Finding Data Patterns with R Markdown
Part 1

Professor Jian Zhang
SMSAS, University of Kent
E-mail: j.zhang-79@kent.ac.uk

1 R markdown

This chapter describes how to ceate an R markdown document in RStudio and how to covert this document into a document of many different types, including HMTL, PDF, and MS Word document.

1.1 What is R markdown?

R markdown is a way to combine plain text language with your R code, results from your data analysis (including plots and tables) and written commentary to produce a single nicely formatted and reproducible document.

Usual workflow: Data are imported from your favourite spreadsheet software into RStudio (or R). Write your R code to explore and analyse your data. Save plots as external files, copy tables of analysis output and then manually combine all of this and your written prose into a single document. The workflow separates your R code from the final document.

Markdown workflow: Data are imported into RStudio (or R) as before but this time all of the R code you used to analyse your data, your plots and your written text are contained within a single R markdown document which is then used (along with your data) to automatically create your final document.

1.2 Get started with R markdown

To use R markdown you will first need to install the rmarkdown package in the RStudio (or in the R console if you're not using RStudio) and any package dependencies. You can find instructions on how to do this for both Windows and Mac OSX operating systems (https://rmarkdown.rstudio.com/lesson-1.html).

1.3 Create an R markdown document

Within RStudio, click on the menu File ⇒ New File ⇒ R Markdown.... In the pop up window, give the document a 'Title' and enter the 'Author' information (your name) and select HTML as the default output. We can change all of this later so don't worry about it at the moment.

When your new R markdown document is created, it includes some example R markdown code. Normally you would just highlight and delete everything in the document except the information at the top between the — delimiters (this is called the YAML header which we will discuss in a bit) and then start writing your own code. However, just for now we will use this document to practice converting R markdown to both HTML and MS Word formats.

Once you've created your R markdown document it's good practice to save this file somewhere convenient. You can do this by selecting File ⇒ Save from RStudio menu (or use the keyboard shortcut ctrl + s on Windows or cmd + s on a Mac) and enter an appropriate file name (maybe call it My First R markdown). Notice the file extension of your new R markdown file is .Rmd.

Normally each R markdown document is composed of 3 main components, 1) a YAML header, 2) formatted text and 3) one or more code chunks.

An example of YAML header:

```
---
title: My First R Markdown Document
author: Bob Done
date: March 01, 2023
output: html_document
---
```

An example of marking up text formatting:

```
#### Benthic Biodiversity experiment
These data were obtained from a mesocosm experiment which aimed to examine the effect
of benthic polychaete biomass on sediment nutrient release.
####
```

To convert your .Rmd file to a HTML document click on the little black triangle next to the Knit icon at the top of the source window and select knit to HTML

If everything went smoothly, a new HTML file will have been created and saved in the same directory as your .Rmd file.

You can also knit an .Rmd file using the command line in the console rather than by clicking on the knit icon. To do this, just use the render() function from the rmarkdown package as shown below. Again, you can change the output format using the output format = argument as well as many other options.

```
library(rmarkdown)
render('myrmarkdown.Rmd', output_format = 'html_document')
#See ?render for more options.
```

RStudio will now 'knit' (or render) your .Rmd file into a HTML file. Notice that there is a new R Markdown tab in your console window which provides you with information on the rendering process and will also display any errors if something goes wrong.

2 Data formats

A common format for storing data in text files is comma-separated values (CSV) or excel spreadsheet. For example, the data might be stored in "tsex.csv":

```
date       value
01/11/2019  73
02/13/2019  82
02/14/2019  83
03/11/2019  99
...
```

The first line in the file declares the columns "date" and "value". The subsequent lines contain the actual rows of data, with the date first and then the value. As CSV files are a common format, so there are many tools that can read and write CSV files. Some of those tools are user-facing, like spreadsheet applications. For developers, there are commands to read and write CSV files from their applications.

3 Independent data

Independent data contain data points that do not depend on each other. In statistics and machine learning, you usually need to do a random permutation of the data points or randomly sample a subset from a finite population. For example, you can evaluate a model's performance by dividing the data randomly into two parts for training and validation. respectively. You can use the R-command sample() to get a permutation or a subset of the data as follows:

```
x<- 6:10
set.seed(1)
#Produce a random permutation
sample(x)
#>[1] 6  9  8  10  7
#Sample a subset of size 2 from x
sample(x, size = 2)
#> [1] 10   8
```

Here, the size argument specify the size of the subset. By default, the command takes a sample without replacement, i.e. the results sample has no duplicated elements.

Sometimes, you may want to get a sample with replacements. You are still using the command sample(), but setting the argument *replace = TRUE*.

```
sample(x, size = 10, replace = TRUE)
#> [1] 6  10  10  6  6  10  10  7  7  6
#Produce a so-called bootstrap sample
sample(x, replace = TRUE)
#> [1] 9  6  9  8  7
```

4 Dependent data

Dependent data contain data points that depend on each other. A time series is a dependent dataset, containing a series of data points ordered in time.

5 Mining data patterns

We often collect data so that we can find patterns in the data, like trending upwards or correlations between two time series. Depending on the data and the patterns, sometimes we can see that pattern in a simple tabular presentation of the data. Other times, it helps to visualize the data in a chart, like a time series or scatter plot. Let us explore examples of patterns that we can find in the data around us.

5.1 Visualising trends

A trending quantity is a number that is generally increasing or decreasing. Consider this data on babies per woman in India from 1955-2015:

```
Year Babies per woman
1960 5.91
1970 5.59
1980 4.83
1990 4.05
2000 3.31
2010 2.60
...
Source: Gapminder, Children per woman (total fertility rate).
```

In this case, the time series are steadily decreasing decade by decade, so this is a downward trend.

Now consider this data about US life expectancy from 1920-2000:

```
Year Life expectancy
1920 55.38
1930 59.57
1940 63.24
1950 68.07
1960 69.86
1970 70.86
1980 73.91
1990 75.4
...
2000  76.9
Source: Gapminder, Life expectancy at birth.
```

In this case, the numbers are steadily increasing decade by decade, so this an upward trend.

5.2 Finding correlations

Another goal of analyzing time series data is to reveal the correlation, the statistical relationship between two time series. A correlation can be positive, negative, or not exist at all. A time series plot can be used to visualize the correlation between two time series. There are strong positive correlation coefficients of fertility rates between American, European, African and Asian regions. There are strong negative correlations between fertility rates and life expectancy.

```
library(tidyverse)
library(seasonal)
library(fpp2)
GM_life <- read_csv("GM-Life Expectancy.csv")
head(GM_life)
ts.plot(ts(GM_life[,3],start=1800,end=2024),xlab="year",ylab="Expectancy",
        main = "Life Expectancy")
Fertility <- read_csv("Total fertility rate_simplified.csv")
head(Fertility)
x<-matrix(0,dim(Fertility)[2],4)
Fertility1<-Fertility[,1:dim(Fertility)[2]]
x<-t(Fertility1[,1:dim(Fertility)[2]])
ts.plot(ts(cbind(x[,1],
        x[,2],
        x[,3],x[,4]),start=1800,end=2024),
        gpars = list(xlab = "Year",
                     ylab = "Fertility",
                     main = "Fertility in four regions",
                     lwd = rep(2,4),
                     lty = c(1:4),
                     col = c("darkred","darkblue","darkgreen","grey")
                     )
        )
legend("topright", c("Amr","Eur","Afr","Asi"), lwd = rep(2,4), lty = c(1:4), col = c("darkred'

cor12<-cor(x[,1],x[,2])
cor13<-cor(x[,1],x[,3])
cor14<-cor(x[,1],x[,4])
cor23<-cor(x[,2],x[,3])
cor24<-cor(x[,2],x[,4])
cor34<-cor(x[,3],x[,4])
corgm1<-cor(x[1:225,1],GM_life[1:225,3])
corgm2<-cor(x[1:225,2],GM_life[1:225,3])
corgm3<-cor(x[1:225,3],GM_life[1:225,3])
corgm4<-cor(x[1:225,4],GM_life[1:225,3])
```

```
c(cor12,cor13,cor14,cor23,cor24,cor34,corgm1,corgm2,corgm3,corgm4)

# Pearson correlation test
test <- cor.test(x[1:225,1],GM_life[1:225,3])
test
```

Regarding the direction of the relationship: On the one hand, a negative correlation implies that the two variables under consideration vary in opposite directions, that is, if a variable increases the other decreases and vice versa. On the other hand, a positive correlation implies that the two variables under consideration vary in the same direction, i.e., if a variable increases the other one increases and if one decreases the other one decreases as well.

Regarding the strength of the relationship: The more extreme the correlation coefficient (the closer to -1 or 1), the stronger the relationship. This also means that a correlation close to 0 indicates that the two variables are independent, that is, as one variable increases, there is no tendency in the other variable to either decrease or increase.

5.3 Data analysis in finance

In finance, logarithmic returns, also known as log-returns, are a way of measuring the percentage change in the price of a stock over a period of time. The formula for calculating logarithmic returns is:

Logarithmic Return at time t = ln(Price at the current time point t / Price at the previous time point t)

Your task: (1) Download historical daily stock price data for IBM and Amazon. (2) Modify/use the codes below to analyse the data in R, including identifying data patterns and conduct a correlation analysis. (3) Write a Markdown note.

```
#Start by setting the Working directory
#Then install the packages below if not already installed (go to Tools on the main panel)
library(tseries)
ibm<-read.csv("IBM.csv",header=T)
names(ibm)
dim(ibm)

date1 <- ibm[,1]
date1 <- as.Date(date1, format = "%Y-%m-%d")

date <- ibm[2:dim(ibm)[1],1]
date <- as.Date(date, format = "%Y-%m-%d")

T<-nrow(ibm);
closepr<-ibm[,6];
#the command diff() allow you to calculate the log-price changes over two consective time poir
```

```
lreturn_IBM<-diff(log(closepr)); #log returns
T1<-length(lreturn_IBM);

lpreturn_IBM<-lreturn_IBM*100; #convert to rates

c(mean(lpreturn_IBM),var(lpreturn_IBM))

plot(closepr~date1, type="l")

plot(lpreturn_IBM ~ date, type="l")

plot(lpreturn_IBM^2 ~ date, type="l")

plot(lreturn_IBM ~ date, type="l")

plot(lpreturn_IBM ~ date, type="l", ylim=c(-16, 16))

amzn<-read.csv("AMZN.csv",header=T)

T<-nrow(amzn);
closepr<-amzn[,6];
lreturn_amzn<-diff(log(closepr)); #log returns
T1<-length(lreturn_amzn);
lpreturn_amzn<-lreturn_amzn*100; #convert to rates

plot(lpreturn_amzn ~ date, type="l", ylim=c(-16, 16))
```

Now plot the histogram of `lpreturn` for the two returns and then fit the curve of the normal distribution. What do you observe? Is the fit good?

```
hist(lpreturn_IBM, 30, xlim=c(-16, 16), main="",freq=FALSE)
x<-seq(-10,+10,by=2)
curve(dnorm(x), add=TRUE)

hist(lpreturn_amzn, 30, xlim=c(-16, 16), main="",freq=FALSE)
x<-seq(-10,+10,by=2)
curve(dnorm(x), add=TRUE)
```