A Notebook on Probability, Statistics, and Data Science

To my family, friends and communities members who have been dedicating to the presentation of this notebook, and to all students, researchers and faculty members who might find this notebook helpful.

Contents

Fo	preword	$\mathbf{i}\mathbf{x}$
Pı	reface	xi
Li	st of Figures	xiii
Li	st of Tables	$\mathbf{x}\mathbf{v}$
Ι	Probability	1
1	Probability Basics 1.1 Sample Space, Event, and Probability 1.2 Classic Probability 1.3 Geometric Probability 1.4 Conditional Probability: A Glance at Bayes Theorem 1.5 Random Variable 1.6 Expectation, Variance, and More 1.7 Sample Space, Event, and Probability 1.8 Sample Space, Event, and Probability 1.9 Sample Space, Event, and Probability 1.1 Sample Space, Event, and Probability 1.2 Classic Probability 1.3 Geometric Probability 1.4 Conditional Probability: A Glance at Bayes Theorem 1.5 Random Variable	3 3 3 4 4 4
2	Random Variables 2.1 Discrete Random Variables and Distributions	5 5 5 5
3	Joint Distributions3.1 Joint Probability Density Function3.2 Correlation3.3 Conditional Probability	7 7 7
II	Statistics	9
4	Sampling4.1 Sampling Methods4.2 Model of Population4.3 Sample Statistics	11 11 12 15
5	Estimation	17

vi			Contents
6	Reg	gression	19
7	Нур	pothesis Testing	21
8	Bay	resian Methods	23
II	I F	R for Data Science	25
9	RВ	Basics	27
	9.1	R and RStudio Installation	28
	9.2	R Packages Management	28
	9.3	R Programming Basics	30
		9.3.1 Data Types	30
		9.3.2 Conditionals and Loops	
		9.3.3 User-Defined Functions	36
		9.3.4 Vectors	36
		9.3.5 Matrices	40
	9.4	Data Frames	45
		9.4.1 Data Import	46
		9.4.2 Basic Operations	
		9.4.3 Filtering	
		9.4.4 Building Data Frames	
	9.5	Basic Data Visualizations Using qplot()	
	9.6	Advanced Data Visualizations Using ggplot()	
	0.0	9.6.1 Grammar of Graphics	
		9.6.2 Data, Aesthetics and Geometries Layers	
		9.6.3 Statistics Layers	
		9.6.4 Facets Layers	
		9.6.5 Coordinates Layers	
		9.6.6 Themes Layers	
	9.7	Data Preparation	
	<i>0.1</i>	9.7.1 Data Type Conversion	
		9.7.2 Handling Missing Data	
	9.8	Connectivity with Data Sources	
10	R fo	or Data Science	71
ττ	, T	Outhor for Data Science	79
I	′ F	Python for Data Science	73
11	Basi	ic Tools	7 5
		NumPy	
	11.2	SciPy	78
	11.3	Matplotlib and Seaborn	78
		11.3.1 Matplotlib	
		11.3.2 Seaborn	79

Co	ntent	s	vii
12	Data	a Frame Processing Using Pandas	83
	12.1	Data Importing	83
	12.2	Series and Data Frame	85
13	AN	N Engines	87
	13.1	Quick Review	88
		13.1.1 AI Pipeline	88
		13.1.2 Human Input	89
		13.1.3 Commonly Seen ANN Use Cases	89
		13.1.4 Computer Vision	89
		13.1.5 Natural Language Processing	90
	13 2	TensorFlow	90
	10.2	13.2.1 TensorFlow Basics	90
			91
		13.2.2 Classification and Regression	-
		13.2.3 Computer Vision	95
		13.2.4 General Sequential Data Processing	95
		13.2.5 Natural Language Processing	95
		13.2.6 TensorFlow on Different Platforms	95
	13.3	PyTorch	95
		13.3.1 PyTorch Basics	96
		13.3.2 Classification and Regression	96
		13.3.3 Computer Vision	96
		13.3.4 General Sequential Data Processing	96
		13.3.5 Natural Language Processing	96
		13.3.6 PyTorch on Different Platforms	96
\mathbf{V}	Se	emantic Web	97
14	Brie	f Introduction to Semantic Web	99
		Web of Data	99
			100
			100
			101
			102
			$102 \\ 107$
	149		
	14.2	00	111
		T J	111
			112
		14.2.3 Knowledge and Logic	112
15			15
		, ,	115
	15.2	- ()	117
		15.2.1 Triple Representation	118
		15.2.2 Multi-valued Relation and Blank Node	119
			120

viii	Contents
15.2.4 Reification	. 122
15.2.5 Converting RDB to RDF	. 122
15.2.6 Conclusion	. 122
15.3 RDF Schema (RDFS)	. 123
15.3.1 RDFS Motivation	
15.3.2 RDF Versus RDF/RDFS via an Example	. 124
15.3.3 RDFS Expanded Class and Properties	. 125
15.3.4 Semantics inside RDF/RDFS	. 126
15.4 SPARQL Protocol and RDF Query Language (SPARQL) .	
15.5 Web Ontology Language (OWL)	
16 Semantic Web Practice	129
16.1 Triplestore	. 129
Bibliography	131

Foreword

If software or e-books can be made completely open-source, why not a note-book?

This brings me back to the summer of 2009 when I started my third year as a high school student in Harbin No. 3 High School. In around the end of August when the results of Gaokao (National College Entrance Examination of China, annually held in July) are released, people from photocopy shops would start selling notebooks photocopies that they claim to be from the top scorers of the exam. Much curious as I was about what these notebooks look like, never have I expected myself to actually learn anything from them, mainly for the following three reasons.

First of all, some (in fact many) of these notebooks were more difficult to understand than the textbooks. I guess we cannot blame the top scorers for being so smart that they sometimes make things extremely brief or overwhelmingly complicated.

Secondly, why would I want to adapt to notebooks of others when I had my own notebooks which in my opinion should be just as good as theirs.

And lastly, as a student in the top-tier high school myself, I knew that the top scorers of the coming year would probably be a schoolmate or a classmate. Why would I want to pay that much money to a complete stranger in a photocopy shop for my friend's notebook, rather than requesting a copy from him or her directly?

However, things had changed after my becoming an undergraduate student in 2010. There were so many modules and materials to learn in a university, and as an unfortunate result, students were often distracted from digging deeply into a module (For those who were still able to do so, you have my highest respect). The situation became even worse as I started pursuing my Ph.D. in 2014. As I had to focus on specific research areas entirely, I could hardly split much time on other irrelevant but still important and interesting contents.

This motivated me to start reading and taking notebooks for selected books and articles, just to force myself to spent time learning new subjects out of my comfort zone. I used to take hand-written notebooks. My very first notebook was on *Numerical Analysis*, an entrance level module for engineering background graduate students. Till today I still have on my hand dozens of these notebooks. Eventually, one day it suddenly came to me: why not digitalize them, and make them accessible online and open-source, and let everyone read and edit it?

x Foreword

As most of the open-source software, this notebook (and it applies to the other notebooks in this series as well) does not come with any "warranty" of any kind, meaning that there is no guarantee for the statement and knowledge in this notebook to be absolutely correct as it is not peer reviewed. **Do NOT cite this notebook in your academic research paper or book!** Of course, if you find anything helpful with your research, please trace back to the origin of the citation and double confirm it yourself, then on top of that determine whether or not to use it in your research.

This notebook is suitable as:

- a quick reference guide;
- a brief introduction for beginners of the module;
- a "cheat sheet" for students to prepare for the exam (Don't bring it to the exam unless it is allowed by your lecturer!) or for lecturers to prepare the teaching materials.

This notebook is NOT suitable as:

- a direct research reference;
- a replacement to the textbook;

because as explained the notebook is NOT peer reviewed and it is meant to be simple and easy to read. It is not necessary brief, but all the tedious explanation and derivation, if any, shall be "fold into appendix" and a reader can easily skip those things without any interruption to the reading experience.

Although this notebook is open-source, the reference materials of this notebook, including textbooks, journal papers, conference proceedings, etc., may not be open-source. Very likely many of these reference materials are licensed or copyrighted. Please legitimately access these materials and properly use them.

Some of the figures in this notebook is drawn using Excalidraw, a very interesting tool for machine to emulate hand-writing. The Excalidraw project can be found in GitHub, *excalidraw/excalidraw*.

Preface

This notebook introduces probability and statistics, which is one of the fundamental undergraduate-level mathematics courses for science and engineering background students at a university.

In Part I of the notebook, probability theory is introduced. Probability theory studies how likely an event is to occur or not, and it offers rich models and tools to model and describe random values and stochastic events.

In Part II of the notebook, statistics is introduced. Statistics is a collection of methods to analyze and observe insights from data, verify statistics hypothesis and draw conclusions and predictions.

In Parts III and IV of the notebook, some of the most widely known and commonly used software solutions to statistics analysis and data science are introduced. Different from Parts I and II of the notebook that focus more on theory, Part III focuses more on using tools to solve practical problems.

As a bonus, in Part V, semantic web, the database framework defined under Web 3.0, is introduced. Semantic web does not necessarily contribute to solving a specific problem. However, it might become the "playground" and the "data pool" where other data science tools need to read and write

Key references of this notebook are summarized as follows. For probability and statistics parts:

- Spiegel, Murray, John Schiller, and Alu Srinivasan. Probability and statistics. 2020.
- Dekking, Frederik Michel, et al., A Modern Introduction to Probability and Statistics: Understanding why and how. Vol. 488. London: Springer, 2005.

For data science part:

- Kirill Eremenko, R Programming A-Z: R For Data Science With Real Exercises, Udemy Course.
- Lakshmanan, Valliappa, Martin Görner, and Ryan Gillard. Practical Machine Learning for Computer Vision. "O'Reilly Media, Inc.", 2021.
- Jose Portilla, Complete TensorFlow 2 and Keras Deep Learning Bootcamp, Udemy

Online materials such as tutorials from YouTube, Bilibili, etc., are also used in forming this notebook. ChatGPT-4 is used as a consultant in forming this notebook.

List of Figures

4.1	Sample with replacement, $N = 100, M = 500$	13
4.2	Sample without replacement, $N = 100, M = 500$	13
4.3	Sample with replacement, $N = 10000$, $M = 500$	14
4.4	Sample without replacement, $N=10000,M=500.$	14
9.1	Graphical interface to manage packages provided by RStudio.	30
9.2	A demonstration of a matrix in R	40
9.3	A demonstration of using matplot to plot trends	43
9.4	Plot of penalty success rate of the 3 players in 10 matches	45
9.5	Plot of average point gained per throw attempt for the 3 players	10
0.0	in 10 matches.	46
9.6	A demonstration of qplot	53
9.7	A demonstration of qplot on mortgage price data frame	54
9.8	A second demonstration of qplot on mortgage price data	٠.
	frame.	54
9.9	Multiple layers in chart design	55
9.10	An example of box plot of the mortgage price data frame using	
	<pre>ggplot() and geom_boxplot()</pre>	58
	An example of using geom_smooth() for scatter point fitting.	59
9.12	An example of histogram plot of house price per unit area in	
	different regions in a single plot	60
9.13	Use facets to plot the histogram of price per unit are of the	
	house in different regions (subplots in rows)	62
9.14	Use facets to plot the histogram of price per unit are of the	
	house in different regions (subplots in columns)	62
	Add coordinates layer using xlim() and ylim()	63
	Add coordinates layer using coord_cartesian()	64
9.17	Mortgage price chart with theme	65
	Plot Fibonacci series as scatter plot	79
11.2	Histogram plot using Seaborn	80
	Count plot using Seaborn	80
11.4	Box plot using Seaborn. The box gives IQR. The bars below	
	and above the box give $Q_1 - 1.5 \times IQR$ and $Q_3 + 1.5 \times IQR$,	
	respectively. The dots are outliers	81

12.1 The simplest data frame importing using pandas	84
ing pandas	85
14.1 Semantic web stack introduced in [1]	103
15.1 Semiotic triangle. The symbol is an "representation" of a category of items. The concept is the abstracted general properties	
of an item	116
15.2 Identification flowchart	117
15.3 Graph representation of triple for knowledge "Einstein was	
born in Ulm"	118
15.4 An example that demonstrate when and where a lecture takes	110
place	119
15.5 An example that demonstrate a lecture taking place at multiple	110
locations and time slots using multi-valued relations and blank	
nodes	120
15.6 An example of a container.	121
15.7 An example of a collection	121
15.8 An example of RDF reification	122
15.9 Semantic web of a few books, and their authors and publishers.	123
15.10Semantic web of a fruits, with the green color RDF and green	
+ red RDF/RDFS implementations	124
red 1021/1021 s implementations.	

List of Tables

9.1	Commonly used data types
9.2	Numerical calculations
9.3	Logical comparisons
	String operations
	Probability density related operations
9.6	Aggregate and statistics functions
9.7	Commonly used commands for data frame exploration 47
9.8	Commonly used commands for data frame exploration 56
9.9	Functions that fit smooth lines to scatter points

Part I Probability

Probability Basics

CONTENTS

1.1	Sample Space, Event, and Probability	3
1.2	Classic Probability	3
1.3	Geometric Probability	3
1.4	Conditional Probability: A Glance at Bayes Theorem	3
1.5	Random Variable	4
1.6	Expectation, Variance, and More	4

This chapter introduces the basic concepts, axioms, theorems, and fundamental calculations of probability theory.

1.1 Sample Space, Event, and Probability

"nobreak

1.2 Classic Probability

 ${\rm ``nobreak'}$

1.3 Geometric Probability

"nobreak

1.4 Conditional Probability: A Glance at Bayes Theorem

 ${\rm ``nobreak'}$

1.5 Random Variable

Discrete

Continuous

1.6 Expectation, Variance, and More

Expectation

Median

Mode

Variance

Skewness

Kurtosis

Random Variables

CONTENTS

2.1	Discrete Random Variables and Distributions	ļ
2.2	Continuous Random Variables and Distributions	,
2.3	Expectation, Variance, and More	ţ
2.4	Special Distributions	ļ

2.1 Discrete Random Variables and Distributions

 ${\rm ``nobreak'}$

2.2 Continuous Random Variables and Distributions

"nobreak

2.3 Expectation, Variance, and More

 ${\rm ``nobreak}$

2.4 Special Distributions

[&]quot;nobreak

Joint Distributions

CONTENTS

3.1	Joint Probability Density Function	,
3.2	Correlation	,
3.3	Conditional Probability	,

3.1 Joint Probability Density Function

"nobreak

3.2 Correlation

 ${\rm ``nobreak'}$

3.3 Conditional Probability

[&]quot;nobreak

Part II Statistics

Sampling

CONTENTS

4.1	Sampling Methods	11
4.2	Model of Population	12
4.3	Sample Statistics	15

The data used for statistics analysis is usually a small portion collected from a huge group. The collected data is called a sample, and the huge group from which the sample is collected, the population. For example, the incomes of a random 1000 citizens as a sample may reflect the incomes of the entire city. Another example is that the working efficiency of a machine for the past months as a sample may reflect its working efficiency in its entire lifespan.

An important underlying assumption behind statistics is that the insights observed from the sample also apply to the population. With that regard, questions naturally come up. Why does this assumption hold? When does this assumption hold? In what extend does this assumption hold? What can be done to make the sample more effective and efficient when reflecting the population?

This chapter tries to answer the above questions. As it is introduced later, the inference from sample to population is not certain, as there is a chance (however small it might be) that the samples are completely biased from the population. Therefore, we must use probability in any statement drawn from statistics analysis.

4.1 Sampling Methods

When selecting elements from the population, make sure that all elements have a equal probability of being selected, hence, random sampling. Depending on how many times a member can be sampled, we have

- Sampling with replacement: a member can be chosen more than once.
- Sampling without replacement: a member can be chosen no more than once.

Sometimes it is interesting to compare the differences of the two methods, especially then the population is finite. An obvious difference is that by using sampling with replacement all the samples can be considered as "independent event", while by using sampling without replacement, previous samples may change the distributions in the remaining population, thus making the samples relevant. In this case, using sampling with replacement can theoretically be considered as sampling from an infinite population (by thinking that the population is duplicated as many times as necessary).

In practice, the population is usually so large, that sampling from a finite population can be considered as sampling from an infinite population, and the two methods would make no differences as far as it is concerned.

Consider the following examples. A set of N random variables are generated from a Gaussian distribution as the population. Sample the population M times using sampling with replacement and sampling without replacement, respectively. Calculate the sampled mean and variance after each sampling instance, and see how it converges to the mean and variance of the population.

In the first example, let N=100 and M=500. Figures 4.1 and 4.2 gives the cumulative mean and variance of sampling with and without replacement, respectively. The mean and variance are given by red and blue curves, respectively. The statistics obtained from the cumulative samples and from the population are given by the solid and dashed curves, respectively. Notice that in Fig. 4.2, after number of samples exceeding 100, the entire population has been sampled, and thus the sampling stops. This explains why its mean and variance stop fluctuating and converge to the population mean and variance, respectively.

In practice, however, the population size is often orders of magnitudes larger than the number of samples. In the second example, let N=10000 and M=500. The corresponding figures are given in Figs. 4.3 and 4.4. There is no obvious differences of the two figures from statistics perspective.

4.2 Model of Population

The features of the population is often not known, or at least not known entirely. It is possible to make some preliminary assumptions to the distribution of population, with parameters to be further confirmed using the samples.

For example, let X be a variable of the population. It could be, for example, the heights of all teenagers in a city. We can make an assumption that X follows some distribution f(x). A widely used assumption, in this scenario, is that f(x) is a Gaussian distribution with mean μ and standard deviation σ , and each element in the population, X_i , can be taken as a random variable generated from f(x). In the case of Gaussian distribution, since it is uniquely

Sampling 13

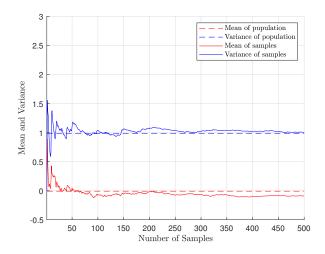


FIGURE 4.1 Sample with replacement, $N=100,\,M=500.$

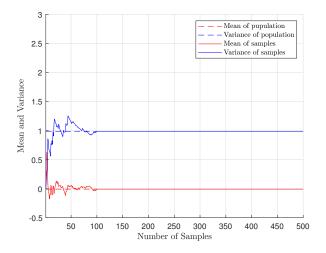


FIGURE 4.2 Sample without replacement, $N=100,\,M=500.$

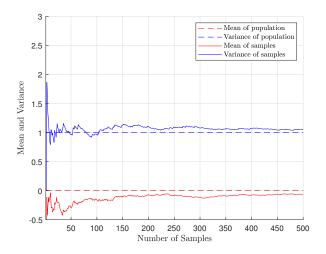


FIGURE 4.3 Sample with replacement, $N=10000,\,M=500.$

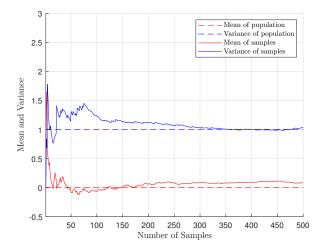


FIGURE 4.4 Sample without replacement, $N=10000,\,M=500.$

Sampling 15

characterized by μ and σ , other quantities such as the median, moments, skewness, etc., can be derived once μ and σ is calibrated.

The questions rise sequentially are:

- What are the parameters in the assumed distribution?
- Does it indeed follow the assumed distribution?

The answers to the above questions need to be found out via the samples, or more precisely, from the sample statistics.

4.3 Sample Statistics

Estimation

CONTENTS

Regression

CONTENTS

7

Hypothesis Testing

CONTENTS

8

Bayesian Methods

CONTENTS

Part III R for Data Science

CONTENTS

9.1	R and	RStudio Installation
9.2	R Pac	kages Management
9.3	R Pro	gramming Basics
	9.3.1	Data Types 3
	9.3.2	Conditionals and Loops
	9.3.3	User-Defined Functions
	9.3.4	Vectors 3
	9.3.5	Matrices 3
9.4	Data 1	Frames 4
	9.4.1	Data Import 4
	9.4.2	Basic Operations 4
	9.4.3	Filtering 4
	9.4.4	Building Data Frames
9.5	Basic	Data Visualizations Using qplot() 5
9.6	Advar	nced Data Visualizations Using ggplot() 5
	9.6.1	Grammar of Graphics
	9.6.2	Data, Aesthetics and Geometries Layers 5
	9.6.3	Statistics Layers 5
	9.6.4	Facets Layers 5
	9.6.5	Coordinates Layers 6
	9.6.6	Themes Layers 6
9.7	Data I	Preparation 6
	9.7.1	Data Type Conversion 6
	9.7.2	Handling Missing Data
9.8	Conne	ectivity with Data Sources

This chapter introduces R language basics, including the import (from files or databases), tidying, transformation, modeling, verification, and visualization of data. The use of R language in machine learning is also briefly covered.

Given that R is less popular than the more well-known and widely appreciated Python and MATLAB among engineering background researchers, this chapter serves as a basic introduction to R, before digging into its use in data science.

The first part of the chapter from Section 9.1 to Section 9.6 introduces

the basic grammar and visualization tools, and the second part of the chapter from Section 9.7 onwards more advanced practice of R.

9.1 R and RStudio Installation

R is a programming language for statistical computing and visualization. It is widely used among statisticians and data miners for developing statistical software and carrying out data analysis. R is free and can be downloaded from [2], where more details about R can also be found.

RStudio, also known as Posit, is an IDE widely used for R programming and testing. RStudio IDE is open-source and free of charge for personal use. It can be downloaded from [3].

Download R and RStudio from the aforementioned web sites, and install them sequentially.

9.2 R Packages Management

Before introducing the syntax, libraries, data frames and tools of R, it is worth introducing package management methods in R.

R packages, both built-in and third-party, provide power functions, data, and compiled codes for data analysis and visualization in a well-defined format. The packages can be published and shared online. CRAN is by far the most popular platform to store and share R packages.

To install or remove a package, use

```
install.packages("<package>")
remove.packages("<package>")
respectively. For example,
install.packages("pacman")
    To load a package, use
library(<package>)
for example
library(pacman)
```

After loading a package, the data frames and functions defined in that package can be used normally. Otherwise, to refer to a data frame or a function, the package name has to be used as a prefix as package>::<re>:<re>, which is inconvenient if the resource is used frequently.

```
To unload a package, use detach("package:<package>", unload = TRUE) for example detach("package:pacman", unload = TRUE)
```

The above methods work for both built-in packages (which often does not require installation) and third-party packages.

There are third-party packages that provides package management functions. The package pacman is an example of such package. With pacman installed and loaded, use the following commands to install, load and unload packages respectively.

```
p_install(<package>, ...) # install
p_load(<package>, ...) # install and load
p_unload(<package>, ...) # unload
p_unload(all) # unload all
```

An example of using pacman to load packages are given as follows.

```
pacman::p_load(
pacman, # package management
dplyr, # data manipulation
GGally, # data visualization
ggplot2, # data visualization
ggthemes, # data visualization
ggvis, # data visualization
httr, # url and http
lubridate, # date and time manipulation
plotly, # data visualization
rio, # io
rmarkdown, # documentation
shiny, # web apps development
stringr, # string operation
tidyr # data tidying
)
```

where notice that the above command can be executed before the loading of pacman itself, which is the reason pacman::p_load() prefix is used. These packages are commonly used in R projects. A brief explanation to them are given as comments following #. Notice that the first time installation of all the above packages may take a few minutes.

RStudio provides a graphical interface to manage packages as shown in Fig. 9.1.

Files	Plots	Packages	Help	Viewer	Presentation										
0	nstall (Update											Q,		
	Name			D	escription							Ve	ersion		
User	Library														4
	askpass			Sa	afe Password En	try for R,	Git, and SS	SH				1.	1	•	8
	asserttha	t		Ea	asy Pre and Post	Assertic	ns					0.	2.1	0	0
	base64er	nc		To	ools for base64	encoding)					0.	1-3	•	0
	bit			CI	lasses and Meth	ods for f	ast Memon	ry-Efficient E	Boolean Sel	ections		4.	0.5	•	0
	bit64			А	S3 Class for Ved	tors of 6	4bit Integer	ers				4.	0.5	•	0
	bslib			Ci	ustom 'Bootstra	p' 'Sass'	Themes for	'shiny' and	'rmarkdow	n'		0.	4.2	•	0
	cachem			C	ache R Objects v	with Auto	omatic Pruni	ning				1.	0.6	0	0
	cellrange	r		Tr	ranslate Spreads	heet Cel	l Ranges to	Rows and (Columns			1.	1.0	•	0
	cli			Н	elpers for Devel	oping Co	ommand Lin	ne Interface	s			3.	5.0	•	0
	clipr			Re	ead and Write fr	om the	System Clipb	board				0.	8.0	•	8
	colorspac	ce		Α	Toolbox for Ma	nipulatin	g and Asses	ssing Color	s and Palett	es		2.	0-3	•	⊗ (
	common	mark		Н	igh Performance	e Commo	onMark and	d Github Ma	rkdown Re	ndering in l	R	1.	8.1	•	0
	cpp11			Α	C++11 Interfac	e for R's	C Interface					0.	4.3	•	⊗ (
	crayon			C	olored Terminal	Output						1.	5.2	•	0
	crosstalk			In	nter-Widget Inte	ractivity	for HTML W	Vidgets				1.	2.0	•	0
	curl			Α	Modern and Fle	exible We	eb Client for	r R				4.	3.3	•	0
	data.table	e		Ex	xtension of `data	a.frame`						1.	14.6	•	0
	digest			Ci	reate Compact I	Hash Dig	ests of R Ob	bjects				0.	6.31	•	0
	dplyr			Α	Grammar of Da	ta Manip	oulation					1.	0.10	0	0
	ellipsis			To	ools for Working	with						0.	3.2	•	8
	evaluate			Pa	arsing and Evalu	ation To	ols that Prov	vide More [Details than	the Defaul	t	0.	19	0	0

FIGURE 9.1

Graphical interface to manage packages provided by RStudio.

9.3 R Programming Basics

This section introduces the basics of R programming, including the data types and syntax for basic R commands.

As the fundamentals, it is worth introducing here that R is case sensitive. Use # to lead a comment in R. Use print() to print a variable on the console. Typing the name of a variable often also prints it out, with a few exceptions such as when in a loop. Finally, use? followed by a function or a data frame name to check the help document for that function or data frame.

9.3.1 Data Types

R provides many data types. Commonly used data types are summarized in Table 9.1, where notice that <- is used to assign a value to a variable. Use typeof() to check the type of a variable. Alternatively, use is.numeric(), is.integer(), is.double(), is.character(), etc., to check whether a variable belongs to a particular data type.

Examples of assigning variables and checking their types are given as follows.

> n <- 2L

> typeof(n)

TABLE 9.1

Commonly used data types.

	Syntax (Example)	Description
integer	n <- 2L	An integer. Define an integer by a value
		followed by L.
double	x <- 2	An double float value.
complex	z <- 3+2i	A complex value.
character	a <- "a"	A character or a string.
logical	q <- T	A boolean value. Use T, TRUE and F,
		FALSE to represent true and false repec-
		tively.

```
[1] "integer"
> x <- 2
> typeof(x)
[1] "double"
> z <- 3+2i
> typeof(z)
[1] "complex"
> a <- "h"
> typeof(a)
[1] "character"
> q <- T
> typeof(q)
[1] "logical"
```

To transform data from one type to another, use as.<data-type>(). Examples of transforming data types are given as follows.

```
> n1 <- as.integer(2)
> typeof(n1)
[1] "integer"
> n2 <- as.integer("2")
> typeof(n2)
[1] "integer"
> x1 <- as.double(2L)
> typeof(x1)
[1] "double"
> x2 <- as.double("2")
> typeof(x2)
```

```
[1] "double"
> z1 <- as.complex("3+2i")
> typeof(z1)
[1] "complex"
> a1 <- as.character(2L)
> typeof(a1)
[1] "character"
> a2 <- as.character(2)
> typeof(a2)
[1] "character"
```

R supports arithmetic calculations of variables, including +, -, *, /, %/% (integer division), %% (modulus) and $\hat{}$ exponential. Examples of arithmetic calculations are given as follows.

```
> a <- 16
> b <- 3
> add <- a + b
> sub <- a - b
> multi <- a * b
> division <- a / b</pre>
> int_division <- a %/% b
> modulus <- a %% b
> exponent <- a ^ b
> add
[1] 19
> sub
[1] 13
> multi
[1] 48
> division
[1] 5.333333
> int_division
[1] 5
> modulus
[1] 1
> exponent
[1] 4096
```

R supports built-in and third-party functions which extend the capability of data manipulation. There is a rich set of functions for numerical calculations, string operations, probability density calculations and statistics analysis. Some of them are summarized in Tables 9.2, 9.3, 9.4, 9.5 and 9.6.

9.3.2 Conditionals and Loops

The if statement syntax is given as follows.

TABLE 9.2

3 T			
N 11	ımerical	calcu	lations

Syntax (Example)	Description
abs(x)	Absolute value.
sqrt(x)	Square root.
<pre>ceiling(x)</pre>	Smallest larger/equal integer.
floor(x)	Largest smaller/equal integer.
trunc(x)	Integer part of a variable.
round(x, n=0)	Round to n digit after decimal.
sin(x)	Trigonometric sin function.
cos(x)	Trigonometric cos function.
tan(x)	Trigonometric tan function.
log(x)	Natural logarithm.
log10(x)	Common logarithm.
exp(x)	Exponent.

TABLE 9.3

Logical comparisons.

nogream comparisons	•
Syntax (Example)	Description
х == у	Equal.
x != y	Not equal.
x > y, x < y	Greater than; less than.
x >= y, x <= y	Greater than or equal to; less than or equal to.
! x	Not.
х & у	And.
х І у	Or.
isTRUE(x)	Is true.

TABLE 9.4

String operations.

	ouring operations.	
Syntax (Example)		Description
	substr(s, n1, n2)	Segment of a string, from the n_1 -th charac-
		ter to n_2 -th character, both characters in-
		cluded.
	<pre>grep(p, s)</pre>	Searching of a pattern in a string.
	sub(s1, s2, s)	Find and replace patterns in a string.
	paste(s1, s2,, p="")	Concatenate strings with selected pattern to
		separate them.
	strsplit(s, p)	Split string into multiple strings at selected
		split points.
	tolower(s)	Convert to lower case.
	toupper(s)	Convert to upper case.

TABLE 9.5

Probability density related operations.

perations.
Description
Calculate the PDF of Gaussian distribution.
Calculate the CDF of Gaussian distribution.
Inverse function of pnorm().
Generate Gaussian distribution samples.
Calculate the probability of a binominal dis-
tribution.
Calculate the comulative probability of a bi-
nominal distribution.
Inverse function of pbinom().
Generate binominal distribution samples.
Calculate the probability of a Poisson distri-
bution.
Calculate the comulative probability of a Pois-
son distribution.
Inverse function of ppois().
Generate Poisson distribution samples.
Calculate the PDF of uniform distribution.
Calculate the CDF of uniform distribution.
Inverse function of punif().
Generate uniform distribution samples.

TABLE 9.6

Aggregate and statistics functions.

Syntax (Example)	Description
mean(1)	Mean.
sd(1)	Standard deviation.
median(1)	Median.
range(1)	Minimum and maximum.
min(1)	Minimum.
$\max(1)$	Maximum.
sum(1)	Sum

```
if(<condition>){
       <command>
} else if(<condition>){
        <command>
} else{
       <command>
}
An example of using if statement is givne below.
> x <- rnorm(1)
> if(x > 0){
       + y <- x
       + \} else if(x < 0){
       + y <- -x
       + } else{
          y <- 0
       + }
> print(x)
[1] -1.981445
> print(y)
[1] 1.981445
   The for loop syntax is given as follows.
for(<variable> in <vector>){
       <command>
}
where the <vector> can be a list of not only numbers but also characters.
Examples of using for loop are given below.
> for(i in 1:5){
             print(i)
       + }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
> for(i in c("a", "b", "c")){
             print(i)
[1] "a"
[1] "b"
[1] "c"
   The while loop syntax is given as follows.
while(<condition>){
   <command>
}
```

An example of using while loop is given below.

```
> counter <- 0
> while(counter < 5){
+    print(counter)
+    counter <- counter + 1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4</pre>
```

An example of using the above to verify the law of the large number is given below.

9.3.3 User-Defined Functions

Define a simple function as follows. Note to run the codes where the function is described before calling the function.

9.3.4 Vectors

There are different types of vectors in R. The commonly used vector types include numeric vector (including both double and integer vector) and character vector. All elements in a vector must have the same data type. When different data type values are stored in a vector, they will be transferred to the most general data type. A single number or character is stored as a vector of length 1.

Notice that the index of a vector in R starts from 1 instead of 0. This is different with many other computer languages.

Use the following syntax to create a vector.

```
<vector> <- c(<value>, ...)
where <value> can be single element or a vector. For example,
> 1 <- c(1,2,3,4,5)
> print(1)
[1] 1 2 3 4 5
> typeof(1)
[1] "double"
```

Alternative ways to create a vector are given as follows. Use sequence to create a vector as follows.

```
<vector> <- seq(<from>, <to>, <by=1>)
<vector> <- <from>:<to> # equivalent to seq() with by=1
```

Use replica to create a vector as follows.

```
<vector> <- rep(<value>, <repeate>)
```

where <value> can be a numeric number, a character, or a vector. For example,

```
> 1 <- rep(c("a", "b", "cde"), 2)
> print(1)
[1] "a" "b" "cde" "a" "b" "cde"
```

Replica can also be used to create empty vector vy rep(NA, n).

A character vector can also be created by splitting strings using strsplit(). For example,

```
> a <- "Hello World!"
> b <- strsplit(a, "")
> print(b)
[[1]]
[1] "H" "e" "l" "l" "o" " " "W" "o" "r" "l" "d" "!"
```

To access the element in a vector, use <vector>[<index>]. Again, notice that the first element in a vector has the index of 1 instead of 0. The index can be an integer, or a list of integer such as a sequence. Examples are given below.

```
> s <- c("a", "b", "c", "d", "e", "f", "g")
> s[1]
[1] "a"
> s[7]
[1] "g"
> s[2:5]
[1] "b" "c" "d" "e"
> s[c(1L, 3L, 5L)] # s[c(1.1, 3.5, 5.9)] gives the same result; data
    type auto transferred
[1] "a" "c" "e"
```

Notice that accessing a single element in a vector is rarely used in practice, because most operations in R are done by the vector basis. Vectorization operation, also known as single-instruction-multiple-data operation, significantly speeds up the calculation in R, which is quite commonly seen in high-layer languages such as R and Python. This is due to the intepreting and wrapping techniques a high-layer language uses to communicate with the underlying low-layer languages, and also the support many processors have for parallel computing.

Most, if not all, of the numerical calculations, including +, -, *, /, %/%, %%, $^{\circ}$. Examples are given below.

```
> a <- c(1,2,3,4,5)
> b < -c(5,4,3,2,1)
> a + b
[1] 6 6 6 6 6
> a - b
[1] -4 -2 0 2 4
> a * b
[1] 5 8 9 8 5
> a / b
[1] 0.2 0.5 1.0 2.0 5.0
> a %/% b
[1] 0 0 1 2 5
> a %% b
[1] 1 2 0 0 0
> a ^ b
[1] 1 16 27 16 5
```

It is also possible to apply logic operations using vectors. Examples are given below.

```
> a <- c(1,2,3,4,5)
> b <- c(5,4,3,2,1)
> a < b
[1] TRUE TRUE FALSE FALSE FALSE
> a > b
[1] FALSE FALSE FALSE TRUE TRUE
> a == b
[1] FALSE FALSE TRUE FALSE FALSE
```

When the sizes of the vectors are not consistent, the shorter vector will repeat and populate to align with the longer vector. Examples are given below.

```
> a <- c(1,10)
> b <- c(1,2,3,4)
> a + b
[1] 2 12 4 14
> a - b
[1] 0 8 -2 6
> a * b
```

```
[1] 1 20 3 40
> a / b
[1] 1.0000000 5.0000000 0.3333333 2.5000000
> a %/% b
[1] 1 5 0 2
> a %% b
[1] 0 0 1 2
> a ^ b
[1] 1 100 1 10000
```

The vector can also play as the input argument or output return of a function, which can be difficult for some computer languages by nature. Again, a scalar is treated as a vector with length 1 in R.

An example of using the above to analyze the profit of a company is given below.

```
> revenue <- round(rnorm(12, 10000, 500), 2)
> expenses <- round(rnorm(12, 9500, 500), 2)
> # profit for each month
> profit.month <- revenue - expenses
> print(profit.month)
[1] -861.33 974.12 665.84 275.72 2374.99 1231.07 953.87 396.83 -536.62
    1202.72 529.28 522.62
> # profit after tax for each month (tax rate 30%)
> profit.month.aftertax <- 0.3*profit.month
> print(profit.month.aftertax)
[1] -258.399 292.236 199.752 82.716 712.497 369.321 286.161 119.049
    -160.986 360.816 158.784 156.786
> # profit margin for each month
> profit.month.margin <- round(100 * profit.month.aftertax / revenue,
> print(profit.month.margin)
[1] -2.85 2.69 1.97 0.84 6.88 3.40 2.99 1.16 -1.71 3.25 1.60 1.56
> # is good month
> profit.month.aftertax > mean(profit.month.aftertax)
[1] FALSE TRUE TRUE FALSE TRUE TRUE TRUE FALSE FALSE FALSE
> # is bad month
> profit.month.aftertax < mean(profit.month.aftertax)
[1] TRUE FALSE FALSE TRUE FALSE FALSE TRUE TRUE FALSE TRUE TRUE
> # is the best month
> profit.month.aftertax == max(profit.month.aftertax)
[1] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
    FALSE
> # is the worst month
> profit.month.aftertax == min(profit.month.aftertax)
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
    FALSE
```

	L,13	[,2]	[5,1	L,4]	L,5]	L,6]
[1,]	E1,13					
[2,]						
[3,]			[3,3]			
[4,]						[4,6]

FIGURE 9.2

A demonstration of a matrix in R.

9.3.5 Matrices

A matrix in R is a recording of a table of data. Matrices are important because they are how data is often naturally organized, and they are also the build blocks of data frame in R.

A demonstration of a matrix in R is given by Fig. 9.2. Let the matrix be named A. The elements in the matrix can be accessed by the name of the table followed by the index coordinates. For example, in the figure, A[1,1] refers the first element and A[4,6] the last element.

It is possible to refer to an entire row or column. For example, use A[1,] to represent the first row of the matrix, by not specifying the column index. The same applies to the column.

Notice that all elements in a matrix must have the same data type.

A matrix can be created from scratch by stacking rows as follows. First, consider creating rows in the matrix. Then, use rbind() to bind rows. Finally, give names to each column and row.

```
# build rows
<row1> <- c(<value11>, ..., <value1n>)
...
<rowm> <- c(<valuem1>, ..., <valuemn>)
# build matrix
<matrix> <- rbind(<row1>, ..., <rowm>)
# (optional) clean rows
rm(<row1>, ..., <rowm>)
# give names
colnames(<matrix>) <- c("<column-name1>", ..., "<column-namen>")
rownames(<matrix>) <- c("<row-name1>", ..., "<row-namem>")
```

There are alternative ways, other than rbind(), to create a matrix. For example, matrix() convert a vector into a matrix. Similar with rbind(), cbind() binds the columns to form a matrix. Examples to create matrices using different methods are given below.

```
> A <- matrix(1:9, 3, 3)
> print(A)
[,1] [,2] [,3]
                  7
[1,]
        1
             4
        2
             5
                  8
[2,]
[3,]
        3
             6
> B \leftarrow rbind(c(1, 4, 7), c(2, 5, 8), c(3, 6, 9))
> print(B)
[,1] [,2] [,3]
                  7
[1,]
        1
             4
[2,]
        2
             5
                  8
[3,]
        3
> C \leftarrow cbind(c(1, 2, 3), c(4, 5, 6), c(7, 8, 9))
> print(C)
[,1] [,2] [,3]
                  7
[1,]
        1
             4
[2,]
        2
             5
                  8
[3,]
```

The name of the columns and rows can also be used to access an element, just by replacing the index with the name (with quotation mark) of the associated column or row. The same applies to vectors, as they can be treated as a one dimensional matrix. More details about naming a vector and columns and rows of a matrix are illustrated as follows.

To check the names relevant to a matrix, use names(<vector>), rownames(<matrix>) and colnames(<matrix>), depending on dealing with either a vector or a matrix. These commands can also be used to assign names. Examples are given below.

```
> v <- c(1, 2, 3, 4, 5)
> names(v) <- c("e1", "e2", "e3", "e4", "e5")
> print(v)
e1 e2 e3 e4 e5
1  2  3  4  5
> print(v[3])
e3
3
> print(v["e3"])
e3
3
> A <- matrix(1:9, 3, 3)
> colnames(A) <- c("col1", "col2", "col3")
> rownames(A) <- c("row1", "row2", "row3")
> print(A)
col1 col2 col3
```

```
row1
            4
                7
row2
           5
                8
       3
row3
            6
> print(A[2,2])
[1] 5
> print(A["row2", "col2"])
[1] 5
> print(A[2,])
col1 col2 col3
   5
       8
> print(A["row2",])
col1 col2 col3
    5
        8
> print(A[,2])
row1 row2 row3
   5 6
> print(A[,"col2"])
row1 row2 row3
    5
        6
```

To remove the names, simply assign NULL to the name.

Like the vector, operators are defined in matrix level as well. For example, for two matrices with the same shape, numerical operations such as +, -, *, /, %/%, %% and $\hat{}$ can be implemented.

R provides flexible and powerful data visualization tools, many of which more advanced than what is to be introduced in this section. This section introduces a simple matrix visualization function called matplot(), which plots the columns of a matrix against each other.

To demonstrate matplot(), consider the following example.

```
professor <- c(1130, 1026, 893, 922, 776)
student <- c(2, 14, 24, 49, 46)
citation <- rbind(professor, student)</pre>
colnames(citation) <- c("2018", "2019", "2020", "2021", "2022")
rownames(citation) <- c("Professor", "Student")</pre>
print(citation)
citation.ratio <- citation
citation.ratio["Professor",] <- round(citation["Professor",] / mean(</pre>
    citation["Professor",]) * 100, 1)
citation.ratio["Student",] <- round(citation["Student",] / mean(</pre>
    citation["Student",]) * 100, 1)
print(citation.ratio)
matplot(
       2018:2022, # x axis
       t(citation.ratio), # y axis
       type="b", # line and point selection
       pch = 15:16, # point shape
       col = 1:2, # color
       xlab = "Year",
```

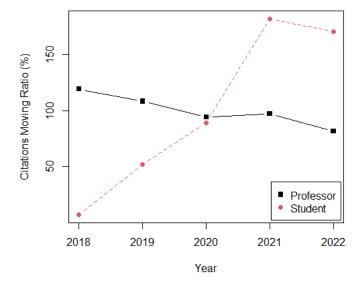


FIGURE 9.3

A demonstration of using matplot to plot trends.

```
ylab = "Citations Moving Ratio (%)"
)
legend("bottomright", inset = 0.01, legend = rownames(citation.ratio),
    pch = 15:16, col = 1:2, horiz = F)
```

where t() used inside matplot() calculates the transpose of a matrix. Save the above in a script and execute the code, to get the following Fig. 9.3.

Notice that matplot() is not widely used in particular in R.

As introduced earlier, a matrix or a vector can be split and segmented to form a smaller matrix or vector. It is worth mentioning that when a single column or row is selected, R will automatically treated the return as a vector instead of a matrix. An example is given below. When a matrix downgrades to a vector, the row name (if it has only one row), or the column name (if it has only one column) will be removed.

```
> A <- matrix(1:9, 3, 3)
> is.matrix(A)
[1] TRUE
> is.vector(A)
[1] FALSE
> is.matrix(A[1,])
[1] FALSE
> is.vector(A[1,])
[1] TRUE
```

To get consistent results, when segmenting matrix to get a single row or

plot

column vector, deliberately ask R to not drop the matrix dimensions. This can be done as follows. By doing this, the names assigned to columns and rows preserve.

```
> A <- matrix(1:9, 3, 3)
> is.matrix(A[1,,drop=F]) # select a row/column
[1] TRUE
> is.matrix(A[2,3,drop=F]) # select an element
[1] TRUE
```

An example of using the above to analyze the performance of players through a series of basketball games are given below.

```
# generate table
player_name <- c("player1", "player2", "player3")</pre>
match_name <- c("match1", "match2", "match3", "match4", "match5", "</pre>
    match6", "match7", "match8", "match9", "match10")
penalty_attempt <- abs(matrix(round(rnorm(3*10, 5, 2)), 3, 10))</pre>
penalty_point <- abs(penalty_attempt - matrix(abs(round(rnorm(3*10, 1,</pre>
    1))), 3, 10))
throw_attempt <- abs(matrix(round(rnorm(3*10, 15, 3)), 3, 10))</pre>
total_point <- abs(3*throw_attempt - abs(matrix(round(rnorm(3*10, 5, 1)
    ), 3, 10))) + penalty_point
rownames(penalty_attempt) <- player_name
colnames(penalty_attempt) <- match_name</pre>
rownames(penalty_point) <- player_name</pre>
colnames(penalty_point) <- match_name</pre>
rownames(throw_attempt) <- player_name</pre>
colnames(throw_attempt) <- match_name</pre>
rownames(total_point) <- player_name
colnames(total_point) <- match_name</pre>
# claim function
myplot <- function(table, xlab, ylab){
   row_name = rownames(table)
    column_name = colnames(table)
    matplot(
       1:length(column_name), # x axis
       t(table), # y axis
       type="b", # line and point selection
       pch = 1:length(row_name), # point shape
       col = 1:length(row_name), # color
       xlab = xlab,
       ylab = ylab
    legend("bottomleft", inset = 0.01, legend = row_name, pch = 1:
        length(row_name), col = 1:length(row_name), horiz = F)
}
```

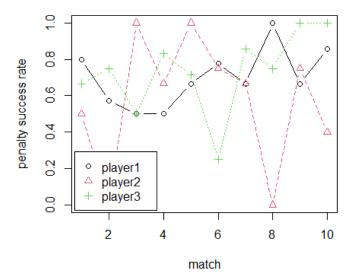


FIGURE 9.4 Plot of penalty success rate of the 3 players in 10 matches.

```
myplot(penalty_point / penalty_attempt, "match", "penalty success rate
    ") # penalty successful rate
myplot((total_point - penalty_point) / throw_attempt, "match", "average
    gained point per throw") # average point gained per throw
```

The results of the above codes are given in Figs. 9.4, 9.5.

9.4 Data Frames

Data frame, just like vector and matrix, is another data structure defined in R.

Both matrix and data frame use a table structure to store data, but data frame does not require all data to be with the same data type. Therefore, data frame is by nature the most closest format to represent a data structure from the real life. The aforementioned flexibility makes it maybe the most important and commonly used data structure in R.

In many applications and sample examples, data are stored and processed in data frame structure. When importing data from the real world, such as from a CSV file, the data is often read into a data frame before further processing.

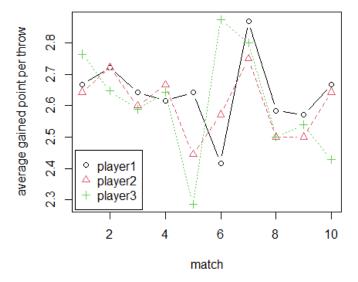


FIGURE 9.5
Plot of average point gained per throw attempt for the 3 players in 10 matches.

9.4.1 Data Import

One of the most common sources of data is CSV files. R provides convenient functions to read data from CSV files into data frames. Use the following commands to import data from a CSV file into a data frame.

The following command pops up a separate window that allows the user to choose a CSV file manually.

```
<data-frame> <- read.csv(file.choose()) # manual selection</pre>
```

The following commands import a specified CSV file.

```
setwd("<directory>") # navigate to the directory of the csv file
<data-frame> <- read.csv("<csv-file>.csv")
```

where notice that getwd() and setwd() are used to get and set current working directory, respectively.

9.4.2 Basic Operations

There are a few ways to access an element in a data frame. The methods used for accessing matrix element, including

```
<df>[<row-index>, <column-index>] <df>[<row-index>, "<column-name>"]
```

still work fine. Do notice that different from a matrix, the rows in a data frame

TABLE 9.7 Commonly used commands for data frame exploration.

Syntax (Example)	Description
nrow(df)	Number of rows.
<pre>ncol(df)</pre>	Number of columns.
head(df, n=6L)	Display the first few rows.
tail(df, n=6L)	Display the last few columns.
str(df)	A summary of the data frame, including the struc-
	ture of each column.
<pre>summary(df)</pre>	A summary of the data frame, including some of
	its statistics features.
<pre>levels(df\$<column>)</column></pre>	The level of the column.

have only indices but not names, while columns have both indices and names. In the case of data frame, \$ can be used to access a column as follows.

```
<df>$<column-name> # equivalent to <df>[, <column-name>]
```

which returns all elements in the column as a vector. The row index [<row-index>] can follow up to further specify an element if necessary.

Table 9.7 summarizes the commonly used commands for data frame exploration, such as checking its shape and data types.

An example of applying the above functions to iris data frame from the built-in datasets package is given below.

```
> library(datasets)
> nrow(iris)
[1] 150
> ncol(iris)
[1] 5
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          5.1
                     3.5
                                  1.4
                                             0.2 setosa
2
          4.9
                     3.0
                                  1.4
                                             0.2 setosa
3
          4.7
                     3.2
                                  1.3
                                             0.2 setosa
4
          4.6
                     3.1
                                  1.5
                                             0.2 setosa
5
          5.0
                     3.6
                                  1.4
                                             0.2 setosa
          5.4
6
                     3.9
                                  1.7
                                             0.4 setosa
> tail(iris)
    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
145
            6.7
                       3.3
                                   5.7
                                               2.5 virginica
146
                       3.0
                                   5.2
                                               2.3 virginica
            6.7
147
            6.3
                       2.5
                                   5.0
                                               1.9 virginica
                                   5.2
148
            6.5
                       3.0
                                               2.0 virginica
149
            6.2
                       3.4
                                   5.4
                                               2.3 virginica
150
            5.9
                       3.0
                                    5.1
                                               1.8 virginica
> str(iris)
```

```
'data.frame': 150 obs. of 5 variables:
$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
             : Factor w/ 3 levels "setosa", "versicolor", ...: 1 1 1 1 1
$ Species
     1 1 1 1 1 ...
> summary(iris)
 Sepal.Length
              Sepal.Width
                             Petal.Length Petal.Width
                                                              Species
Min. :4.300 Min. :2.000 Min. :1.000 Min.
                                                 :0.100
                                                                  :50
                                                         setosa
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300 versicolor
     :50
Median: 5.800 Median: 3.000 Median: 4.350 Median: 1.300 virginica
     :50
Mean :5.843 Mean :3.057
                            Mean
                                  :3.758
                                           Mean
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max. :7.900 Max.
                    :4.400 Max. :6.900 Max.
> levels(iris$Species) # only works discrete-value columns
[1] "setosa"
              "versicolor" "virginica"
```

Data frame sub-setting works similarly with matrix. A sub-setting of multiple rows and columns of a data frame is a data frame. Notice that unlike the matrix case where if only one row is segmented the return is treated as a vector by default, in the case of a data frame the structure preserves. When a single column is segmented, however, in both matrix and data frame scenarios, the result will be treated as a vector by default, and <code>drop=F</code> can be used to preserve data frame structure.

An example is given below.

```
> library(datasets)
> print(iris[1:5,])
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
                     3.5
                                            0.2 setosa
          5.1
                                 1.4
2
          4.9
                     3.0
                                 1.4
                                            0.2 setosa
3
          4.7
                     3.2
                                 1.3
                                            0.2 setosa
4
          4.6
                     3.1
                                 1.5
                                            0.2 setosa
5
          5.0
                     3.6
                                 1.4
                                            0.2 setosa
> print(iris[1,])
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          5.1
                     3.5
                                 1.4
                                            0.2 setosa
> is.data.frame(iris[1,])
> print(iris[,1]) # equivalent to print(iris@Sepal.Length)
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7
      5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1
 [25] 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1
     5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6
 [49] 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1
     5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
```

```
[73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3
      5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7
 [97] 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4
      6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0
 [121] \  \, 6.9 \  \, 5.6 \  \, 7.7 \  \, 6.3 \  \, 6.7 \  \, 7.2 \  \, 6.2 \  \, 6.1 \  \, 6.4 \  \, 7.2 \  \, 7.4 \  \, 7.9 \  \, 6.4 \  \, 6.3 \  \, 6.1 \  \, 7.7 
     6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
> is.data.frame(iris[,1])
[1] FALSE
> print(iris[,1,drop=F]) # preserve data frame
    Sepal.Length
              5.1
1
2
              4.9
3
              4.7
     # WRAPPED #
148
              6.5
149
              6.2
150
              5.9
> is.data.frame(iris[,1,drop=F])
[1] TRUE
```

To add a new column to an existing data frame, just assign values to a new column name as follows.

```
<df>$<new-column> <- <vector>
```

if there is a mismatch in size, <vector> will be cycled.

Ro remove a column, assign NULL to all the elements in that column as follows.

```
<df>$<column> <- NULL
```

9.4.3 Filtering

Filtering is about selecting specific rows from a data frame that meet specific criteria. A true-false vector can be used as a filter as follows.

```
<filter-name> <- <true-false-vector> # use true-false vector as filter <df>[filter,] # implement filter on data frame
```

An example is given below.

```
> library(datasets)
> filter <- iris$Sepal.Length >= 7
> print(iris[filter,])
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
51
            7.0
                       3.2
                                   4.7
                                              1.4 versicolor
103
            7.1
                       3.0
                                   5.9
                                              2.1 virginica
106
            7.6
                       3.0
                                   6.6
                                              2.1 virginica
108
            7.3
                       2.9
                                   6.3
                                              1.8 virginica
110
            7.2
                       3.6
                                   6.1
                                              2.5 virginica
```

```
118
            7.7
                       3.8
                                   6.7
                                               2.2 virginica
119
            7.7
                       2.6
                                   6.9
                                               2.3 virginica
123
            7.7
                       2.8
                                   6.7
                                               2.0 virginica
126
            7.2
                       3.2
                                   6.0
                                               1.8 virginica
130
            7.2
                       3.0
                                   5.8
                                               1.6 virginica
131
            7.4
                       2.8
                                               1.9 virginica
                                   6.1
                       3.8
132
            7.9
                                   6.4
                                               2.0 virginica
136
            7.7
                       3.0
                                    6.1
                                               2.3 virginica
```

> filter <- iris\$Sepal.Length >= 7 & iris\$Sepal.Width >= 3.5
> print(iris[filter,])

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width Species
110	7.2	3.6	6.1	2.5 virginica
118	7.7	3.8	6.7	2.2 virginica
132	7.9	3.8	6.4	2.0 virginica

As shown above, it is possible to use &, | to form a more complex filter. The commands can be merged together as follows.

> print(iris[iris\$Sepal.Length >= 7,])

- 1		- + I 0	', -,			
	Sepal.Length	Sepal.Width	Petal.Length	Petal.W	idth Specie	es
51	7.0	3.2	4.7	1.4	versicolor	
103	7.1	3.0	5.9	2.1	virginica	
106	7.6	3.0	6.6	2.1	virginica	
108	7.3	2.9	6.3	1.8	virginica	
110	7.2	3.6	6.1	2.5	virginica	
118	7.7	3.8	6.7	2.2	virginica	
119	7.7	2.6	6.9	2.3	virginica	
123	7.7	2.8	6.7	2.0	virginica	
126	7.2	3.2	6.0	1.8	virginica	
130	7.2	3.0	5.8	1.6	virginica	
131	7.4	2.8	6.1	1.9	virginica	
132	7.9	3.8	6.4	2.0	virginica	
136	7.7	3.0	6.1	2.3	virginica	
	/· · г· ·	40 3 5			٦.	

> nrow(iris[iris\$Sepal.Length >= 7,]) # count the result number
[1] 13

9.4.4 Building Data Frames

To create a data frame from scratch, use function data.frame() as follows.

```
<df> <- data.frame(<vector>, ...) # add a column colnames(<df>) <- c("<column-name>", ...)

or
<df> <- data.frame(<column-name> = <vector>, ...)
```

An example is given below, where a data frame of mortgage price at 3 types of areas, namely "CBD", "city" and "suburbs", are is created. Arbitrary data is used.

```
# create data frame
vec_region <- c(rep("CBD", 100), rep("City", 100), rep("Suburbs", 100))</pre>
vec_size_cbd <- rnorm(100, 75, 10)</pre>
vec_size_city <- rnorm(100, 100, 15)</pre>
vec_size_suburbs <- rnorm(100, 150, 25)</pre>
vec_size = c(vec_size_cbd, vec_size_city, vec_size_suburbs)
vec_price_cbd <- vec_size_cbd*rnorm(100, 12500, 2500)</pre>
vec_price_city <- vec_size_city*rnorm(100, 7500, 1000)</pre>
vec_price_suburbs <- vec_size_suburbs*rnorm(100, 5000, 1000)</pre>
vec_price <- c(vec_price_cbd, vec_price_city, vec_price_suburbs)</pre>
mortgage_price <- data.frame(Region = vec_region, Size = vec_size,</pre>
    Price = vec_price)
rm(vec_region, vec_size_cbd, vec_size_city, vec_size_suburbs, vec_size,
      vec_price_cbd, vec_price_city, vec_price_suburbs, vec_price)
which gives the following result
> head(mortgage_price)
  Region
            Size
                     Price
     CBD 81.84889 1154873.0
1
     CBD 77.78946 831468.7
    CBD 84.60477 735265.2
     CBD 62.42625 829977.5
5
     CBD 65.42723 933851.3
     CBD 82.43867 1208589.0
```

A data frame can also be created from two existing data frames by merging them together. It works like the "JOIN" function in SQL, and in this sense it supports all "INNER JOIN", "LEFT JOIN", "RIGHT JOIN" and "OUTER JOIN". The syntax is given below.

In case the two data frames have duplicated columns other than the joining columns pair, use <df>\$<column> <- NULL to unnecessary columns.

9.5 Basic Data Visualizations Using qplot()

The package ggplot2 provides useful tools for visualization of a data frame. For example, both qplot() and ggplot() in ggplot2 provide plot function. Notice that in the late versions of ggplot2, qplot() is deprecated to encourage using of the more powerful ggplot(). With that been said, both functions are smart and flexible enough to produce many different types of plots.

An example of qplot() is given below, just to show some of its capability. Run the following codes, and Fig. 9.6 is displayed. It can be seen that qplot() is smart enough to automatically choose plot dype, background color, etc., to simplify the plot function.

```
library(datasets)
library(ggplot2)
qplot(
    data=iris,
    x=Sepal.Length*Sepal.Width,
    y=Petal.Length*Petal.Width,
    color=Species,
    size=I(3),
    xlab = "Sepal Area",
    ylab = "Petal Area"
)
```

As a recap, the mortgage_price data frame created previously can be visualized as follows. Figures 9.7 and 9.8 can be obtained.

```
library(ggplot2)
rm(list=ls())
# create data frame
vec_region <- rep(c("CBD", "City", "Suburbs"), each = 100)</pre>
vec_size_cbd <- rnorm(100, 75, 10)</pre>
vec_size_city <- rnorm(100, 100, 15)</pre>
vec_size_suburbs <- rnorm(100, 150, 25)</pre>
vec_size = c(vec_size_cbd, vec_size_city, vec_size_suburbs)
vec_price_cbd <- vec_size_cbd*rnorm(100, 12500, 2500)</pre>
vec_price_city <- vec_size_city*rnorm(100, 7500, 1000)</pre>
vec_price_suburbs <- vec_size_suburbs*rnorm(100, 5000, 1000)</pre>
vec_price <- c(vec_price_cbd, vec_price_city, vec_price_suburbs)</pre>
mortgage_price <- data.frame(Region = vec_region, Size = vec_size,</pre>
    Price = vec_price)
rm(vec_region, vec_size_cbd, vec_size_city, vec_size_suburbs, vec_size,
      vec_price_cbd, vec_price_city, vec_price_suburbs, vec_price)
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size</pre>
# plot
qplot(data=mortgage_price, x=Size, y=Price, color=Region, geom=c("point
     ", "smooth"))
```

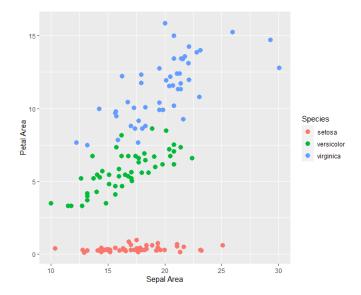


FIGURE 9.6 A demonstration of qplot.

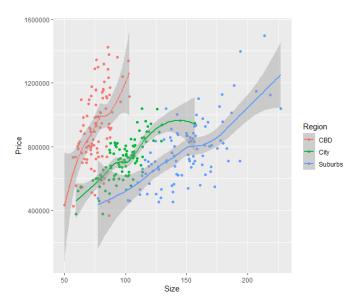
qplot(data=mortgage_price, x=Region, y=Price.Unit, geom="boxplot")

9.6 Advanced Data Visualizations Using ggplot()

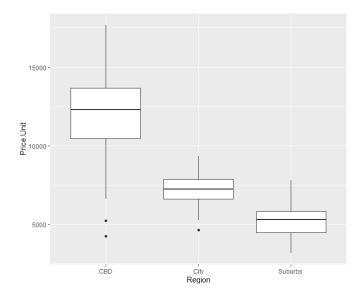
Function ggplot() is the main data visualization tool in ggolot2 package. It provides very flexible approaches for data plotting.

9.6.1 Grammar of Graphics

As proposed by Leland Wilkinson's Grammar of Graphics, a chart shall contain multiple independent and reusable layers including "data" (as data in data frames), "aesthetics" (how data maps to the chart, i.e., the logic of the plot; for example sample dots, curve, color block, or length of lines/bars), "geometries" (the actual color and shape of each element on the chart), "statistics" (information derived from the data being represented in the chart), "facets" (subplots of the same style align together for comparison), "coordinates" (the meaning and range of axis) and "theme" (overall design, such as title, label, etc.). A demonstrative Fig. 9.9 is given to illustrate the different layers in a chart.



 $\begin{tabular}{ll} FIGURE~9.7\\ A~demonstration~of~qplot~on~mortgage~price~data~frame. \end{tabular}$



 $\begin{tabular}{ll} {\bf FIGURE~9.8} \\ {\bf A~second~demonstration~of~qplot~on~mortgage~price~data~frame.} \\ \end{tabular}$

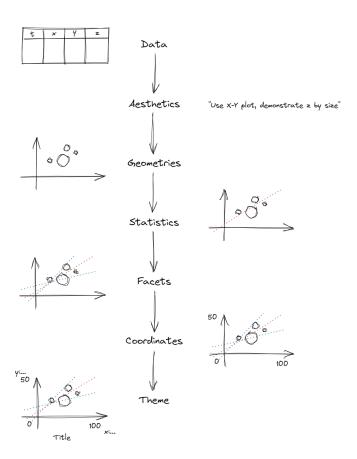


FIGURE 9.9 Multiple layers in chart design.

TABLE 9.8Commonly used commands for data frame exploration.

Geom	Description
<pre>geom_point()</pre>	Scatter plots and dot plots.
<pre>geom_line()</pre>	Line plots.
<pre>geom_bar()</pre>	Bar plots.
<pre>geom_histogram()</pre>	Histograms.
<pre>geom_boxplot()</pre>	Box plots.
<pre>geom_violin()</pre>	Violin plots.
<pre>geom_density()</pre>	Density plots.
<pre>geom_density2d()</pre>	2-dimensional Density plots.
<pre>geom_text()</pre>	Text Annotation.
<pre>geom_label()</pre>	Label on the observations.

9.6.2 Data, Aesthetics and Geometries Layers

Function ggplot() is a very good practice of implementing the above chart design and plotting philosophy. A simple example for ggplot(), just for quick demonstration purpose, is given below.

where aes() is used to build mappings in the aesthetics.

An interesting fact when using ggplot() is that, when adding a layer to the chat, the layer is literally added to ggplot(). In the program, this step by step build up an object, where ggplot() provides the most basic layers. Therefore, the above simple example is equivalent to

and the added layers are able to inherit the aesthetics settings, if it is not overwritten. And speaking of overwriting, even the x and y axis can be overwritten. The displaying name of the labels can be overwritten by stack xlab("") and ylab("") into the chart.

Function ggplot() provides many choices for geometries. The most commonly used ones are summarized in Table 9.8.

9.6.3 Statistics Layers

Similar to the case of geometries layers, statistics layers can also be stacked to ggplot(). As introduced earlier, statistics layers are often "add-on" layers that derives statistical features from the data and provide additional insights to the users.

Many functions in Table 9.8 are statistics layer built-in, such as geom_boxplot() which by nature is a statistics result presentation in the first place. Regression functions such as geom_smooth() also reveals statistical insights of the data. More details of these functions are as follows.

Consider using geom_boxplot() to visualize the mortgage_price data frame that was used in the previous section. Examples are given below.

```
library(ggplot2)
# create data frame
vec_region <- rep(c("CBD","City","Suburbs"), each = 100)</pre>
vec_size_cbd <- rnorm(100, 75, 10)</pre>
vec_size_city <- rnorm(100, 100, 15)</pre>
vec_size_suburbs <- rnorm(100, 150, 25)
vec_size = c(vec_size_cbd, vec_size_city, vec_size_suburbs)
vec_price_cbd <- vec_size_cbd*rnorm(100, 12500, 2500)</pre>
vec_price_city <- vec_size_city*rnorm(100, 7500, 1000)</pre>
vec_price_suburbs <- vec_size_suburbs*rnorm(100, 5000, 1000)</pre>
vec_price <- c(vec_price_cbd, vec_price_city, vec_price_suburbs)</pre>
mortgage_price <- data.frame(Region = vec_region, Size = vec_size,
    Price = vec_price)
rm(vec_region, vec_size_cbd, vec_size_city, vec_size_suburbs, vec_size,
     vec_price_cbd, vec_price_city, vec_price_suburbs, vec_price)
# processing
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size</pre>
# plot
p <- ggplot(data=mortgage_price, aes(x=Region, y=Price.Unit, color=
    Region)) + ggtitle("Mortgage uprice per area VS region") + xlab("
    Region") + ylab("Price per area")
p + geom_boxplot() + geom_jitter(aes(size=Size, color=Region), alpha
    =0.25)
```

and the result is shown in Fig. 9.10. Notice that ggtitle(), xlab(), ylab(), alpha are used in the plot. They are self-explanatory. A new geometry geom_jitter() is used, which works similarly with geom_point() except the additional vibration in the horizontal axis which makes the points clearer to see.

Function geom_smooth() is widely used for curve fitting. An example is given below.

```
library(ggplot2)
# generate data
t <- 1:500
var1 <- 1.5*t + rnorm(500, 0, 100)
var2 <- 0.5*t + rnorm(500, 200, 10) + t^1.3*rnorm(500, 0, 0.1)</pre>
```

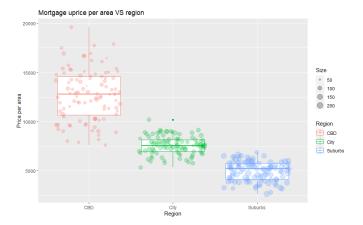


FIGURE 9.10

An example of box plot of the mortgage price data frame using ggplot() and geom_boxplot().

```
df <- data.frame(t=t, x=var1, y=var2)
# plot data
p <- ggplot(data=df) +
ggtitle("Plot of x and y VS t.") +
xlab("t") +
ylab("x and y") +
geom_point(aes(x=t, y=x), color="blue", shape=1, size=1.5) +
geom_smooth(aes(x=t, y=x), color="blue") +
geom_point(aes(x=t, y=y), color="red", shape=2, size=1.5) +
geom_smooth(aes(x=t, y=y), color="red")
p</pre>
```

Do note that aesthetics needs to be given to geom_smooth() in the above example. This is because aesthetics is not given in the base ggolot(). Notice that geom_smooth() can inherit aesthetics from the previous ggplot(), but not from the previous geom_point(). The plot is given by Fig. 9.11.

More functions similar to geom_smooth() are summarized in Table 9.9.

9.6.4 Facets Layers

The facets layer allows subplot of data. Consider the following example, where the distribution of mortgage price is studied using histogram. The following code can be used to plot the result in a single plot without the facets layer. The plot is given in Fig. 9.12.

```
library(ggplot2)
# create data frame
Region = rep(c("CBD","City","Suburbs"), each = 500)
```

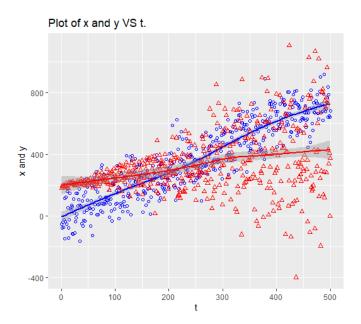


FIGURE 9.11 An example of using geom_smooth() for scatter point fitting.

TABLE 9.9
Functions that fit smooth lines to scatter points.

Function	Description
loess()	Non-parametric method for fitting a smooth line to a
smooth.spline()	scatter plot using locally weighted regression algorithm. Fits a smoothing spline to the data, which is a type of regression spline where the degree of smoothing is
	chosen automatically by cross-validation.
lm()	Linear Model, fits a linear relationship between inde-
	pendent and dependent variables by minimizing the residuals between the data points and the line.
glm()	Generalized Linear Model, similar to linear model, but
	it allows different distribution of error other than normal.
gam()	Generalized Additive Model, it is similar to GLM, but
	it allows non-parametric smooth functions to be added
	to the linear predictor.
<pre>geom_smooth()</pre>	A function in ggplot2 that is used to add a smooth line
	to a scatter plot, it uses method = "loess" by default
	but also allow to use other smoothing method like lm,
	gam etc.

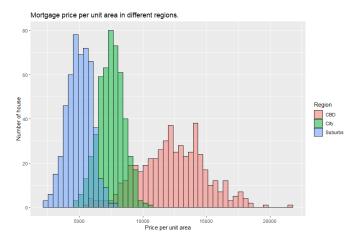


FIGURE 9.12

An example of histogram plot of house price per unit area in different regions in a single plot.

```
vec_size = list(vec_size_cbd = rnorm(500, 75, 10), vec_size_city =
    rnorm(500, 100, 15), vec_size_suburbs = rnorm(500, 150, 25))
vec_price = list(vec_price_cbd = vec_size$vec_size_cbd*rnorm(500,
    12500, 2500),
                vec_price_city = vec_size$vec_size_city*rnorm(500,
                    7500, 1000),
                vec_price_suburbs = vec_size$vec_size_suburbs*rnorm
                    (500, 5000, 1000))
mortgage_price <- data.frame(Region = Region,</pre>
                           Size = unlist(vec_size),
                           Price = unlist(vec_price))
mortgage_price$Region <- as.factor(mortgage_price$Region)</pre>
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size</pre>
# plot data
p <- ggplot(data=mortgage_price, aes(x=Price.Unit))</pre>
p + geom_histogram(aes(fill=Region), bins=50, color="black", alpha=0.5,
     position="identity") +
 ggtitle("Mortgage price per unit area in different regions.") +
  xlab("Price per unit area") +
 ylab("Number of house")
```

To use facets layer, revise the code as follows. Notice that facet_grid() is added to the plot, and its input <column>~. or .~<column> (it is okay to use <column1>~<column2> as well) decide the design of the subplots (how to arrange the rows and columns of the subplots).

```
library(ggplot2)
# create data frame
Region = rep(c("CBD","City","Suburbs"), each = 500)
```

```
vec_size = list(vec_size_cbd = rnorm(500, 75, 10),
                vec_size_city = rnorm(500, 100, 15),
                vec_size_suburbs = rnorm(500, 150, 25))
vec_price = list(vec_price_cbd = vec_size$vec_size_cbd*rnorm(500,
    12500, 2500),
                vec_price_city = vec_size$vec_size_city*rnorm(500,
                    7500, 1000),
                vec_price_suburbs = vec_size$vec_size_suburbs*rnorm
                    (500, 5000, 1000))
mortgage_price <- data.frame(Region = Region,</pre>
                           Size = unlist(vec_size),
                           Price = unlist(vec_price))
mortgage_price$Region <- as.factor(mortgage_price$Region)</pre>
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size</pre>
# plot data
p <- ggplot(data=mortgage_price, aes(x=Price.Unit))</pre>
p \leftarrow p + geom\_histogram(aes(fill=Region), bins=50, color="black", alpha
    =0.5, position="identity") +
  ggtitle("Mortgage price per unit area in different regions.") +
  xlab("Price per unit area") +
  ylab("Number of house")
p + facet_grid(Region~.) # put subplots for different regions in rows
p + facet_grid(.~Region) # put subplots for different regions in
    columns
```

The results are given in Figs. 9.13 and 9.14, depending on the subplot designs.

9.6.5 Coordinates Layers

Coordinate control is important. The coordinate layer allows setting limits to the axis and zooming in to the chart. An example of adding coordinates layers to a plot is given as follows. The same mortgage price data frame is used for illustration.

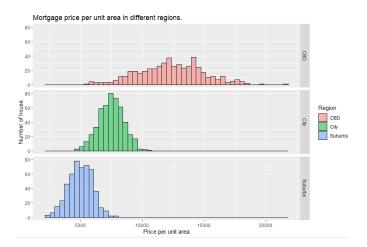


FIGURE 9.13

Use facets to plot the histogram of price per unit are of the house in different regions (subplots in rows).

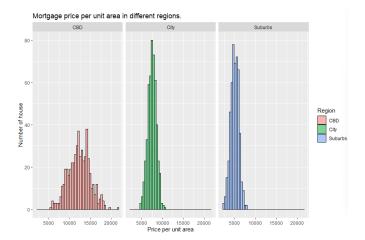


FIGURE 9.14

Use facets to plot the histogram of price per unit are of the house in different regions (subplots in columns).

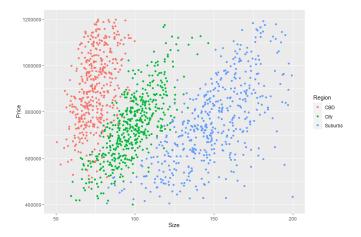


FIGURE 9.15

Add coordinates layer using xlim() and ylim().

where notice that two charts are generated. The first chart using xlim(), ylim removes all samples outside the boundary from the chart. While in the second chart using coord_cartesian(), all samples preserves and the chart zooms in towards the boundary. The results are given in Figs. 9.15 and 9.16, respectively. The difference can be observed near the boundary.

9.6.6 Themes Layers

Theme layers mainly refer to titles, labels, and other comments on the chart that help with understanding the content of the chart. As already demonstrated in previous examples, use xlab(), ylab() to add labels, ggtitle() to add title.

Use theme() to change the themes of the labels. An example is given below.

```
library(ggplot2)
# create data frame
Region = rep(c("CBD","City","Suburbs"), each = 500)
```

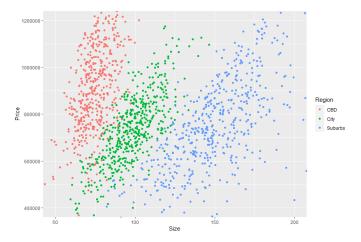


FIGURE 9.16

Add coordinates layer using coord_cartesian().

```
vec_size = list(vec_size_cbd = rnorm(500, 75, 10), vec_size_city =
    rnorm(500, 100, 15), vec_size_suburbs = rnorm(500, 150, 25))
vec_price = list(vec_price_cbd = vec_size$vec_size_cbd*rnorm(500,
    12500, 2500),
               vec_price_city = vec_size$vec_size_city*rnorm(500,
                    7500, 1000),
               vec_price_suburbs = vec_size$vec_size_suburbs*rnorm
                    (500, 5000, 1000))
mortgage_price <- data.frame(Region = Region,</pre>
                          Size = unlist(vec_size),
                          Price = unlist(vec_price))
mortgage_price$Region <- as.factor(mortgage_price$Region)</pre>
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size</pre>
# plot data
p <- ggplot(data=mortgage_price, aes(x=Price.Unit))</pre>
p <- p + geom_histogram(aes(fill=Region), bins=50, color="black", alpha
    =0.5, position="identity")
p + ggtitle("Mortgage price per unit area in different regions.") +
 xlab("Price per unit area") +
 ylab("Number of house") +
 theme(axis.title.x = element_text(color = "DarkGreen", size=15),
       axis.title.y = element_text(color = "DarkRed", size=15),
       axis.text.x = element_text(size=10),
       axis.text.y = element_text(size=10),
       legend.title = element_text(size=10),
       legend.text = element_text(size=8),
       legend.position = c(1,1), # right top corner of chart
       legend.justification = c(1,1), # legend align point
       plot.title = element_text(color = "DarkBlue", size = 15)
```



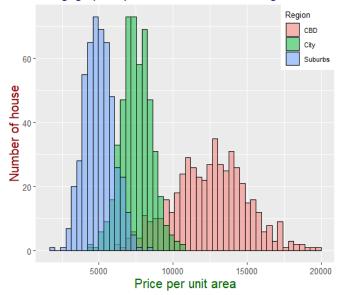


FIGURE 9.17

Mortgage price chart with theme.

)

The resulted chart is given in Fig. 9.17. Compare it with Fig. 9.12 to see the differences by applying theme() in the themes layer.

9.7 Data Preparation

The data downloaded from sensors usually needs to go through pre-processing procedures such as filtering, normalization, etc., before it can be used by a controller or an AI engine.

9.7.1 Data Type Conversion

Data preparation including data tidy is one of the most tedious and time consuming parts when using R for data analysis. The section introduces useful techniques helpful with data preparation.

It is importnt that the data types of all the columns meet expectation, especially for numeric and factor (categorical) data types. Use str(<df>) to check the column data types of a data frame, and convert data types as follows.

```
<df>$<column> <- factor(<df>$<column>) # character/numeric to factor
<df>$<column> <- as.numeric(<df>$<column>) # character to numeric
<df>$<column> <- as.numeric(as.character(<df>$<column>)) # factor to
numeric
```

Notice that when converting factor type to other types, R may deal with the factor using the underlying "factorization integers" instead of the factor item names. An example is given below. It can be seen that the original 5.1, after being converted to factor then back to numeric, becomes 9. This is because the factorization integer for 5.1 is 9, as given by printing my_factor to the console.

```
to the console.
> library(datasets)
> iris$Sepal.Length
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7
 [17] 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4
 [33] 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6
 [49] 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1
 [65] 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7
 [81] 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7
 [97] 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4
[113] 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1
[129] 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
> my_factor <- factor(iris$Sepal.Length)
> my_numeric <- as.numeric(my_factor)</pre>
> my_numeric
 [1] 9 7 5 4 8 12 4 8 2 7 12 6 6 1 16 15 12 9 15 9 12 9
 [23] 4 9 6 8 8 10 10 5 6 12 10 13 7 8 13 7 2 9 8 3 2 8
 [45] 9 6 9 4 11 8 28 22 27 13 23 15 21 7 24 10 8 17 18 19 14 25
 [67] 14 16 20 14 17 19 21 19 22 24 26 25 18 15 13 13 16 18 12 18 25 21
 [89] 14 13 13 19 16 8 14 15 15 20 9 15 21 16 29 21 23 33 7 31 25 30
[111] 23 22 26 15 16 22 23 34 34 18 27 14 34 21 25 30 20 19 22 30 32 35
[133] 22 21 19 34 21 22 18 27 25 27 16 26 25 25 21 23 20 17
> typeof(my_factor)
[1] "integer"
> my_factor
 [1] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7
 [17] 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5 5 5.2 5.2 4.7 4.8 5.4
 [33] 5.2 5.5 4.9 5 5.5 4.9 4.4 5.1 5 4.5 4.4 5 5.1 4.8 5.1 4.6
 [49] 5.3 5 7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5 5.9 6 6.1
 [65] 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6 5.7
 [81] 5.5 5.5 5.8 6 5.4 6 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5 5.6 5.7
 [97] 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4
[113] 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1
[129] 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6 6.9 6.7 6.9 5.8 6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
35 Levels: 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5 5.1 5.2 5.3 5.4 5.5 ... 7.9
```

When converting factor to other types, special caution is required. To

convert a factor to other types such as numeric, consider converting it to character first as given in the following example.

```
> my_numeric <- as.numeric(as.character(my_factor))
> my_numeric
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7
[17] 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4
[33] 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6
[49] 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1
[65] 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7
[81] 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7
[97] 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4
[113] 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1
[129] 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
```

where my_factor is generated previously.

It is possible for some columns in the data frame to look like a factor and a character string, but indeed should be handled as numeric values. For example, \$\$6,125.50 in many occasions should be treated just as 6125.0. These factor or character values cannot be converted to numeric values directly.

In such cases, consider using sub() or gsub() to replace patterns in a character, then convert it into numeric values. Notice that sub() replaces only the first encounter of the pattern, while gsub() replaces all the encounters. An example of using gsub() is given below.

```
> money_character <- c("S$6,273.15", "S$215.3", "S$8,987,756.00")
> typeof(money)
[1] "character"
> a <- gsub(",", "", money) # replace "," with ""
> a <- gsub("S\\$", "", a) # replace "S$" with ""
> money_numeric <- as.numeric(a)
> money_numeric
[1] 6273.15 215.30 8987756.00
> typeof(money_numeric)
[1] "double"
```

where notice that \$ is a special character defined in R, and to escape from that \\\$ is used. Notice that applying sub() and gsub() on a factor automatically converts it to character as a hidden step.

9.7.2 Handling Missing Data

There can be missing data in the data frame. There are a few ways to deal with missing data as follows.

- If the missing data can be derived from other columns, derive the missing data and fill in the blanks.
- If the missing data does not affect the rest analysis, leave it blank.

- Delete the row.
- Use interpolations to fill in the blank.
- Use correlations and similarities to fill in the blank.
- Argument a new column add a "data-missing" flag to that row.

If the missing data can be derived from other columns, derive the missing data and fill in the blanks.

In R, NA is a special variable used to indicate a missing value, and it is by itself of logical data type in addition to TURE and FALSE. Operations involving NA often return NA. Examples are given below.

```
> typeof(NA)
[1] "logical"
> TRUE == 1 # TRUE is equivalent with 1
[1] TRUE
> TRUE == 2
[1] FALSE
> FALSE == 0 # FALSE is equivalent with 0
[1] TRUE
> FALSE == -1
[1] FALSE
> TRUE == FALSE
[1] FALSE
> NA == NA
[1] NA
> NA == TRUE
[1] NA
> NA == FALSE
[1] NA
```

Use the following to filter for all rows with/without at least one NA.

```
<df>[complete.cases(<df>),] # all complete rows
<df>[!complete.cases(<df>),] # all incomplete rows
```

where complete.cases(<df>) returns a list made up of TRUE and FALSE indicating whether the associated row is complete or now.

Sometimes a blank string "" that we would expected to be treated as NA is not treated as so. To fix that, while importing the data frame (say, from a CSV file), use the following

```
df <- read.csv("<csv-name>", na.string=c("<pattern>", ...))
```

where "<pattern>" are the patterns in the original file to be replaced by NA, for instance, "", "ERROR", etc.

9.8 Connectivity with Data Sources

This section introduces the connectivity of R to the data sources, such as a file, or a database.

10

R for Data Science

CONTENTS

Part IV Python for Data Science

11

Basic Tools

CONTENTS

11.1	NumPy	7!
	SciPy	
	Matplotlib and Seaborn	78
	11.3.1 Matplotlib	78
	11.3.2 Seaborn	79

Python has been increasingly popular for data science in the past few years. Many libraries and tools have been developed for Python to enhance its data analysis and visualization capabilities, just to name a few, numpy, scipy, scikit-learn, pandas, matplotlib tensorflow and pytorch.

This chapter together with a few consequent chapters introduces commonly used tools that data science adopts using Python. This part of the notebook is more application driven, and only the basic implementations are introduced. We are not digging into the theory supporting machine learning and artificial intelligence.

It has been increasingly popular today to use Python together with Conda and jupyter-lab/jupyter notebook. Conda is an open-source language-agnostic package and environment management system. Jupyter notebook is an interactive computing platform for Python and other computer programming languages. The detailed introduction to the installation and usage of Conda and Jupyter notebook is not covered here. They are used when demonstrating the examples in this chapter and the consequent chapters.

11.1 NumPy

When comes to any sort of numerical computation, one of the most popular Python packages is definitely NumPy. NumPy allows quickly deployment of numerical vectors, matrices and tensors, as well as associated efficient numerical calculations. It is the "MATLAB" package in Python.

Details of NumPy can be found at numpy.org.

The following commands can be used to create NumPy arrays and matrices.

```
import numpy as np
# create numpy array from python list
x = np.array([1, 2, 3, 4, 5]) # 1d
x = np.array([[1, 2], [3, 4]]) #2d
# create numpy array/matrix using built-in functions
x = np.arange(0, 5) # array([0, 1, 2, 3, 4])
x = np.linspace(0, 5, 3) # array([0, 2.5, 5])
x = np.zeros(5) # 1d zero vector
x = np.zeros([5, 5]) # 2d zero matrix
x = np.ones(5)
x = np.ones([5, 5])
x = np.eye(5)
# create random vector/matrix
np.random.seed(1) # set seed; optional
x = np.random.rand(5, 5) # uniform distribution [0, 1)
x = np.random.randn(5, 5) # standard normal distribution N(0, 1)
# reshape
x = np.random.randn(16)
y = x.reshape(4, 4)
y = x.reshape(1, 16) # return is always 2d matrix format, not 1d vector
```

Aggregation functions are defined. These functions are used to calculate maximum, minimum, sum, etc., of a vector or a matrix. Examples are given below.

```
import numpy as np
x = np.random.randn(10)
xmax = np.max(x)
xargmax = np.argmax(x)
xmin = np.min(x)
xargmin = np.argmin(x)
xsum = np.sum(x)
xprod = np.prod(x)
xmean = np.mean(x)
xstd = np.std(x)
xvar = np.var(x)
xmedian = np.median(x)
```

When some of these functions such as sum() are applied to matrix, it is important to specify the axis along which the calculation would be carried out.

```
import numpy as np
```

Basic Tools 77

```
x = np.array([[1,2,3,4,5], [6,7,8,9,10]])
np.sum(x, axis=0) # array([7, 9, 11, 13, 15])
np.sum(x, axis=1) # array([15, 40])
```

Accessing (reading and modifying) values in numpy arrays or matrices using the index. Examples are given below. Notice that all examples are about vector accessing.

```
import numpy as np
```

```
x = np.random.randn(10)

x[1:5] # x[1], x[2], x[3] and x[4] are returned in an numpy array

<math>x[:5] # same as x[0:5]

x[5:] # same as x[5:10]
```

Matrix accessing is a bit more tricky. A matrix in NumPy is stored like a nested array. Examples are given below to access a single item, a row, and a column or a matrix.

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
x[1, 2] # 6 (2nd row, 3rd column)
```

x[0] # 1st row as a numpy array
x[:,1] # 2nd column as a numpy array

NumPy provides efficient and convenient vector and matrix level calculations, such as broad casting, matrix multiplication, etc. Broad casting essentially allows element-by-element basis adding, subtracting or assigning scalar value to a vector or a matrix. An example is given below.

```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
x[:] = 10 # all elements become 10
```

NumPy array and matrix support boolean selection. An example is given below.

NumPy supports both element-by-element or vector-level operations, including +, -, *, /, **, numpy.sqrt(), numpy.log(), etc. Most of these operators are executed element-by-element by default.

11.2 SciPy

SciPy is a library collections of "comprehensive" algorithms widely used in scientific and technical calculations. Notice that NumPy also has built in basic and commonly algorithms in the library, such as FFT. For those algorithms not included in NumPy, there is a chance that it is in SciPy, such as K-means clustering.

A detailed documentation of SciPy functions put into different categories are given here docs.scipy.org/doc/scipy/reference/.

11.3 Matplotlib and Seaborn

Data visualization is important through out the entire data analysis process. It is about not only demonstrating the results to the audiences, but also helping the developers to understand the data and improving the inefficient designs in the pipeline. Matplotlib and Seaborn are two important data visualization libraries. They are briefly introduced in this section.

11.3.1 Matplotlib

Matplotlib is the "basic" visualization library. Many data visualization features provided by other packages such as pandas are essentially realized using Matplotlib internally.

Simple line and scatter plots using Matplotlib can be drawn easily. Examples are given below. Pandas series is used as the axis to the plots, but in reality they can be Python arrays or NumPy arrays. The scatter plot used in the example is given in Fig. 11.1.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

index_s = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
values_s = pd.Series([1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
df_dict = {
    "F_Index": index_s,
    "F_Values": values_s
}
fibonacci = pd.DataFrame(df_dict)
plt.plot(index_s, values_s) # line
plt.scatter(fibonacci["F_Index"], fibonacci["F_Values"], color="red") #
    scatter
```

Basic Tools 79

```
plt.title("Fibonacci_Series")
plt.xlabel("n")
plt.ylabel("f(n)")
```

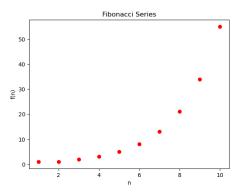


FIGURE 11.1

Plot Fibonacci series as scatter plot.

The presentation of the plot can be customized. For example, use plt.xlim(x1, x2), plt.ylim(y1, y2) to change the axis limitations, etc. It is possible to change curve color, size, style, and marker size, style, etc.

11.3.2 Seaborn

Seaborn is another visualization library built on top of the Matplotlib library. It tries to standardize the plots with a simple function call. It scarifies some flexibility that Matplotlib offers, but makes plotting easier.

An example of using Seaborn to plot a histogram is given below. The result is given in Fig. 11.2.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

x = np.random.randn(1000)
plt.figure()
sns.histplot(x, color="red")
plt.title("Histogram_Example")
plt.xlabel("x")
plt.ylabel("Count")
```

An example of using Seaborn to plot a count plot for discrete values (usually category values) is given below. Notice that the attribute whose values are put into categories is assigned to x of ${\tt seaborn.countplot()}$. The result is given in Fig. 11.3.

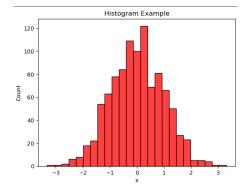


FIGURE 11.2

Histogram plot using Seaborn.

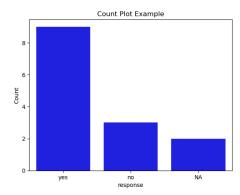


FIGURE 11.3

Count plot using Seaborn.

Bot plot gives the maximum, minimum, and interquartile range (IQR) of the data. In the box plot, the data is split into 4 parts, namely $x < Q_1$

Basic Tools 81

(0%-25%), $Q_1 < x < Q_2$ (25%-50%), $Q_2 < x < Q_3$ (50%-75%), and $Q_3 < x$ (75%-100%). The IQR gives the range between Q_1 and Q_3 , i.e., the half in the middle 25%-75%. An example is given below. The result is given in Fig. 11.4.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

x = np.random.randn(1000)
plt.figure()
sns.boxplot(x, color="blue")
plt.title("Box_Plot_Example")
plt.xlabel("Input")
plt.ylabel("Value")
```

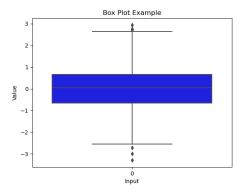


FIGURE 11.4

Box plot using Seaborn. The box gives IQR. The bars below and above the box give $Q_1 - 1.5 \times IQR$ and $Q_3 + 1.5 \times IQR$, respectively. The dots are outliers.

There are many more plot functions in Seaborn. For example, scatter plot seaborn.scatterplot() works similar with Matplotlib as shown by Fig. 11.1. The marker size and color can be adjusted to reflect different parameters, just like in R language. Pairplot seaborn.pairplot() generates a matrix of plots to demonstrate associations of columns in a data frame.

12

Data Frame Processing Using Pandas

CONTENTS

12.1	Data Importing	83
12.2	Series and Data Frame	85

Many Python packages provide functions to handle structured data such as tables, series, and data frames. Among all these packages, pandas is the all-time star that is very widely used by developers and data scientists. With pandas, Python gains the ability to easily, flexibly and efficiently deal with data frames. The pandas package is introduced in this section.

A large portion of this chapter, including codes and examples, are from online resources such as *Data Analysis by Pandas and Python* on Udemy. My special thanks go to them.

The data frames used in the examples of this chapter may come from different public data resources. It is worth mentioning *kaggle.com*, a place where tens of thousands of data sets, code examples, and notebooks are collected and shared. Some data frames used in the examples come from Kaggle.

Two Python packages, numpy and pandas, are almost certainly used in all the examples to be presented in this chapter. Import these packages as follows.

```
import numpy as np
import pandas as pd
```

Unless otherwise mentioned, these packages are always assumed imported in this chapter. To display the packages version, use something like

```
print("Numpy version: {}.".format(np.__version__))
print("Pandas version: {}".format(pd.__version__))
```

12.1 Data Importing

Pandas provide variety of ways to import data from different resources, including plain texts, CSV files, databases, etc. One of the most commonly seen data sources is CSV files. Importing data from CSV files is introduced here.

Pandas provide .read_csv() function to import data from CSV files. Its basic usage is introduced below.

```
best_selling_games_df = pd.read_csv("best-selling-video-games.csv")
best_selling_games_df
```

which reads all the information in the CSV table into a data frame as shown by Fig. 12.1. It is possible to import only selected columns as follows.

F	lank	Title	Sales	Series	Platform(s)	Initial release date	Developer(s)	Publisher(s)
0	1	Minecraft	238000000	Minecraft	Multi-platform	November 18, 2011	Mojang Studios	Xbox Game Studios
1	2	Grand Theft Auto V	175000000	Grand Theft Auto	Multi-platform	September 17, 2013	Rockstar North	Rockstar Games
2	3	Tetris (EA)	100000000	Tetris	Multi-platform	September 12, 2006	EA Mobile	Electronic Arts
3	4	Wii Sports	82900000	Wiii	Wii	November 19, 2006	Nintendo EAD	Nintendo
4	5	PUBG: Battlegrounds	75000000	PUBG Universe	Multi-platform	December 20, 2017	PUBG Corporation	PUBG Corporation
5	6	Mario Kart 8 / Deluxe	60460000	Mario Kart	Wii U / Switch	May 29, 2014	Nintendo EAD	Nintendo
6	7	Super Mario Bros.	58000000	Super Mario	Multi-platform	September 13, 1985	Nintendo R&D4	Nintendo
7	8	Red Dead Redemption 2	50000000	Red Dead	Multi-platform	October 26, 2018	Rockstar Studios	Rockstar Games
8	9	Pokémon Red / Green / Blue / Yellow	47520000	Pokémon	Multi-platform	February 27, 1996	Game Freak	Nintendo
9	10	Terraria	44500000	None	Multi-platform	May 16, 2011	Re-Logic	Re-Logic / 505 Games
10	11	Wii Fit / Plus	43800000	Wii	Wii	December 1, 2007	Nintendo EAD	Nintendo

FIGURE 12.1

The simplest data frame importing using pandas.

```
best_selling_games_df = pd.read_csv("best-selling-video-games.csv",
    usecols = ["Title", "Sales", "Publisher(s)"])
best_selling_games_df
```

When no index column is specified, pandas will add an additional auto-incremental column and use it as the index column, as shown in Fig. 12.1 by the most-left column. When a column index is specified, pandas will use that column as index column. An example is given below. The result is shown in Fig. 12.2.

```
best_selling_games_df = pd.read_csv("best-selling-video-games.csv",
    index_col = "Title", usecols = ["Title", "Sales", "Publisher(s)"])
best_selling_games_df
```

It is possible to read the full-size data frame into pandas first, then subsequently select only a few (or event only one) columns to form a new sub data frame. It is possible to take only one column from the data frame, and convert it into a series. Notice that a single-column data frame is different from a series from data type perspective. Examples are given below.

Publisher(s)	Sales	
		Title
Xbox Game Studios	238000000	Minecraft
Rockstar Games	175000000	Grand Theft Auto V
Electronic Arts	100000000	Tetris (EA)
Nintendo	82900000	Wii Sports
PUBG Corporation	75000000	PUBG: Battlegrounds
Nintendo	60460000	Mario Kart 8 / Deluxe
Nintendo	58000000	Super Mario Bros.
Rockstar Games	50000000	Red Dead Redemption 2
Nintendo	47520000	Pokémon Red / Green / Blue / Yellow
Re-Logic / 505 Games	44500000	Terraria
Nintendo	43800000	Wii Fit / Plus

FIGURE 12.2

Specifying index column and reading only selected columns using pandas.

```
sub_games_df
# convert single-column data frame to series
best_selling_games_titles = sub_games_df.squeeze('columns')
best_selling_games_titles
# get series from data frame directly
best_selling_games_df = pd.read_csv("best-selling-video-games.csv")
best_selling_games_titles = best_selling_games_df["Title"]
best_selling_games_titles
```

It is worth mentioning that when generating series from data frames, the index column of the data frame is inherited by the series. This introduces an important feature of pandas series: unlike Python array where it is just single-stream sequence of data, pandas series has a separate measure of index for each element in the series, essentially making it multi-stream of data. More are introduced in later sections.

12.2 Series and Data Frame

13

ANN Engines

CONTENTS

13.1	Quick	Review	88
	13.1.1	AI Pipeline	88
	13.1.2	Human Input	88
	13.1.3	Commonly Seen ANN Use Cases	89
	13.1.4	Computer Vision	89
	13.1.5	Natural Language Processing	90
13.2	Tensor	Flow	90
	13.2.1	TensorFlow Basics	90
	13.2.2	Classification and Regression	91
	13.2.3	Computer Vision	95
	13.2.4	General Sequential Data Processing	95
	13.2.5	Natural Language Processing	95
	13.2.6	TensorFlow on Different Platforms	95
13.3	PyTore	ch	95
	13.3.1	PyTorch Basics	95
	13.3.2	Classification and Regression	96
	13.3.3	Computer Vision	96
	13.3.4	General Sequential Data Processing	96
	13.3.5	Natural Language Processing	96
	13.3.6	PyTorch on Different Platforms	96

This chapter focuses on the introduction of commonly used Python-supported ANN engines used in data science. ANN relationships with AI, machine learning and deep learning as well as theories and mechanisms behind ANN can be found on other notebooks, hence is not given here. The introduction only contains the basic usage of these ANN engines from the implementation perspective, and it may not reflect the state-of-the-art technologies such as transformer, LLM, instruction-tuned LLM, etc. These state-of-the-art technologies are introduced on other notebooks.

There are many ANN engines for Python. Among all, TensorFlow and PyTorch are very popular and powerful generic-purpose ANN engines. They both cover a large range of supervised, reinforcement and unsupervised learning applications including classification, regression, pattern recognition, computer vision, natural language processing, clustering, abnormality detection,

and many more. They both offer variety of tools to quickly and flexibly design and deploy different types of AI models such as conventional dense networks, CNN models, RNN models, and many more. Both of them can be used to train, evaluate and run networks. Both of them provide server solutions, cloud solutions and edge computing solutions. TensorFlow and PyTorch are introduced in this chapter.

Installation of TensorFlow and PyTorch can be found in there websites. Although it is possible to run all the calculations on CPU, these ANN engines are more powerful when GPU/TPU are enabled. Depends on the OS and the GPU/TPU brands of the local system, different methods may apply to enable GPU/TPU. For example, if NVIDIA GPU is used, a software called CUDA can be used to configure and enable GPU for ANN training. The installation of TensorFlow, PyTorch, and the enabling of GPU/TPU modules are not covered here.

Alternative to running the code on a local system, consider using online platforms such as Google Colaboratory, which already have all necessary packages pre-installed and the CPU/GPU/TPU pre-configured.

13.1 Quick Review

This section briefly reviews the basic concepts used in this chapter. Details of the concepts can be found elsewhere in other notebooks.

13.1.1 AI Pipeline

AI pipeline is a set of (automated) steps used to build, train, evaluate and deploy AI models. An AI pipeline usually includes at least the following steps (for supervised learning):

- 1. Data collection
- 2. Data preparation
- 3. Model design
- 4. Model training
- 5. Model evaluation and analysis
- 6. Model deployment and testing

where notice that model design and training might need to be carried out iteratively. After the training, the performance of the model is validated using the validation set, according to which the model and its hyper parameters can be modified.

ANN Engines 89

13.1.2 Data Preparation and Model Evaluation

Model training is where the magic happens. Nowadays, with the help of AI engines, it is done automatically via back propagation and other techniques. Given the same training data and model design (including training methods), the almost-the-same trained model can be reproduced by the machine. It is done in a standardized, systematic and consistent manner, hence does not distinguish the performance of the model.

It is rather the data preparation (pre-processing), model design, and model evaluation that require human guidance. Depending on the experience and skill level of the data scientist and engineer, this is where the model performance may differ. Model design and fine-tuning are closely related to model evaluation, as model evaluation results and analysis decides how the model should be tuned. Therefore, it can be concluded that good data preparation, model evaluation and analysis skills are the critical factors that affect model behavior.

To be more precise by breaking down the aforementioned critical factors:

• Data preprocessing:

- Data cleaning. This is a critical step that greatly influences the model's performance. It includes cleaning the data, handling missing values, and dealing with outliers. Often, domain knowledge plays a significant role in these steps. In practice, some of the above procedures are done using AI engine built-in functions, while others are done using other packages such as pandas.
- Feature engineering. This involves creating new features from the existing data that might help improve the model's performance. Feature selection is also crucial to reduce overfitting and improve the model's interpretability. Again, domain knowledge can be very beneficial here.

• Model design, evaluation and tuning:

- Model selection. Choosing the right model or architecture for the problem at hand requires a solid understanding of the strengths and weaknesses of different models.
- Hyperparameter tuning. While there are systematic approaches like grid search or random search, often, practical experience and intuition play a significant role in choosing the right hyperparameters.
- Model evaluation and analysis. Evaluating a model goes beyond looking at a single metric. It involves understanding the model's errors, checking its performance on different subsets of data, and considering aspects like fairness and interpretability.

Given that many, if not all, of the above steps involve a lot of human interaction, data visualization tools often play a very important role to assist humans on their tasks.

13.1.3 Commonly Seen ANN Use Cases

"nobreak

13.1.4 Computer Vision

CV, as an important part of AI, has evolved in the past decades. In the early 2010s, ANN was not used in CV. Instead, conventional deterministic approaches were widely used. With the development in deep learning, the primary approach for CV has changed to CNN. There are a few milestones along the way that together make the change happen:

- Development of GPU
- CNN with deep neural network
- Introduction of rectified linear unit (ReLU) activation function
- Regularization techniques

Recently, with the development in transformer model and large language model, CV is able to be combined with LLM for image comprehension, reasoning, and even artwork generation.

The commonly seen objectives of CV include:

- Image classification
- Object detection
- Image generation
- Image search
- Image comprehension and generation

13.1.5 Natural Language Processing

"nobreak

13.2 TensorFlow

TensorFlow is an open-source software library for machine learning developed by Google in 2015. TensorFlow 2.X is released in 2019 and it is know the official latest major updated version. TensorFlow is backed up by a large community and it supports Python as well as a few other programming languages.

ANN Engines 91

Don't confuse TensorFlow with Keras, later of which is a Python library built on top of deep learning libraries such as TensorFlow, and provides a simple and useful API. In TensorFlow 2.X, Keras is officially adopted as its API. Therefore, when TensorFlow 2.X is used, there is no need to install or import Keras separately.

13.2.1 TensorFlow Basics

Unless otherwise mentioned, the following packages are imported and the command executed in the beginning of all the relevant scripts.

```
import numpy as np
import pandas as pd
import tensorflow as tf

tf.test.is_gpu_available()
```

where .is_gpu_available() tests whether GPU is enabled on the machine. The well-known numpy package defines "NumPy arrays" to store vectors, matrices and tensors. Package tensorflow also defines counterparts "Tensor-Flow tensors" for the same purpose. NumPy arrays and TensorFlow tensors can be converted from one to the other. A key difference of the two is that TensorFlow tensors related calculations are executed on GPU wherever possible, making it more efficient when comes to large-scale calculation that can be paralleled. An example is given below.

```
x = np.array([1, 2, 3, 4, 5])
y = tf.convert_to_tensor(x, dtype=tf.float64)
x = x*0.3 // cpu calculation
y = y*0.3 // gpu parallel calculation
```

13.2.2 Classification and Regression

Classification

Regression

The following data frame *kaggle.com/datasets/shree1992/housedata* is used in this example to as a demonstration to predict house pricing using regression model.

The following class SimpleRegressionModel serves as an example that can be used for the above task. Notice that this model design is only a demonstration, and it is not optimized.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import seaborn as sns
class SimpleRegressionModel:
       SimpleRegressionModel trains and tests a regression model using
           the given data frame.
       def __init__(self):
              self.num_input = None
              self.num_output = None
              self.scalar = None
              self.num_training = None
              self.X_train = None
              self.Y_train = None
              self.num_validation = None
              self.X_val = None
              self.Y_val = None
              self.history = None
              self.num_test = None
              self.X_test = None
              self.Y_test = None
              self.model = None
       def import_dataset(self, df_total: pd.DataFrame, iplist: list,
           oplist: list, validation_size: float, test_size: float):
       total_X = df_total[df_total.columns.intersection(iplist)]
       object_columns = total_X.select_dtypes(include=['object'])
       if not object_columns.empty:
          dummy_columns = pd.get_dummies(object_columns)
           total_X = pd.concat([total_X.drop(object_columns, axis=1),
               dummy_columns], axis=1)
       self.num_input = len(total_X.columns)
       total_Y = df_total[df_total.columns.intersection(oplist)]
       self.num_output = len(total_Y.columns)
       train_val_X, self.X_test, train_val_Y, self.Y_test =
           train_test_split(total_X, total_Y, test_size=test_size,
           random_state=None)
       self.num_test = len(self.X_test.index)
              if validation_size == 0:
                     self.X_train = train_val_X
                     self.Y_train = train_val_Y
                     self.X_val = []
                     self.Y_val = []
                     self.num_training = len(self.X_train.index)
                     self.num\_validation = 0
              else:
                     self.X_train, self.X_val, self.Y_train, self.
                          Y_val = train_test_split(train_val_X,
```

ANN Engines 93

```
train_val_Y, test_size=validation_size/(1-
                   test_size), random_state=None)
               self.num_training = len(self.X_train.index)
               self.num_validation = len(self.X_val.index)
       print("Dataset \_ size: \_ training: \_ \{\}, \_ validation: \_ \{\}, \_ test:
            _{\sqcup}\{\}". \texttt{format}(\texttt{self.num\_training}, \ \texttt{self.num\_validation},
            self.num_test))
       self.scalar = MinMaxScaler()
       self.X_train = self.scalar.fit_transform(self.X_train)
       if validation_size == 0:
               pass
       else:
               self.X_val = self.scalar.transform(self.X_val)
       self.X_test = self.scalar.transform(self.X_test)
def design_model(self, hidden_layer_model: list, optimizer: str,
     learning_rate: float, loss: str):
       The input model describes the design of the model. It is
             a list of layers. Each layer is given by a
            dictionary describing layer type, number of nodes,
       self.model = tf.keras.Sequential()
       for ind in range(len(hidden_layer_model)):
               if ind == 0:
                       if hidden_layer_model[ind]["type"] == "
                           dense":
                               layer = tf.keras.layers.Dense(
                                   hidden_layer_model[ind]["node"
                                   ], activation =
                                   hidden_layer_model[ind]["
                                   activation"], input_shape = (
                                   self.num_input, ))
                       else:
                              pass
               else:
                       if hidden_layer_model[ind]["type"] == "
                           dense":
                              layer = tf.keras.layers.Dense(
                                   hidden_layer_model[ind]["node"
                                   ], activation =
                                   hidden_layer_model[ind]["
                                   activation"])
                       else:
                              pass
               self.model.add(layer)
       self.model.add(
               tf.keras.layers.Dense(self.num_output, activation
                   ='relu')
```

```
if optimizer == 'adam':
                       optimizer = tf.keras.optimizers.Adam()
               self.model.compile(optimizer=optimizer, loss=loss,
                   metrics=['mae'])
       def train_model(self, epochs: int, batch_size: int):
               self.history = self.model.fit(self.X_train, self.Y_train
                    , epochs=epochs, batch_size=batch_size,
                   validation_split=0.2, verbose=2)
       def evaluate_model(self):
               if self.num_validation == 0:
                      {\tt print("Validation} \sqcup {\tt set} \sqcup {\tt is} \sqcup {\tt empty."})
               else:
                       loss, mae = self.model.evaluate(self.X_val, self.
                           Y_val, verbose=2)
                       print("Validation\_set\_test\_result:\_loss=\{\},\_mae
                           ={}".format(loss, mae))
       def test_model(self):
               loss, mae = self.model.evaluate(self.X_test, self.Y_test
                    , verbose=2)
               print("Test_set_test_result:_loss={},_mae={}".format(
                   loss, mae))
   The following code uses the above defined class to predict house price.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import seaborn as sns
df_house_pricing = pd.read_csv("house_pricing.csv").dropna()
srm = SimpleRegressionModel()
srm.import_dataset(
    df_total=df_house_pricing,
    iplist=["bedrooms", "bathrooms", "sqft_living", "sqft_lot", "floors
        ", "sqft_above", "sqft_basement", "yr_built", "yr_renovated", "
        condition", "city"],
    oplist=["price"],
    validation_size=0, test_size=0.2)
hidden_layer_model = [
       {
```

"type": "dense", "node": 128,

"type": "dense", "node": 64,

},

"activation": "relu"

ANN Engines 95

```
"activation": "relu"
       },
               "type": "dense",
               "node": 64,
               "activation": "relu"
       },
               "type": "dense",
               "node": 32,
               "activation": "relu"
       },
       {
               "type": "dense",
               "node": 16,
               "activation": "relu"
       },
]
srm.design_model(hidden_layer_model=hidden_layer_model, optimizer='adam
    ', learning_rate=0.001, loss='mse')
srm.train_model(epochs=100, batch_size=50)
srm.evaluate_model()
srm.test_model()
```

The above example is self-explanatory. Some key components of the codes are:

- Use train_test_split() from sklearn.model_selection to split the data set into training set, validation set and testing set.
- Use MinMaxScaler() from sklearn.preprocessing to normalize the inputs to the AI model.
- Use tf.keras.Sequential() and tf.keras.layers to design the AI model structure.
- Use .compile() to configure optimization engine, loss function, validation matrix, etc., during the training.
- Use .fit() to train the model using the training set.
- Use .evaluate() to evaluate the AI model performance.
- Use .predict() to carry out prediction of input points.

Use .save("name") to save a model, and load_model() from tensorflow.keras.models to load a model. Some popular formats to store a model include H5 file.

13.2.3 Computer Vision

"nobreak

13.2.4 General Sequential Data Processing

"nobreak

13.2.5 Natural Language Processing

"nobreak

13.2.6 TensorFlow on Different Platforms

"nobreak

13.3 PyTorch

TensorFlow is another open-source software library for machine learning originally developed by Meta AI in 2016. It is now under the Linux foundation umbrella.

13.3.1 PyTorch Basics

"nobreak

13.3.2 Classification and Regression

"nobreak

13.3.3 Computer Vision

"nobreak

13.3.4 General Sequential Data Processing

"nobreak

13.3.5 Natural Language Processing

 ${\rm ``nobreak'}$

13.3.6 PyTorch on Different Platforms

Part V Semantic Web

14

Brief Introduction to Semantic Web

CONTENTS

14.1	Web of	f Data	96
	14.1.1	Web 1.0 and 2.0	100
	14.1.2	Web 3.0	100
	14.1.3	Semantic Web Vision	101
	14.1.4	Semantic Web Stack	102
	14.1.5	Semantic Web Limitations and Challenges	107
14.2	Ontolo	gy	111
	14.2.1	Philosophy Perspective	111
	14.2.2	Semantic Web Perspective	112
	14.2.3	Knowledge and Logic	112

Ontology is the philosophical study of the nature of being, existence, or reality. In the context of the Semantic Web, ontology refers to a formal specification of concepts, relationships, and categories of a particular domain or knowledge base. Ontology allows the creation of a shared understanding of a particular domain, enabling machines to interpret and reason about data more effectively. Ontology is essentially about "what is everything". It is about how we as humans mark our existence, and how we preserve and add to our knowledge base for the generations to come.

Ontology inspires people in computer science how we can store and exchange information more efficiently by building an internet based knowledge base. The solution is called the semantic web. An internet powered by semantic web (together with other technologies) defines web 3.0.

14.1 Web of Data

The Internet, as we call it today, is a technology that enables information exchange between machines, as well as between machines and human beings. With the power of the Internet, humans can obtain the information they need from remote servers, databases, or knowledge bases.

Initially, this information exchange was difficult, and professional computer

skills were required to operate the Internet and its human-machine interface. Nowadays, everyone can access the Internet by just using a graphical browser. Public knowledge bases like Wikipedia make obtaining information much easier

Following this trend, AI knowledge bases combined with human-readable semantic web technologies such as RDF and OWL, will become more and more popular and provide additional help for knowledge transfer and sharing.

14.1.1 Web 1.0 and 2.0

In the age of Web 1.0, knowledge was stored on individual servers, often in a static manner. Users could open a browser and check the knowledge as it was presented in the form designed by the server owner. It was essentially a one-way data transmission. The experts provide information, and the ordinary users consume the information.

In the age of Web 2.0, thanks to more advanced communication technologies, users can interact with web service providers. They can search and filter data from the servers and even upload their own data to the servers and share it with others. The Internet became easier and more flexible to use for information exchange, and data transfer became bidirectional. Ordinary users are both providers and consumers of information on the Internet. Databases are used in the backend which allows the user to push and pull data from the server.

Web 2.0 is useful but still far from perfect, especially given the background of industry 4.0 and data-driven everything.

One of the biggest challenges that we encounter with Web 2.0 is how to quickly locate the information we want hidden among the vast amount of data on the Internet. Among all the information available, which is important, and which is redundant? Humans have contextual knowledge and experience to solve the problem, but it will take forever.

Keyword search engines are widely used in the Web 2.0 framework, and it is the main method one can rely on when pulling information from the Internet. Keyword search engines may struggle with polysemous, synonyms, unintended personalizing, and implicit information from pictures.

Today most of the information on the Internet is stored in HTML format. HTML tells the contents links to the information, but not the meaning behind it. It is difficult for a machine to understand the meaning of the information from HTML files. As will be introduced later, this potentially makes it difficult for a machine to retrieve data efficiently.

14.1.2 Web 3.0

The ultimate goal of Web 3.0 is to allow more efficient information search and transfer using the Internet, given the vast amount of existing data hosted on distributed servers and sensors.

In order to search, exchange, and abstract useful information from distributed sources in an efficient manner, there are at least two things we can do:

- Let the machine do the job. The AI should be smart enough to precisely capture what the user wants and get back to him with useful and accurate results.
- Make the information stored on the Internet more readable for both humans and machines, i.e., make the information semantic (meaningful).

The above forms the fundamentals of Web 3.0. The first point leads to smart chatbot, and the second to semantic web.

Web 3.0 relies on advanced data storage, data searching, and data processing technologies. We need to develop powerful AI so that it can interpret the user demands, search for useful information in its knowledge base efficiently, and finally come back with a human-readable answer. For that, we need advanced natural language processing techniques, and efficient (and potentially online) AI training techniques to keep the sophisticated knowledge base up to date.

In addition, we need to formally express the semantics (meanings) of the information for everything put online. This helps both humans and machines to interpret information and decide whether it is useful. This introduces the concept of the semantic web, which is a significant aspect of Web 3.0. In fact, semantics are so important that they are often considered synonymous with Web 3.0, although the broader vision of Web 3.0 includes other features as well.

Distributed storage has both advantages and disadvantages. On one hand, it provides higher resilience against data loss. On the other hand, it adds difficulty to the searching and collecting of information. Technologies like IPFS, blockchain, and distributed databases are being used to address these challenges.

This notebook focuses on the technologies used in semantic web, some of which such as RDF and OWL have already been implemented. We will see how we can use semantic web to collect and organize information, and from where how we can create a knowledge base for both humans and AI.

14.1.3 Semantic Web Vision

How to make contents on the Internet meaningful to machines?

Recall the 2 bullet points in the previous section. We always have the choice to use AI with powerful natural language processing capability. This of course relies on technologies in machine learning. We need to design ANN models that can be easily massively trained and it needs to be able to capture long text that humans usually use. With the development of transformer, this is becoming promising. There is another challenge though, which is that the

knowledge base formed by AI is a black box, and cannot be interpreted by other machines with a different framework. In other words, you cannot build a knowledge base with AI and use it everywhere without any tuning. Transfer learning and knowledge distillation may help, but they also have constraints.

In the semantic web vision, we mainly focus on another approach, which is **annotating explicit semantic metadata to the existing content**. The semantic metadata shall be encoded the meaning of the content in a machine-readable way.

Semantic web approach is complementary with AI approach. It has some advantages:

- The knowledge is more transparent, unlike AI where the knowledge is hidden in the ANN as a black box.
- The knowledge is reusable and scalable in a easier way than AI knowledge base which relies on knowledge distilling (reuse) and transfer learning (rescale).
- RDF and OWL, the tools we use to organize and store knowledge in semantic web, are more reader-friendly for both humans and machines.

Many research facilities and organizations have tried building semantic web for either global or specific domains. An example is DBPedia *dbpedia.org*. The goal of DBPedia is to create something like Wikipedia, but semantic.

14.1.4 Semantic Web Stack

A commonly seen semantic web stack looks like Fig. 14.1. It is a summary of components and tools used in building a semantic web.

In the very bottom is the platform that hosts everything. It is also the representation of all the existing knowledge. Uniform Resource Identifier (URI) is the "address" that can be used to identify a information source. It can be in the form of an URL to the web page that introduces an item. For example, "About: Semantic Web (dbpedia.org)" dbpedia.org/page/Semantic Web is the URL to the defination of semantic web in DBPedia. The web page represents semantic web as part of the existing knowledge (this web page is not stand alone though, as later we will see that it links to other sources using linked data).

N3, TURTLE, XML, JSON and Resource Description Framework (RDF) model are used to organize and store data. N3, TURTLE, XML and JSON is the syntax of the text, while RDF demonstrates the knowledge representation structure. The following is an example of using semantic web stored in TURTLE. Only the first few lines are given. Details to the syntax are introduced in later chapters.

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbr: <http://dbpedia.org/resource/> .
```

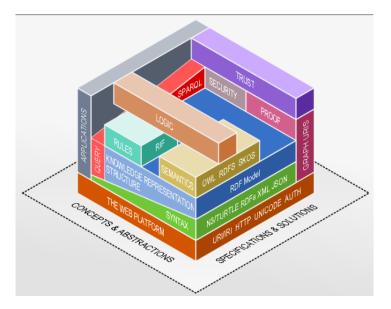


FIGURE 14.1 Semantic web stack introduced in [1].

```
dbr:Semantic_web
                      dbo:wikiPageWikiLink dbr:Semantic_Web ;
       dbo:wikiPageRedirects dbr:Semantic_Web .
                             dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:Knowledge_management
dbr:Intelligent_agent dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:Wiki_software
                     dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:List_of_Brown_University_alumni dbo:wikiPageWikiLink dbr:
    Semantic_Web .
dbr:NitrosBase dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:Arabic_Ontology dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:Semantic_Web_Stack dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:Giant_Global_Graph dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:Computational_semantics dbo:wikiPageWikiLink dbr:Semantic_Web .
dbr:Deborah_McGuinness dbo:wikiPageWikiLink dbr:Semantic_Web ;
       dbo:academicDiscipline dbr:Semantic_Web .
```

Information is tagged in RDF. RDF, together with RDF Schema (RDFS) provides limited semantics, and it created a foundation of semantic that allows Web Ontology Language (OWL) to build more complicated semantics on top of it, such as adding rules on restrictions and disjoint classes. In this sense, OWL is an extension of RDF.

SPARQL (SPARQL Protocol and RDF Query Language) is the standard query language for querying and manipulating RDF data on the Semantic Web. It allows users to search, retrieve, and modify information stored in RDF format.

The syntax of SPARQL is similar to SQL, as both are query languages designed for structured data. SPARQL was influenced by SQL, and many concepts in SPARQL have counterparts in SQL, such as SELECT, WHERE, GROUP BY, and ORDER BY. However, the backend technologies differ largely.

SQL is designed to query and manipulate data stored in relational databases, which are based on tables with rows and columns. On the other hand, SPARQL is designed for querying and manipulating RDF data, which is based on graphs with nodes and edges. In SPARQL, you work with RDF triples and graph patterns, and use operations such as graph pattern matching to find data that satisfies specific conditions. Their interfaces look similar, but the backend technologies differ largely.

An example of a knowledge base using RDF/RDFS/OWL is given below, together with a SPARQL query.

Example: Knowledge Base on Animals

The following examples demonstrates how to use RDF/RDFS and OWL to create a small knowledge base about animal. The example is provided by *ChatGPT-4*.

RDF/RDFS

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
ex:Animal rdf:type rdfs:Class .
ex:Mammal rdf:type rdfs:Class ;
   rdfs:subClassOf ex:Animal .
ex:Reptile rdf:type rdfs:Class;
   rdfs:subClassOf ex:Animal .
ex:hasLegs rdf:type rdf:Property ;
   rdfs:domain ex:Animal;
   rdfs:range rdfs:Literal .
ex:Dog rdf:type rdfs:Class;
   rdfs:subClassOf ex:Mammal .
ex:Lizard rdf:type rdfs:Class;
   rdfs:subClassOf ex:Reptile .
ex:Max rdf:type ex:Dog ;
```

```
ex:hasLegs 4 .

ex:Lizzy rdf:type ex:Lizard ;
  ex:hasLegs 4 .

In this example, we used RDF and RDFS to:
```

- Define classes (Animal, Mammal, Reptile, Dog, and Lizard)
- Define a property (hasLegs)
- Set the domain and range of the property
- Create subclass relationships (Mammal and Reptile are subclasses of Animal; Dog is a subclass of Mammal; Lizard is a subclass of Reptile)
- Define individuals (Max and Lizzy) and their properties

OWL

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ex: <http://example.org/> .
ex:hasParent rdf:type owl:ObjectProperty ;
   rdfs:domain ex:Animal;
   rdfs:range ex:Animal .
ex:hasChild rdf:type owl:ObjectProperty ;
   owl:inverseOf ex:hasParent .
ex:isWarmBlooded rdf:type owl:Class ;
   rdfs:subClassOf ex:Animal .
ex:Mammal rdfs:subClassOf ex:isWarmBlooded .
ex:Reptile owl:disjointWith ex:isWarmBlooded .
ex:Max ex:hasParent ex:Buddy .
ex:Buddy rdf:type ex:Dog ;
   ex:hasLegs 4 .
   In this example, we used OWL to:
```

• Define object properties (hasParent and hasChild)

- Specify an inverse relationship between properties (hasParent and hasChild)
- Define a new class (isWarmBlooded) and set it as a superclass of Mammal
- Specify a disjoint relationship between Reptile and isWarmBlooded
- Define a new individual (Buddy) and his properties
- Specify a relationship between individuals (Max and Buddy)

In this demonstration example, RDF and RDFS provided the basic structure and hierarchy for the knowledge base, while OWL added more expressivity by defining complex relationships, additional semantics, and constraints.

Finally, the above knowledge base can be queried by SPARQL as follows.

```
PREFIX ex: <a href="http://example.org/">
SELECT ?mammal
WHERE {
    ?mammal rdf:type/rdfs:subClassOf* ex:Mammal .
}
```

SPARQL query is not user-friendly. Commercialized semantic web applications such as WolframAlpha wolframalpha.com often provides a "search bar" with limited NLP capability to make the searching of the semantic web easier. It is worth mentioning that although this searching engine may look like chatbot at a first glance, their underlying information source and techniques differently largely. See ChatGPT-4's comments as follows.

I understand that chatbot like you works differently compared with these engines such as WolframAlpha. While their knowledge comes from semantic web, your knowledge comes from years of training using natural-language-written books and articles. Am I right?

Yes, that's correct. As an AI language model, I am based on the GPT-4 architecture developed by OpenAI. My primary source of knowledge comes from the vast amount of text data that I have been trained on, which includes books, articles, and other documents written in natural language. My training data consists of diverse sources and covers a wide range of topics up to my knowledge cutoff date in September 2021.

Unlike search engines like WolframAlpha or tools that rely on the Semantic Web, my knowledge representation and understanding are not based on formalized structures like RDF or OWL. Instead, I analyze and generate text based on patterns and relationships learned from my training data.

This approach has some advantages and limitations. On the one hand, I can provide more human-like, natural language responses and can generate text on a wide range of topics, even if I have not been specifically trained on a particular domain. On the other hand, my understanding of the underlying structure and semantics of the data may be less precise than that of systems that use formal knowledge representation languages like RDF or OWL.

In summary, chatbots like me and engines like WolframAlpha operate differently in terms of their knowledge sources and representation, but both approaches can be useful for providing information and answering questions in their respective domains.

Chatbot and semantic web can work together. Chatbot is used to provide a human interface and basic domain knowledge, while the semantic web provides up-to-date precise informative facts.

14.1.5 Semantic Web Limitations and Challenges

It is worth mentioning that though semantic web is powerful, building semantic web can be challenging because it requires a lot of human labor. As of 2023, the majority of web sites and applications have not yet embraced the full potential of the semantic web, some of which only partially adopt semantic web concepts or technologies.

ChatGPT-4 comments on the current status of semantic web applications as follows.

The semantic web has made significant progress since its inception, but it hasn't been adopted as widely as initially envisioned. Many organizations, research institutions, and some large corporations have adopted semantic web technologies for specific use cases, such as knowledge management, data integration, and domain-specific applications. However, the majority of websites and web applications have not yet embraced the full potential of the semantic web.

That being said, some aspects of the semantic web have become popular and are widely used:

1. Structured data markup: Many websites use structured data markup, such as schema.org, to provide additional semantic information about their content. This helps search engines like Google to better understand the content and present it more effectively in search results. While not the full realization of the semantic web vision, this is a step towards making the web more machine-readable.

- 2. Linked Data: The concept of Linked Data, which is closely related to the semantic web, has gained some traction. Linked Data is a method of publishing structured data on the web so that it can be interlinked and easily accessed by machines. Examples include DBpedia, Wikidata, and various governmental and organizational datasets.
- 3. Research and academia: The semantic web has been a popular research topic in academia, with numerous conferences, workshops, and publications dedicated to its development and application. This has led to the development of various tools, algorithms, and methodologies that have been adopted in specific domains.
- 4. Domain-specific applications: Some industries and sectors, such as healthcare, life sciences, and digital libraries, have adopted semantic web technologies to manage their complex data and facilitate data integration and interoperability.

While the semantic web has not been adopted as widely as originally anticipated, its core ideas and technologies have had an impact on various areas of the web, and its principles continue to influence the development of new standards and technologies. It's possible that as AI and machine learning technologies continue to advance, there will be renewed interest in the semantic web and its potential to facilitate machine understanding of web content.

There are several challenges that have contributed to the semantic web not being embraced to its full potential:

- Complexity: Semantic web technologies, such as RDF and OWL, can be complex and difficult to understand for many web developers, especially those without a background in knowledge representation, logic, or artificial intelligence. This complexity can act as a barrier to adoption.
- 2. Lack of incentives: Many website owners and content creators may not see immediate benefits or incentives for adopting semantic web technologies. Implementing semantic annotations and structured data can be time-consuming and resource-intensive, especially for smaller organizations or individuals without dedicated technical teams.
- 3. Incomplete coverage: The vision of the semantic web relies on a large amount of high-quality, machine-readable data being available. However, not all websites provide structured data or semantic annotations, which limits the effectiveness of semantic web applications and discourages further adoption.
- 4. Ontology development and maintenance: Creating and maintaining

ontologies, which are key components of the semantic web, can be a complex and time-consuming task. Achieving consensus on common vocabularies and ontologies across different domains and stakeholders can also be challenging.

- 5. Scalability and performance: Semantic web technologies need to handle large-scale data processing and reasoning tasks, which can be computationally expensive. Some traditional semantic web tools and algorithms may not scale well for big data or real-time applications.
- 6. Privacy and security concerns: The semantic web promotes data sharing and integration, which can raise privacy and security concerns for both individuals and organizations. Ensuring that sensitive data is protected while still enabling the benefits of the semantic web can be a challenging balance to strike.
- 7. Competition from other technologies: Other technologies and approaches, such as keyword-based search engines, relational databases, and machine learning, have continued to evolve and provide effective solutions for many data management and retrieval tasks, which may have reduced the urgency for adopting semantic web technologies.

Overcoming these challenges will require continued research, development, and collaboration among researchers, developers, and stakeholders. As new tools, algorithms, and techniques are developed, and as the importance of machine understanding of data increases, there may be renewed interest in and adoption of the semantic web.

Despite the challenges in the widespread adoption of the semantic web, its core ideas and principles have had a significant impact on various fields, especially in the development of domain-specific knowledge bases and applications. The concepts of ontology, structured data, and linked data have inspired and facilitated numerous projects and use cases, such as:

- 1. Domain-specific ontologies: The development of domain-specific ontologies has allowed organizations and researchers to model and represent complex knowledge structures and relationships in their respective fields. Examples include the Gene Ontology (GO) in the life sciences, the CIDOC Conceptual Reference Model (CRM) in cultural heritage, and the Financial Industry Business Ontology (FIBO) in finance.
- 2. Data integration and interoperability: Semantic web technologies have been instrumental in integrating and sharing data across different systems, platforms, and organizations. By utilizing shared ontologies and vocabularies, data can be mapped and linked more eas-

ily, enabling better interoperability and reducing the need for manual data reconciliation.

- 3. Knowledge management and discovery: Semantic web technologies have been used to build intelligent knowledge management systems that can infer new information and discover hidden relationships by reasoning over structured data. This can help organizations make better decisions, discover new insights, and optimize their processes.
- 4. Personalization and recommendation systems: By incorporating semantic web technologies and leveraging domain-specific knowledge bases, recommendation systems can provide more accurate and personalized suggestions based on users' preferences, behaviors, and the underlying relationships between items.
- 5. Natural language processing and AI: The ideas behind the semantic web have inspired research in natural language processing (NLP) and AI, as they both strive to enable machines to better understand and process human-generated content. Semantic web technologies can be used to enrich NLP tasks and enhance AI models with structured knowledge.

While the semantic web has not yet reached its full potential, its principles and concepts continue to influence and inspire the development of new technologies and applications, especially in the context of building domain-specific knowledge bases and fostering knowledge sharing and discovery.

With the population of LLM powered chatbot and IoT as part of Industrial 4.0, things might change. It has been extremely difficult to check the correctness and consistency of the existing data bases (both structured and unstructured) due to the large volume of data, let along converting them into semantic web. With the help from chatbot, the checking and conversion might be automated. In addition, as Industry 4.0 and IoT technologies continue to advance, the amount of structured data generated will increase significantly. This structured data can be a valuable resource for building semantic web applications more easily and effectively. A knowledge base that combines semantic web and chatbot might become a future standard. ChatGPT-4 comments as follows.

In the future, the combination of AI chatbots and semantic web technologies could become a powerful standard for handling various tasks. AI chatbots would serve as the natural language interface for users, providing basic knowledge and understanding of context, while the semantic web would provide accurate, up-to-date, and domain-specific data.

Integrating AI chatbots with SPARQL or other semantic web query-

ing capabilities can enable them to search, analyze, and interpret structured data more effectively. This way, chatbots could provide users with more precise and relevant information based on the available structured data.

Such a combination of technologies would not only make it easier for users to access and interact with the vast amount of structured data, but it would also enable AI systems to better understand the context and semantics of information, leading to improved performance and more accurate results.

14.2 Ontology

Ontology has very rich meanings with both philosophy and semantic web perspective.

14.2.1 Philosophy Perspective

Knowledge is the intersect of ground truth and human beliefs, i.e., it is the truth that humans know of being truth. Ontology is a way of communicating knowledge.

In the context of philosophy, ontology discusses the meaning of an object "exists", how objects can be categorized, and how objects relate to each other. Ontology mainly discusses the following questions:

- What is existence?
- What are the fundamental categories of existence?
- What does it mean for something to exist or not exist?
- What types of entities exist, and what is their nature?
- What is the nature of abstract entities like numbers, properties, and relations?
- How do different entities relate to each other and interact?
- Can something exist independently of our perception or thought?

The discussion of these questions can be traced back to BC300 and earlier. Aristotle defined a system to structure and reasoning knowledge. The famous Aristotelian logic "major premise + minor premise -; conclusion" is in fact a standard and systematic way of reasoning unknown from knowledge. Aristotelian logic is an important tool of ontology, and it inspires how we build semantic web, even after these many years.

The first idea of creating a machine to categorize things came in 1200s. The first idea of creating a "universal precise language" to record knowledge was proposed in 1600s.

14.2.2 Semantic Web Perspective

In the context of the Semantic Web, an ontology is a formal, machine-readable representation of a specific domain's concepts, their properties, and the relationships between them. It serves as a shared vocabulary (unlike natural language, which is often ambiguous) for describing and reasoning about the knowledge within that domain. In practice, RDF/RDFS and OWL can be used to express the ontology.

The followings are defined in RDF/RDFS and OWL.

- Class. Classes are used to represent components and models. It represents a concept. It is an abstraction of objects sharing some similarities.
- Attribute. Attributes are used to describe classes.
- Relations. Relations are special attributes whose values are objects of (other) classes.
- Constraints. The attributes of classes might be restricted. For example, a class may have an multiple attributes of similar kind (consider a person has many houses). A class may not have two attributes at the same time (consider a person cannot be man and woman at the same time).
- Instance. An instance is an individual of an ontology. For example, "John is a person", where "person" is a class.

14.2.3 Knowledge and Logic

Knowledge should be stored in such a way that new knowledge can be derived from existing knowledge, and there should be a standardized way of deriving new knowledge from existing knowledge. This introduces the important concept of logic.

Propositional logic is the fundamental of the logic that we know of today. In propositional logic, knowledge is represented by simple facts (simple proposition), and facts connected with AND, OR, NOT, IF THEN, IF AND ONLY IF (compound proposition).

First-order logic (FOL) is the most commonly used logic as of today. In FOL, quantifiers and logic connectives are introduced, including universal quantification \forall , existential quantification \exists , conjunction, disjunction and negation. A formal way of representing logic expressions are defined. For example, using the following to represent "all humans are mortal"

 $\forall x (\operatorname{Human}(x) \to \operatorname{Mortal}(x))$

where CLASS(x) is equivalent of a proposition "x belongs to CLASS", which can be either true or false. The above proposition says "for any item, if that it belongs to human is true, then that it belongs to mortal must be true". Given the current biology development, this proposition is still true.

Description logic (DL) is a subset of the first-order logic. DL is specifically designed to represent and reason about ontologies and their concepts, relations, and instances. It focuses on capturing the semantics of structured knowledge with a restricted set of constructs and reasoning capabilities. The idea of semantic web builds on top of DL. Some key features of DL are:

- Defining classes and subclasses
- Defining properties (roles) and their domain and range:
- Defining restrictions on classes
- Using existential and universal quantifiers

Semantic web is essentially an implementation of DL on computers. As can be seen later, many tools used in semantic web, such as RDF/RDFS and OWL, are essentially practicing DL in a machine-readable way.

15

Semantic Web Architecture

CONTENTS

15.1	Unifori	m Resource Identifier (URI)	115
15.2	Resour	ce Description Framework (RDF)	117
	15.2.1	Triple Representation	118
	15.2.2	Multi-valued Relation and Blank Node	119
	15.2.3	Lists	120
	15.2.4	Reification	120
	15.2.5	Converting RDB to RDF	122
	15.2.6	Conclusion	122
15.3	RDF S	chema (RDFS)	123
	15.3.1	RDFS Motivation	123
	15.3.2	RDF Versus RDF/RDFS via an Example	124
	15.3.3	RDFS Expanded Class and Properties	125
	15.3.4	Semantics inside RDF/RDFS	126
15.4	SPARG	QL Protocol and RDF Query Language (SPARQL)	126
15.5	Web O	ntology Language (OWL)	127

Semantic web uses URI to identify an existing name or resource, RDF/RDFS and OWL to store new information, and SPARQL to query from the knowledge base. The semantic web stack is already given earlier in Section 14.1.4. More details are introduced in this chapter.

15.1 Uniform Resource Identifier (URI)

Humans uses symbol and concept to help interpret things and link to objects in real life. This is demonstrated by semiotic triangle as shown in Fig. 15.1. The interpretation of "apple" is used as an example. The concept is shared among humans in the form of knowledge.

Uniform resource identifier (URI) is the way to identify a resource of information or knowledge. URI is unique from resource to resource.

For example, DBpedia gives concept of the apple here: dbpedia.org/page/Apple. This is an URL to the web page that contains information of the apple. This

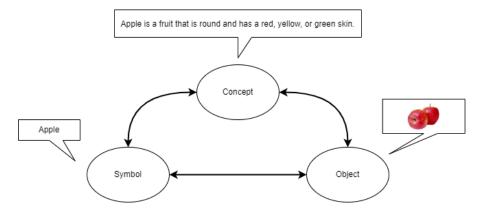


FIGURE 15.1

Semiotic triangle. The symbol is an "representation" of a category of items. The concept is the abstracted general properties of an item.

URL is an example of URI. In this example, the information can be accessed by HTTP call to the URL.

In semantic web, the URI of a resource identifies an item or a class. Using the URI, the representation (meta data) of the item/class can be accessed. The representation is like the symbol in Fig. 15.1, and although it is not a specific instance of the item, it is usually sufficient for us to use the symbol to discuss the item. This is shown in Fig. 15.2.

URI shall contain at least two pieces of information: the address (locator), and the identity (name). URI is often ASCII encoded, but it is possible to extend to unicode. The generic syntax is given below, and the URL is a subset of the syntax.

scheme:[//authority]path[?query][#fragment]

where

- scheme specifies the protocol used to access the resource. Common schemes include http, https, ftp, mailto, file, and data. The scheme is followed by a colon:.
- authority component is optional and typically includes the user information, host, and port, separated by an at sign @ and a colon:, respectively. The authority is preceded by a double forward slash //.
- path identifies the specific resource within the context of the scheme and authority. It is a sequence of segments separated by forward slashes /.
- query component is optional and provides additional information that the resource can use for processing. It is a series of key-value pairs separated by an ampersand &. The query starts with a question mark?.

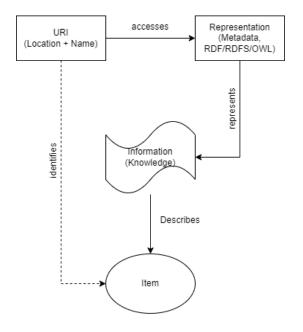


FIGURE 15.2 Identification flowchart.

• fragment component is optional and allows for the identification of a specific part or section within the resource. It is typically used with HTML documents to indicate a specific anchor or location within the page. The fragment starts with a hash sign #.

More details can be found on

https://www.rfc-editor.org/rfc/rfc3986.html#section-3.1

which happens to be a good example of an URI in https scheme.

15.2 Resource Description Framework (RDF)

The representations of the knowledge shown in 15.2 need to be presented in a consistent way. In practice, RDF is introduced to represent knowledge.

Notice that RDF itself is not a programming language. It is the structure using which information is stored. Therefore, it needs a markup language to host the RDF framework. Commonly used markup language the following. There are tools to automatically transfer them from one to another.

• RDF/XML: has good compatibility with old machines.

- Terse RDF Triple Language (Turtle): easy to use, human-readable.
- JSON for Linked Data (JSON-LD): popular in web applications and APIs where JSON-based format is required.
- N-Triples: simple, machine-readable format for data exchange between tools

Turtle is very self-explanatory. Unless otherwise mentioned, Turtle is used in this notebook.

15.2.1 Triple Representation

RDF uses "node-edge-node" triples to represent knowledge. The triple refers to the combination of:

- Subject
- Predicate
- Object

For example, consider representation of knowledge "Einstein was born in Ulm". In RDF, "Einstein" is the subject of the knowledge, and "Ulm" the object. The predicate is "has birthPlace", where "birthPlace" is a property assigned to "Einstein". The graph representation is given by Fig. 15.3. The

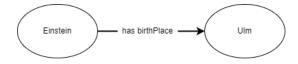


FIGURE 15.3

Graph representation of triple for knowledge "Einstein was born in Ulm".

RDF is given as follows.

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbr: <http://dbpedia.org/resource/> .
```

dbr:Albert_Einstein dbo:birthPlace dbr:Ulm .

where Albert_Einstein and Ulm are defined in dbpedia.org/resource/ as name and place, and birthPlace in dbpedia.org/ontology/ as an attribute, respectively. To give more details:

- Qprefix keyword is used to define prefixes for namespaces, making it easier to write URIs.
- dbr:Albert_Einstein is the subject, representing Albert Einstein as a resource in the DBpedia namespace.

- dbo:birthPlace is the predicate, representing the "birthPlace" property from the DBpedia ontology.
- dbr:Ulmis the object, representing the city of Ulm as a resource in the DBpedia namespace.
- . indicates the end of a statement.

We use

```
<object> <predicate> <subject> .
```

to claim a statement, and

```
<object> <predicate1> <subject1>; <predicate2> <subject2>; <predict3> <
    subject3> .
```

to assign multiple predicates to a subject to avoid repeating the same object in statements.

15.2.2 Multi-valued Relation and Blank Node

It is possible to use multi-valued relations and blank nodes to enforce combining of information. Consider an example given in Fig. 15.4, where the graph is used to demonstrate a lecture taking place at a specific room at given time slot. What if the lecture takes place twice a week, each time at a different location? With the help of multi-valued relations and blank nodes, this can be demonstrated clearly as shown in Fig. 15.5.

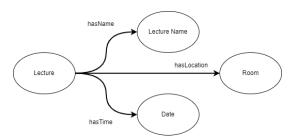


FIGURE 15.4

An example that demonstrate when and where a lecture takes place.

Turtle and other markup languages provides syntax for creating blank nodes that looks like the following.

To refer to a blank node, it is also possible to give the blank node a name. The syntax looks like the following.

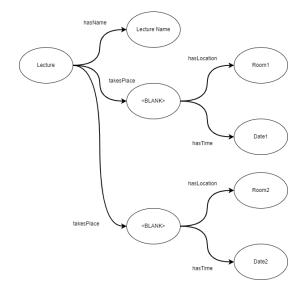


FIGURE 15.5

An example that demonstrate a lecture taking place at multiple locations and time slots using multi-valued relations and blank nodes.

```
<subject1> <predicate1> _:<blank-node-name> .
_:<blank-node-name> <predicate2> <object2> .
_:<blank-node-name> <predicate3> <object3> .
```

where _:<black-node-name> is used to declare a blank node and assign it a name.

15.2.3 Lists

Lists help to make the code clean and readable. There are two types of lists, the container (open list, extendable) and the collections (closed list, fixed). Container is helpful to handle the situation given in Fig. 15.6. Notice that the blank node has a type rdf:Seq. This tells that the items stored in the container follows an ordered set. There are other container types, such as rdf:Bag (unordered set) and rdf:Alt (alternatives of elements; only one element is relevant for the application).

The collection, on the other hand, defines a closed list as shown by Fig. 15.7. In turtle, this can be done by using nested [] iteratively. A short cut is to use (), with the items in the collection listed down in the bracket as follows.

```
<subject>  ct1> <object2> <object3>) .
```

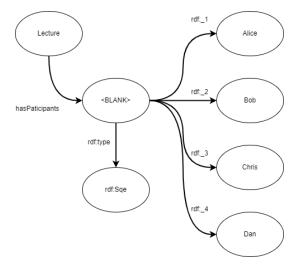


FIGURE 15.6

An example of a container.

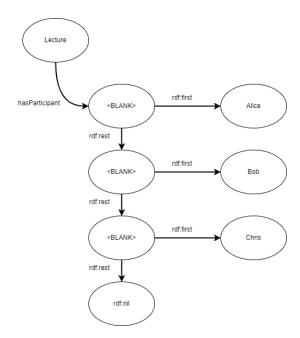


FIGURE 15.7

An example of a collection.

15.2.4 Reification

RDF permits interleaving of statements, i.e., to make a statement about another statement. For example, consider "Alice says that Bob ate the cake". This example contains nested statement, where "Bob ate the cake" is a statement, and "Alice says ..." is a statement on that statement. RDF reification follows Fig. 15.8.

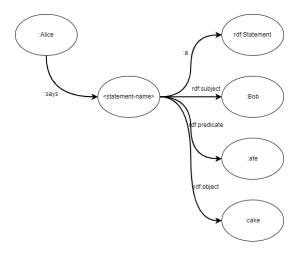


FIGURE 15.8

An example of RDF reification.

15.2.5 Converting RDB to RDF

For small-scale relational database, it is possible to convert it to semantic web RDF manually. For example, consider an RDB for all the books in a study. There are three tables in the database, for books, authors and publishers, respectively. Each table contains a few dozens of entries. The RDB can be converted to RDF as shown in Fig. 15.9. It can be realized very easily, simply by defining everything as nodes and stack triples for all relationships.

However, when comes to a large and complicated database, converting it to RDF manually would consume too much labor. Several ways to systematically do the conversion have been proposed. Details are not covered here. See [4] for more details.

15.2.6 Conclusion

In summary, RDF, different from a plain XML or JSON which can also store independent classes, defines not only the objects (nodes) themselves but also the relationships among objects. This is essentially how RDF differs from a

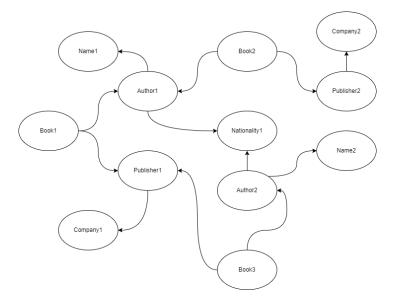


FIGURE 15.9

Semantic web of a few books, and their authors and publishers.

conventional key-value based NoSQL. RDF can be easily scaled up as nodes with the same name naturally merge together.

The underlying mechanism behind RDF, such as how machine stores graphical database including nodes and relationships, and how it enables query using SPARQL, is out of the scope of this notebook. In short, there are several ways to do that, for example by transforming the graph links into multiple small tables, etc. Different approaches may have their pros and cons. Details are neglected here.

15.3 RDF Schema (RDFS)

RDFS enforces schema to the RDF model. By using RDF/RDFS, a more consistent and semantic RDF model can be achieved compared with using RDF along.

15.3.1 RDFS Motivation

RDF is flexible. It is so flexible that sometimes it becomes difficult to maintain consistency of the RDF model, let alone performing sophisticated reasoning from it. RDFS, also known as RDF vocabulary description language, enforce

schema to the RDF model by adding more "meta information" which builds more connections among the nodes.

RDFS expands the vocabulary of RDF. It introduces the concepts of "class" and "subclass" to RDF. It provides built-in predefined classes such as rdfs:Literal, rdfs:Resource, rdfs:Datatype, etc., and enforce the nodes to be linked to these classes. RDF already defines rdf:Property. RDFS further expands the properties and relations. All above makes the RDF modeling more consistent and semantic.

15.3.2 RDF Versus RDF/RDFS via an Example

The following example applies RDF versus RDF/RDFS on the same statement. Consider statement "banana is yellow, apple is red, and orange is orange color". In a RDF implementation, this would be simply be Fig. 15.10 green-colored elements. It is a disconnected graph, where the three statements are completely independent.

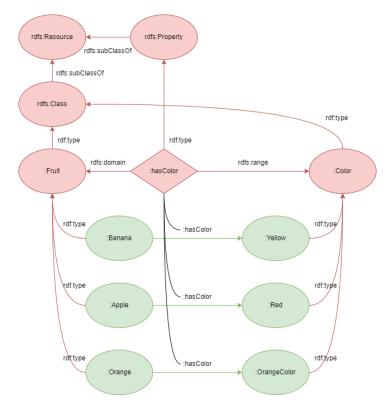


FIGURE 15.10

Semantic web of a fruits, with the green color RDF and green + red RDF/RDFS implementations.

In a RDF/RDFS implementation, class/subclass and property are enforced in the RDF model, with property domain and range specified. All classes eventually root to rdfs:Resource. This is shown by the green + red color in Fig. 15.10. The corresponding Turtle is

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix example: <http://example.org/> .
# Define classes (optional; they are implicit)
rdfs:Class rdfs:subClassOf rdfs:Resource .
rdf:Property rdfs:subClassOf rdfs:Resource .
example:Fruit rdf:type rdfs:Class;
rdfs:subClassOf rdfs:Resource .
example:Color rdf:type rdfs:Class;
rdfs:subClassOf rdfs:Resource .
# Define properties
example:hasColor rdf:type rdf:Property ;
rdfs:domain example:Fruit;
rdfs:range example:Color .
# Define fruits and colors
example:Banana rdf:type example:Fruit .
example:Apple rdf:type example:Fruit .
example:Orange rdf:type example:Fruit .
example:Yellow rdf:type example:Color .
example:Red rdf:type example:Color .
example:Green rdf:type example:Color .
# Define relationships between fruits and colors
example:Banana example:hasColor example:Yellow .
example:Apple example:hasColor example:Green .
example:Orange example:hasColor example:Orange
```

where only the last 3 lines would be used in RDF implementation. It can be seen from this example that RDF/RDFS uses more "complicated" structure to enforce schema of the model. In this example, "a fruit has color of a color" is enforced. Notice that predicate rdf:type can be used interchangeably with turtle keyword a. They both declares a instance of a class.

15.3.3 RDFS Expanded Class and Properties

Special classes and properties are defined, some of which already used in the above example. Some commonly seen ones include:

• rdfs:Resource

• rdfs:Class

• rdf:Property

• rdfs:subClassOf

• rdfs:subPropertyOf

• rdfs:domain

• rdfs:range

Commonly seen further properties of a node include the following. They help to make the RDF model more human-readable.

- rdfs:seeAlso points to a resource where a detailed explanation of the node can be found.
- isDefinedBy defines the relation of a resource to its definition.
- rdfs:comment points to text comment.
- rdfs:label assign a more human-readable name to the node.

More details of the latest version of RDF/RDFS defined classes and properties are given by W3C and can be found at w3.org/TR/rdf-schema/.

15.3.4 Semantics inside RDF/RDFS

The semantics in RDF/RDFS, such as Fig. 15.10, is stored in the format of connections linked via properties, with domain and range specified. The name of a class or a property makes no sense to a machine when it is doing reasoning and query. The connection matters.

The following gives some intuitive examples for semantic reasoning.

- Via class inheritance. In Fig. 15.10, if apple is a subclass of fruit, and fruit a subclass of plant, then apple must also be a subclass of a plant. There is an "hidden arrow" pointing from the apple to the plant.
- Via property domain and range. In Fig. 15.10, if an unknown object "has—Color", then its object must be a fruit, and the subject a color. There are associated "hidden arrows", even if these arrows are not specified in the programming.
- Via property inheritance. Consider two properties, "isMotherOf" and "is-ParentOf". Both properties have the same domain and range of "person", and "isMotherOf" is a sub property of "isParentOf". In this case "A is the mother of B" can derive "A is the parent of B".

Though not given by the input directly, the hidden information exists in the semantic web and should be accessible from API

 ${\rm ``nobreak'}$

15.5 Web Ontology Language (OWL)

16

Semantic Web Practice

CONTENTS

"nobreak

16.1 Triplestore

Bibliography

- [1] The common layered semantic web technology stack.
- [2] The R project for statistical computing.
- [3] Rstudio.
- [4] Franck Michel, Johan Montagnat, and Catherine Faron Zucker. A survey of RDB to RDF translation approaches and tools. PhD thesis, I3S, 2014.