Chapter4: Data Manipulation and Basic Plots

Kevin Kam Fung Yuen
PhD, Senior Lecturer, School of Business, Singapore University of Social Sciences
kfyuen@suss.edu.sg, kevinkf.yuen@gmail.com

May 27, 2019

Contents

1 Data Manipulation

1.1 Built-in Data

A number of functions are used to manipulate a dataset, such as to navigate, change, import and export the dataset. A default datasets package in R has been loaded into memory when R starts running. When we browse the datasets package, there are variety of datasets. Take a dataset called airquality as an example. The following function is used to navigate and understand a dataset.

```
airquality
class(airquality)
# list the variables of the dataset
names(airquality)
# list the structure of the dataset
str(airquality)
# list levels of a factor of the dataset
levels(airquality$Ozone)
# dimensions of the dataset
dim(airquality)
# display first 5 rows
head(airquality, n=5)
# display last 10 rows
tail(airquality, n=10)
# display the summary of the data
summary(airquality)
```

We can use *View()* function to open a spreadsheet-style data viewer to browse the data.

```
View(airquality)
```

For some purposes, we need to export a dataset into a file. There is a number of data files to store data. The csv file is one of the common ones. We can use write.csv() function to write a dataset in computer memory into a file. Therefore, the file can be used in other time and other areas.

```
write.csv(airquality, "data/mydata.csv")
```

From the above statement, the content of airquality dataset variable is written into mydata.csv file, which will be stored in the working directory. Alternatively, the file can be written into a specific location which is discussed in Unit 1. The csv file is supported by many software applications. For example, we can use Microsoft Excel or notepad to open the csv file.

Conversely, when we have a data file such as csv format, we can use read.csv() to read the data into the computing memory in R.

```
airquality.data=read.csv("data/mydata.csv")
airquality.data
```

From the above statement, the data in mydata.csv has been read and store in the variable defined by user, such as airquality.data. Alternatively, we can use RStudio to read the data file. So we click

File ->Import Data Set

We found that the options include CSV, Excel, SPSS, SAS, and Stata. So we choose "From CSV…". A window is promoted to ask us if we need to install the packages, if such packages have not been installed. After we click "yes" to proceed with the installation of packages, a user interface is promoted to guide us to import the data.

We select the file by using "Browse…", and the data is loaded as default. After we click "Import", the data is loaded. And the console window will automatically generate the code.

We can reuse the code above in the future without performing a series of click events.

```
# use code generated from import
library(readr)
mydata <- read_csv("mydata.csv")
```

2 Statistics Charts and Diagrams

A diagram can help us to learn about the overview of the distribution of a sample. The graph can be effective to present data rather than a mass of numbers because we can see where data clusters and where there are only a few data values. The base diagram functions are introduced.

2.1 Data set

a sample of smartphones imported from *mobilePhoneData.csv* is shown as below.

```r
smartphones <- read.csv("data/mobilePhoneData.csv")
# dimensions of the dataset
dim(smartphones)
# display first 5 rows
head(smartphones, n=5)
# display last 10 rows
tail(smartphones, n=10)
# display the summary of the data
summary(smartphones)
```

2.2 Frequency Tables

A frequency table presents the numbers of observations (class frequencies) with respect to mutually exclusive classes.

For category data, we use R to produce the frequency table.

```r
# Use table() function to calculate the frequences for all brands in the data set
# recall the barnds in smartphone data set and assign to x
(x=smartphones$Brands) # row view
class(x) # if it is not factor data, we use as.level()
#Create frequency table
(freq.table=table(x))
#check the data object class
class(freq.table)
#converted to matrix class
(freq.table=as.matrix(freq.table))
#give a colume name to the matrix
colnames(freq.table) = c("frequences")
freq.table # colume view
```

By dividing each class frequency by the total number of observations, class frequencies can be rescaled to relative class frequencies to show the proportion of the total observations for each class.

```r
#test
sum(freq.table)
(relative.freq.table=freq.table/sum(freq.table))
sum(relative.freq.table)
```

```r
#alternatively, colmune view
table(smartphones$Brands)
cbind(table(smartphones$Brands))
class(table(smartphones$Brands))
```

Summation of the relative frequencies is always equal to 1.0, as summation of observations in all classes is always equal to the total number of observations in the data.

If the data is not the categorical data, but metric data, we need to divide the metric data into classes with certain interval length, for example, 5 or 10 units as an interval for the common practice, but subject to different scenarios. Too few or too many classes do not have good insights for the data. The class interval should meet the basic condition if the number of classes is defined. , where

Using R as an example is shown as below.

```r
# final exam results of a population of students
X=c(69,58,64,69,68,43,67,71,89,61,63,52)
min(X)
max(X)
no.class=5
# defined the minumum interval length
(min.Interval.length=(max(X)-min(X))/no.class)
# according the result above, we choose 10 as an interval length
# we generate the sequence number from 40 to 90 which is stepped by 10
(breaks <- seq(40,90, by=10))

# Alternatively, we can use Pretty Breakpoints:Compute a sequence of about n+1
# equally spaced 'round' values which cover the range of the values in x.
breaks <-pretty(X, n=5)

# use cut() to seperate different interval
# right = FALSE means the intervals should not be closed on the right
grade.intervals= cut(X,breaks,right=FALSE)
grade.frequency=table(grade.intervals)

#converted to a matrix class
(grade.frequency=as.matrix(grade.frequency))
#give a colume name to the matrix
colnames(grade.frequency) = c("frequences")
# we may define grades in this way
rownames(grade.frequency)= c("pass","B","B+","A","A+")
grade.frequency# colume view
```

2.3 Histograms

Histogram is used to depict the frequency distribution of discrete or continuous variables. To make a histogram, data are divided into a number of class intervals (or bins). For each class interval, we need to count the number of observations falling in each class interval, i.e. frequency for each class interval. So we can make a frequency table to produce the histogram.

```r
# The final exam results of a population of students
X=c(69,58,64,69,68,43,67,71,89,61,63,52)
# hist() is used to plot a histogram
hist(X)
```

The codes above produce the basic histogram by *hist()* with default settings. When we click Export button, the graph is saved as an image, or PDF file.

We set different properties for the histogram. The customised plot of histogram is shown.

```r
#breaks: the least number of intervals.pretty() is default function used by hist()
#        to set the breaks for class intervals so the breaks of default setting are 5.
#ylim: y axis range
#xlim: x axis range
#main: Histogram Title
#xlab: lable for x axis
#ylab: lable for y axis
#labels: set if the frequency number should be displayed?
```

```
#col: set the colour for the diagram
hist(X, col = "salmon2" , breaks = 7, ylim=c(0,5),xlim=c(30,100), main="Histogram of Final Exam Marks",
     xlab = "Marks",  ylab="No of Students", labels = TRUE)
```

The *pretty()* function is used to create break points

```
# break points: extra
pretty(X,7)
```

The *colours()* function is used to check the availability of colour names. The total number of colours is 657.

```
length(colours())
colours()
??col
cl <- colors()
length(cl); cl[1:5]
```

We can use number for the colour for the bins.

```
# number of intervals
(length(pretty(X,7))-1)
# plot histogram
hist(X, col = 1:(length(pretty(X,7))-1), breaks = 10, ylim=c(0,5), main="Histogram of Final Exam Marks"
     xlab = "Marks",  ylab="No of Students", labels = TRUE)

# the colour palette values
palette()
```

We should use help function to check the parameter settings for the function for more customised settings.

3.3 Bar Chart

When the data is categorical, but not continuous, a bar chart could be used. The category classes are presented on the horizontal axis and the class frequencies (or relative frequencies) on the vertical axis. The class frequencies (or relative frequencies) are proportional to the heights of the bars, whilst each bar width should be equal and there is a gap between adjacent bars. The difference between a bar chart and a relative frequency bar chart is due to the scaling of the data.

```
# recall the code to create frequency table
(X.freq=table(smartphones$Brands))
# bar plot
barplot(X.freq)
```

2.4 Pie Chart

A pie chart is a circle (pie) cutting into various portions presenting the proportions of all classes.

```
# recall the code to create frequency table
(X.freq=table(smartphones$Brands))
# bar plot
pie(X.freq)
```

3.5 Line Chart

A line chart displays a series of joint lines (or line segments) connecting the data points (or markers) in sequence.

```
#X=c(69,58,64,69,68,43,67,71,89,61,63,52)
# recall source data
X
# recall frequency table
grade.frequency
# Basic plot by plot()
plot(grade.frequency)

# o: overplotted points and lines
plot(grade.frequency, type="o", col="blue",
     xlab = "Marks",ylab="No. of Students")
```

The plot above shows the plot with points only. We use type="o" to join the points in sequence.

```
# o: overplotted points and lines
plot(grade.frequency, type="o", col="blue",
     xlab = "Marks",ylab="No. of Students")
```

The parameter, type="l", is used for a line shape.

```
#Line plot by using type="l"
plot(grade.frequency, type="l", col="blue",
     xlab = "Marks",ylab="No. of Students")
```

The label for the marks above is not clear.

```
plot(grade.frequency, type="l", col="blue",
     xlab = "Marks",ylab="No. of Students", axes=FALSE)
# Make x axis using labels of "Pass", ..., "A+"
# The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
axis(1, at=1:5, lab=rownames(grade.frequency))
axis(2,  at=0:8)
# Use a box to cover the all sides
box()
```

2.5 Box Plot

In R, the box plot for a variable shows the information including 1. extreme of the lower whisker (the lowest value within 1.5 IQR of the lower quartile) 2. lower hinge (1st quartile or lower quartile) 3. median 4. upper hinge (3rd quartile) 5. extreme of the upper whisker (the highest value within 1.5 IQR of the upper quartile)

There are various versions of boxplot; R's boxplot() in the default grDevices package produces Tukey Boxplot.

```
#recall student final exam results data
X
#Vertical  box plot by default
boxplot(X)
#horizontal box plot by customised settings.
boxplot(X, horizontal=TRUE, col="green", main = "Student Final Exam Results", xlab = "Scores")
```

In the boxplot diagram shown above, two little circles are outliers. We also can use boxplot object to find statistics of the data.

```
#assign the boxplot object to a variable
result.boxplot=boxplot(X, horizontal=TRUE, col="green", main = "Student Final Exam Results", xlab = "Sc

# boxplot$stats, which is same as boxplot.stats
#  the extreme of the lower whisker, the lower 'hinge', the median, the upper 'hinge'
# and the extreme of the upper whisker.
#Boxplot statistics
result.boxplot$stats

#Alternatively, we can use boxplot.stats. $out indicates the outlier
boxplot.stats(X)

#out: outliers
result.boxplot$out
```

Many other advanced graph packages are available to produce the fantastic statistics charts, but beyond the scope of this course. Interested learners can refer to the topic of R for data visualisation. Once you have the basic knowledge from this course, it will not be difficult for you to pick up the concepts by your self-learning. However, the chart techniques above should address many common graph presentation requirements. A good rule is that "less is more", and "simple is beauty".

3 References

for the data source Kevin Kam Fung Yuen, The fuzzy cognitive pairwise comparisons for ranking and grade clustering to build a recommender system: An application of smartphone recommendation,Engineering Applications of Artificial Intelligence, Volume 61, 2017, pp.136-151, https://doi.org/10.1016/j.engappai.2017.02.001