STAT9006: Data Manipulation with *R*

CIT CORK INSTITUTE OF TECHNOLOGY

INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

MATHEMATICS

Outline

1. Indexing
2. Sorting
3. Ordering a Dataframe and Renaming a Variable
4. Data types
5. Grouping a variable

Outline

6. Merging Rows/Columns

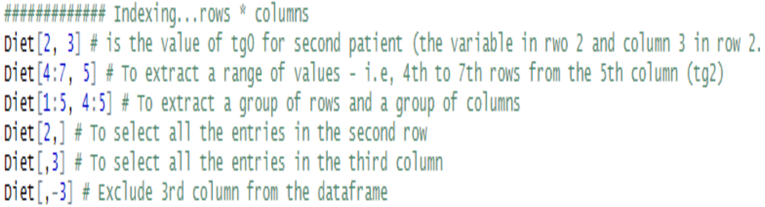
7. Missing Values

8. Outliers

9. Filtering

Indexing

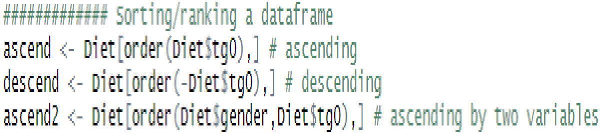
A dataframe/tibble is a two dimensional object consisting of **rows and columns**. The rows are referred to by the first (left-hand) subscript and the columns by the second (right-hand) subscript.



```
############ Indexing...rows * columns
Diet[2, 3] # is the value of tg0 for second patient (the variable in rwo 2 and column 3 in row 2.
Diet[4:7, 5] # To extract a range of values - i.e, 4th to 7th rows from the 5th column (tg2)
Diet[1:5, 4:5] # To extract a group of rows and a group of columns
Diet[2,] # To select all the entries in the second row
Diet[,3] # To select all the entries in the third column
Diet[,-3] # Exclude 3rd column from the dataframe
```

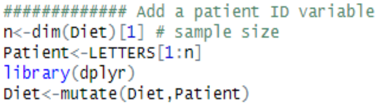
Sorting

Variables can be sorted using a vector of choice. For example:



```
############# Sorting/ranking a dataframe
ascend <- Diet[order(Diet$tg0),] # ascending
descend <- Diet[order(-Diet$tg0),] # descending
ascend2 <- Diet[order(Diet$gender,Diet$tg0),] # ascending by two variables
```

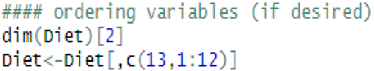
Before sorting data it is important that there is a subject ID that allows you to return data to its original form.



```
############# Add a patient ID variable
n<-dim(Diet)[1] # sample size
Patient<-LETTERS[1:n]
library(dplyr)
Diet<-mutate(Diet,Patient)
```

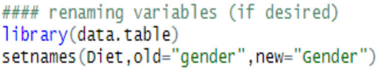
Ordering a Dataframe and Renaming a Variable

Ordering



```
#### ordering variables (if desired)
dim(Diet)[2]
Diet<-Diet[,c(13,1:12)]
```

Renaming a variable



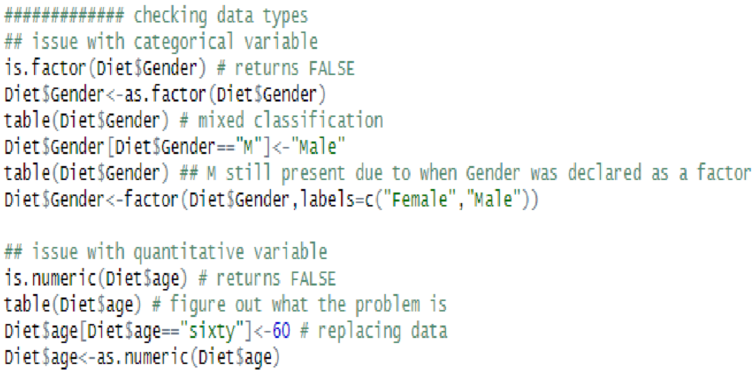
```
#### renaming variables (if desired)
library(data.table)
setnames(Diet,old="gender",new="Gender")
```

Data types

Types of variables encountered in *RStudio* are often:

- *int* stands for integers;
- *dbl* stands for doubles, or real numbers;
- *chr* stands for character vectors, or strings;
- *dttm* stands for date-times (a date + a time).

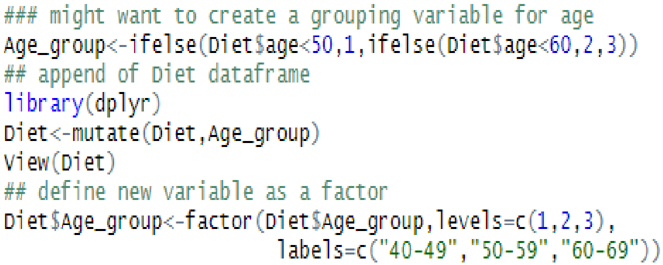
Data types



```
############# checking data types
## issue with categorical variable
is.factor(Diet$Gender) # returns FALSE
Diet$Gender<-as.factor(Diet$Gender)
table(Diet$Gender) # mixed classification
Diet$Gender[Diet$Gender=="M"]<-"Male"
table(Diet$Gender) ## M still present due to when Gender was declared as a factor
Diet$Gender<-factor(Diet$Gender,labels=c("Female","Male"))

## issue with quantitative variable
is.numeric(Diet$age) # returns FALSE
table(Diet$age) # figure out what the problem is
Diet$age[Diet$age=="sixty"]<-60 # replacing data
Diet$age<-as.numeric(Diet$age)
```

Grouping a variable

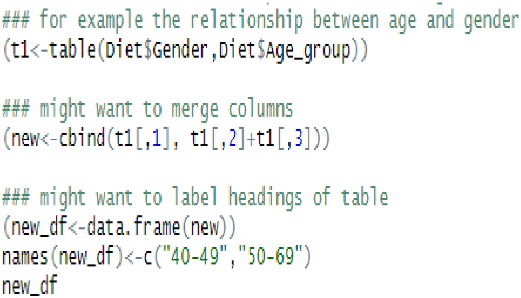
In some instances you might like to group a continuous measure - e.g., Age from the Diet dataframe. When you create a new variable, you will then want to append the new variable to the original dataframe.



```
### might want to create a grouping variable for age
Age_group<-ifelse(Diet$age<50,1,ifelse(Diet$age<60,2,3))
## append of Diet dataframe
library(dplyr)
Diet<-mutate(Diet,Age_group)
View(Diet)
## define new variable as a factor
Diet$Age_group<-factor(Diet$Age_group,levels=c(1,2,3),
                       labels=c("40-49","50-59","60-69"))
```

Merging Rows/Columns

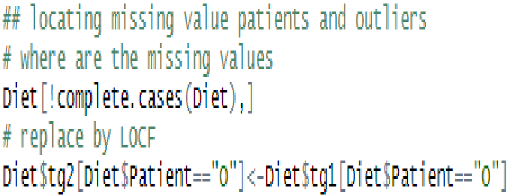
In some instances you might like to merge rows and/or columns. For example:



```
### for example the relationship between age and gender
(t1<-table(Diet$Gender,Diet$Age_group))

### might want to merge columns
(new<-cbind(t1[,1], t1[,2]+t1[,3]))

### might want to label headings of table
(new_df<-data.frame(new))
names(new_df)<-c("40-49","50-69")
new_df
```

Missing Values

- One important feature of *R* that can make comparison tricky are missing values, or **NA** (not available).
- Missing values are *contagious*: almost any operation involving an unknown value will also be unknown.
- To isolate missing cases !*complete*.*cases*() can be useful.

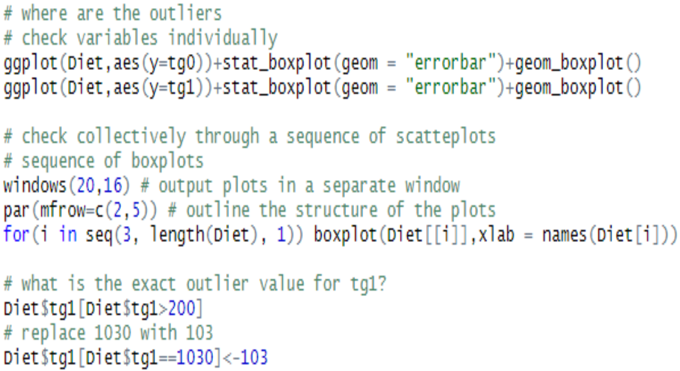


```
## locating missing value patients and outliers
# where are the missing values
Diet[!complete.cases(Diet),]
# replace by LOCF
Diet$tg2[Diet$Patient=="0"]<-Diet$tg1[Diet$Patient=="0"]
```

Outliers

- An outlier is a data point that d**iffers significantly from other observations**.
- An outlier may be due to variability in the measurement or it may indicate **experimental error**. The latter are sometimes excluded from the data set.
- An outlier **may cause serious problems** in statistical analyses and should always be investigated before proceeding with the statistical analysis.
- Descriptive statistics (numerical and/or graphical) can be used to **isolate outliers**.

Outliers

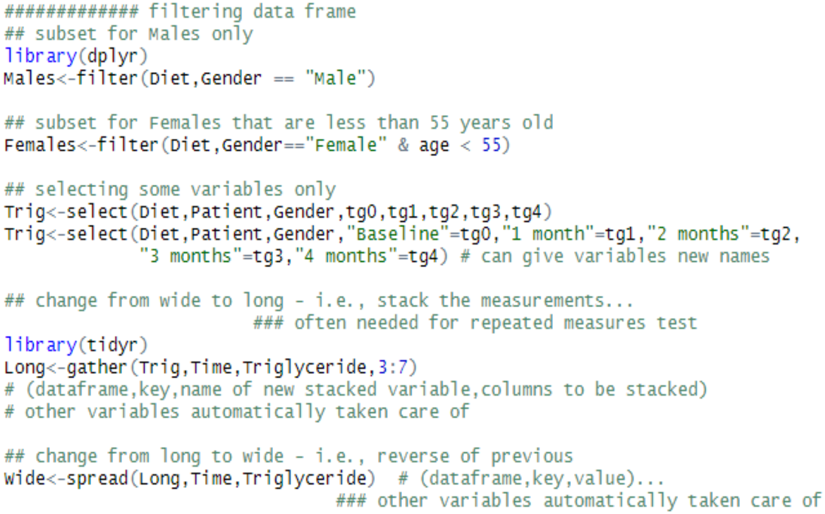


```
# where are the outliers
# check variables individually
ggplot(Diet,aes(y=tg0))+stat_boxplot(geom = "errorbar")+geom_boxplot()
ggplot(Diet,aes(y=tg1))+stat_boxplot(geom = "errorbar")+geom_boxplot()

# check collectively through a sequence of scatteplots
# sequence of boxplots
windows(20,16) # output plots in a separate window
par(mfrow=c(2,5)) # outline the structure of the plots
for(i in seq(3, length(Diet), 1)) boxplot(Diet[[i]],xlab = names(Diet[i]))

# what is the exact outlier value for tg1?
Diet$tg1[Diet$tg1>200]
# replace 1030 with 103
Diet$tg1[Diet$tg1==1030]<-103
```

Filtering

- To use filtering effectively, you have to know how to select the observations that you want using the comparison operators.
- *R* provides the standard suite: $>, >=, <, <=, ! =$ (not equal), and $==$ (equal).
- When starting out with *R*, the easiest mistake to make is to use $=$ instead of $==$ when testing for equality. When this happens an **informative error** appears.
- Multiple arguments to *filter()* can be combined using Boolean/logical operators:
  - & is *and*
  - $|$ is *or*
  - ! is *not*

Filtering



```
############# filtering data frame
## subset for Males only
library(dplyr)
Males<-filter(Diet,Gender == "Male")

## subset for Females that are less than 55 years old
Females<-filter(Diet,Gender=="Female" & age < 55)

## selecting some variables only
Trig<-select(Diet,Patient,Gender,tg0,tg1,tg2,tg3,tg4)
Trig<-select(Diet,Patient,Gender,"Baseline"=tg0,"1 month"=tg1,"2 months"=tg2,
             "3 months"=tg3,"4 months"=tg4) # can give variables new names

## change from wide to long - i.e., stack the measurements...
                        ### often needed for repeated measures test
library(tidyr)
Long<-gather(Trig,Time,Triglyceride,3:7)
# (dataframe,key,name of new stacked variable,columns to be stacked)
# other variables automatically taken care of

## change from long to wide - i.e., reverse of previous
Wide<-spread(Long,Time,Triglyceride)  # (dataframe,key,value)...
                                ### other variables automatically taken care of
```