  3.6744213
100%  92  94.35495 180.95547              241           12.2 42.7481191
------------------------------------------------------------------------
Mb$AdvEvent: Yes
     Age Diastolic  Systolic Cross_Clamp_Time RFLastA1cLevel  Euroscore
0%    24  39.42266  92.34727             13.0          4.600  0.5546841
25%   62  52.93295 119.55877             48.0          5.500  1.1644286
50%   69  59.94571 131.48461             64.0          5.800  1.8710907
75%   77  65.97349 153.50990             88.5          6.425  3.7733095
100%  88  92.51766 193.75080            298.0         11.700 42.9068563
```

6. Create a table with the comparison between the groups having or not having adverse events for all the variables available in the combined database. Use the appropriate measures for each type of variable.

```
> table$pvalue <- apply(Mb[,which(sapply(Mb,is.numeric))],2,function(x)
wilcox.test(x[which(Mb$AdvEvent=="No")],x[which(Mb$AdvEvent=="Yes")])$p.value)

> table
```

	Age	Diastolic	Systolic	Cross_Clamp_Time	RFLastA1cLevel	Euroscore
0%	25	26.42047	88.70903	15	4.7	0.5017431
25%	59	53.30694	115.03977	51	5.6	1.1325681
50%	67	60.75382	129.50300	68	5.8	1.9099247
75%	76	68.64874	143.12238	83	6.4	3.6744213
100%	92	94.35495	180.95547	241	12.2	42.7481191

	Age	Diastolic	Systolic	Cross_Clamp_Time	RFLastA1cLevel	Euroscore
0%	24	39.42266	92.34727	13.0	4.600	0.5546841
25%	62	52.93295	119.55877	48.0	5.500	1.1644286
50%	69	59.94571	131.48461	64.0	5.800	1.8710907
75%	77	65.97349	153.50990	88.5	6.425	3.7733095
100%	88	92.51766	193.75080	298.0	11.700	42.9068563

Age	Diastolic	Systolic	Cross_Clamp_Time	RFLastA1cLevel	Euroscore
0.09297764	0.41375034	0.09079886	0.76575185	0.78804579	0.82851108

6. Create a table with the comparison between the groups having or not having adverse events for all the variables available in the combined database. Use the appropriate measures for each type of variable.

```
> table2 <- by(Mb[,which(sapply(Mb,is.factor))], INDICES = Mb$AdvEvent, FUN =
function(x){sapply(x,table)})
```