Statistical Computing
Master Data Science
Winter Semester 2019/20

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Prof. Tim Downie

Workshop 3
Basics summary statistics and box plots
Matrices *with solutions*

1 Introduction

In this workshop you will learn how to do obtain summary statisics for data in a data frame. This includes the measures of location and quantiles covered in Lecture 3. Section 3 is concerned with box plots and strip charts from the lecture. Section 4 covers how matrices are handled in R, a subject not included in the lectures, but the subject is still examinable!

Section 6 has three exercises as home work, for you to consolidate the subjects in the lecture and workshop

Preliminaries

► Open a new word document (or similar). ► Start R-Studio using the Icon on the desktop
► *Strg* + *Shift* + *N* opens a new R script.
► Type the comment

```
#Statistical Computing: Workshop 3 -
#Basic Statistics and exploratory data I
```

as a title to the script file.
► Save the file in your `H:\\StatComp` folder with the name `Workshop3.R` using *File > Save as*
► Set your *working directory* to be `H:\\StatComp`. The code to do this is

```
> setwd(H://StatComp)
```

By now it is possible that you have accidentally saved your workspace when leaving R-Studio.
► If your Environment Window is displaying any objects then clear the workspace using
Session > clear workspace.

Work through the examples as you did last week. In many of the code examples the output is not included, you should always read and understand the output. Please add comments (in your own words) to your

code so that it will make sense to you when you return several weeks (or years) later.

When there is a question in the text e.g. *"How many students are included in the data set?"* , write the answer to the question in the word document. You can also copy and paste your graphics into this document.

2 The Prestige data set and basic summary statistics

When analysing data you will usually start by carrying out some simple procedures to get a feel for the data. Typically, we will produce some appropriate plots and calculate some useful summary statistics.

In this section we will be using the Canadian **`Prestige`** data set which contains the following variables:

`occupation` 102 occupation groups as defined by the Canadian Census Bureau in 1971.

`education` Average education of occupational group, years, in 1971.

`income` Average income of group, dollars, in 1971.

`women` Percentage of group who are women.

`prestige` "Pineo-Porter prestige score" for occupation, from a social survey conducted in the mid-1960s.

`census` Canadian Census occupational code.

`type` Type of occupation. A factor with levels: `bc`, Blue Collar; `prof`, Professional, Managerial, and Technical; `wc`, White Collar.

Notice that there is a variable called `prestige` in the data frame called `Prestige`!

The data is directly available in R, from within the "`car`" package. This is pre-installed on the Lab-PCs. To load the package use

```
> require(car)
```

If this command caused an error:

Then you need to install the package:

```
> install.packages("car")
```

And follow the instructions. Once the installation has worked, then you need to load the library into R.

```
> require(car)
```

Installing packages only needs to be done once.

To make the data available, type

```
> data(Prestige)
```

2.1 Summarising numeric variables

First of all we should find out how many rows and columns our data set has

```
> dim(Prestige)
```

To find the names of the variables type:

```
> names(Prestige)
```

A quick method of seeing what the data type for each variable is, is

```
> str(Prestige)
```

The basic commands to summarise the numeric variables in the data set are straightforward.

`mean()`	arithmetic mean
`median()`	median
`sd()`	standard deviation
`var()`	variance
`quantile()`	quantile

You will learn about the standard deviation and variance next week, but the measure how spread out the data values are.

```
> mean(Prestige$income)
> median(Prestige$income)
> sd(Prestige$income)
> var(Prestige$income)
> sqrt(var(Prestige$income))
```

The units of the mean, median and standard deviation are the same as the measured variable e.g. Canadian dollars. The units of the variance are squared units e.g. dollars^2. In terms of understanding the numbers, the standard deviation is much more helpful than the variance.

These functions all have arguments that control the treatment of missing values. These arguments are there for convenience and readability they're not really necessary. For example,
`mean(x, na.rm=TRUE)` is just a convenient and more readable way of expressing
`mean(x[!is.na(x)])`.

Exercise: Try this yourself. Define a vector `x` with the following definition
`> x<-c(1:5,NA,10:15)`
investigate the latter command by inspecting the following

```
> is.na(x)
> !is.na(x)
> x[!is.na(x)]
```

The first command returns a logical vector. What does the operator `!` do to a logical vector? Verify that both the following commands give the same result.

```
> mean(x, na.rm=TRUE)
> mean(x[!is.na(x)])
```

We may also want to compute the quantiles of a set of numeric data. The command is `quantile`. For example, to obtain the *lower* and *upper quartiles* of income:

```
> quantile(Prestige$income,probs=c(0.25,0.75))
```

If you read about another type of *summary statistic*, there is probably a function in R or in a package to do calculate it.

Exercise:

(a) What is the mean and standard deviation of the prestige Scores?

(b) What is the median of the census codes? Does this statistic even make sense?

2.2 Summarising categorical variables

The `type` and `code` variables in our data frame `Prestige` represent categorical variables (i.e. those that take values in a discrete set of non-ordered categories). In this particular case, `code` is represented by numbers, even though the levels do not correspond to amounts. The variable can be analysed as if it is numeric even if this is not sensible (see that last exercise).

The way to summarise categorical variables is in a frequency table, i.e. count the number of times each value occurs:

```
> table(Prestige$type)
```

To find the modal value of a factor variable the quickest method is to obtain the frequency table and read

of which is the largest value. What is the mode of `type`?

The R Command to find the mode is to obtain which position in the table has the highest frequency:

```
> which.max(table(Prestige$type))
```

Add up the counts for `bc`, `prof` and `wc` in the frequency table. How many rows does the data frame have? Why are these two numbers not the same?

There are some `NA`s in `type`. If we want to include these in the table use:

```
> table(Prestige$type,useNA="always")
```

If you want the proportions then we can use the function `prop.table`. Note that `prop.table` expects an object of class `table` as input, which means that most of the time we use the slightly odd expression `prop.table(table(...))`, e.g..

```
> prop.table(table(Prestige$type,useNA="always"))
```

2.3 The `summary` command

One of the most useful commands in R is `summary`. Try it:

```
> summary(Prestige$income)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    611    4106    5930    6798    8187   25879
```

Since *income* is a numeric variable, you get a standard 6-number summary of the data (minimum, lower quartile, median, mean, upper quartile and maximum). Such a summary would be unsuitable for a categorical variable, so:

```
> summary(Prestige$type)
  bc prof   wc NA's
  44   31   23    4
```

Notice that the `summary` command automatically reports missing values. The row names contain the types of job, so

```
> row.names(Prestige)[is.na(Prestige$type)]
```

returns the 4 jobs, for which the value of `type` is missing.

The `summary` command can be applied to almost any R object. It is an example of a generic function this means that it does different things depending on the `class` of the object. At present we're only interested in simple summaries of data objects. Another important use of the command is to summarise the results of fitting a regression model. More on this later in the course.

Exercise: What does summary do if you give it a data frame as an argument? Try it and find out.

3 Box plots and strip charts

Box plots

A box plot (a.k.a. box-and-whisker plot) is used to compare a numeric variable across several groups in a data frame. The exact format of a box plot differs between packages; in R the box is defined by the lower quartile, median and upper quartile, while the "whiskers" extend to the data point that is no more than 1.5 times the interquartile range from the box. Any observation outside the whiskers is defined as an *extreme value* (or a box plot outlier) and is shown as a point.

```
> boxplot(prestige ~ type, data = Prestige, xlab="job type"
+ col = "lightblue",pch=16)
```

Notice the strange use of the tilde sign (~) in: `prestige ~ type`. This is called an R *model formula*. Model formulae are common in model fitting, and the details are not important now; you will come across them later in the course. For now, read the formula as "`prestige` *depends on* `type`" using the data frame `Prestige`.

What do you notice about the ordering of the three values in the variable `type`?

A more sensible ordering of these values would be `bc`, `wc` and `prof`. To change this ordering, redefine the factor variable with the levels specified as an argument.

```
> factor(Prestige$type,levels=c("bc","wc","prof"))
> Prestige$newtype<-factor(Prestige$type,levels=c("bc","wc","prof"))
> boxplot(prestige ~ newtype, data = Prestige, xlab="job type",
+ col = "lightblue",pch=16)
```

The second command stores the reordered levels in the variable `newtype` within the `Prestige` data frame. The data have not been reordered only the names of the levels have been reordered.

Warning: Box plots are often great for investigating or summarising data, but at other times they can be very misleading. Things to be wary of are: when there are very many observations in each box, very few observations in each box or the number of observations for each group are quite different. Sometimes one or two extremely *extreme values* can compress the boxes into a small part of the vertical axis, making visual interpretation difficult.

Strip charts

If there are too few observarions, a strip chart is often better

```
> stripchart(prestige ~ newtype, data = Prestige, ylab="job type")
> stripchart(prestige ~ newtype, data = Prestige, ylab="job type",
```

```
+ method="stack")
```

The second example stacks identical values (ties) on top of each other

4 Matrices in R

A matrix is a two dimensional "table" of numbers. In mathematical notation for a matrix is

$$M = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}.$$

M is a matrix with 2 rows and 3 columns. $M_{1,3}$ is the element of M in the 1st row and 3rd column and equals 5.

The R code for this uses the `matrix()` function:

```
> M <- matrix(1:6,nrow=2)
> M
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Note that the vector `1:6`=$(1, 2, 3, 4, 5, 6)$ is assigned to the matrix by column. `matrix()` takes an option called `byrow=TRUE` if you want to assign the values by row.

```
> M <- matrix(1:6,nrow=2,byrow=TRUE)
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

You specify the number of rows and/or the number of columns using `nrow` or `ncol`, you probably want to supply at least one of them. `dim()` returns the number of rows and columns of a matrix:

```
> M <- matrix(1:6,nrow=2)
> M
> dim(M)
> M <- matrix(1:6,ncol=3)
> M
> dim(M)
> M <- matrix(1:6)
> M
> dim(M)
```

Subsetting a Matrix A subset of elements in a vector can be accessed using square brackets. An example from Week 1 was:

```
> fibonacci<-c(1,1,2,3,5,8,13,21,34,55)
> fibonacci[7]
```

The elements of a matrix are identified by the row and the column, so the square bracket notation for a matrix requires two indices separated by a comma `[.,.]`. The row index always comes before the column index.

```
> M <- matrix(1:6,nrow=2)
> M[1,3]
[1] 5
```

You can specify multiple rows and/or columns:

`> M[1:2, 1]` : 1st and 2nd rows, 1st column
`> M[2, c(1,3)]` : 2nd row, 1st and 3rd columns
`> M[1:2, c(1,3)]` : 1st and 2nd rows, 1st and 3rd columns

The subsetting of vectors, matrices, arrays and data frames is a very useful tool and we will be using it a lot.

To get one complete row of a matrix omit the term after the comma:

```
> M[2, ]
```

To get one complete column of a matrix omit the term before the comma:

```
> M[,3]
```

Note that in the last example R returns the column as a vector. This is usually what one wants so is the default behaviour.

To drop one row of a matrix use a minus sign in the row field:

`> M[-2, ]` : all rows except the 2nd, all columns

To drop one column of a matrix use a minus sign in the column field:

`> M[, -1]` : all rows, all columns except the 1st

You can even specify the rows or columns using an R-object `> x<-c(1,3)`

`> M[,x]` : all rows, 1st and 3rd columns.

Matrix arithmetic Matrices have their special arithmetic rules. Addition of two matrices is as one would expect:

$$\text{If } A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \text{ and } B = \begin{bmatrix} 5 & 7 & 9 \\ 6 & 8 & 10 \end{bmatrix}, \text{ then } A + B = \begin{bmatrix} 6 & 10 & 14 \\ 8 & 12 & 16 \end{bmatrix}.$$

Which is what + in R does:

```
> A <- matrix(1:6,nrow=2)
> dim(A)
> B <- matrix(5:10,nrow=2)
> dim(B)
> A+B
     [,1] [,2] [,3]
[1,]    6   10   14
[2,]    8   12   16
```

Multiplication of a scalar and a matrix is also straightforward:

$$2A = \begin{bmatrix} 2 & 6 & 10 \\ 4 & 8 & 12 \end{bmatrix}.$$

```
> 2*A
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    4    8   12
```

However, **the multiplication of two matrices follows special rules**:

$$\text{If } A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \text{ and } C = \begin{bmatrix} 5 & 8 \\ 6 & 9 \\ 7 & 10 \end{bmatrix}, \text{ then } AC = \begin{bmatrix} 58 & 85 \\ 76 & 112 \end{bmatrix}.$$

In R `A*B` performs elementwise multiplication and `A%*%B` performs the matrix multiplication used in mathematics.

```
> A <- matrix(1:6,nrow=2)
> B <- matrix(5:10,nrow=2)
> C <- matrix(5:10,nrow=3)
> A*B
     [,1] [,2] [,3]
[1,]    5   21   45
[2,]   12   32   60
> dim(A)
[1] 2 3
> dim(C)
[1] 3 2
> A%*%C
     [,1] [,2]
[1,]   58   85
[2,]   76  112
>
```

`t` transposes a matrix and `solve` computes the inverse.

```
> t(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> D <- matrix(1:4,nrow=2)
> solve(D)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> D%*%solve(D)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

There are many more functions which apply to matrices, you will learn them when they are needed.

Comparing data frames and matrices

In the Further Reading document, the concept of a data frame was introduced. It is easy to think that a data frame is a type of matrix because it stores data in a rectangular format. There are similarities, but they are not the same:

- All entries in a *matrix* must be numeric, otherwise matrix addition and multiplication won't work.
- A *data frame* consists of a number of variables and each variable is represented as a column in the data frame. The length of each variable is equal.
- A *data frame* can have a mix of "data types": numeric, factor, logical or character. All the elements of one variable must have the same "data type".

The main similarity between a matrix and a data frame is that they are both represented as rows and columns and the `[.,.]` notation can be used for both.

Example

```
> data(iris)
> iris[10,]  #returns the 10th row of the iris data frame
> iris[1:10,4] #first 10 rows of the 4th variable in iris
> iris[1:10,"Species"] #first 10 rows of the Species variable
```

The following three commands all give the same output

```
> iris[1:10,4]
> iris[1:10,"Petal.Width"]
> iris$Petal.Width[1:10]
```

► In the *Environment Window* top left you can see a list of all objects you have used today, one of which is called `iris`. On the right hand side of this row there is an icon symbolising a data matrix. Click on this icon Click on this icon. Notice that there are 4 continuous variables and one nominal variable. The R function `str()` outputs a summary of the object, which for a data frame includes the variable type.

```
> str(iris)
```

5 Tidying up

Make sure your script file has sensible comments. At least one comment for each main section. Try to tidy up your code a bit. If you entered a command just trying something out, but which has little to do with that section then delete the non-essential commands.

► Save the script file (source file) again: Strg + s or *File* > *Save as*.

► Leave RStudio by typing the command:

```
> q()
```

When R asks you *Save workspace image ...?*, click on **Don't save**!

► Have a great weekend!

6 Homework exercises

Exercise 1 Linear transformation of a mean and median

In Europe temperatures are usually measured in Celsius.
In the USA temperatures are usually measured in Fahrenheit.

To convert a temperature from Celsius x into Fahrenheit y use the linear transformation formula

$$y = 1.8x + 32.$$

Suppose you have data for the maximum temperatures in Berlin between March and May, and have already calculated the mean to be 15.2C and the median to be 14.5C.

Calculate the mean median temperatures for the data in Fahrenheit. Note that you do not need the original data to calculate these.

Mean $\overline{y} = 1.8\overline{x} + 32 = 59.36$
Median $y_{0.5} = 1.8x_{0.5} + 32 = 58.1$

Exercise 2 Box plot

Shortly after introducing the metric system into Australia in the 1960s each of 44 Students was asked to estimate, to the nearest meter, the width of the leacture hall in which they were sitting. The true width was 13.1 m. The data were collected by Professor T. Lewis (A Handbook of Small Data Sets by David J. Hand, Fergus Daly, K. McConway, D. Lunn, E. Ostrowski, 1994)

The data are:

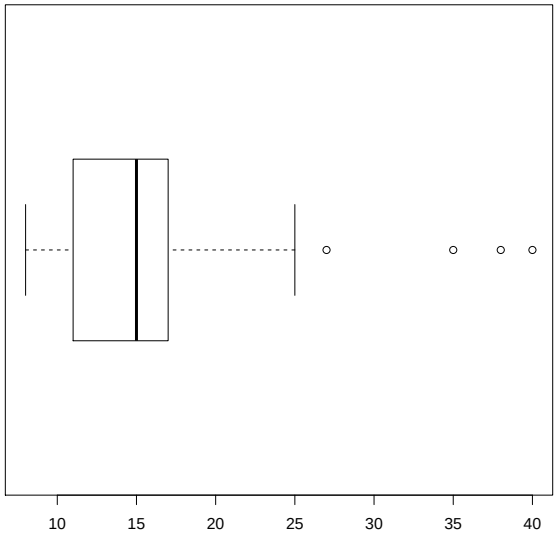
8,	9,	10,	10,	10,	10,	10,	10,	11,	11,	11,
11,	12,	12,	13,	13,	13,	14,	14,	14,	15,	15,
15,	15,	15,	15,	15	15,	16,	16,	16,	17,	17,
17,	17,	18,	18,	20,	22,	25,	27,	35,	38,	40.

(a) Obtain the Median, Q_1, Q_3 and the IQR. *15,11,17,6*

(b) Find the whisker values. Remember that the whisker values are values contained in the data set. *8, 26*

(c) Which values are box plot-outliers? *27, 35, 38, 40*

(d) Draw the box plot.



Exercise 3 Matrix arithmetic

These are a few easy exercises. If you are unsure about matix addition or matrix multiplication then

$$\text{Let } A = \begin{bmatrix} 3 & 5 \\ 2 & 8 \end{bmatrix} \qquad B = \begin{bmatrix} -2 & 0 \\ 5 & -1 \end{bmatrix} \qquad \text{and} \qquad C = \begin{bmatrix} 5 & 0 \\ 4 & 5 \end{bmatrix}$$

Calculate

(a) $A + B$

(b) $3B$

(c) AC

The solutions are on the next page to check that you are doing things properly

Spoiler

$$A + B = \begin{bmatrix} 1 & 5 \\ 7 & 7 \end{bmatrix} \qquad 3B = \begin{bmatrix} -6 & 0 \\ 15 & -3 \end{bmatrix} \qquad AC = \begin{bmatrix} 35 & 25 \\ 42 & 40 \end{bmatrix}.$$