Lu Sun, and many more.

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

Foreword **xi**

Preface **xiii**

List of Figures **xv**

List of Tables **xix**

I Probability **1**

1 Randomness **3**
- 1.1 Randomness and Stochasticity 3
  - 1.1.1 Random Experiment 4
  - 1.1.2 Sample Space 4
  - 1.1.3 Event 5
- 1.2 Probability 5
  - 1.2.1 Classical Probability 6
  - 1.2.2 Geometric Probability 6
  - 1.2.3 Empirical Probability 8
  - 1.2.4 Axioms 8
  - 1.2.5 Conditional Probability 9

2 Distributions of Random Variables **13**
- 2.1 Discrete and Continuous Random Variables 13
  - 2.1.1 Discrete Random Variable 14
  - 2.1.2 Continuous Random Variable 14
  - 2.1.3 Distribution of Derived Variable 15
  - 2.1.4 Expectation 15
  - 2.1.5 Variance and Standard Deviation 16
  - 2.1.6 Moment 17
- 2.2 Joint Distribution 18
  - 2.2.1 Joint Probability 19
  - 2.2.2 Conditional Distribution 20
  - 2.2.3 Covariance and Correlation 21
- 2.3 Parameter Estimation 23
  - 2.3.1 Posteriori Mean 23
  - 2.3.2 MLE and MAP 24

2.3.3 Parameter Estimation Benchmark 25
2.4 Important Theorems 27
2.4.1 Law of Large Numbers 27
2.4.2 Central Limit Theorem 27

3 Commonly Seen Distributions 29
3.1 Bernoulli and Binomial Distributions 29
3.2 Normal Distribution 31
3.3 Poisson Distribution 32
3.4 Exponential Distribution 34
3.5 Uniform Distribution 35
3.6 Cauchy Distribution 35
3.7 Γ, χ^2, β and F Distributions 35
3.7.1 Γ Distribution 36
3.7.2 χ^2 Distribution 37
3.7.3 β Distribution 38
3.7.4 F-Distribution 38
3.8 Student's t-Distribution 39

II Statistics 41

4 Basic Statistics 43
4.1 Two Types of Statistical Problems 43
4.2 Sampling 44
4.3 Moments Estimation 47
4.3.1 Mean and Variance 47
4.3.2 Other Moments 49
4.4 Confidence Interval Estimation 49

5 Statistical Hypothesis Testing 53
5.1 Hypothesis 53
5.1.1 Motivating Examples 53
5.1.2 Hypothesis Categories 55
5.1.3 Acceptance Region and Rejection Region 55
5.2 Commonly Seen Hypothesis Tests 56
5.2.1 The Mean of a Normal Distribution 56
5.2.2 The Variance of a Normal Distribution 58
5.2.3 The Comparison of Means of Two Normal Distribution 59
5.2.4 Exponential Distribution 59
5.2.5 Binomial Distribution 59
5.2.6 Poisson Distribution 60

6 Bayesian Methods 61

III Tools 63

7 R (Part I: Basics) 65
- 7.1 Brief Introduction . . . 66
  - 7.1.1 Installation . . . 66
  - 7.1.2 R Packages Management . . . 66
- 7.2 Basic Data Types and Syntax . . . 68
  - 7.2.1 Variable Names . . . 69
  - 7.2.2 Different Types of Scalars . . . 69
  - 7.2.3 Vectors . . . 73
  - 7.2.4 Matrices . . . 78
  - 7.2.5 Lists . . . 82
  - 7.2.6 Conditional and Loop Statements . . . 84
  - 7.2.7 User-Defined Function . . . 86
- 7.3 Data Frames . . . 86
  - 7.3.1 Data Importing . . . 86
  - 7.3.2 Data Accessing . . . 87
  - 7.3.3 Filtering . . . 90
  - 7.3.4 Data Frame from Integration of Data . . . 91
- 7.4 Basic Data Visualizations Using `qplot()` . . . 92
- 7.5 Advanced Data Visualizations Using `ggplot()` . . . 93
  - 7.5.1 Grammar of Graphics . . . 95
  - 7.5.2 Data, Aesthetics and Geometries Layers . . . 95
  - 7.5.3 Statistics Layers . . . 97
  - 7.5.4 Facets Layers . . . 99
  - 7.5.5 Coordinates Layers . . . 102
  - 7.5.6 Themes Layers . . . 103

8 R (Part II: Practice) 107
- 8.1 Data Preparation . . . 107
  - 8.1.1 Data Type Conversion . . . 107
  - 8.1.2 Handling Missing Data . . . 109
- 8.2 Connectivity with Data Sources . . . 112

9 Python (Part I: Basics) 113
- 9.1 NumPy . . . 113
- 9.2 SciPy . . . 116
- 9.3 Matplotlib and Seaborn . . . 116
  - 9.3.1 Matplotlib . . . 116
  - 9.3.2 Seaborn . . . 117
- 9.4 Pandas . . . 119
  - 9.4.1 Data Importing . . . 120
  - 9.4.2 Series and Data Frame . . . 122

10 Python (Part II: Python for Data Science) 123

- 10.1 Quick Review . . . 124
  - 10.1.1 AI Pipeline . . . 124
  - 10.1.2 Data Preparation and Model Evaluation . . . 125
  - 10.1.3 Commonly Seen ANN Use Cases . . . 126
  - 10.1.4 Computer Vision . . . 126
  - 10.1.5 Natural Language Processing . . . 126
- 10.2 TensorFlow . . . 126
  - 10.2.1 TensorFlow Basics . . . 127
  - 10.2.2 Classification and Regression . . . 127
  - 10.2.3 Computer Vision . . . 132
  - 10.2.4 General Sequential Data Processing . . . 132
  - 10.2.5 Natural Language Processing . . . 132
  - 10.2.6 TensorFlow on Different Platforms . . . 132
- 10.3 PyTorch . . . 132
  - 10.3.1 PyTorch Basics . . . 132
  - 10.3.2 Classification and Regression . . . 132
  - 10.3.3 Computer Vision . . . 132
  - 10.3.4 General Sequential Data Processing . . . 132
  - 10.3.5 Natural Language Processing . . . 132
  - 10.3.6 PyTorch on Different Platforms . . . 133

IV Semantic Web 135

11 Semantic Web Basics 137

- 11.1 Web of Data . . . 138
  - 11.1.1 Web 1.0 and 2.0 . . . 138
  - 11.1.2 Web 3.0 . . . 139
  - 11.1.3 Semantic Web Vision . . . 140
  - 11.1.4 Semantic Web Stack . . . 140
  - 11.1.5 Semantic Web Limitations and Challenges . . . 144
- 11.2 Ontology . . . 145
  - 11.2.1 Philosophy Perspective . . . 145
  - 11.2.2 Semantic Web Perspective . . . 145
  - 11.2.3 Ontology Types and Categories . . . 146
- 11.3 Logic . . . 147
  - 11.3.1 Different Semantics from the Same Syntax . . . 147
  - 11.3.2 Logic Framework . . . 148
  - 11.3.3 Logical Expression . . . 150
  - 11.3.4 Logical Equivalence . . . 151
  - 11.3.5 Logical Reasoning . . . 152

12 Resource Description Framework **155**
- 12.1 Uniform Resource Identifier 155
- 12.2 Resource Description Framework (RDF) 157
  - 12.2.1 Triple Representation 157
  - 12.2.2 Multi-valued Relation and Blank Node 158
  - 12.2.3 Lists 160
  - 12.2.4 Reification 160
  - 12.2.5 Converting RDB to RDF 162
- 12.3 RDF Schema 163
  - 12.3.1 RDF Versus RDF/RDFS 163
  - 12.3.2 RDFS Expanded Class and Properties 165
  - 12.3.3 Semantics inside RDF/RDFS 166
- 12.4 SPARQL Protocol and RDF Query Language (SPARQL) . . 166
  - 12.4.1 SPARQL for Basic Query 167
  - 12.4.2 SPARQL for Advanced Operations 169
  - 12.4.3 Default Graph and Named Graph 171
  - 12.4.4 SPARQL Programming Returns 171
  - 12.4.5 Underlying Data Structure of Triplestores 172

13 Web Ontology Language **175**
- 13.1 RDF/RDFS Limitations 175
- 13.2 OWL Vision 177
- 13.3 OWL Basic Syntax 178
- 13.4 OWL Advanced Syntax 179
- 13.5 Semantic Web with Rules 184

14 Semantic Web Practice **185**
- 14.1 Ontological Engineering 185
- 14.2 Ontology Design 186
  - 14.2.1 General Tasks 186
  - 14.2.2 Ontology Design Basics 187
  - 14.2.3 Semantic Web Design for Enterprise 188
- 14.3 Linked Data Engineering 189
  - 14.3.1 Web of Data 190
  - 14.3.2 Semantic Search in Semantic Web 192
- 14.4 Triplestore 192
- 14.5 Example: Semantic Web for Home Assets 193
  - 14.5.1 Define Classes Hierarchy 195
  - 14.5.2 Define Properties Hierarchy 195
  - 14.5.3 Add OWL 195
  - 14.5.4 Data Retrieval Examples 195
- 14.6 Reference: Commonly Used Namespace 195

Bibliography **197**

Foreword

If software and e-books can be made completely open-source, why not a notebook?

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 the end of August when the results of Gaokao (National College Entrance Examination of China, annually held in July) were released, people from photocopy shops would start selling notebooks photocopies that they claimed 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 read than the textbooks. I guess we cannot blame the top scorers for being so smart that they sometimes made 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 were probably my schoolmates. Why would I pay money to a stranger in a photocopy shop for my friends' notebooks, rather than requesting a copy from them directly?

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

To make a difference, I enforced myself reading articles beyond my comfort zone, which ended up motivating me to take notes to consolidate the knowledge. I used to work with 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 dozens of these notebooks on my bookshelf. Eventually, it came to me: why not digitizing them, making them accessible online and open-source and letting everyone read and edit it?

Similar with most open-source software, this notebook does not come with any "warranty" of any kind, meaning that there is no guarantee that everything in this notebook is correct, and it is not peer reviewed. **Do NOT cite this notebook in your academic research paper or book!** If you find anything helpful here with your research, please trace back to the origin of the knowledge and confirm by yourself.

This notebook is suitable as:

- a quick reference guide;
- a brief introduction for beginners of an area;
- a "cheat sheet" for students to prepare for the exam or for lecturers to prepare the teaching materials.

This notebook is NOT suitable as:

- a direct research reference;
- a replacement of the textbook.

The notebook is NOT peer reviewed, thus is more of a notebook than a book. It is meant to be easy to read, not to be comprehensive and very rigorous.

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, should you decided to trace the origin of the knowledge.

Some of the figures in this notebook are plotted using Excalidraw, a very convenient tool to emulate hand drawings. The Excalidraw project can be found on GitHub, *excalidraw/excalidraw*. Other figures may come from MATLAB, R, Python, and other computation engines. The source code to reproduce the results are intended to be included in the same repository of the notebook, but there might be exceptions.

This work might have benefited from the assistance of large language models, which are used exclusively for editing purposes such as correcting grammar and rephrasing sentences, without introducing new content, generating novel information, or changing the original intent of the text.

Preface

This notebook introduces probability and statistics, which are one of the most important, fundamental and commonly used skills in almost all science and engineering subjects.

In Part I of the notebook, probability theory is introduced. Probability theory studies how likely an event is to occur. It offers rich models and tools to 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 statistical hypotheses, and draw conclusions and predictions.

In Part III of the notebook, commonly used software and toolkit for statistical analysis and data science are introduced. Different from Parts I and II which focus more on theory, Part III focuses more on the tools to solve practical problems.

Artificial intelligence has become notably popular in recent years for data analysis. There is a separate notebook introducing AI, and hence in this notebook AI-based data analysis is not introduced in detail, but only briefly covered.

As a bonus, in Part IV of the notebook the semantic web is introduced. Semantic web is both a philosophical concept and a data framework defined under Web 3.0. It does not necessarily contribute to solving a specific data science problem, but rather a concept that allows convenient and flexible information storage and exchange, thus serving as a probable backbone to improve efficiency in data analysis.

Key references of this notebook are listed below.

Probability and statistics:

- 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.

Data science:

- 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

List of Figures

1.1 Sample space of two people arriving at the park from 8:00 AM to 9:00 AM. 7

2.1 Demonstration of PDF with different skewness. 18
2.2 Demonstration of PDF with different excess kurtosis. 19

3.1 Poisson distribution with different λ. 32
3.2 Poisson distribution approximation using binomial distribution. 33
3.3 Exponential distributions with different λ. 34
3.4 Cauchy Distribution. 36
3.5 Gamma Distribution. 37
3.6 The χ^2 Distribution. 38

4.1 Sample with replacement, population size $N = 100$, sample size $0 < M \leq 500$. 45
4.2 Sample without replacement, population size $N = 100$, sample size $0 < M \leq 500$. 45
4.3 Sample with replacement, population size $N = 10000$, sample size $0 < M \leq 500$. 46
4.4 Sample without replacement, population size $N = 10000$, sample size $0 < M \leq 500$. 46

5.1 Distribution of μ as a function of $\bar{X}$ and σ^2. 57

7.1 RStudio's graphical interface for package management. 68
7.2 A demonstration of a matrix in R. 78
7.3 A demonstration of using `matplot` to plot trends. 81
7.4 Plot of penalty success rate of the 3 players in 10 matches. . . 83
7.5 Plot of average point gained per throw attempt for the 3 players in 10 matches. 83
7.6 A demonstration of `qplot`. 93
7.7 A demonstration of `qplot` on mortgage price data frame. . . 94
7.8 A second demonstration of `qplot` on mortgage price data frame. 94
7.9 Multiple layers in chart design. 96

7.10 An example of box plot of the mortgage price data frame using `ggplot()` and `geom_boxplot()`. 98
7.11 An example of using `geom_smooth()` for scatter point fitting. 99
7.12 An example of histogram plot of house price per unit area in different regions in a single plot. 101
7.13 Use facets to plot the histogram of price per unit are of the house in different regions (subplots in rows). 102
7.14 Use facets to plot the histogram of price per unit are of the house in different regions (subplots in columns). 103
7.15 Add coordinates layer using `xlim()` and `ylim()`. 104
7.16 Add coordinates layer using `coord_cartesian()`. 104
7.17 Mortgage price chart with theme. 106

9.1 Plot Fibonacci series as scatter plot. 117
9.2 Histogram plot using Seaborn. 118
9.3 Count plot using Seaborn. 118
9.4 Box plot using Seaborn. The box gives IQR. The bars below and above the box give $Q_1 - 1.5 \times \text{IQR}$ and $Q_3 + 1.5 \times \text{IQR}$, respectively. The dots are outliers. 119
9.5 The simplest data frame importing using pandas. 121
9.6 Specifying index column and reading only selected columns using pandas. 121

11.1 Semantic web stack [2]. 141
11.2 A demonstrative example of ontology level using control engineering. 146

12.1 Semiotic triangle of Apple. 156
12.2 Graph representation of triple for knowledge "Einstein was born in Ulm". 158
12.3 An example that demonstrate when and where a lecture takes place. 159
12.4 An example that demonstrate a lecture taking place at multiple locations and time slots using multi-valued relations and blank nodes. 159
12.5 An example of a container. 160
12.6 An example of a collection. 161
12.7 An example of RDF reification. 161
12.8 Semantic web of a few books, and their authors and publishers. 162
12.9 Semantic web of fruits and their colors, with RDF implementation in green RDF/RDFS in red. 164

13.1 An RDF model that demonstrates "animal eats food". 176

14.1 Semantic web design workflow. 189
14.2 Linked Data example: web browsing. 190

14.3 The linked open data cloud from *lod-cloud.net*. 191
14.4 An example of an RDF model in GraphDB that describes house assets. This is only a demonstration graph and the information inside is artificial and not true. 194

List of Tables

7.1 Commonly used data types. 70
7.2 Numerical calculations. 72
7.3 Logical comparisons. 72
7.4 String operations. 72
7.5 Probability related operations. 73
7.6 Statistics related functions. 73
7.7 Commonly used commands for data frame exploration. 87
7.8 Commonly used `ggplot` geometries and statistics layers commands. A full list can be found at [4]. 97
7.9 Functions that fit smooth lines to scatter points. 100

11.1 Numerical calculations. 152

14.1 Commonly used name spaces in RDF models. URI is neglected since they can be easily found online. 195

Part I

Probability

1

Randomness

CONTENTS

1.1 Randomness and Stochasticity 3
  1.1.1 Random Experiment 4
  1.1.2 Sample Space 4
  1.1.3 Event 5
1.2 Probability 5
  1.2.1 Classical Probability 6
  1.2.2 Geometric Probability 6
  1.2.3 Empirical Probability 7
  1.2.4 Axioms 8
  1.2.5 Conditional Probability 9

This chapter introduces the basic concepts, axioms and theorems in probability theory.

1.1 Randomness and Stochasticity

The terms "random" and "stochastic" are used to describe variables or models whose measurements or outcomes are not precisely predictable.

Randomness typically refers to the case where a variable (as in "random variable") is not predictable. Its value is not known precisely until it is measured. The chance of it taking particular values may follow some patterns known as the statistical properties of the variable. For example, the result of tossing a coin is a random variable with the following statistic property. It is either head ($X = 1$) or tail ($X = 0$), each with a 50% chance. The precise value remains unknown until the coin is tossed.

Stochasticity typically refers to the case where the outcome of a process (as in "stochastic process") is not predictable. This is likely caused by the incomplete modeling or random disturbance of the system. As a result, the process becomes non-deterministic, i.e., the output of the system cannot be precisely determined by the model and the input. A stochastic process can still be described by a parametric model, but with some unknowns (usually in the form of random variables) in the equations.

This part of the notebook mainly studies random variables. Stochastic process and control of stochastic systems are introduced elsewhere in control system related notebooks.

1.1.1 Random Experiment

Experiments are among the most important activities in science and engineering. Very often, experiments are used to verify theories. In these cases, experiments are carefully designed so that their outcomes are deterministic and predictable as long as the theory is correct. By observing the results of the experiments matching the prediction of the theory, we build confidence in the theory.

However, this notebook is concerned with another type of experiment where the result contains randomness due to the lack of information or incomplete modeling. Such experiments include tossing a coin, predicting the GDP of a country next year, etc. These experiments are known as **random experiments**. The result of a random experiment can still be meaningful because it reflects some insights of the system. The challenge is to design the experiments so that useful information can be revealed from the experiment results efficiently.

1.1.2 Sample Space

A set S that consists of all possible outcomes of a random experiment is called a **sample space**.

Cardinality refers to the number of elements in a set. Depending on the cardinality of a sample space, the space can be categorized as follows. If a sample space has a finite number of elements, it is called a **finite sample space**. Otherwise, it is called an **infinite sample space**. If the elements in an infinite sample space can be mapped to natural numbers, the sample space is also called a **countably infinite sample space**. Otherwise, the sample space is called a **uncountably infinite sample space**. Examples are given below.

- Finite sample space. Randomly pick 5 balls from a bag that contains 50 red balls and 50 green balls. The number of red balls in the 5 picked balls forms a finite sample space, $S = \{0, 1, 2, 3, 4, 5\}$.

- Countably infinite sample space. The number of consecutive "heads" when tossing a coin before the first "tail" forms a countably infinite sample space, $S = \mathbb{N}$.

- Uncountably infinite sample space. Randomly drop a ball into a circle with radius 1. The distance from the ball to the center of the circle forms the sample space $S = [0, 1]$.

Countable Infinity VS Uncountable Infinity

Though both infinity, the total number of natural numbers, i.e., the cardinality of $\mathbb{N}$, is less than the total number of real numbers, i.e., the cardinality of $\mathbb{R}$.

The cardinality of $\mathbb{N}$ is known as $\aleph_0$. It is also the cardinality of all rational numbers. In computing, it is also the cardinality of all computable numbers (i.e., numbers that can be computed to any desired precision by a finite terminating algorithm) and computable functions (i.e., algorithms). The total number of real numbers is known as $\aleph_1$. It is also the cardinality of all irrational numbers and complex numbers, the number of points on a line or in a high dimensional space $\mathbb{R}^n$ where n is a finite integer.

Larger infinity quantities such as $\aleph_2, \aleph_3, \ldots$ are also defined, though they may not have an intuitive explanation as $\aleph_0$ and $\aleph_1$.

Deeper discussion of this topic requires advanced set theory and is not given in this notebook.

Finite sample spaces and countably infinite sample spaces are also known as **discrete sample spaces**, whereas uncountably infinite sample spaces are known as **continuous sample spaces**.

1.1.3 Event

An **event** is defined as a subset $A \subseteq S$, i.e., it is a portion of all possible outcomes. An event may or may not occur depending on the outcome of the experiment. In the special case where A has only one element, the event is also called an **elementary event**, or a **sample**. Notice that all elementary events are mutually exclusive as each of them contains an independent outcome of the experiment. In the special case where $A = S$, the event is also called a **certain event**.

1.2 Probability

Given an experiment and an event, it is unsure whether the event will occur, and **probability** is a measurement of the likelihood that the event is going to occur. For example, if the probability is 50%, it means that the event has an equal chance of happening or not happening.

There are different ways to calculate the probability of events. A rigorous mathematical definition of probability, known as axiomatic approach, is introduced in Section 1.2.4. But before that, classical approach, empirical approach and geometric probability are introduced. One may get confused with which is the "true" definition of probability. As will be shown later, though defined from different angles, they all point to essentially the same concept.

1.2.1 Classical Probability

In the **classical approach**, as a prerequisite all elementary events have the same chance of occurrence, and the total number of elementary events is n. Define an event that contains h elementary events. The **classical probability** of the event is given by h/n.

Calculating h, the cardinality of event A, and n, the cardinality of the sample space S, is the key to solving the classical probability. Permutation and combination are widely used for such calculations. Suppose that there are n distinct objects, and we would like to select $r \leq n$ objects from them and put them into a sequence. The **permutation** defined by

$$nPr = n(n-1)(n-2)\cdots(n-r+1) \tag{1.1}$$

gives the number of possible outcomes. In the special case where $r = n$,

$$nPn = n(n-1)(n-2)\cdots\times 1$$

where $n(n-1)(n-2)\cdots\times 1$ is often denoted by $n!$. We can use that notation to rewrite (1.1) as

$$nPr = \frac{n!}{(n-r)!}$$

In permutation, the order of the selected r objects matters. When the order does not matter, the **combination** defined by

$$\begin{aligned} nCr &= \frac{nPr}{r!} \\ &= \frac{n!}{r!\,(n-r)!} \end{aligned}$$

is used to calculate the total number of outcomes. Notice that nCr is also denoted by $\left(\begin{array}{c} n \\ r \end{array}\right)$.

1.2.2 Geometric Probability

Geometric probability is an extension of classical probability to infinite sample spaces. Similar to classical probability, it assumes that all elementary events share the same probability. The probability of an event can be obtained by measuring the area or volume associated with the event. An example is used to illustrate the use of geometric probability.

Geometric Probability Example

Consider two people trying to meet up at the park, but they forgot to tell each other the time to meet. Both of them will arrive at the park at anytime between 8 : 00 AM and 9 : 00 AM with equal chance, and wait for the other for 15 minutes or until 9 : 00 AM, whichever is earlier, and then will leave the park if the other person does not show up.

Calculate the probability of the two people successfully meeting up.

Since there are infinite number of combinations of the timestamps the two people arrive, we cannot calculate the total number of the sample space, nor the cardinality of the event of them successfully meeting up. Both of them are infinite sets. To solve the problem, consider drawing the sample space in a 2-D plot as shown in Fig. 1.1, where the x-axis and y-axis are the arriving time of the two people, respectively. The shaded area represents the samples where the two would meet up successfully.

Divide the shared area by the total sample space area to get $P = 7/16 = 43.75\%$ which is the probability that the two would meet up successfully.

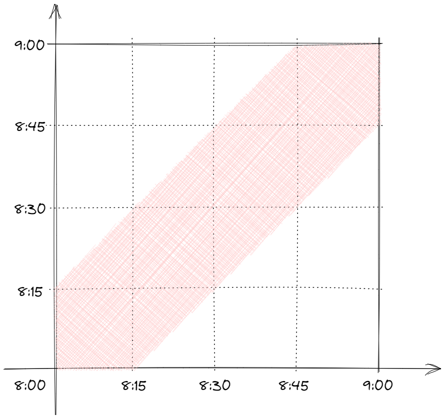


FIGURE 1.1
Sample space of two people arriving at the park from 8:00 AM to 9:00 AM.

Note that in the problem description, the two people arrive at the park between 8 : 00 AM and 9 : 00 AM with equal probability. Here "equal probability" is an important prerequisite for using geometric probability. Otherwise, the probability of the event would not be proportional to the area given in Fig. 1.1. This applies to most use cases of geometric probability.

1.2.3 Empirical Probability

In the **frequency approach**, the experiment can be repeated n times where n is a large number, and the number of instances where the event occurred is recorded as h. The **empirical probability** of the event is obtained by calculating h/n, which should converge to the true probability as n increases.

1.2.4 Axioms

The **axiomatic approach** puts probability theory into mathematical perspective.

Let the sample space be denoted by S, and events by A_i. For simplicity of illustration, assume that S is discrete. Define $P(\cdot)$ as the **probability function** and $P(A_i)$ as the probability of event A_i, subject to the following axioms:

1. For every event A_i, $P(A_i) \geq 0$.
2. For the certain event S, $P(S) = 1$.
3. For mutually exclusive events A_1 and A_2, $P\left(A_1 \cup A_2\right) = P(A_1) + P(A_2)$.

Using the above axioms, a bunch of well-known theorems can be derived, such as

- If $A_1 \subseteq A_2$, $P(A_2 - A_1) = P(A_2) - P(A_1)$.
- For every event A_i, $0 \leq P(A_i) \leq 1$.
- For the impossible event $\varnothing$, $P(\varnothing) = 0$.
- For the complement of an event A', $P(A') = 1 - P(A)$.
- For mutually exclusive events $A_i, i = 1, ..., n$, if $A = \bigcup_{i=1}^{n} A_i$, $P(A) = \sum_{i=1}^{n} P(A_i)$.
- For two events A_1 and A_2, $P\left(A_1 \cup A_2\right) = P(A_1) + P(A_2) - P\left(A_1 \cap A_2\right)$.

With the above axioms, we can revisit classical probability as follows. Assume that a discrete and finite sample space S consists of the following elementary events $A_i, i = 1, ..., n$, i.e.,

$$S = \bigcup_{i=1}^{n} A_i$$

and assume equal probabilities for all the elementary events, i.e., the probability of each event is given by

$$P(A_i) = \frac{1}{n}, i = 1, ..., n$$

Define an event A that is made up of h such elementary events out of A_i. The probability of A can then be calculated by

$$P(A) = \frac{h}{n}$$

where h, n are the cardinality of A and S with respect to the elementary events.

Recall the definition of a certain event. A certain event must have a probability of 1. However, an event with probability 1 is not necessarily a certain event. For example, consider dropping a ball in a circle. The probability of it landing precisely in the center of the circle is 0 (yet possible), and the probability of otherwise is 1 (yet not certain).

Axiomatic approach Versus Frequency Approach

Our intuitive understanding of probability should align with the frequency approach, where the probability of an event should describe how frequently an event occurs if the experiment is repeated many times. The axiomatic approach, though very clearly defined mathematically, seems to disconnect from the intuition.

As will be introduced in a later Section 2.4.1, the law of large numbers actually bridges the axiomatic approach and frequency approach, and guarantees the consistency of the two. As a result, the probability defined in axiomatic approach can also reflect the likelihood of an event happening the similar way the frequency approach does, and there is no gap in between.

1.2.5 Conditional Probability

Consider two events A and B. The **conditional probability** $P(B|A)$ describes the probability of B given that A has occurred. The definition is given below. This definition can be thought of as replacing S with A since A is confirmed to have occurred.

$$P(B|A) \equiv \frac{P(A \cap B)}{P(A)}$$

Or equivalently,

$$P(A \cap B) = P(A)P(B|A)$$

From the above,

$$P(B|A) = \frac{P(B)P(A|B)}{P(A)}$$

which is known as the **Bayes' rule**.

An example of implementing Bayes' rule is given below.

Example of Bayes' Rule

A couple has an equal chance of giving birth to a boy or a girl. Gather a group of couples with 2 children. Randomly pick one couple.

1. Without further information, what is the chance of them having two girls?
2. From an earlier interview, we know that they have at least one girl. What is the chance of them having two girls?
3. From an earlier interview, we know that their first child is a girl. What is the chance of their second child being a girl as well?

In this example, the sample space for a couple is

$$S = \{[\text{Boy Boy}], [\text{Boy Girl}], [\text{Girl Boy}], [\text{Girl Girl}]\} \tag{1.2}$$

each with an equal probability of 25%. For convenience of illustration, define the following events.

- Event A: both children are girls
- Event B: at least one child is a girl
- Event C: the first child is a girl
- Event D: the second child is a girl

Intuitively, we can see that the events are correlated in the following manner.

$$\begin{aligned} A &= C \cap D \\ A, C, D &\subset B \end{aligned}$$

Consider the first question. From (1.2), the chances of the event A happening is obviously

$$P(A) = \frac{1}{4}$$

which is the answer to the first question.

Consider the second question. Using Bayes' rule,

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(A)}{P(B)} \\ &= \frac{1}{3} \end{aligned}$$

where note that $P(A \cap B) = P(A)$ since $A \subset B$. Intuitively, this makes sense. From the interview, it is possible to drop [Boy Boy] from the sample space for this couple. As a result, the probability of them having two girls, $\frac{1}{3}$, is higher than in the first question, $\frac{1}{4}$, where no prior information was available. In this example, the prior information is useful for obtaining a more precise answer.

Consider the third question. Using Bayes' rule,

$$\begin{aligned} P(D|C) &= \frac{P(D \cap C)}{P(C)} \\ &= \frac{P(A)}{P(C)} \\ &= \frac{1}{2} \end{aligned}$$

Note that from (1.2), $P(D) = \frac{1}{2}$. Therefore, $P(D|C) = P(D)$. In this example, the prior information about whether C occurs is not helpful for calculating the probability that D occurs. This makes sense. The genders of both children are independent. The first child, whether a boy or a girl, does not affect the likelihood of the gender of the second child.

If $P(B|A) = P(B)$, or equivalently $P(A \cap B) = P(A)P(B)$, the two events A and B are known as **independent events**. In this case, the prior knowledge of whether A has occurred does not affect the probability of event B, and vice versa.

2

Distributions of Random Variables

CONTENTS

2.1 Discrete and Continuous Random Variables 13
  2.1.1 Discrete Random Variable 13
  2.1.2 Continuous Random Variable 14
  2.1.3 Distribution of Derived Variable 14
  2.1.4 Expectation 15
  2.1.5 Variance and Standard Deviation 16
  2.1.6 Moment 17
2.2 Joint Distribution 18
  2.2.1 Joint Probability 19
  2.2.2 Conditional Distribution 20
  2.2.3 Covariance and Correlation 21
2.3 Parameter Estimation 23
  2.3.1 Posteriori Mean 23
  2.3.2 MLE and MAP 24
  2.3.3 Parameter Estimation Benchmark 25
2.4 Important Theorems 26
  2.4.1 Law of Large Numbers 27
  2.4.2 Central Limit Theorem 27

The definition of random variables has been introduced in the previous chapter. This chapter explores the different types of random variables, their properties, and how they are described mathematically.

2.1 Discrete and Continuous Random Variables

A random variable may be discrete or continuous depending on the sample space of the variable.

2.1.1 Discrete Random Variable

If a random variable X takes only discrete values $x_1, x_2, \ldots$, it is called a **discrete random variable**. The probability of X taking a particular value x is denoted by $P(X = x)$ or simply $f(x)$ if no ambiguity exists. In this case, $P(X = x)$ and $f(x)$ are the probability function (also known as **probability mass function**) of X.

Furthermore, define $P(X \leq x)$ or $F(x)$ as the **cumulative distribution function** of x. It is easy to prove that $F(x)$ is nondecreasing, and $\lim_{x\rightarrow -\infty} F(x) = 0$, $\lim_{x\rightarrow\infty} F(x) = 1$. Also, $F(x)$ "jumps" at each $P(X = x_k) > 0$ and it is continuous from the right.

2.1.2 Continuous Random Variable

A random variable X may also take continuous values in many applications. For example, let X denote the time consumption to finish a task, in which case X is a random variable whose value can be any positive number.

In this case, the probability for X to take a precise value, say finishing a task using precisely 25 minutes 13 seconds 750 milliseconds, is very small (in fact, zero, if the precision approaches infinity). The probability function $P(X = x)$ becomes meaningless. The cumulative distribution function $F(x) = P(X \leq x)$ still makes sense, as it calculates the probability of X within a range rather than at a precise value.

Inspired by this, define **probability density function** (PDF) or $f(x)$ for continuous random variable as follows.

$$f(x) = \frac{d}{dx}F(x)$$

With this definition, $f(x)$ is such a function that

$$P(a < X < b) = \int_a^b f(x)dx$$

gives the probability of X in a range, and

$$\begin{aligned} F(x) &= P(X \leq x) \\ &= \int_{-\infty}^{x} f(\epsilon)d\epsilon \end{aligned}$$

Notice that $f(x)$ itself is not probability. It is $f(x)dx$ accumulating in range $x \in (a, b)$ that forms the probability, hence the name "probability density".

In science and engineering problems, continuous random variables are more common than discrete random variables. In engineering, discrete random variables can sometimes be described by impulse PDF.

2.1.3 Distribution of Derived Variable

Let X be a random variable with PDF $f_X(x)$. Let U be another random variable which is a function of X, $U = \phi(X)$. The PDF of U can be calculated from f_X and ϕ. Details are given below.

For simplicity, assume that $U = \phi(X)$ is an injective function (one-to-one function), and $X = \phi^{-1}(U) = \psi(U)$. In that case, the PDF of U, $g(u)$, can be obtained as follows.

$$g(u) = |\psi'(u)| f(\psi(u))$$

For example, let $U = aX$, $X = \frac{U}{a}$.

$$g(u) = \left|\frac{1}{a}\right| f\left(\frac{u}{a}\right)$$

Let X, Y be two random variables with joint distribution $f(x, y)$. Let $U = X + Y$. The PDF of U can be obtained as follows.

$$g(u) = \int_{-\infty}^{\infty} f(x, u - x)dx \tag{2.1}$$

In the special case where X and Y are independent, $f(x, y) = f_X(x)f_Y(y)$, and (2.1) becomes

$$\begin{aligned} g(u) &= \int_{-\infty}^{\infty} f_X(x)f_Y(u - x)dx \\ &= f_X * f_Y \end{aligned}$$

where $*$ denotes the convolution operator.

2.1.4 Expectation

Given the probability function of a discrete random variable or the PDF of a continuous random variable, a lot of insights can be extracted. Commonly seen measures of a random variable are introduced in this section and the following sections. Usually, they can be calculated from their associated probability functions or PDFs.

In probability and statistics sense, **expectation** (also known as **mean**) describes the average value of a random variable, if the variable is generated many times. For discrete random variable X, the expectation is given below.

$$\mathrm{E}(X) = \sum_{i=1}^{n} x_i P(x_i) \tag{2.2}$$

where $\mathrm{E}(\cdot)$ is used to denote the expectation, and n the cardinality of the sample space. In the case of countably infinite sample space, replace n with ∞ in (2.2). For continuous random variable, it is

$$\mathrm{E}(X) = \int_{-\infty}^{\infty} xf(x)dx \tag{2.3}$$

Expectation is sometimes denoted by μ in literatures.

Some features of expectation calculation are given below.

$$\begin{aligned} \mathrm{E}(cX) &= c\mathrm{E}(X) \\ \mathrm{E}(X+Y) &= \mathrm{E}(X) + \mathrm{E}(Y) \end{aligned}$$

where c is a constant and X, Y are two random variables. These features can be easily derived from (2.2) and (2.3). Furthermore, if X, Y are independent, recall $f(x,y) = f_X(x)f_Y(y)$,

$$\begin{aligned} \mathrm{E}(XY) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyf(x,y)dxdy \\ &= \int_{-\infty}^{\infty} xf_X(x)dx \times \int_{-\infty}^{\infty} yf_Y(y)dy \\ &= \mathrm{E}(X)\mathrm{E}(Y) \end{aligned}$$

2.1.5 Variance and Standard Deviation

Variance and **standard deviation** describe how spread samples are from its expectation. It is defined as follows.

$$\begin{aligned} \mathrm{Var}(X) &= \mathrm{E}\left((X - \mathrm{E}(X))^2\right) &(2.4) \\ &= \mathrm{E}\left(X^2 - 2X\mathrm{E}(X) + \mathrm{E}(X)^2\right) \\ &= \mathrm{E}\left(X^2\right) - \mathrm{E}(X)^2 &(2.5) \\ \mathrm{Std}(X) &= \sqrt{\mathrm{Var}(X)} \end{aligned}$$

Variance and standard deviation are sometimes denoted as σ^2 and σ respectively. Notice that (2.5) also implies that $\mathrm{E}\left(X^2\right) \geq \mathrm{E}(X)^2$, a conclusion used in many lemma derivations.

For continuous random variables, from (2.5) the variance is

$$\mathrm{Var}(X) = \int_{-\infty}^{\infty} (x - \mathrm{E}(X))^2 f(x)dx$$

Some features of variance calculation are given below.

$$\mathrm{Var}(cX) = c^2\mathrm{Var}(X)$$

For independent random variables X and Y,

$$\mathrm{Var}(X \pm Y) \quad = \quad \mathrm{Var}(X) + \mathrm{Var}(Y)$$

Mean and standard deviation can be used to standardize a random variable as follows.

$$X^* \quad = \quad \frac{X - \mu}{\sigma}$$

where μ, σ are the mean and standard deviation of random variable X respectively. The standardized random variable, X^*, has a mean of 0 and standard deviation of 1.

2.1.6 Moment

In mathematics, the **r-th moment** of a continuous function $f(x)$ about c is defined as follows.

$$\mu_n \quad = \quad \int_{-\infty}^{\infty} (x - c)^n f(x) dx$$

By simply saying "moment" without specifying c, $c = 0$ by default. Let $f(x)$ be a PDF. In this sense, the 0-th order and the 1st order moment of a probability density function can be calculated as follows.

$$\mu_0 = \int_{-\infty}^{\infty} f(x) dx = 1$$
$$\mu_1 = \int_{-\infty}^{\infty} x f(x) dx = \mathrm{E}(X)$$

where it can be seen that the 0th and 1st moments of a PDF are 1 and its mean, respectively.

Further more, let $c = \mu_1$ be the mean of the random variable to calculate the 2nd-order central moment μ_2 as follows.

$$\mu_2 = \int_{-\infty}^{\infty} (x - \mu_1)^2 f(x) dx = \mathrm{Var}(X)$$

which is the variance of the random variable.

Using mean and variance to further define standardized moments as shown below. Define $\bar{\mu}_k$ for $k \geq 3$ as follows.

$$\bar{\mu}_k = \frac{\mu_k}{\sigma^k}$$

where

$$\mu_k = \mathrm{E}\left((X - \mu_1)^k\right) = \int_{-\infty}^{\infty} (x - \mu_1)^k f(x) dx$$

$$\sigma^k = (\mu_2)^{\frac{k}{2}} = \left(\int_{-\infty}^{\infty} (x - \mu_1)^2 f(x) dx\right)^{\frac{k}{2}}$$

with μ_1 and μ_2 the mean (1st order moment) and variance (2nd order central moment) of the random variable respectively.

The 3-rd and 4-th order standardized moments are known as the **skewness** and **kurtosis** of the PDF respectively. In some literatures, skewness is denoted by $\gamma_1 = \bar{\mu}_3$, and kurtosis $\gamma_2 = \bar{\mu}_4$.

The skewness γ_1 is a measure of asymmetry of the PDF. When $\gamma_1 > 0$ or positive skew, the distribution has a long tail on the right side of the PDF. When $\gamma_1 < 0$ or negative skew, the distribution has a long tail on the left side. When $\gamma_1 = 0$, the PDF might be symmetric (but not necessarily so). Examples are given in Fig. 2.1.

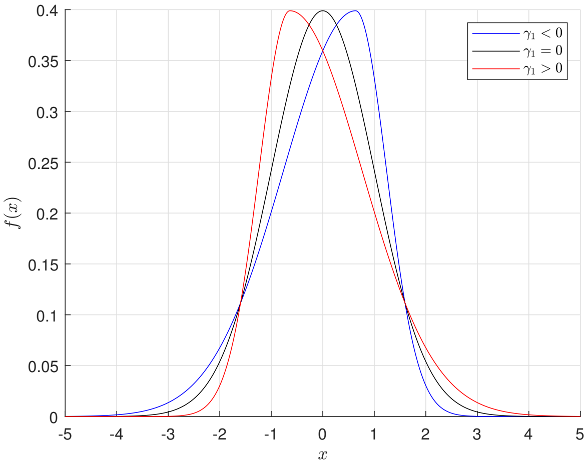


FIGURE 2.1
Demonstration of PDF with different skewness.

The kurtosis γ_2 measures the “tailedness” of a probability distribution, i.e., whether the PDF has heavy tail or thin tail. The normal distribution, which has $\gamma_2 = 3$, is often used as a benchmark. Excess kurtosis is kurtosis subtracting 3, making the normal distribution having the excess kurtosis of 0. A positive excess kurtosis would mean a “heavier” tail than the normal distribution. Examples are given in Fig. 2.2.

2.2 Joint Distribution

Joint distribution describes the distribution of multiple random variable combinations. It is especially useful when these variables are correlated, in which case the joint probability function or PDF can reflect their correlation.

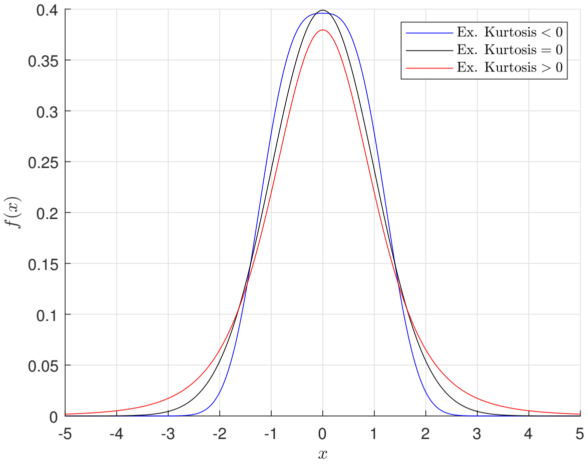


FIGURE 2.2
Demonstration of PDF with different excess kurtosis.

For example, in a system identification problem the unknown system parameters are correlated with the measurements and their correlations are described by their joint PDF. System identification tries to estimate the unknown system parameters given the measurements.

2.2.1 Joint Probability

Joint probability describes the distribution of two or more random variables. For simplicity, in the remainder of this section, two random variables X and Y are considered. The same concepts can be extended to more variables.

In the case of discrete variables X and Y, define joint probability function and joint cumulative distribution function as follows.

$$\begin{aligned} f(x,y) &= P\left(X = x, Y = y\right) \\ F(x,y) &= P\left(X \leq x, Y \leq y\right) \\ &= \sum_{u \leq x} \sum_{v \leq y} f(u,v) \end{aligned}$$

In the case of continuous random variables, let

$$F(x,y) = P(X \leq x, Y \leq y)$$

be the cumulative distribution function, and joint PDF of X and Y is defined by

$$f(x,y) = \frac{d^2}{dxdy} F(x,y)$$

Therefore,

$$\begin{aligned}\int_{x=a}^{b}\int_{y=c}^{d} f(x,y)dxdy &= P\left(a < X < b, c < y < d\right) \qquad (2.6)\\ F(x,y) &= \int_{u=-\infty}^{x}\int_{v=-\infty}^{y} f(u,v)dudv\end{aligned}$$

The cumulative distribution function and PDF of one of the variables, for example X, can be derived from the joint PDF as follows. By definition,

$$\begin{aligned}F_X(x) &= P(X \leq x)\\ &= \int_{u=-\infty}^{x}\int_{v=-\infty}^{\infty} f(u,v)dudv\end{aligned}$$

Thus,

$$\begin{aligned}f_X(x) &= \frac{d}{dx}F_X(x)\\ &= \int_{y=-\infty}^{\infty} f(x,y) \qquad (2.7)\end{aligned}$$

which is the integration of (2.6) w.r.t. all other variables from $-\infty$ to ∞.

Equation (2.7) can be interpreted as follows. If (X_i, Y_i) samples are generated from $f(x,y)$ in (2.6), and we only look at the X_i of these samples, they should follow (2.7).

2.2.2 Conditional Distribution

Equation (2.7) gives the distribution of X regardless of the value of its corresponding Y. Conditional distribution, on the other hand, determines the distribution of X when Y is observed. For example, in (2.6), calculate $f_{X|Y}(x|Y = y)$, i.e., the PDF of X given $Y = y$. In many literatures, $f_{X|Y}(x|Y = y)$ is denoted by $f_{X|Y}(x|y)$ for simplicity.

The conditional PDF $f_{X|Y}(x|y)$ can be obtained as follows. It is essentially Bayes' rule applied on continuous variables.

$$f_{X|Y}(x|y) = \frac{f(x,y)}{f_Y(y)} \qquad (2.8)$$

where $f_Y(y)$ is obtained using (2.7). Equation (2.8) is a function of both x and y. Substituting the observed value of y into (2.8) reduces it to the PDF of x alone. Its integration with respect to x from $-\infty$ to ∞, like any PDF, equals 1. This can be verified as follows.

$$\begin{aligned}\int_{x=-\infty}^{\infty} f_{X|Y}(x|y)dx &= \int_{x=-\infty}^{\infty} \frac{f(x,y)}{f_Y(y)}dx \\ &= \frac{\int_{x=-\infty}^{\infty} f(x,y)dx}{f_Y(y)} \\ &= \frac{f_Y(y)}{f_Y(y)} \\ &= 1\end{aligned}$$

Note that given a particular value of y, $f_Y(y)$ is a constant value independent of x, and hence can be taken out of the integration in the above derivation.

If X and Y are independent variables, $f(x,y) = f_X(x)f_Y(y)$. In this case, (2.8) becomes

$$f_{X|Y}(x|y) = f_X(x)$$

which implies that the information of $Y = y$ does not affect our understanding of X, just as if the information is absent.

Equation (2.8) can be recurrently re-written as

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_Y(y)} \tag{2.9}$$

2.2.3 Covariance and Correlation

Covariance and correlation are defined for a joint distribution with multiple random variables. For simplicity, consider only two random variables X, Y whose joint distribution is given by $f_{XY}(x,y)$. The idea derived from here can be generated to more variables.

The PDF of one variable can be derived from the joint distribution using (2.7). It is straight forward to get the expectation and variance for that variable as follows.

$$\begin{aligned}\mathrm{E}(X) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xf(x,y)dxdy \\ \mathrm{Var}(X) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\mathrm{E}(X))^2 f(x,y)dxdy\end{aligned}$$

The **covariance** of two variables is defined and calculated as follows.

$$\begin{aligned}\mathrm{Cov}(X,Y) &= \mathrm{E}\left((x-\mathrm{E}(X))(y-\mathrm{E}(Y))\right) &(2.10)\\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\mathrm{E}(X))(y-\mathrm{E}(Y))f(x,y)dxdy &(2.11)\end{aligned}$$

where $\mathrm{Cov}(X,Y)$ is sometimes denoted by σ_{XY}. Notice that unlike variance that is always positive, covariance can be zero or negative. If X, Y are independent variables, $f(x,y) = f_X(x)f_Y(y)$. From (2.11)

$$\begin{aligned} \mathrm{Cov}(X,Y) &= \int_{-\infty}^{\infty} (x - \mathrm{E}(X)) f_X(x) dx \times \int_{-\infty}^{\infty} (y - \mathrm{E}(Y)) f_Y(y) dy \\ &= 0 \end{aligned}$$

If the covariance of the two variables is zero, the two variables are called **uncorrelated**. Independent variables are always uncorrelated. However, uncorrelated variables are not necessarily independent.

Furthermore,

$$\mathrm{Cov}(X,Y)^2 \leq \mathrm{Var}(X)\mathrm{Var}(Y)$$

The proof is given below. Notice that lemma (2.12) is used in the proof.

Lemma

For two random variables X and Y,

$$\mathrm{E}(XY)^2 \leq \mathrm{E}(X^2)\mathrm{E}(Y^2) \tag{2.12}$$

Proof:

$$\begin{aligned} 0 &\leq \mathrm{E}\left(\left(X - Y\frac{\mathrm{E}(XY)}{\mathrm{E}(Y^2)}\right)^2\right) \\ &= \mathrm{E}\left(X^2 - 2XY\frac{\mathrm{E}(XY)}{\mathrm{E}(Y^2)} + \mathrm{E}(Y^2)\frac{\mathrm{E}(XY)^2}{\mathrm{E}(Y^2)^2}\right) \\ &= \mathrm{E}(X^2) - 2\mathrm{E}(XY)\frac{\mathrm{E}(XY)}{\mathrm{E}(Y^2)} + \mathrm{E}(Y^2)\frac{\mathrm{E}(XY)^2}{\mathrm{E}(Y^2)^2} \\ &= \mathrm{E}(X^2) - 2\frac{\mathrm{E}(XY)^2}{\mathrm{E}(Y^2)} + \frac{\mathrm{E}(XY)^2}{\mathrm{E}(Y^2)} \\ &= \mathrm{E}(X^2) - \frac{\mathrm{E}(XY)^2}{\mathrm{E}(Y^2)} \end{aligned}$$

Therefore

$$\begin{aligned} \frac{\mathrm{E}(XY)^2}{\mathrm{E}(Y^2)} &\leq \mathrm{E}(X^2) \\ \mathrm{E}(XY)^2 &\leq \mathrm{E}(X^2)\mathrm{E}(Y^2) \end{aligned}$$

Using (2.12) on (2.10), (2.4) gives

$$\begin{aligned} \mathrm{Cov}(X,Y)^2 &= \mathrm{E}\left((x - \mathrm{E}(X))(y - \mathrm{E}(Y))\right)^2 \\ &\leq \mathrm{E}\left((x - \mathrm{E}(X))\right)^2 \mathrm{E}\left((y - \mathrm{E}(Y))\right)^2 \\ &= \mathrm{Var}(X)\mathrm{Var}(Y) \end{aligned}$$

or equivalently

$$\sigma_{XY}^2 \leq \sigma_X^2\sigma_Y^2$$

Noticing that while σ_X, σ_Y are always nonnegative, σ_{XY} is not,

$$-\sigma_X\sigma_Y \leq \sigma_{XY} \leq \sigma_X\sigma_Y \tag{2.13}$$

The **correlation** of two variables is defined and calculated as follows.

$$\begin{aligned}\rho &= \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X)}\sqrt{\mathrm{Var}(Y)}} \\ &= \frac{\sigma_{XY}}{\sigma_X\sigma_Y}\end{aligned}$$

From (2.13), apparently $-1 \leq \rho \leq 1$. When two variables are uncorrelated or independent, $\rho = 0$. If $\rho = 1$, the two variables X and Y are said to be **perfect positive correlated**. This happens usually because the two variables are positively linearly depended, for example, $X = 2Y$ or $X = Y + 1$. If $\rho = -1$, they are said to be **perfect negative correlated**, and the similar idea applies.

2.3 Parameter Estimation

Parameter estimation is an important use case of conditional distribution. Let θ be the parameter to be estimated, and x the measurement that reflects θ via measurement model $f(\theta, x)$ which is given in the form of joint PDF. Both θ and x can be vectors.

2.3.1 Posteriori Mean

Without measurement x, the estimate of θ is given by

$$\hat{\theta} = \int_{-\infty}^{\infty} \theta f_\theta(\theta) d\theta$$

where $f_\theta(\theta)$ is obtained using (2.7). This is known as the **priori estimation** of θ.

Given measurement x, the **posteriori estimation** of θ can be obtained as follows. From (2.9),

$$f_{\theta|X}(\theta|x) = \frac{f_{X|\theta}(x|\theta)f_\theta(\theta)}{f_X(x)} \tag{2.14}$$

$$\hat{\theta} = \int_{-\infty}^{\infty} \theta f_{\theta|X}(\theta|x) d\theta \tag{2.15}$$

where $f_{X|\theta}(x|\theta)$ is known as the **likelihood function** that describes the likelihood of measuring x if the actual parameter(s) is θ. The PDF $f_\theta(\theta)$ is from the priori estimation of θ. Finally, $f_X(x)$ is known as the **evidence**. With fixed x, $f_X(x)$ becomes a constant.

Equation (2.14) is essentially

$$\text{posteriori} = \frac{\text{likelihood} \times \text{priori}}{\text{evidence}} \tag{2.16}$$

Notice that all the components in (2.14) and (2.16) are derived from the joint distribution $f(\theta, x)$. With known analytical $f(\theta, x)$ and a set of measurement x, (2.14) can be used to calculate the estimate. However, in practice $f(\theta, x)$ is often unknown and other estimation methods need to be used.

2.3.2 MLE and MAP

Alternative to (2.15), there are at least two other types of commonly seen estimations, namely **Maximum Likelihood Estimation** (MLE) and **Maximum A Posteriori** (MAP) estimation. MLE and MAP are widely used in system identification. Hence, more details of them can be found in control system relevant notebooks. A brief introduction is given below.

In many occasions, the joint distribution $f(\theta, x)$ is unknown. Instead, the measurement x can be formulated as a function of θ plus noise. In this case, only $f_{X|\theta}(x|\theta)$ the likelihood function can be derived, whereas the priori estimation and evidence are unknown. As a result, (2.14) and (2.15) cannot be used to calculate the estimate.

In this case, consider maximizing the likelihood function to obtain the estimate as follows.

$$\hat{\theta} = \arg\max_{\theta} f_{X|\theta}(x|\theta)$$

This is known as MLE. MLE is useful when only the output model is known.

There are many ways to solve an MLE problem. In the special cases where $f_{X|\theta}(x|\theta)$ is a special distribution, such as Gaussian, Laplace, etc., the MLE becomes **Weighted Least Squares** (WLS) estimator, **Least Absolute Value** (LAV) estimator, etc., respectively.

A general way of solving MLE is given below. Let cost function

$$\begin{aligned} J &= -\sum_{i=1}^{m} \ln f_{X|\theta}(x|\theta) \\ \hat{\theta} &= \arg\min_{x} J \end{aligned}$$

which can be solved using commonly used optimization algorithms such as Newton-Raphson Method.

MAP estimation is also known as the Bayesian estimation. Like the posteriori mean method, MAP also calculates the posteriori distribution (2.14). But instead of (2.15), the estimate is calculated using

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} f_{\theta|X}(\theta|x) \\ &= \arg\max_{\theta} \frac{f_{X|\theta}(x|\theta)f_{\theta}(\theta)}{f_X(x)} \qquad (2.17)\end{aligned}$$

Notice that (2.17) is equivalent to

$$\hat{\theta} = \arg\max_{\theta} f_{X|\theta}(x|\theta)f_{\theta}(\theta) \qquad (2.18)$$

as $f_X(x)$ becomes constant for a given set of x. In this sense, comparing with the posteriori mean method, MAP does not require $f_X(x)$ nor the joint distribution $f(\theta, x)$ in the calculation. Comparing with MLE, MAP further requires $f_{\theta}(\theta)$ the priori estimate.

Equation (2.18) can be solved as follows.

$$\begin{aligned}J &= -\sum_{i=1}^{m} \ln f_{X|\theta}(x|\theta) - \ln f_{\theta}(\theta) \\ \hat{\theta} &= \arg\min_{x} J\end{aligned}$$

likewise.

MLE versus MAP versus Posteriori Mean

It is clear that MLE require the least information among the three to carry out the calculation. MAP requires more information. Posteriori mean method requires the most information.

MLE maximizes the likelihood function $f_{X|\theta}(x|\theta)$ only. MAP maximizes the product of likelihood function and the priori estimation, $f_{X|\theta}(x|\theta)f_{\theta}(\theta)$. From that sense, MLE can be taken as a special case of MAP where $f_{\theta}(\theta)$ is constant for all θ.

2.3.3 Parameter Estimation Benchmark

Estimation error is almost always unavoidable due to the limited size of the samples. However, it is desirable that with an appropriate estimation method, the estimates are unbiased from the true value of the population, i.e., the expectation of the estimates should equal the corresponding values of the population. In that case, the estimation is known as **unbiased estimation**.

Consider unbiased estimation. The accuracy of an estimation method can be evaluated as follows.

The maximum information contained in the samples can be described by

Fisher Information Matrix named after Sir Ronald Fisher. The following equation describes the relation between $\mathrm{Var}(\hat{\theta})$ and Fisher Information Matrix $I(\theta)$ for a scalar unbiased estimator.

$$\mathrm{Var}(\hat{\theta}) \quad \geq \quad I(\theta)^{-1} \tag{2.19}$$

which is known as the **Cramér–Rao lower bound** (CRLB) of the estimator. From (2.19), it can be seen that the "larger" $I(\theta)$, the lower $\mathrm{Var}(\hat{\theta})$ can possibly reach.

It is possible (although in many cases difficult) to check and prove whether an estimator hits the bound and provides the smallest possible $\mathrm{Var}(\hat{\theta})$ by comparing it with its corresponding CRLB. If its variance equals to CRLB, the estimator an **optimal estimator**. A more general CRLB for biased estimator can also be derived. The detailed derivations of Fisher Information Matrix and general CRLB are not given in this notebook.

The magnitude of the estimation error can be evaluated using **Mean Squared Error** (MSE) and **Root Mean Squared Error** (RMSE) as defined below.

Let θ be a parameter and $\hat{\theta}_i$ its estimation in the ith Monte-Carlo run. Let n be the total number of Monte-Carlo runs. Define MSE and RMSE over the n Monte-Carlo runs as follows.

$$\begin{aligned} \mathrm{MSE} &= \frac{1}{n}\sum_{i=1}^{n}\left(\hat{\theta}_i - \theta\right)^2 \\ \mathrm{RMSE} &= \sqrt{\mathrm{MSE}} \end{aligned}$$

Notice that in practice, MSE and RMSE are meaningful only when n is large enough so that their values converge. Let n approaches infinity and we can observe that

$$\mathrm{Var}(\hat{\theta}) = E\left[\left(\hat{\theta} - E\left[\hat{\theta}\right]\right)^2\right] = E\left[\left(\hat{\theta} - \theta\right)^2\right] = \mathrm{MSE}$$

for unbiased estimation because $E\left[\hat{\theta}\right] = \theta$, and the MSE approaches the variance of the estimate. From this point of view, we can use the variance of the estimate, $\mathrm{Var}(\hat{\theta})$, interchangeably with MSE as an evaluation of the performance of the estimation. In most literatures, $\mathrm{Var}(\hat{\theta})$ is used, implying that the result is theoretical and derived under the assumption of "infinite runs".

The smaller $\mathrm{Var}(\hat{\theta})$, the more precise the estimation. There is a limitation to how small $\mathrm{Var}(\hat{\theta})$ can be regardless of the design of the estimator. This is intuitive because there is only so much information contained in the samples, given limited sample size and measurement error. So long as the estimation is done using those samples, its precision is limited.

2.4 Important Theorems

There are a few important theorems frequently used in the study of probability and statistics. They are introduced here.

2.4.1 Law of Large Numbers

The **Law of Large Numbers** (LLN) is a theorem that basically says if performing the same experiment a large number of times, the average of the outcomes of the experiments should eventually converge to a certain value which is the empirical expectation of the experiment. The larger number of trails, the closer the average to the empirical expectation.

In mathematical expression, let X be a random variable which represents the outcome of an experiment. Let X_i be a sample of the outcome. According to LLN,

$$\lim_{n \to \infty} \sum_{i=1}^{n} \frac{X_i}{n} = \bar{X}$$

2.4.2 Central Limit Theorem

Central Limit Theorem (CLT) states the following observation. For independent and identically distributed (i.i.d.) random variables not necessarily following normal distribution, the empirical mean of the samples taken from these distributions tends towards normal distribution when the number of samples is large.

Let X be a random variable not necessarily following normal distribution, and it has mean and variance of μ and $\sigma^2 < \infty$ respectively. Let X_i be samples of the random variable. The empirical mean of the samples is calculated by

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

CLT states that $\bar{X}_n$ follows normal distribution when n is large. The mean and variance of the normal distribution are μ and $\frac{\sigma^2}{n}$ respectively, i.e.,

$$\frac{\bar{X}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$$

follows standard normal distribution.

3

Commonly Seen Distributions

CONTENTS

3.1 Bernoulli and Binomial Distributions 29
3.2 Normal Distribution 31
3.3 Poisson Distribution 32
3.4 Exponential Distribution 34
3.5 Uniform Distribution 34
3.6 Cauchy Distribution 35
3.7 Γ, χ^2, β and F Distributions 35
  3.7.1 Γ Distribution 35
  3.7.2 χ^2 Distribution 37
  3.7.3 β Distribution 37
  3.7.4 F-Distribution 38
3.8 Student's t-Distribution 39

Commonly seen special distributions are introduced here. Some of them are very useful in statistics analysis and are introduced in more details in later part of the notebook. Both discrete and continuous distributions are considered.

3.1 Bernoulli and Binomial Distributions

Bernoulli distribution is a discrete probability distribution of random variable X that can take only 2 values, 0 or 1. The probability of X taking 1 is denoted by p, while 0 is $q = 1 - p$ as shown below.

$$f(x) = P(X = x) = \begin{cases} p & x = 1 \\ q & x = 0 \end{cases}$$

subject to $0 \leq p \leq 1$, $0 \leq q \leq 1$ and $p + q = 1$. Each test is also called a **Bernoulli trail**.

The expectation, variance, skewness and excess kurtosis of the distribution are p, pq, $\dfrac{q-p}{\sqrt{pq}}$ and $\dfrac{1-6pq}{pq}$, respectively.

Run Bernoulli trails repeatedly. Each Bernoulli trail has a probability of

p to take value 1, and $q = 1 - p$ to take value 0. The test is carried out n times. The number of the tests with outcome 1 is a discrete random variable $0 \leq X \leq n$. In this case, X follows **binomial distribution**, whose probability function is given by

$$\begin{aligned} f(x) &= P(X = x) \\ &= \left(\begin{array}{c} n \\ x \end{array} \right) p^x (1-p)^{n-x} \\ &= \frac{n!}{x!\,(n-x)!} p^x (1-p)^{n-x} \end{aligned} \tag{3.1}$$

Bernoulli distribution can be taken as a special case of binomial distribution with $n = 1$, $x = 1$. The expectation, variance, skewness and excess kurtosis of the binomial distribution are np, npq, $\frac{q-p}{\sqrt{npq}}$ and $\frac{1-6pq}{npq}$, respectively.

If n is large, Binomial distribution becomes a verification of CLT on Bernoulli trail. Indeed, when n is large, binomial distribution approaches normal distribution.

Binomial distribution can be extended to **multinomial distribution**, where instead of a single event A happening with probability p or not happening with probability q, s.t. $p + q = 1$, consider multiple events A_1, A_2, ..., A_m, each with probability p_1, p_2, ..., p_m, respectively, s.t. $p_1 + p_2 + \ldots + p_m = 1$. Consider a total of n tests. The number of A_1 occurring is a random variable X_1, event A_2, X_2, and so on. Multinomial distribution studies the probability of

$$\begin{aligned} P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m) &= \frac{n!}{x_1!\,x_2!\ldots x_m!} p_1^{x_1} p_2^{x_2} \ldots p_m^{x_m} \\ \text{s.t.} &\quad \sum x_i = n \end{aligned}$$

Binomial distribution is different from **hypergeometric distribution**. Binomial distribution is used to model "sampling with replacement" process: each Bernoulli trail backing up the binomial distribution is i.i.d. and one's result is not affected by its previous trails. Consider picking up a marbles from a bag containing mixture of red and blue marbles whose numbers are given by r and b respectively. Repeat the test n times. After each test, put the marble back to the bag. The number of blue marbles collected is a random variable X that follows binomial distribution

$$f(x) = \left(\begin{array}{c} n \\ x \end{array} \right) \left(\frac{b}{b+r} \right)^x \left(\frac{r}{b+r} \right)^{n-x}$$

On the other hand, if after each test the marble is not returned to the bag, it becomes a hypergeometric distribution and the probability follows

$$f(x) = P(X = x) = \frac{\begin{pmatrix} b \\ x \end{pmatrix}\begin{pmatrix} r \\ n-x \end{pmatrix}}{\begin{pmatrix} b+r \\ n \end{pmatrix}}$$

with $\max(0, n-r) \leq x \leq \min(n, b)$.

When b, r are far larger than n, the hypergeometric distribution can be approximated by the binomial distribution. When b and r approaches infinity with constant ratio $b/(b+r)$ and $r/(b+r)$, hypergeometric distribution converges to binomial distribution.

3.2 Normal Distribution

Normal distribution, also known as **Gaussian distribution**, is a continuous distribution. It is one of the most widely used distributions to model random noise. One of the explanations is given as follows. Many measurements such as sensor readings are in fact aggregated values. For example, consider a sensor whose reading refreshes at $1Hz$. Behind the screen, it might be sampling the signal at $1000Hz$, and the reading is the average of every 1000 samples. According to CLT, the reading shall follow normal distribution.

We will discuss single normal distribution first, followed by joint multivariate normal distribution.

The PDF of the normal distribution is given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ, σ^2 are the mean and variance of the distribution respectively. The skewness and excess kurtosis of normal distribution are zero.

Let X be a random variable following normal distribution with mean μ and variance σ^2. This can be denoted by $X \sim \mathcal{N}(\mu, \sigma^2)$. Let $Z = \frac{X-\mu}{\sigma}$, and Z would be a standard normal distribution with mean 0 and variance 1.

For a random variable X following normal distribution, the probabilities of its value falling between $\pm\sigma$, $\pm 2\sigma$ and $\pm 3\sigma$ are 0.6827, 0.9545 and 0.9973 respectively. In statistics, sometimes we will take samples with residuals larger than $|3\sigma|$ as outliers.

Multivariate normal distribution describes a vector of random variables $X = [X_1, \ldots, X_m]$ that follows joint normal distribution. The joint PDF is given by

$$f(x) = \frac{1}{\sqrt{(2\pi)^m|\Sigma|}} e^{-\dfrac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}}$$

where $\mu \in \mathbb{R}^m$ is the mean of x and $\Sigma \in \mathbb{R}^{m\times m}$ the covariance matrix. The off diagonal elements in Σ describes the correlation of elements in X. The $|\Sigma|$ is the determinant of Σ.

3.3 Poisson Distribution

Poisson distribution is a discrete probability distribution that takes non-negative integer values 0, 1, 2, ... The probability function of poisson distribution is given by

$$f(x) = P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where $\lambda > 0$ is the shape parameter as shown in Fig. 3.1. Notice that the sum of two Poisson distribution with shape parameters λ_1 and λ_2 is also Poisson distribution with $\lambda = \lambda_1 + \lambda_2$.

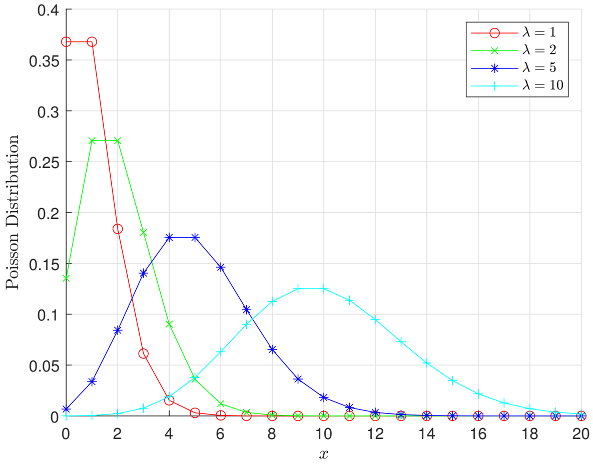


FIGURE 3.1
Poisson distribution with different λ.

Poisson distribution is often used to describe the probability of an event happening a particular number of times in a given window, assuming each occurrence of the event is independent. The parameter λ reflects the average number of occurrence. For example, it can model the number of times a machine fails in a year, assuming each failure is independent from another (the failures are completely random and independent).

Binomial distribution is closely connected with Poisson distribution. Poisson distribution can be taken as a limiting case of binomial distribution when $n \to \infty$, $p \to 0$ is small, and $\lambda = np$ a decent value. A demonstration is given in Fig. 3.2.

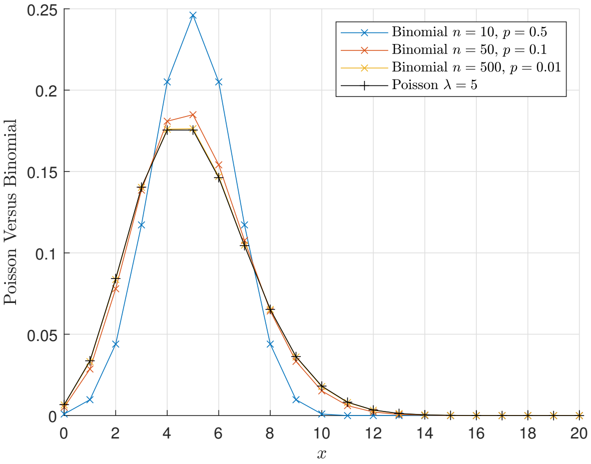


FIGURE 3.2
Poisson distribution approximation using binomial distribution.

An intuitive explanation is given using the following example. Consider studying the number of failure of a machine in one year. In each second, the machine has a failure rate of p, which of course is very small. The total number of seconds in a year is $31,536,000$, which is a large number and it is associated with the number of trails. The number of failure of the machine in a year can be approximately described by binomial distribution with $n = 31536000$. The underlying assumption is that the machine can either fail or survive every second with the probability of p and $1 - p$ respectively, and its status in each second are independent, ignoring the downtime of the machine.

Another example of Poisson distribution is the number of visitors of a website in a given period of time. Each internet user has a small probability p to visit the website. The total number of internet users is n which is a large number. The number of visitors to the website follows Poisson distribution with $\lambda = np$ being the expected visitor number.

The expectation, variance, skewness and excess kurtosis of the distribution are λ, λ, $\frac{1}{\sqrt{\lambda}}$ and $\frac{1}{\lambda}$, respectively.

3.4 Exponential Distribution

Exponential distribution can be used to model the duration of time between two events in a Poisson distribution. For example, it can model the operating time of a machine between two failures, assuming that the failures are independent and random.

Exponential distribution is an important special case of the Γ distribution. More about Γ distribution will be introduced in more details later.

The PDF is the exponential distribution is given by

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where λ is the shape parameter that reflects the average occurrence rate of the event within a unit time window. Exponential distributions with different choice of λ is plotted in Fig. 3.3.

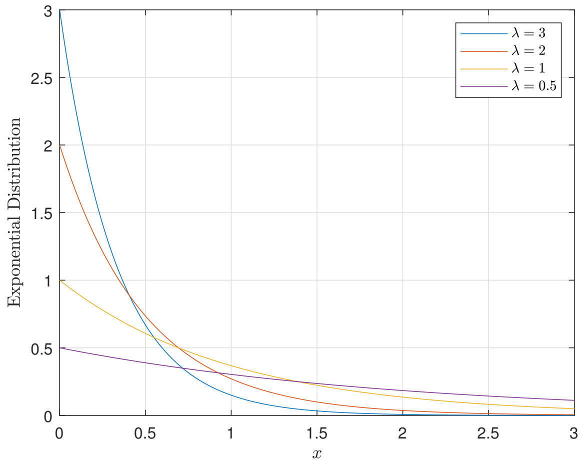


FIGURE 3.3
Exponential distributions with different λ.

The expectation, variance, skewness and excess kurtosis of the distribution are $\frac{1}{\lambda}$, $\frac{1}{\lambda^2}$, 2 and 6, respectively.

3.5 Uniform Distribution

Uniform distribution can be either continuous or discrete. Under the scope of this discussion, continuous uniform distribution is studied.

The PDF of continuous uniform distribution is given by

$$f(x) = \begin{cases} \dfrac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

The expectation, variance, skewness and excess kurtosis of the distribution are $\dfrac{a+b}{2}$, $\dfrac{(b-a)^2}{12}$, 0 and $-\dfrac{6}{5}$, respectively.

3.6 Cauchy Distribution

Cauchy distribution, named after Augustin Cauchy, is a continuous distribution. The PDF is given by

$$\begin{aligned} f(x) &= \frac{1}{\pi\gamma\left(1+\left(\dfrac{x-x_0}{\gamma}\right)^2\right)} \\ &= \frac{1}{\pi}\frac{\gamma}{(x-x_0)^2+\gamma^2} \end{aligned}$$

and it is plotted in Fig. 3.4.

From mathematics perspective, Cauchy distribution describes the ratio of two independent normal distributed random variables with mean zero. It is a useful distribution in physics.

3.7 Γ, χ^2, β and F Distributions

The Γ (gamma), χ^2 (chi-square), β (beta) and F distributions are introduced as follows.

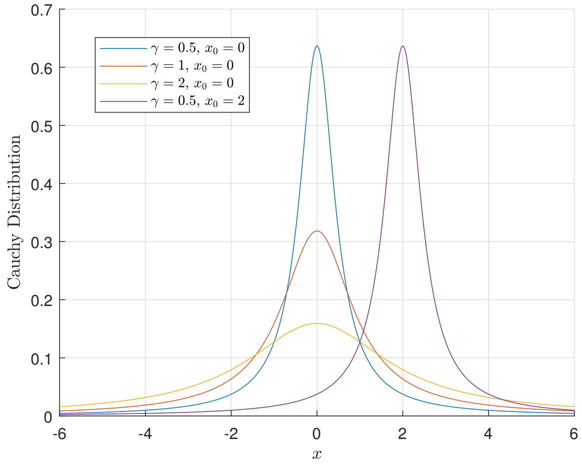


FIGURE 3.4
Cauchy Distribution.

3.7.1 Γ Distribution

The Γ distribution is a continuous distribution with the following PDF

$$f(x) = \begin{cases} \dfrac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where $\alpha > 0$ and $\beta > 0$ are the shape parameters (some literatures uses scale parameter $\theta = 1/\beta$ in the equation), and $\Gamma(\cdot)$ denotes the Gamma function

$$\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt$$

for $\alpha > 0$. Γ distribution is the superset of many distributions. For example, the exponential distribution introduced in the earlier section is a special case of Γ distribution with $\alpha = 1$, and it is used to model the time interval between two events in the Poisson distribution.

The plot of the PDF of Gamma distribution is given in Fig. 3.5.

A Little Bit about Gamma Function

Gamma function is given by

$$\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt$$

It is obvious that $\Gamma(1) = 1$. If $\alpha > 1$, an integration by parts shows

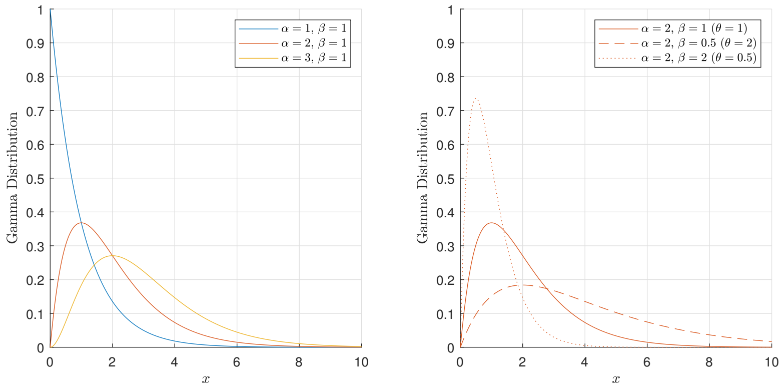


FIGURE 3.5
Gamma Distribution.

that

$$\Gamma(\alpha) \quad = \quad (\alpha - 1)\Gamma(\alpha - 1)$$

Therefore, $\Gamma(\alpha) = (\alpha-1)(\alpha-2)\ldots \times 3 \times 2 \times 1 \times \Gamma(1) = (\alpha-1)!$ for $\alpha \in \mathbb{N}^+$, with $0! = 1$.

The expectation, variance, skewness and excess kurtosis of the distribution are $\frac{\alpha}{\beta}$, $\frac{\alpha}{\beta^2}$, $\frac{2}{\sqrt{\alpha}}$ and $\frac{6}{\alpha}$, respectively.

3.7.2 χ^2 Distribution

The χ^2 distribution is a special case of Gamma distribution. In (3.3), let $\alpha = \frac{r}{2}$ and $\beta = \frac{1}{2}$ to get the PDF of χ^2 distribution as follows.

$$f(x) \quad = \quad \begin{cases} \dfrac{1}{\Gamma(r/2)2^{r/2}} x^{r/2-1} e^{-x/2} & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Equation (3.4) is also denoted by $\chi^2(r)$, where r is known as the degrees of freedom. The plots of PDFs with different γ are shown in Fig. 3.6.

The χ^2 distribution can be used to model the sum of the squares of r independent standard normal distributions, making it an important distribution is statistics, such as in outlier test.

The expectation, variance, skewness and excess kurtosis of the distribution are r, $2r$, $\sqrt{\frac{8}{r}}$ and $\frac{12}{r}$, respectively.

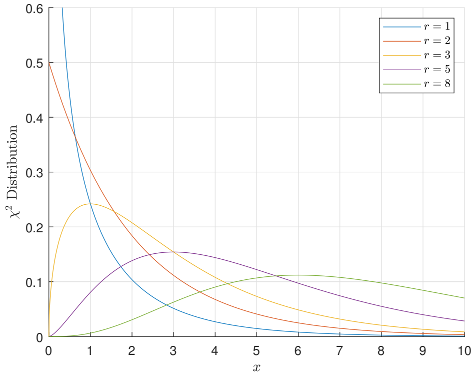


FIGURE 3.6
The χ^2 Distribution.

3.7.3 β Distribution

Let X_1 and X_2 be two independent random variables following Gamma distributions with shape parameters α_1 and α_2 respectively and with $\beta = 1$ in the PDF (3.3). Denote $X = \dfrac{X_1}{X_1 + X_2}$. In this case, X would be a random variable following β distribution whose PDF can be derived from (3.3) and it is given by

$$f(x) = \begin{cases} \dfrac{\Gamma(\alpha_1+\alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)} x^{\alpha_1-1}(1-x)^{\alpha_2-1} & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

In some literatures, notation (a, b) or (α, β) are used to denote (α_1, α_2) in (3.5).

3.7.4 F-Distribution

Let X_1, X_2 be two random variables following χ^2 distribution with degree of freedom ν_1 and ν_2 respectively. Let

$$X = \frac{\dfrac{X_1}{\nu_1}}{\dfrac{X_2}{\nu_2}}$$

then X follows F-distribution, named after R.A. Fisher.

The PDF of the F-distribution is given by

$$f(x) = \begin{cases} \dfrac{\Gamma\left(\dfrac{\nu_1+\nu_2}{2}\right)}{\Gamma\left(\dfrac{\nu_1}{2}\right)\Gamma\left(\dfrac{\nu_2}{2}\right)} \nu_1^{\frac{\nu_1}{2}} \nu_2^{\frac{\nu_2}{2}} x^{\left(\frac{\nu_1}{2}-1\right)} (\nu_2+\nu_1 x)^{-\dfrac{\nu_1+\nu_2}{2}} & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

where ν_1, ν_2 are the shape parameters of the distribution.

3.8 Student's t-Distribution

The zero-mean **Student's t-distribution** (also referred as t-distribution for simplicity) PDF is given by

$$f(x) = \frac{\Gamma\left(\dfrac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\sigma\Gamma\left(\dfrac{\nu}{2}\right)} \left(1+\frac{x^2}{\nu\sigma^2}\right)^{-\dfrac{\nu+1}{2}} \tag{3.6}$$

where ν and σ are known as the shape and scale parameters respectively. When $\nu \to \infty$, (3.6) reduces to normal distribution.

The t-distribution can also be derived from normal and χ^2 distributions as follows. Let X_1 and X_2 be two random variables following standard normal distribution and χ^2 distribution with degree of freedom ν. Let

$$X = \frac{X_1}{\sqrt{\dfrac{X_2}{\nu}}}$$

then X follows t-distribution (3.6).

Student's t-distribution is famous for its heavy-tail characteristics, and it is good at emulating noise with outliers.

The expectation, variance, skewness and excess kurtosis of the t-distribution given by (3.6) are 0, $\dfrac{\nu}{\nu-2}$ $(\nu > 2)$, 0 $(\nu > 3)$ and $\dfrac{6}{\nu-4}$ $(\nu > 4)$ respectively.

Part II

Statistics

4

Basic Statistics

CONTENTS

4.1 Two Types of Statistical Problems 43
4.2 Sampling 44
4.3 Moments Estimation 47
  4.3.1 Mean and Variance 47
  4.3.2 Other Moments 49
4.4 Confidence Interval Estimation 49

Statistics is widely used in science and engineering. Broadly speaking, any attempt to estimate an unknown property of a system using direct or indirect measurements can be considered an application of statistics. Due to factors such as measurement errors and limited sample sizes, the exact value of the unknown property cannot be determined. Instead, statistics provides methodologies for efficiently estimating the most probable value of the unknown, given the available data and its limitations. It also offers tools to quantify the estimation error and the confidence level of the result.

This chapter introduces various sampling methods and demonstrates how to use samples to solve basic statistical problems.

4.1 Two Types of Statistical Problems

At least two common types of problems in science and engineering involve the widespread use of statistics.

1. There is a property of the system whose true value exists but is unknown. We have either direct or indirect measurements of this value, subject to measurement noise. The goal is to estimate the true value from the measurements, while minimizing the effect of noise.

   Example: Consider a power system with many buses, each having a voltage phasor. The objective is to estimate the voltage phasors using various sensor measurements deployed across the system.

2. There is a large population of items whose properties follow an unknown underlying distribution. It is impractical to examine every item due to the size of the population. Hence, we collect some samples and record their properties. The goal is to estimate the distribution of the entire population using only the samples.

   Example: Suppose a factory produces a large number of light bulbs. The objective is to estimate the average lifespan of the bulbs by randomly selecting a few and testing their durability.

The two types of problems, though sound different, are deeply connected in the essence. Specifically, the first type will be introduced with greater details in control system relevant notebooks.

4.2 Sampling

When selecting samples from the population, it is critical to ensure that the selection is done randomly, and all elements in the population have an equal probability of being selected.

Depending on whether an element can be selected multiple times, we have

- Sampling with replacement. In this scenario, a member can be chosen more than once, and the samples are **independent and identically distributed** (i.i.d.)
- Sampling without replacement. In this scenario, a member can be chosen no more than once, and the later samples distribution is slightly affected by earlier samples.

Sampling with replacement is applicable in many engineering problems, such as measuring a physical property of a system and calculating the mean. Sampling without replacement is applicable in many social science problems, such as conducting a survey to estimate the average income of the population. They do have some subtle differences and they are explained using the following example.

Let a set of N random variables be the population. Sample the population M times using sampling with/without replacement. Calculate the mean and variance of the samples, and compare them with their counterparts in the population.

Let $N = 100$ and $M = 500$. Figures 4.1 and 4.2 give 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.

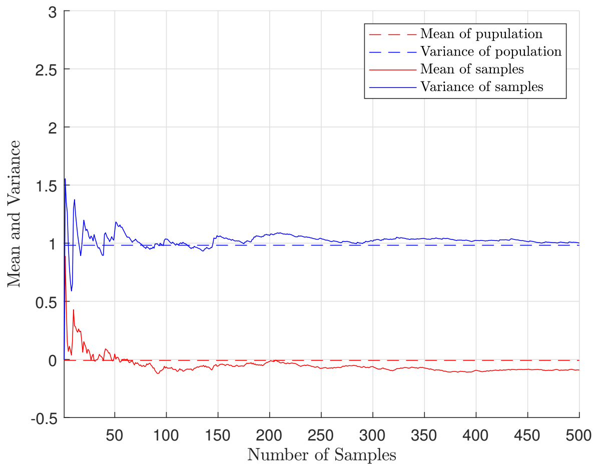


FIGURE 4.1
Sample with replacement, population size $N = 100$, sample size $0 < M \leq 500$.

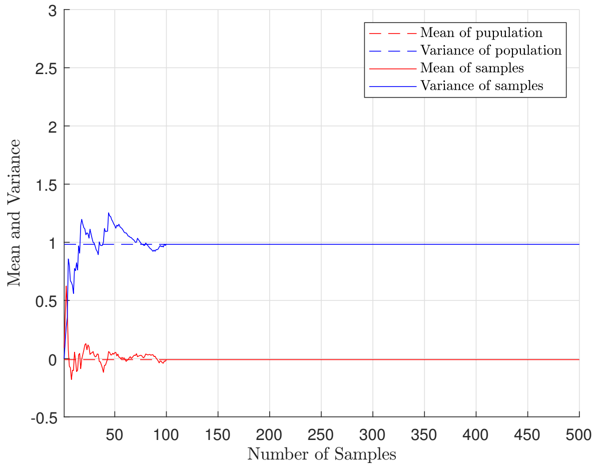


FIGURE 4.2
Sample without replacement, population size $N = 100$, sample size $0 < M \leq 500$.

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.

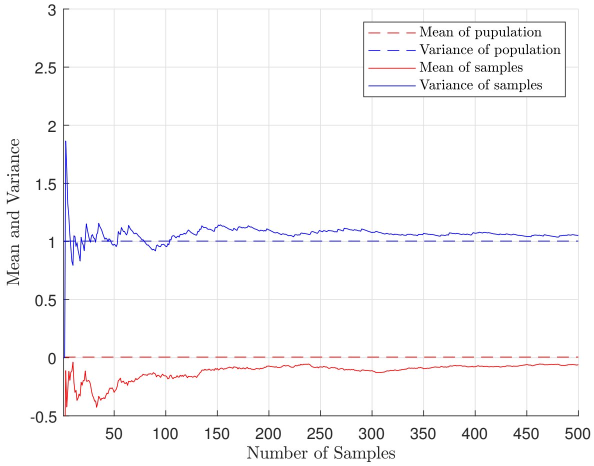


FIGURE 4.3
Sample with replacement, population size $N = 10000$, sample size $0 < M \leq 500$.

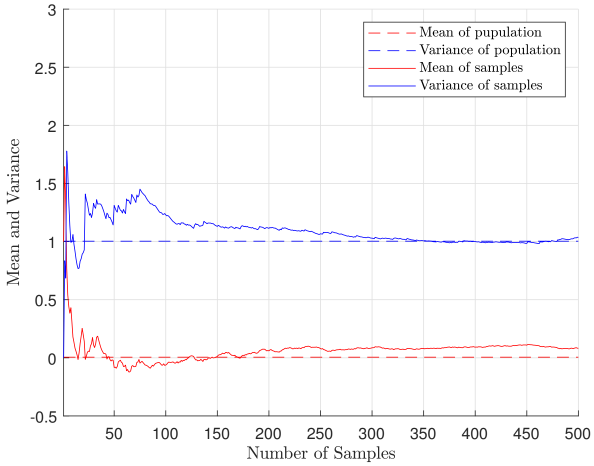


FIGURE 4.4
Sample without replacement, population size $N = 10000$, sample size $0 < M \leq 500$.

4.3 Moments Estimation

It is reasonable to assume that the statistical properties of the samples are connected with those of the population. The statistics of the population can be estimated using the samples. The larger the sample size, the more likely this statement holds true.

In practice, certain types of distributions of the population are often assumed based on prior knowledge or common sense, for example, Gaussian distribution for the salaries of people living in a city. Samples are taken from the population and used to estimate the properties of the distribution, such as its mean and variance.

More details are given below.

4.3.1 Mean and Variance

The **sample mean** and **sample variance** are calculated as follows. Let $X_1, \ldots, X_m$ be the samples. For sampling size $m \geq 2$,

$$\bar{X} = \frac{1}{m}\sum_{i=1}^{m} X_i \tag{4.1}$$

$$S^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(X_i - \bar{X}\right)^2 \tag{4.2}$$

Do not confuse the sample mean in (4.1) and sample variance in (4.2) with the true population mean and variance, typically denoted by μ and σ^2. Although they are closely related, they are not the same.

According to LLN, the sample mean $\bar{X}$ converges to the population mean μ as the number of samples m increases. It can be proved that the sample variance S^2 converges to the population variance σ^2 as well. The proof is given below.

From (4.2),

$$\begin{aligned}
E\left[S^2\right] &= E\left[\frac{1}{m-1}\sum_{i=1}^{m}\left(X_i-\bar{X}\right)^2\right] \\
&= \frac{1}{m-1}E\left[\sum_{i=1}^{m}\left(X_i^2-2X_i\bar{X}+\bar{X}^2\right)\right] \\
&= \frac{1}{m-1}E\left[\sum_{i=1}^{m}X_i^2-2\bar{X}\sum_{i=1}^{m}X_i+m\bar{X}^2\right] \qquad (4.3) \\
&= \frac{1}{m-1}E\left[\sum_{i=1}^{m}X_i^2-m\bar{X}^2\right] \\
&= \frac{1}{m-1}\sum_{i=1}^{m}E\left[X_i^2\right]-\frac{m}{m-1}E\left[\left(\frac{1}{m}\sum_{i=1}^{m}X_i\right)^2\right] \\
&= \frac{1}{m-1}\sum_{i=1}^{m}E\left[X_i^2\right]-\frac{1}{m(m-1)}E\left[\left(\sum_{i=1}^{m}X_i\right)^2\right] \qquad (4.4) \\
&= \frac{1}{m-1}\sum_{i=1}^{m}\left(\mathrm{Var}(X_i)+E[X_i]^2\right) \\
&\quad -\frac{1}{m(m-1)}\left(\mathrm{Var}\left(\sum_{i=1}^{m}X_i\right)+E\left[\sum_{i=1}^{m}X_i\right]^2\right) \\
&= \frac{m}{m-1}\left(\sigma^2+\mu^2\right)-\frac{1}{m(m-1)}\left(m\sigma^2+m^2\mu^2\right) \\
&= \frac{1}{m-1}\left(m\sigma^2+m\mu^2-\sigma^2-m\mu^2\right) \\
&= \sigma^2
\end{aligned}$$

where note that in (4.3)

$$\sum_{i=1}^{m}X_i = m\bar{X}$$

which is from (4.1), and in (4.4)

$$E\left[X^2\right] = \mathrm{Var}(X)+E[X]^2$$

for any random variable X as given in (2.5).

The reason for dividing by $m-1$ in (4.2), rather than m, is that the sample mean $\bar{X}$ is also estimated from the data. This adjustment ensures that S^2 is an unbiased estimation of the population variance σ^2.

Imagine a scenario where the population mean μ is known in advance. In that case, we could have calculated sample variance alternatively as follows

$$S^{*2} = \frac{1}{m}\sum_{i=1}^{m}(X_i-\mu)^2$$

This estimator would also be unbiased for σ^2, and does not require the $m-1$ correction.

4.3.2 Other Moments

Recall from Section 2.1.6 that the mean and the variance are the 1st order moment and the 2nd order central moment respectively. The mean and variance obtained from the samples are closely related their correspondents in the population. It is natural to expand the idea to other moments. This is known as the **method of moments** in population estimation.

Commonly used moments and central moments are of 4th order or lower. Note that bias might be introduced when using central moments where the center is obtained from the samples, as illustrated earlier for the sample variance.

4.4 Confidence Interval Estimation

The estimation obtained from empirical data often differs from the true value due to the randomness introduced by the samples. From CLT, we know that the more samples, the more confident we are that the true value is close to the estimate.

To put it into perspective, we can derive the probability of the true value θ actually lying with a given interval $\left[\hat{\theta}_{\min}, \hat{\theta}_{\max}\right]$, $P\left(\hat{\theta}_{\min} \leq \theta \leq \hat{\theta}_{\max}\right)$, as a function of estimation algorithm, measurement noise and samples number. The interval is known as the **Confidence Interval** (CI). Each CI corresponds with a probability. CI estimation tries to find the probability of a CI, or find the smallest CI for a given probability.

As an example, consider estimating the CI of the mean of a normal distribution using samples taken from that distribution. Let $\mathcal{N}(\mu, \sigma^2)$ be a normal distribution, and $X_1, \ldots, X_m$ a set of m samples generated from the distribution. From the samples, the sample mean $\bar{X}$ and sample variance S^2 are calculated.

Apparently, μ does not necessarily equal to $\bar{X}$. The smaller σ^2 and larger m, the more likely that $\bar{X}$ is close to μ. The error $\tilde{\mu} = \mu - \bar{X}$ is a random variable with

$$
\begin{aligned}
E[\tilde{\mu}] &= 0 \\
\mathrm{Var}[\tilde{\mu}] &= E\left[\left(\mu - \frac{1}{m}\sum_{i=1}^{m} X_i\right)^2\right] \\
&= E\left[\left(\frac{1}{m}\sum_{i=1}^{m} \mu - \frac{1}{m}\sum_{i=1}^{m} X_i\right)^2\right] \\
&= \frac{1}{m^2} E\left[\left(\sum_{i=1}^{m} (\mu - X_i)\right)^2\right] && (4.5) \\
&= \frac{1}{m^2} E\left[\sum_{i=1}^{m} (\mu - X_i)^2\right] && (4.6) \\
&= \frac{1}{m}\sigma^2 && (4.7)
\end{aligned}
$$

where notice that in (4.5),

$$
E\left[(\mu - X_i)(\mu - X_j)\right] = 0
$$

for $i \neq j$.

Though (4.7) can be used to derive the CI, it is useless in the empirical approach because the statistics of the original distribution, μ and σ, is not known exactly. Therefore, (4.7) needs to be reformulated to include $\bar{X}$ and S^2. Rewrite (4.6) as follows.

$$
\begin{aligned}
\mathrm{Var}[\tilde{\mu}] &= \frac{1}{m^2} E\left[\sum_{i=1}^{m} \left(\bar{X} + \tilde{\mu} - X_i\right)^2\right] \\
&= \frac{1}{m^2} E\left[\sum_{i=1}^{m} \left(\bar{X} - X_i\right)^2\right] + \frac{2}{m^2} E\left[\sum_{i=1}^{m} \left(\bar{X} - X_i\right)\tilde{\mu}\right] + \frac{1}{m^2} E\left[\sum_{i=1}^{m} \tilde{\mu}^2\right] \\
&= \frac{m-1}{m^2} S^2 + \frac{1}{m}\mathrm{Var}[\tilde{\mu}] && (4.8)
\end{aligned}
$$

where notice that

$$
E\left[\sum_{i=1}^{m} \left(\bar{X} - X_i\right)\tilde{\mu}\right] = 0
$$

Solving (4.8) for $E[\tilde{\mu}^2]$ gives

$$
\mathrm{Var}[\tilde{\mu}] = \frac{1}{m} S^2 \qquad (4.9)
$$

Equations (4.7) and (4.9) look similar except that σ^2 is replaced by S^2. Notice that (4.9) gives the variance. For the standard deviation,

$$\mathrm{Std}(\tilde{\mu}) = \frac{1}{\sqrt{m}}\sqrt{S^2} = \frac{1}{\sqrt{m}}S \tag{4.10}$$

where S is the estimate of the standard deviation using the samples. Let the interval be $\left[\bar{X} - \mathrm{CI}, \bar{X} + \mathrm{CI}\right]$ where from (4.10),

$$\mathrm{CI} \quad = \quad \frac{u_\alpha}{\sqrt{m}}S \tag{4.11}$$

and u_α a gain factor determines the range of CI. Each u_α corresponds with a CI and a probability. The relation can be derived from the CDF. The result is given below.

$$\mathrm{F}(u_\alpha) = \frac{P+1}{2} = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{u_\alpha}{\sqrt{2}}\right)\right)$$

where $F(\cdot)$ is the CDF of the standard normal distribution and $\operatorname{erf}(\cdot)$ the error function. This is equivalent with

$$\operatorname{erf}\left(\frac{u_\alpha}{\sqrt{2}}\right) \quad = \quad P$$

Some commonly used u_α and confidence pairs are listed below for normal distribution noise assumption:

- $u_\alpha = 1$, $P = 0.683$;
- $u_\alpha = 1.96$, $P = 0.95$;
- $u_\alpha = 2$, $P = 0.954$;
- $u_\alpha = 2.58$, $P = 0.99$;
- $u_\alpha = 3$, $P = 0.997$;
- $u_\alpha = 3.29$, $P = 0.999$.

In the previous example, the problem is to estimate the CI of the mean of a normal distribution. In practice, there are many other problems.

For example, instead of using normal distribution as the noise assumption, t-distribution might be used, especially if the number of samples are small. The t-distribution has the “heavy tail” to emulate outliers, thus making the result more robust.

Depending on the choice of noise assumption, CI may look different and/or u_α may differ. There are CI tables for different types of noise.

5

Statistical Hypothesis Testing

CONTENTS

5.1 Hypothesis 53
    5.1.1 Motivating Examples 53
    5.1.2 Hypothesis Categories 54
    5.1.3 Acceptance Region and Rejection Region 55
5.2 Commonly Seen Hypothesis Tests 56
    5.2.1 The Mean of a Normal Distribution 56
    5.2.2 The Variance of a Normal Distribution 58
    5.2.3 The Comparison of Means of Two Normal Distribution 59
    5.2.4 Exponential Distribution 59
    5.2.5 Binomial Distribution 59
    5.2.6 Poisson Distribution 60

A statistical hypothesis test is a method of statistical inference used to decide whether the empirical data sufficiently support a particular hypothesis. Details are given in the rest of the chapter.

5.1 Hypothesis

This section illustrates the concept of hypothesis and the purpose of hypothesis testings.

5.1.1 Motivating Examples

Two motivating examples are considered.

Example: Verify the Probability of a Coin Tossing "Head"

Consider tossing a coin. The result follows Bernoulli distribution with the chance of head p and tail $1-p$. It is fair to assume that $p = 0.5$. Toss the coin multiple times. Consider the following events:

1. Toss the coin 1 time and the result is "head";
2. Toss the coin 5 times and all the results are "head";
3. Toss the coin 10 times and all the results are "head".

If the above events happen, how confident are we to insist that the coin tossing has a probability of head of $p = 0.5$?

If $p = 0.5$ is indeed correct, the probabilities of the 3 events are $P = 0.5$, $P = 0.5^5 = 0.03125$ and $P = 0.5^{10} = 0.0009765625$ respectively. These figures are consistent with our intuition: the first event can happen quite often, the second very rare, and third seemingly never happen in real life. On the other hand, if the third indeed happens in the first trail, then it is very likely that it is a specially designed coin and $p > 0.5$.

To verify whether $p = 0.5$ or $p > 0.5$, we can make an hypothesis $H_0 : p = 0.5$ as well as its alternative hypothesis $H_1 : p > 0.5$. The Bernoulli trail results then can be used to support or reject the hypothesis with quantified probability. Details are discussed in the remaining of this chapter.

Example: Interview People for Average Annual Income

Consider estimating the average monthly income of the population in a group. It is too costly to interview all the $100,000$ (a very large number) citizens in the city to get the precise average income μ. Therefore, only $1,000 = 1\% \times 100,000$ of the population are randomly selected and interviewed. The sample mean $\bar{X} = 2900$ and variance $S^2 = 100^2$ are calculated from the interview results.

The purpose is to use the above information to evaluate the probability that μ is at least $\mu_0 = 3000$.

In this example, the hypothesis is $H_0 : \mu \geq \mu_0$. Its alternative hypothesis is $H_1 : \mu < \mu_0$. The interview results obtained from the m samples are used to support or reject the hypothesis.

Both the above two examples verify the aggregated mean of a parameter using hypothesis testings. It is possible to use hypothesis testing for other figures as well, including variance and many more.

5.1.2 Hypothesis Categories

The baseline hypothesis, such as "$p = 0.5$ to give head when tossing a coin" in the motivating example, is known as the **null hypothesis** which is often denoted by H_0. One way of interpreting "null" in this context is that it is the default "boring" hypothesis that nulls any changes or innovations. The alternative to the null hypothesis is known as the **alternative hypothesis** which is often denoted by H_1 or H_a.

We take the null hypothesis as the "default truth". There is no need, or it is not possible, to directly prove that the null hypothesis is correct. So we look at a different approach to see whether we can prove it wrong, which is why hypothesis test is introduced. Hypothesis test focuses on looking for evidence to disprove the null hypothesis. The samples try to reject the null hypothesis and the hypothesis test checks whether there is strong evidence in the samples that allow them to do so.

When the samples fail to reject the null hypothesis, it does not necessarily mean that the samples support the hypothesis. It is just that there is no strong statistical evidence in the samples to disprove the hypothesis.

Depending on the statement made in the null hypothesis, it can be divided into two categories: **simple hypothesis** where it specifies a particular precise value for each parameter, or **compound hypothesis** where it specifies multiple values or ranges of values for at least one of the parameters.

5.1.3 Acceptance Region and Rejection Region

A hypothesis is tested using the samples. Since the samples are randomly picked form the population, the result of a hypothesis test is to some extent random. One set of samples may support the hypothesis, while the other set disapprove it.

Let $X_1, \ldots, X_m$ be the m samples taken in the trail. It may just so happen that this particular set of samples passes or fails the hypothesis test. Define

$$A = \{(x_1, \ldots, X_m) | H_0 \text{ is accepted}\} \tag{5.1}$$

$$R = \{(x_1, \ldots, X_m) | H_0 \text{ is rejected}\} \tag{5.2}$$

as the **acceptance region** and **rejection region** respectively. Notice that the rejection region is also known as the **critical region**.

The probability of the null hypothesis being rejected relates to the cardinality of the acceptance and rejection regions. The **power function** is used to describe the probability that the samples reject the null hypothesis. The denotation is given below.

$$\beta_\Phi(\theta) = P\left(\left(X_1, \ \ldots, \ X_m \right) \in R\right)$$

where θ is the true value and Φ the hypothesis testing method. If θ supports the null hypothesis, a good hypothesis testing method Φ should make $\beta_\Phi(\theta)$

close to 0. If θ fails the null hypothesis, a good hypothesis testing method Φ should make $\beta_\Phi(\theta)$ close to 1. The value of $\beta_\Phi(\theta)$ under these two scenarios can be used to evaluate the effectiveness of a hypothesis testing method.

Consider the case where θ is such that H_0 is true. In this case, $\beta_\Phi(\theta)$ describes the probability of the hypothesis test making a **type I error**: reject a true null hypothesis. The probability is described by

$$\alpha_{1\Phi}(\theta) = \begin{cases} \beta_\Phi(\theta) & H_0 \text{ is true} \\ 0 & H_0 \text{ is false} \end{cases} \tag{5.3}$$

Consider the case where θ is such that H_0 is false. In this case, $1 - \beta_\Phi(\theta)$ describes the probability of the hypothesis test making a **type II error**: fail to reject a false null hypothesis. The probability is described by

$$\alpha_{2\Phi}(\theta) = \begin{cases} 0 & H_0 \text{ is true} \\ 1 - \beta_\Phi(\theta) & H_0 \text{ is false} \end{cases} \tag{5.4}$$

Ideally, a good hypothesis test shall ensure that both (5.3) and (5.4) are small. However, notice that there is a trade-off. A very small (5.3), for example by acting extremely conservative and hardly rejecting any hypothesis, may lead to a very large (5.4), and vise versa. In a common practice, a **significance level**, α, is defined for (5.3). The target to minimize (5.4) while keeping (5.3) equal or below α. Typical values of α include 0.05, 0.01, etc.

For two hypothesis testing methods Φ_1 and Φ_2 that both satisfy the same significance level, their $\alpha_{2\Phi}(\theta)$ decide which one is more reliable. The lower $\alpha_{2\Phi}(\theta)$, the better the quality of the hypothesis testing. Among all the hypothesis testing methods that satisfies the same significance level α, if there is a hypothesis testing method Φ that gives the smallest $\alpha_{2\Phi}(\theta)$ in (5.4), Φ is known as the uniformly optimal hypothesis testing method at significance level α. Notice that it is challenging to find and prove the uniform optimal of a hypothesis testing method which usually requires a lot of statistics expertise.

5.2 Commonly Seen Hypothesis Tests

We have seen some hypothesis tests problems in the motivating examples. More are introduced in this section, together with their solutions.

5.2.1 The Mean of a Normal Distribution

Let parameter θ follow normal distribution $\mathcal{N}(\mu, \sigma^2)$, where σ^2 might be unknown. In the case of unknown σ^2, use the empirical variance from (4.2) wherever the variance is used.

Two types of null hypothesis are considered:

- Range. H_0: $\mu \geq \mu_0$ ($\mu \leq \mu_0$); H_1: $\mu < \mu_0$ ($\mu > \mu_0$)
- Particular value. H_0: $\mu = \mu_0$; H_1: $\mu \neq \mu_0$

Consider H_0: $\mu \geq \mu_0$ and H_1: $\mu < \mu_0$. Given samples $X_1, \ldots, X_m$, it is intuitive to survey the mean of the samples $\bar{X} = \frac{1}{m}\sum_{i=1}^{m} X_i$, and let

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \bar{X} \geq C \\ \text{reject } H_0 & \bar{X} < C \end{cases} \tag{5.5}$$

The choice of C influences the power function $\beta_\Phi(\theta)$. Details are discussed as follows.

Recall from (4.7) and (4.9) that given the samples mean $\bar{X}$ and the variance σ^2, the true value of μ follows a normal distribution centered at $\bar{X}$ and with standard deviation $\frac{1}{\sqrt{m}}\sqrt{\sigma^2}$. The distribution of μ is given by Fig. 5.1. Notice that Fig. 5.1 is shifted by $\bar{X}$ in the horizontal axis so that it is centered at zero.

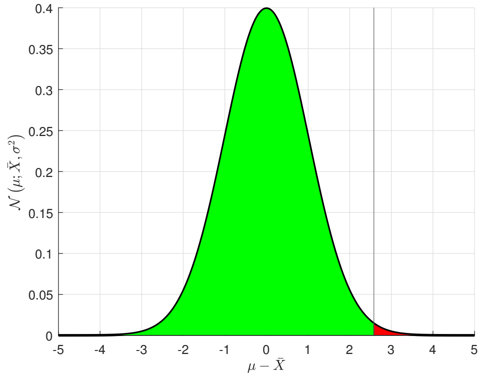


FIGURE 5.1
Distribution of μ as a function of $\bar{X}$ and σ^2.

Draw a vertical bar at $\mu - \bar{X} = C'$ in Fig. 5.1 so that the area of the red zone equals to α and green zone $1 - \alpha$. From (4.11),

$$C' = \frac{u_\alpha}{\sqrt{m}}\sigma$$

where u_α is a gain parameter determined by α as follows.

$$\operatorname{erf}\left(\frac{u_\alpha}{\sqrt{2}}\right) = 1 - 2\alpha$$

For example, if $\alpha = 0.005$, $u_\alpha = 2.58$.

Consider a hypothesis test method where

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \mu_0 - \bar{X} \leq C' \\ \text{reject } H_0 & \mu_0 - \bar{X} > C' \end{cases} \tag{5.6}$$

The hypothesis test (5.6) can be interpreted as follows. From the empirical samples, μ is unlikely to be so large that it falls in the red zone in Fig. 5.1. If the criterion μ_0 is in the red zone, it is unlikely that $H_0 : \mu > \mu_0$ is true. Therefore, the samples would reject H_0. On the other hand, if the criterion μ_0 stays in the green zone, the samples fail to reject $H_0 : \mu > \mu_0$ because it is fairly probable.

For the above hypothesis test, the probability of H_0 being true while rejected equals to the area of the red zone which is α. This hypothesis test meets the significance level of α.

Comparing (5.5) and (5.6) gives

$$C = \mu_0 - C' = \mu_0 - \frac{u_\alpha}{\sqrt{m}}\sigma$$

and (5.5) becomes

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \bar{X} \geq \mu_0 - \dfrac{u_\alpha}{\sqrt{m}}\sigma \\ \text{reject } H_0 & \bar{X} < \mu_0 - \dfrac{u_\alpha}{\sqrt{m}}\sigma \end{cases}$$

which is a hypothesis test of the mean of a normal distribution at significance level of α.

Consider H_0: $\mu = \mu_0$ and H_1: $\mu \neq \mu_0$. The idea is similar except that there would have been two red zones in Fig. 5.1 at both tails. The hypothesis test is given by

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \mu_0 - \dfrac{u_\alpha}{\sqrt{m}}\sigma \leq \bar{X} \leq \mu_0 + \dfrac{u_\alpha}{\sqrt{m}}\sigma \\ \text{reject } H_0 & \text{otherwise} \end{cases}$$

where notice that

$$\operatorname{erf}\left(\frac{u_\alpha}{\sqrt{2}}\right) = 1 - \alpha$$

the $1 - \alpha$ is used instead of $1 - 2\alpha$ on the right side of the above equation because the red zones are at both tails.

5.2.2 The Variance of a Normal Distribution

Variance is important in quality control. Let X_i be m samples taken from $\mathcal{N}(\mu, \sigma^2)$. The variance σ^2 is unknown. The objective is to use the samples to reject the following three types of null hypothesis

- H_0: $\sigma^2 \geq \sigma_0^2$; H_1: $\sigma^2 < \sigma_0^2$
- H_0: $\sigma^2 \leq \sigma_0^2$; H_1: $\sigma^2 > \sigma_0^2$
- H_0: $\sigma^2 = \sigma_0^2$; H_1: $\sigma^2 \neq \sigma_0^2$

where σ_0^2 is the benchmark.

5.2.3 The Comparison of Means of Two Normal Distribution

The evaluation of the difference of two parameters is useful. It is widely used to compare the quality of two products. When the two parameters are described by normal distributions, the problem can be formulated as the comparison of the means of two normal distributions.

Let X_i be m samples taken from $\mathcal{N}(\mu_x, \sigma_x^2)$, and Y_i, n samples from $\mathcal{N}(\mu_y, \sigma_y^2)$. The means μ_x and μ_y are unknown. The objective is to use the samples to reject the following null hypothesis

1. H_0: $\mu_x - \mu_y \geq \mu_0$; H_1: $\mu_x - \mu_y < \bar{X}$
2. H_0: $\mu_x = \mu_y$; H_1: $\mu_x \neq \mu_y$

where μ_0 is a given constant.

5.2.4 Exponential Distribution

As introduced earlier in Section 3.4, exponential distribution can be used to model the duration of time between two independent events. It is useful to model the lifespan of an equipment, or the operating hour between its two consecutive independent and random failures.

Let X_i be m samples taken from (3.2). In practice, it can be m sample products' lifespan. The expectation of (3.2) is $\frac{1}{\lambda}$. The objective is to use the samples to estimate λ and reject the following null hypothesis.

1. H_0: $\lambda \geq \lambda_0$; H_1: $\lambda < \lambda_0$
2. H_0: $\lambda \leq \lambda_0$; H_1: $\lambda > \lambda_0$
3. H_0: $\lambda = \lambda_0$; H_1: $\lambda \neq \lambda_0$

where λ_0 is a given constant.

5.2.5 Binomial Distribution

Let X_i be m samples taken from binomial distribution $\mathcal{B}(n, p)$, where p is the probability of the Bernoulli trail, and n be the number of trails repeated in the binomial test. The objective is to use the samples to reject the following null hypothesis.

1. H_0: $p \geq p_0$; H_1: $p < p_0$
2. H_0: $p \leq p_0$; H_1: $p > p_0$
3. H_0: $p = p_0$; H_1: $p \neq p_0$

where p_0 is a given constant.

5.2.6 Poisson Distribution

6

Bayesian Methods

CONTENTS

Part III

Tools

7

R (Part I: Basics)

CONTENTS

7.1 Brief Introduction 65
  7.1.1 Installation 66
  7.1.2 R Packages Management 66
7.2 Basic Data Types and Syntax 68
  7.2.1 Variable Names 69
  7.2.2 Different Types of Scalars 69
  7.2.3 Vectors 71
  7.2.4 Matrices 78
  7.2.5 Lists 82
  7.2.6 Conditional and Loop Statements 84
  7.2.7 User-Defined Function 86
7.3 Data Frames 86
  7.3.1 Data Importing 86
  7.3.2 Data Accessing 87
  7.3.3 Filtering 89
  7.3.4 Data Frame from Integration of Data 91
7.4 Basic Data Visualizations Using `qplot()` 92
7.5 Advanced Data Visualizations Using `ggplot()` 93
  7.5.1 Grammar of Graphics 94
  7.5.2 Data, Aesthetics and Geometries Layers 95
  7.5.3 Statistics Layers 96
  7.5.4 Facets Layers 99
  7.5.5 Coordinates Layers 102
  7.5.6 Themes Layers 103

This and the next chapter introduce R language, a widely used programming language for data mining and visualization. This chapter focuses on the fundamental setup such as the installation of R and RStudio, basic data types, basic operations such as vector, list, table and data frame manipulations, and basic visualization methods. The next Chapter 8 introduces advanced data analysis tools and skills that R uses to solve practical problems.

When comes to statistics calculation and data visualization, R is often considered simpler and more efficient with large size datasets compared with other programming languages such as Python.

7.1 Brief Introduction

R language 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 [5] where more details about R can also be found.

RStudio, also known as **Posit**, is an Integrated Development Environment (IDE) widely used for R programming and testing. RStudio IDE has an open-source and free-of-charge community version for personal use. It can be downloaded from [6]. Notice that RStudio is not compulsory and there are other IDEs that supports R script programming such as VSCode and Vim. With that been said, most of the R scripts in this notebook are programmed and tested with RStudio.

7.1.1 Installation

Download R and RStudio from [5] and [6], and install them following the instructions. It is required to install R before RStudio.

When working with R language, the user needs to maintain the version of R and its packages (more to be introduced in Section 7.1.2) and to make sure that they are compatible. Anaconda allows the user to create a virtual environment for a project and manage packages within that environment. For that reason, the user might prefer to use Anaconda to install and maintain R and its packages. The installation and maintenance of R via Anaconda can be found on [7] and it is not covered in this notebook.

7.1.2 R Packages Management

R uses packages, also known as libraries, to expand its capability. R packages, both built-in and third-party, provide powerful features for data analysis and visualization. Some packages come with demonstrative sample data frames. Packages can be published and shared online. **CRAN** is by far the most popular platform to store and share R packages.

There are a number of default built-in packages that come along with R installation. Only third-party packages need separate installation, and it can be done using the R console. To install or remove a package, use

```
install.packages("<package>")
remove.packages("<package>")
```

respectively. For example,

```
install.packages("pacman")
```

It is a good practice to load a package before using its functions and datasets. To load a package, both default and third-party, use

```
library(<package>)
```

For example

```
library(pacman)
```

After loading a package, the data frames and functions defined in that package can be used normally.

It is possible to use the functions and data frames in a package without loading it, in which case use prefix `<package>::<function>`, `<package>::<dataframe>`. This is inconvenient if the functions or data frames are to be used frequently.

To unload a package, use

```
detach("package:<package>", unload = TRUE)
```

For example,

```
detach("package:pacman", unload = TRUE)
```

There are third-party packages that provide package management functions, for example `pacman`. With `pacman` installed and loaded, use the following commands to install, load and unload packages respectively.

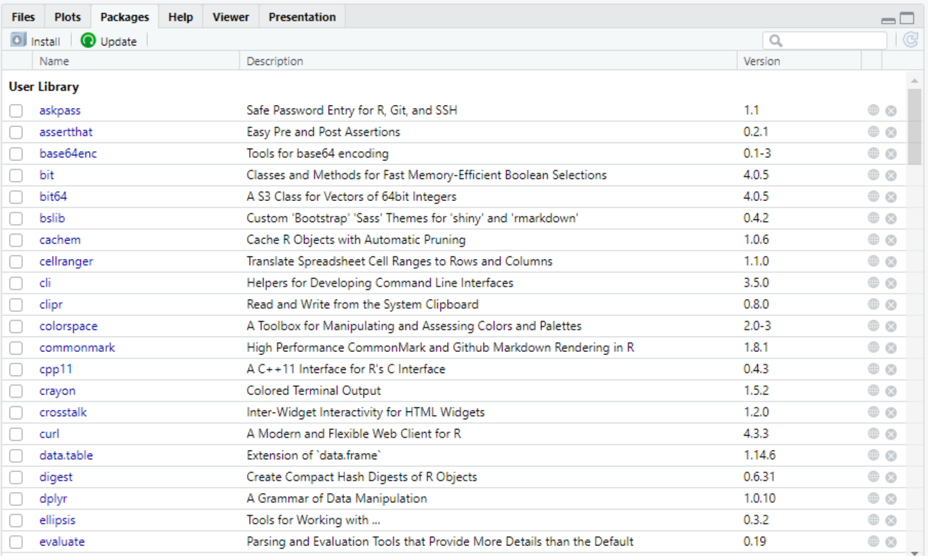
```
p_install(<package>, ...) # install
p_load(<package>, ...) # load (if not installed, also install)
p_unload(<package>, ...) # unload
p_unload(all) # unload all
```

Assume that `pacman` is already installed. 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
)
```

The above command can be executed without loading `pacman` in advance since `pacman::p_load()` prefix is used. The listed packages are commonly used in R projects, and a brief explanation to each of them is given as comments in the above code.

RStudio provides a graphical interface to manage packages as shown in Fig. 7.1. A package can be loaded or unloaded simply by checking or unchecking the corresponding box.



	Name	Description	Version
User Library			
☐	askpass	Safe Password Entry for R, Git, and SSH	1.1
☐	assertthat	Easy Pre and Post Assertions	0.2.1
☐	base64enc	Tools for base64 encoding	0.1-3
☐	bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.0.5
☐	bit64	A S3 Class for Vectors of 64bit Integers	4.0.5
☐	bslib	Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'	0.4.2
☐	cachem	Cache R Objects with Automatic Pruning	1.0.6
☐	cellranger	Translate Spreadsheet Cell Ranges to Rows and Columns	1.1.0
☐	cli	Helpers for Developing Command Line Interfaces	3.5.0
☐	clipr	Read and Write from the System Clipboard	0.8.0
☐	colorspace	A Toolbox for Manipulating and Assessing Colors and Palettes	2.0-3
☐	commonmark	High Performance CommonMark and Github Markdown Rendering in R	1.8.1
☐	cpp11	A C++11 Interface for R's C Interface	0.4.3
☐	crayon	Colored Terminal Output	1.5.2
☐	crosstalk	Inter-Widget Interactivity for HTML Widgets	1.2.0
☐	curl	A Modern and Flexible Web Client for R	4.3.3
☐	data.table	Extension of `data.frame`	1.14.6
☐	digest	Create Compact Hash Digests of R Objects	0.6.31
☐	dplyr	A Grammar of Data Manipulation	1.0.10
☐	ellipsis	Tools for Working with ...	0.3.2
☐	evaluate	Parsing and Evaluation Tools that Provide More Details than the Default	0.19

FIGURE 7.1
RStudio's graphical interface for package management.

7.2 Basic Data Types and Syntax

This section introduces the basic data types and their associated operations. Basic syntax such as conditionals, loops and user-defined functions are also given.

It is worth mentioning in the beginning that R is case sensitive. Use `#` to lead a comment. In the console, use `print(<variable>)` or simply type the name of the variable to print the value of a variable to the console. Use `?<function>`, `?<dataframe>` to access the manual for a function or data frame.

As a quick demonstrative example, the following is a small piece of code written in R. It generates generalized t-distribution noise and save the noise samples into a local CSV file.

```
# Read parameters
generate_sgt_parameters <- read.csv('generate_sgt_parameters.csv',
    header = FALSE)
```

```
sgt_mu <- generate_sgt_parameters[,1]
sgt_sigma <- generate_sgt_parameters[,2]
sgt_lambda <- generate_sgt_parameters[,3]
sgt_p <- generate_sgt_parameters[,4]
sgt_q <- generate_sgt_parameters[,5]
sgt_n <- generate_sgt_parameters[,6]
sgt_seed <- generate_sgt_parameters[,7]

# Generate pseudo random numbers
library('optimx')
library('numDeriv')
library('sgt')
set.seed(sgt_seed)
x = rsgt(n = sgt_n, mu = sgt_mu, sigma = sgt_sigma, lambda = sgt_lambda
    , p = sgt_p, q = sgt_q, mean.cent = TRUE, var.adj = FALSE)

# Write data
write.table(x, 'generate_sgt_data.csv', sep=",", row.names = FALSE, col
    .names=FALSE)
```

7.2.1 Variable Names

Variable names are used as identifiers to variables. Legitimate variable names are a combination of letters, numbers, periods (`.`) and underscores (`_`). The name cannot start with a number of the underscore, but can start with the period. A name cannot contain dash (`-`), space or special characters such as `@` or `$`.

It is worth emphasizing the use of period (`.`) in R and in other languages. Many languages such as Python and MATLAB uses the period to indicate fields in a structure. For example, there might be a structure called "`student`", and `student.name`, `student.age` are the fields under `student`. However, this is not the case with R. In R, `student`, `student.name` and `student.age` are treated as different variables without logical connections.

For R, it uses `<variable>$<field>` to indicate fields of a structure. More details will be given in later sections when list and data frame are introduced.

7.2.2 Different Types of Scalars

R supports many data types. Commonly used data types are summarized in Table 7.1, 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. Notice that `>` is the prompt indicating that the commands are executed in a console.

TABLE 7.1
Commonly used data types.

Data Type	Syntax (Example)	Description
integer	`n <- 2L`	An integer. Define an integer by a value followed by `L`.
double	`x <- 2`	A 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 respectively.

```
> n <- 2L
> typeof(n)
[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.<datatype>()`. 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 `^` (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
> 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 uses built-in and third-party functions which extend its capability. There is a rich set of functions for numerical calculations, string operations, probability calculations and statistics analysis. Some of them are summarized in Tables 7.2, 7.3, 7.4, 7.5 and 7.6.

TABLE 7.2
Numerical calculations.

Syntax	Description
`abs(x)`	Absolute value.
`sqrt(x)`	Square root.
`ceiling(x)`	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 7.3
Logical comparisons.

Syntax	Description
`x == y`	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.
`x & y`	And.
`x | y`	Or.
`isTRUE(x)`	Is true.

TABLE 7.4
String operations.

Syntax	Description
`substr(s, n1, n2)`	Segment of a string, from the n_1-th character to n_2-th character, both characters included. Notice that the index starts from 1 but not 0.
`grep(p, s)`	Searching for a pattern in a string.
`sub(s1, s2, s)`	Find and replace patterns in a string.
`paste(s1, s2, ..., p="")`	Concatenate strings.
`strsplit(s, p)`	Split a string.
`tolower(s)`	Convert to lower case.
`toupper(s)`	Convert to upper case.

TABLE 7.5
Probability related operations.

Syntax	Description
`dnorm(x, m=0, std=1)`	Calculate the PDF of Gaussian distribution.
`pnorm(x, m=0, std=1)`	Calculate the CDF of Gaussian distribution.
`qnorm(x, m=0, std=1)`	Inverse function of `pnorm()`.
`rnorm(n, m=0, std=1)`	Generate Gaussian distribution samples.
`dbinom(n, size, p)`	Calculate the probability of a binominal distribution.
`pbinom(n, size, p)`	Calculate the comulative probability of a binominal distribution.
`qbinom(x, size, p)`	Inverse function of `pbinom()`.
`rbinom(n, size, p)`	Generate binominal distribution samples.
`dpois(x, lambda)`	Calculate the probability of a Poisson distribution.
`ppois(n, lambda)`	Calculate the comulative probability of a Poisson distribution.
`qpois(x, lambda)`	Inverse function of `ppois()`.
`rpois(n, lambda)`	Generate Poisson distribution samples.
`dunif(x, min=0, max=1)`	Calculate the PDF of uniform distribution.
`punif(x, min=0, max=1)`	Calculate the CDF of uniform distribution.
`qunif(x, min=0, max=1)`	Inverse function of `punif()`.
`runif(n, min=0, max=1)`	Generate uniform distribution samples.

TABLE 7.6
Statistics related functions.

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

7.2.3 Vectors

Vector and matrix are the fundamental forms using which R stores and processes data. List is similar with vector to some extend, and it is also introduced in this section.

There are different types of vectors in R depending on the data type of the elements it stores. The commonly used vector types include numeric 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.

Notice that the index of a vector in R starts from 1 instead of 0. This is similar with MATLAB and different from many other computer languages such as C/C++, JavaScript and Python.

Use the following syntax to create a vector.

```
<vector> <- c(<value>, ...)
```

where `<value>` can be single element or a vector. Nested vector is expanded. For example,

```
> x <- c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
> y <- c(c(1,2,3),4,5,6,7)
> y
[1] 1 2 3 4 5 6 7
> z = c(x, 1,2,3,4,5)
> z
[1] 1 2 3 4 5 1 2 3 4 5
> typeof(z)
[1] "double"
```

To append to a vector, or to add vectors to form a longer vector, or to insert a vector in the middle of a vector, use `append()` as follows.

```
> a <- c(1, 2, 3)
> a <- append(a, 4)
> print(a)
[1] 1 2 3 4
> b <- c(5, 6)
> c <- append(a, b)
> print(c)
[1] 1 2 3 4 5 6
> d <- append(c, c(100, 200, 300), 1) # insert a vector after the
    element
> print(d)
[1]   1 100 200 300  2   3   4   5   6
```

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>, <repeat>)
```

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

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

Replica can also be used to create empty vector with `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>]`, where `<index>` can be an integer or a vector of integers. 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(1, 3, 5)]
[1] "a" "c" "e"
```

Notice that as mentioned earlier, the first element in a vector is indexed 1 instead of 0.

Unlike vectors in MATLAB or lists in Python, in R the elements in a vector can have corresponding names, and they can be referenced not only by index but also by names. An example is 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
```

Notice that `NA` will be returned if the index used to access the element in a vector exceeds its length, or if the name used to access the element in a vector is not found. This is different from most programming languages, which will usually raise an exception or an error. To remove the name of a vector, simply assign `NULL` to `names(<vector>)`.

And finally to return the length of a vector, use `length(<vector>)`.

Vectorization operations significantly speed up the calculation comparing with looping over elements. Most numerical calculations, including `+`, `-`, `*`, `/`, `%/%`, `%%`, `^`, support vectorization operations. 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
```

A vector can also be used as an input argument or output return of a function.

Two indexes methods: `[]` versus `[[]]`

So far we have been using `<vector>[<index>]` to access elements in a vector, and it works alright. The `<index>` can be a vector of indexes, in which case the return is also a vector. There is an alternative way of accessing elements, and that is `<vector>[[<index>]]`. An example is given below.

```
> a <- c(1, 2, 3, 4, 5)
> print(a[1])
[1] 1
> print(a[[1]])
[1] 1
> names(a) <- c("first", "second", "third", "fourth", "fifth")
> print(a["first"])
first
1
> print(a[["first"]])
[1] 1
```

From the above example, one may notice that for unnamed vectors `[]` and `[[]]` behave similarly when accessing a single element. For named vectors, `[]` and `[[]]` behave slightly differently.

As introduced earlier, `<index>` can be a vector in `<vector>[<index>]`. However this is not true for `<vector>[[<index>]]`. An example is given below.

```
> print(a[1:3])
first second third
1      2     3
> print(a[c(1, 3, 5)])
first third fifth
1    3    5
> print(a[c("first", "third", "fifth")])
first third fifth
1    3    5
> print(a[[1:3]])
Error in a[[1:3]] :
attempt to select more than one element in vectorIndex
```

There are some semantic differences when using `[]` and `[[]]`. In short, `[]` returns a sub-vector of the original vector (the length of the sub-vector can be 1), while `[[]]` returns the element in the vector. This explains why it triggers an error when selecting multiple elements using `[[]]`.

Notice that numeric values are atomic in R, meaning that they behave the same whether they are extracted as a single-element vector or a number. That is why `print(a[1])` shows the same result as `print(a[[1]])` in the earlier example, except that `[]` preserves the name of an element.

In the context of unnamed numeric vectors and matrices, `[[]]` is rarely used, and `[]` works just fine almost all the time. However, when comes to other data structures like list, etc., it is important to distinguish `[]` and `[[]]`, and choose the correct one depending on the purpose.

7.2.4 Matrices

Similar with vector, all elements in a matrix must have the consistent datatype. A demonstration is shown in Fig. 7.2 with elements indexes. 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 to the first element and `A[4,6]` the last element. It is also possible to index an entire row or column. For example, use `A[1,]` to represent the first row of the matrix.

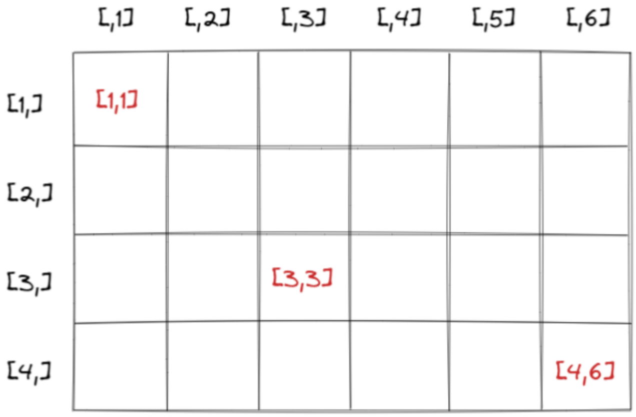


FIGURE 7.2
A demonstration of a matrix in R.

A matrix can be created from scratch by stacking row vectors using `rbind()` as follows.

```
# build rows
<row1> <- c(<value11>, ..., <value1n>)
...
<rowm> <- c(<valuem1>, ..., <valuemn>)
# build matrix
<matrix> <- rbind(<row1>, ..., <rowm>)
```

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

Similar with `names(<vector>)` that can be used to view and assign names to elements in a vector, `rownames(<matrix>)` and `colnames(<matrix>)` can be used to view and assign names to the rows and columns of a matrix. Examples are given below.

```
> A <- matrix(1:9, 3, 3)
> colnames(A) <- c("col1", "col2", "col3")
> rownames(A) <- c("row1", "row2", "row3")
> A
col1 col2 col3
row1    1    4    7
row2    2    5    8
row3    3    6    9
> colnames(A)
[1] "col1" "col2" "col3"
> rownames(A)
[1] "row1" "row2" "row3"
> A[1,1]
[1] 1
> A["row1", "col1"]
[1] 1
> A[1,2]
[1] 4
> A["row1", "col2"]
[1] 4
```

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

Vectorization operators are defined for matrix level as well. For example, for two matrices with the same shape, numerical operations such as `+`, `-`, `*`, `/`, `%/%`, `%%` and `^` can be used.

By default, a single row or column of a matrix is treated as a vector. 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
```

It is possible to specifically 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
```

R provides flexible and powerful data visualization tools, many of which more advanced than the ones 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)
colnames(citation) <- c("2018", "2019", "2020", "2021", "2022")
rownames(citation) <- c("Professor", "Student")
print(citation)
citation.ratio <- citation
citation.ratio["Professor",] <- round(citation["Professor",] / mean(
    citation["Professor",]) * 100, 1)
citation.ratio["Student",] <- round(citation["Student",] / mean(
    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",
       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. 7.3.

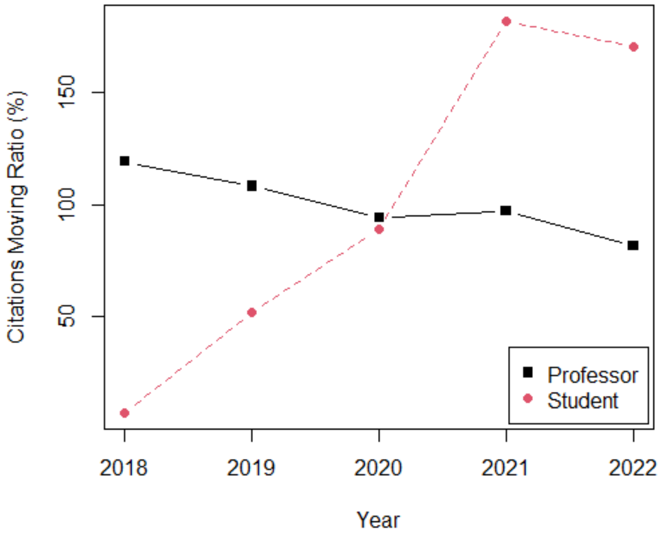


FIGURE 7.3
A demonstration of using `matplot` to plot trends.

Another example of analyzing the performance of players through a series of basketball games are given below.

```
# generate table
player_name <- c("player1", "player2", "player3")
match_name <- c("match1", "match2", "match3", "match4", "match5", "
    match6", "match7", "match8", "match9", "match10")
penalty_attempt <- abs(matrix(round(rnorm(3*10, 5, 2)), 3, 10))
penalty_point <- abs(penalty_attempt - matrix(abs(round(rnorm(3*10, 1,
    1))), 3, 10))
throw_attempt <- abs(matrix(round(rnorm(3*10, 15, 3)), 3, 10))
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
rownames(penalty_point) <- player_name
colnames(penalty_point) <- match_name
rownames(throw_attempt) <- player_name
colnames(throw_attempt) <- match_name
rownames(total_point) <- player_name
colnames(total_point) <- match_name

# 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)
}

# plot
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. 7.4 and 7.5.

Notice that `matplot()` is not widely used in practice comparing with the other visualization tools to be introduced in later sections.

7.2.5 Lists

List is similar with vector in the sense that it is also an indexed 1-D storage. However, the elements in a list does not need to share the same data type, and hence list can be used flexibly, such as to save nested “list of lists”, “list of data frames”, or list of other items. Just like vector where each element can have its own name, elements in a list can also be associated with names. From this perspective, list can be used almost like an “indexed dictionary” of Python.

Examples are given below.

```
> list1 <- list(24, "abc", 5.4, list(17, "def"))
> print(list1[[1]])
[1] 24
```

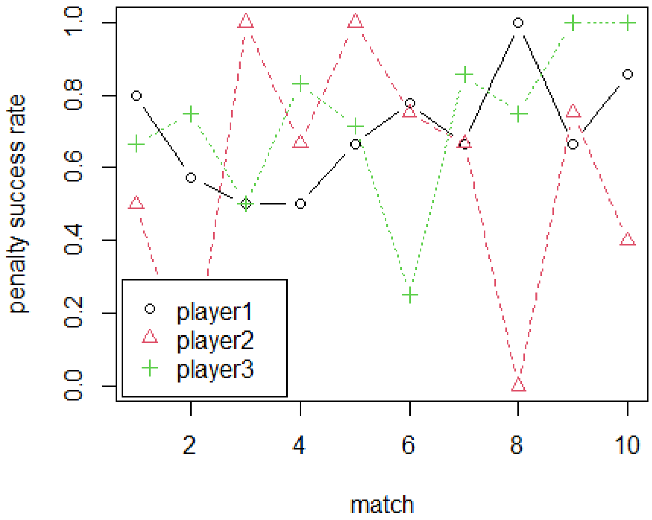


FIGURE 7.4
Plot of penalty success rate of the 3 players in 10 matches.

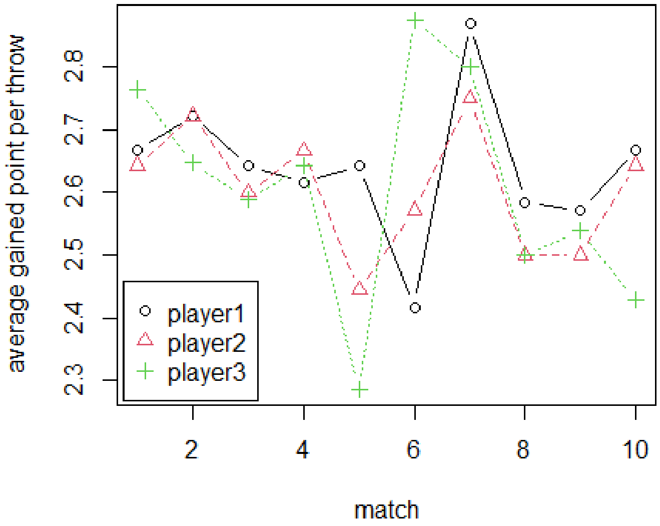


FIGURE 7.5
Plot of average point gained per throw attempt for the 3 players in 10 matches.

```
> print(list1[[4]])
```

```
[[1]]
[1] 17

[[2]]
[1] "def"
```

For elements with names, the list looks very much like a dictionary. The content of the list can be accessed via both index and key.

```
> list2 = list(
+   "name" = "lu",
+   "age" = 33,
+   "number of pc" = 3
+   )

> print(list2[[1]])
[1] "lu"

> print(list2[["age"]])
[1] 33
```

As introduced earlier, there are two ways to index elements in a vector, matrix or list, namely `[]` and `[[]]`. While in the context of vectors and matrices `[]` is almost always used, in the context of list `[]` and `[[]]` both have purposes and they can make a big difference. And finally, `<list>$<name>` is simply a shorthand for `<list>[[<name>]]`. Examples are given below.

```
> print(list2$name)
[1] "lu"

> print(list2$‘number of pc‘)
[1] 3
```

7.2.6 Conditional and Loop Statements

The `if` statement syntax is as follows.

```
if(<condition>){
       <command>
} else if(<condition>){
       <command>
} else{
       <command>
}
```

An example of using `if` statement is given 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
```

where `+` indicates the splitting of commands in multiple lines in the console.

The `for` loop syntax is given as follows.

```
for(<variable> in <vector>){
       <command>
}
```

where `<vector>` is a list of numbers or characters. Examples 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"
```

where `c()` defines a list from items. This command is widely used. More details will be introduced later.

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
```

An example using the above to estimate the ratio of data of a standard normal distribution between ±1 is given below.

```
> N <- 1000000
> counter <- 0
> for(i in rnorm(N)){
       +  if(-1 < i & i < 1){
              +    counter <- counter + 1
              +   }
       + }
> answer <- counter / N # expected: 0.682689492
> print(answer)
[1] 0.682407
```

7.2.7 User-Defined Function

Define a simple function as follows. Run the codes where the function is described before calling the function.

```
<function-name> <- function(<input>[=<default>], ...){
       <command>
       [return(<variable>)]
}
```

7.3 Data Frames

Data frame, just like vector and matrix, is another data structure defined in R. Both matrix and data frame use tables to store data, but data frame does not require all data to have the same data type, which makes it more flexible and hence very widely used in R. Another difference between matrix and data frame is that data frame rows do not have names.

7.3.1 Data Importing

Data frame can be created from scratch by concatenating columns just like matrix, and it is introduced in Section 7.3.4. However, in practice it is more common that a data frame is obtained by importing data from other data sources. Hence, data importing is introduced first.

One of the most common sources of data is a CSV file. R provides convenient functions to read data from CSV files into data frames. Details are given below.

TABLE 7.7
Commonly used commands for data frame exploration.

Syntax (Example)	Description
`nrow(df)`	Number of rows.
`ncol(df)`	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 structure of each column.
`summary(df)`	A summary of the data frame, including some of its statistics features.
`levels(df$<column>)`	The level of the column; only works for descrete-value columns.

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
```

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()` can be used to get and set current working directory respectively.

7.3.2 Data Accessing

There are a few ways to access an element in a data frame as shown below.

```
<df>[<row_index>,<column_index>]
<df>[<row_index>,"<column_name>"]
<df>[,<column-name>][<row_index>]
<df>$<column_name>[<row_index>]
```

where the later two essentially adopt a 2-step precedure and

```
<df>[, <column-name>]
<df>$<column-name>
```

are used to access the entire column in a data frame. Notice that different from a matrix, rows in a data frame do not have names.

Table 7.7 summarizes the commonly used functions on date frames, such as checking its shape and data types. Examples of applying these functions to `iris` data frame from the built-in `datasets` package are given below.

```
> library(datasets)
> nrow(iris)
```

```
[1] 150
> ncol(iris)
[1] 5
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1         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
6         5.4        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         6.7        3.0         5.2        2.3 virginica
147         6.3        2.5         5.0        1.9 virginica
148         6.5        3.0         5.2        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 ...
 $ Species    : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1
     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  setosa   :50
 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  :1.199
 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.  :2.500
> levels(iris$Species) # only works for discrete-value columns
[1] "setosa"   "versicolor" "virginica"
```

Data frame segmentation works similarly with matrix. A segmentation of multiple rows and columns of a data frame is also a data frame. Notice that when a single column is segmented, the result will be a vector by default, and `drop=F` 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
1         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
> print(iris[1,])
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1         5.1        3.5         1.4        0.2  setosa
> is.data.frame(iris[1,])
[1] TRUE
> 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
1           5.1
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. To remove a column, assign `NULL` to the column as follows.

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

7.3.3 Filtering

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

```
<filter> <- <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$Sepal.Length >= 7
> print(iris[filter,])
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
51          7.0         3.2          4.7         1.4 versicolor
103         7.1         3.0          5.9         2.1  virginica
106         7.6         3.0          6.6         2.1  virginica
108         7.3         2.9          6.3         1.8  virginica
110         7.2         3.6          6.1         2.5  virginica
118         7.7         3.8          6.7         2.2  virginica
119         7.7         2.6          6.9         2.3  virginica
123         7.7         2.8          6.7         2.0  virginica
126         7.2         3.2          6.0         1.8  virginica
130         7.2         3.0          5.8         1.6  virginica
131         7.4         2.8          6.1         1.9  virginica
132         7.9         3.8          6.4         2.0  virginica
136         7.7         3.0          6.1         2.3  virginica
> filter <- iris$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 compound filter. The commands can be merged together as follows.

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

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

7.3.4 Data Frame from Integration of Data

So far we have been importing data or using existing `iris` data frame in the well-defined library `datasets` in the case study. Data frame can also be created from scratch using columns of data. To create a new 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))
vec_size_cbd <- rnorm(100, 75, 10)
vec_size_city <- rnorm(100, 100, 15)
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)
vec_price_city <- vec_size_city*rnorm(100, 7500, 1000)
vec_price_suburbs <- vec_size_suburbs*rnorm(100, 5000, 1000)
vec_price <- c(vec_price_cbd, vec_price_city, vec_price_suburbs)
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)
```

which gives the following result

```
> head(mortgage_price)
  Region     Size     Price
1    CBD 81.84889 1154873.0
2    CBD 77.78946  831468.7
3    CBD 84.60477  735265.2
4    CBD 62.42625  829977.5
5    CBD 65.42723  933851.3
6    CBD 82.43867 1208589.0
```

A data frame can also be created from two existing data frames by joining them together using `merge()`. It works like the “JOIN” function in SQL, and it supports “INNER JOIN”, “LEFT JOIN”, “RIGHT JOIN” and “OUTER JOIN”. The syntax is given below.

```
<df> <- merge(
   <df1>,
   <df2>,
   by.x = <column-in-df1>,
   by.y = <column-in-df2>,
   all=FALSE,
   all.x = all, # left join, by default FALSE
   all,y = all, # right join, by default FALSE
   sort = TRUE # sort by the join column
)
```

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

7.4 Basic Data Visualizations Using `qplot()`

The package `ggplot2` provides useful tools for visualization of a data frame including `qplot()` and `ggplot()`. Notice that in the late versions of `ggplot2`, `qplot()` is deprecated to encourage using of the more powerful `ggplot()`. Both functions are powerful enough to produce many different types of plots.

An example of using `qplot()` is given below, just to show some of its capabilities. Run the following codes to get Fig. 7.6. It can be seen that `qplot()` is smart enough to automatically choose plot type, background color, etc.

```
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 7.7 and 7.8 can be obtained.

```
library(ggplot2)
rm(list=ls())
# create data frame
vec_region <- rep(c("CBD","City","Suburbs"),each = 100)
vec_size_cbd <- rnorm(100, 75, 10)
vec_size_city <- rnorm(100, 100, 15)
vec_size_suburbs <- rnorm(100, 150, 25)
vec_size = c(vec_size_cbd, vec_size_city, vec_size_suburbs)
```

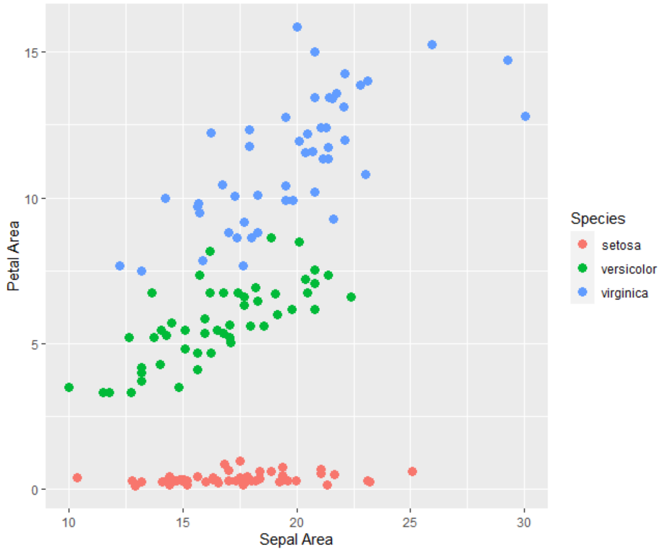


FIGURE 7.6
A demonstration of `qplot`.

```
vec_price_cbd <- vec_size_cbd*rnorm(100, 12500, 2500)
vec_price_city <- vec_size_city*rnorm(100, 7500, 1000)
vec_price_suburbs <- vec_size_suburbs*rnorm(100, 5000, 1000)
vec_price <- c(vec_price_cbd, vec_price_city, vec_price_suburbs)
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
# plot
qplot(data=mortgage_price, x=Size, y=Price, color=Region, geom=c("point
    ", "smooth"))
qplot(data=mortgage_price, x=Region, y=Price.Unit, geom="boxplot")
```

7.5 Advanced Data Visualizations Using `ggplot()`

Function `ggplot()` is the main data visualization tool in `ggolot2` package. It provides very flexible features for data plotting. More details are given in the section. Before digging into the details of `ggolot2`, grammar of graphics is

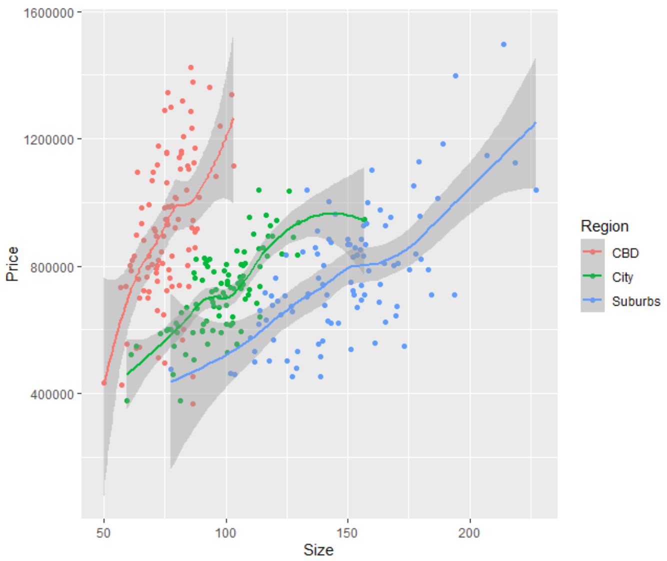


FIGURE 7.7
A demonstration of `qplot` on mortgage price data frame.

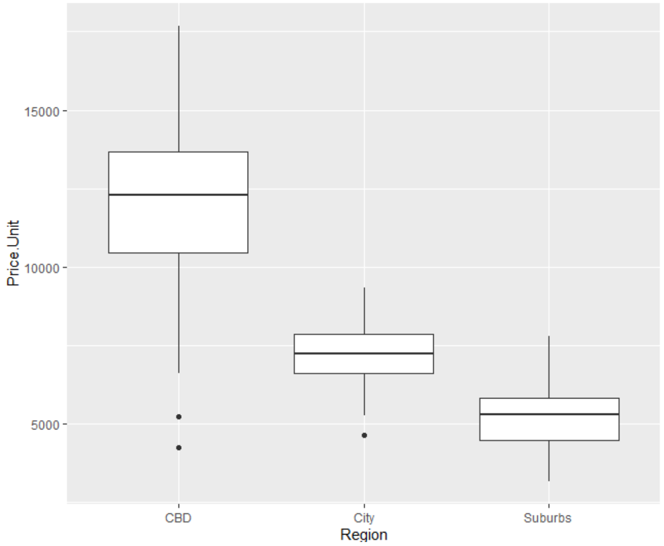


FIGURE 7.8
A second demonstration of `qplot` on mortgage price data frame.

introduced first. Later it will become clear that `ggolot2` is practices grammar of graphics.

7.5.1 Grammar of Graphics

As proposed by Leland Wilkinson's **grammar of graphics**, a chart shall contain multiple independent and reusable layers including

- Data. Original data.
- Aesthetics. How data is mapped to the graph, for example, by dots, curves, color blocks or lines/bars of different length.
- Geometries. The color or shape of each element in the graph.
- Statistics. Information derived from the data.
- Facets. Subplots of different data sets, and how they are aligned and compared.
- Coordinates. The meaning and range of axis.
- Theme. Titles, labels, legends, etc.

A demonstrative Fig. 7.9 is given to illustrate the different layers in a chart.

Notice that the first 3 layers, i.e. data, aesthetics and geometries layers are essential layers, while the rest layers are optional. A graph shall contain at least the first 3 layers for the minimum visualization.

7.5.2 Data, Aesthetics and Geometries Layers

Function `ggplot()` is a very good practice of implementing the grammar of graphics. The basic syntax of `ggplot()`, just for quick demonstration purpose, is given below. In the basic syntax, only data, aesthetics and geometries layers are concerned.

```
p <- ggplot(data=<df>,
          aes(x=<column>, y=<column>, color=<column>, size=<column>)
    ) # data and aesthetics
p <- p + geom_point() # geometries
```

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()`. For example, in

```
p <- ggplot(data=<df>, aes(x=<column>, y=<column>, color=<column>, size
    =<column>))
p1 <- p + geom_point() # scatter plot
p2 <- p + geom_line() # line plot
p3 <- p + geom_line() + geom_point() # both scatter and line
p4 <- p + geom_point(aes(size=<another-column>)) # overwrite aesthetics
     when adding scatter plot
```

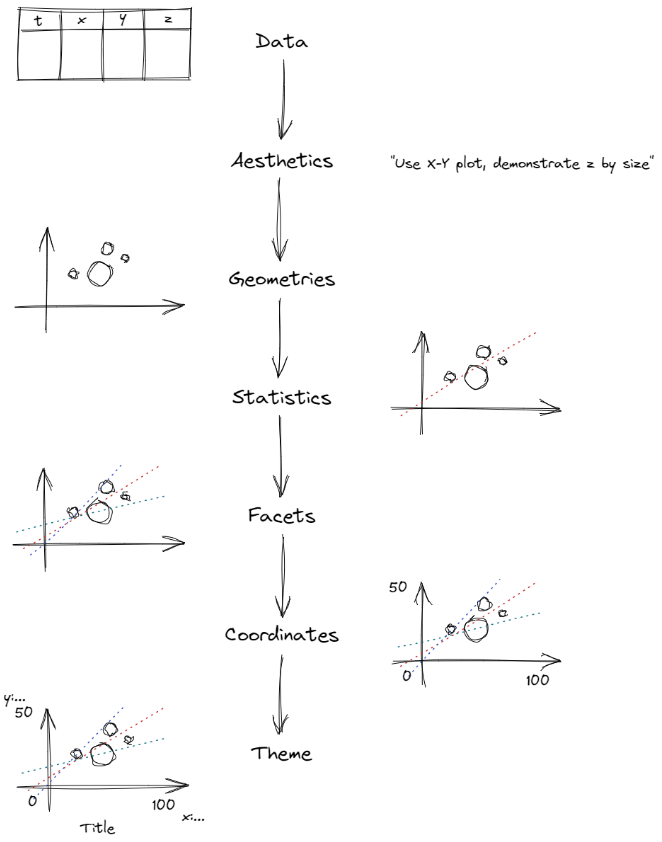

t
x
y
z
Data
Aesthetics
"Use X-Y plot, demonstrate z by size"
Geometries
Statistics
Facets
Coordinates
Theme
50
0
100
y:...
x:...
Title

FIGURE 7.9
Multiple layers in chart design.

the added layers are able to inherit the aesthetics settings, if they are 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 7.8. Notice that while some of the `geom_*` functions are purely geometries layers and only reflect what is in the data itself (such as scatter plot and line plot), others may paired with a statistics layer function (such as histogram). When adding the later type of `geom_*` functions, statistics layers are also implemented behind the screen.

TABLE 7.8
Commonly used `ggplot` geometries and statistics layers commands. A full list can be found at [4].

Geom	Statistics	Description
`geom_point()`	—	Scatter plot.
`geom_line()`	—	Line plot.
`geom_segment()`	—	Drawing a segment.
`geom_polygon()`	—	Polygons plot.
`geom_contour()`	—	Contour plot.
`geom_text()`	—	Text.
`geom_histogram()`	`stat_bin()`	Histogram.
`geom_density()`	`stat_density()`	Kernel density estimate of a distribution.
`geom_bar()`	`stat_count()`	Bar plot (occurrence plot).
`geom_smooth()`	`stat_smooth()`	Linear regression model.
`geom_boxplot()`	`stat_boxplot()`	Distribution with quartiles and outliers.
`geom_violin()`	`stat_ydensity()`	Combination of density estimation and box plot.

7.5.3 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 to provide additional insights. Many functions in Table 7.8 are statistics related. More details are given below.

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)
vec_size_cbd <- rnorm(100, 75, 10)
vec_size_city <- rnorm(100, 100, 15)
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)
vec_price_city <- vec_size_city*rnorm(100, 7500, 1000)
vec_price_suburbs <- vec_size_suburbs*rnorm(100, 5000, 1000)
vec_price <- c(vec_price_cbd, vec_price_city, vec_price_suburbs)
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
```

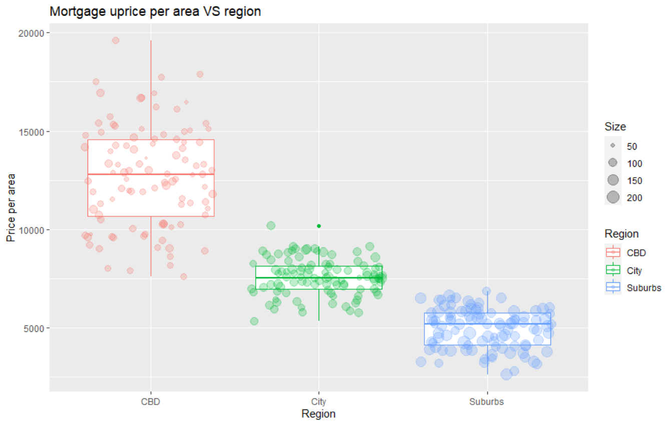


FIGURE 7.10
An example of box plot of the mortgage price data frame using `ggplot()` and `geom_boxplot()`.

```
# 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. 7.10. 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)
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) +
```

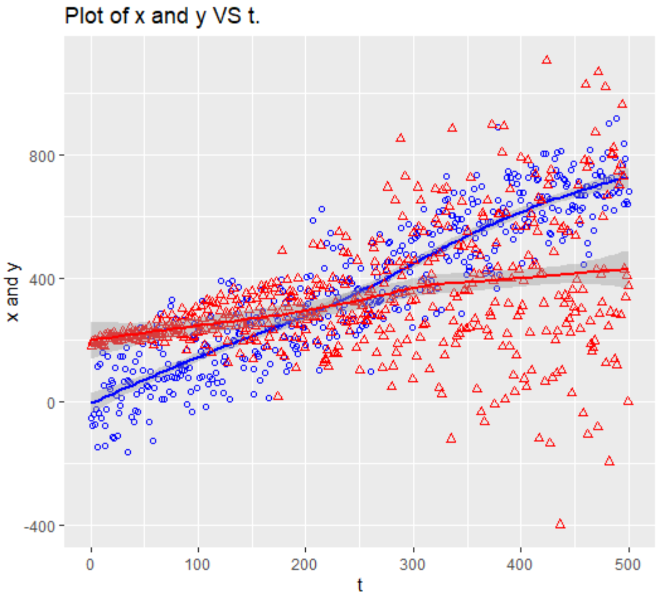


FIGURE 7.11
An example of using `geom_smooth()` for scatter point fitting.

```
geom_smooth(aes(x=t, y=y), color="red")
p
```

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. 7.11.

More functions similar to `geom_smooth()` are summarized in Table 7.9.

7.5.4 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. 7.12.

```
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),
```

TABLE 7.9

Functions that fit smooth lines to scatter points.

Function	Description
`loess()`	Non-parametric method for fitting a smooth line to a scatter plot using locally weighted regression algorithm.
`smooth.spline()`	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 independent 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.
`geom_smooth()`	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.

```
               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,
                          Size = unlist(vec_size),
                          Price = unlist(vec_price))
mortgage_price$Region <- as.factor(mortgage_price$Region)
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size
# plot data
p <- ggplot(data=mortgage_price, aes(x=Price.Unit))
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
```

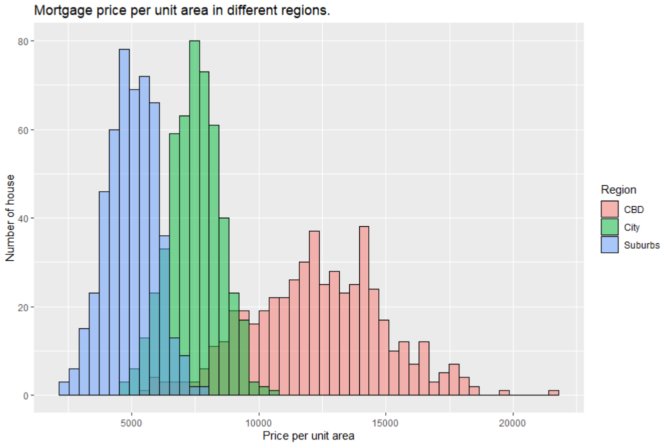


FIGURE 7.12
An example of histogram plot of house price per unit area in different regions in a single plot.

```
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$vec_size_suburbs*rnorm
                   (500, 5000, 1000))
mortgage_price <- data.frame(Region = Region,
                          Size = unlist(vec_size),
                          Price = unlist(vec_price))
mortgage_price$Region <- as.factor(mortgage_price$Region)
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size
# plot data
p <- ggplot(data=mortgage_price, aes(x=Price.Unit))
p <- 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
p + facet_grid(Region˜.) # put subplots for different regions in rows
p + facet_grid(.˜Region) # put subplots for different regions in
    columns
```

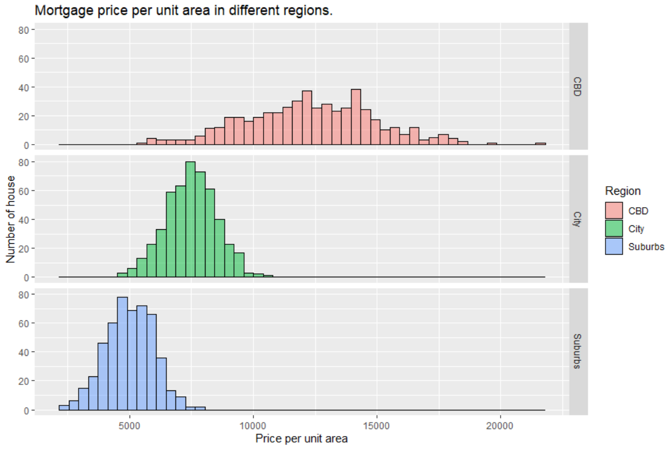


FIGURE 7.13
Use facets to plot the histogram of price per unit are of the house in different regions (subplots in rows).

The results are given in Figs. 7.13 and 7.14, depending on the subplot designs.

7.5.5 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.

```
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$vec_size_suburbs*rnorm
                   (500, 5000, 1000))
mortgage_price <- data.frame(Region = Region,
                          Size = unlist(vec_size),
                          Price = unlist(vec_price))
mortgage_price$Region <- as.factor(mortgage_price$Region)
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size
```

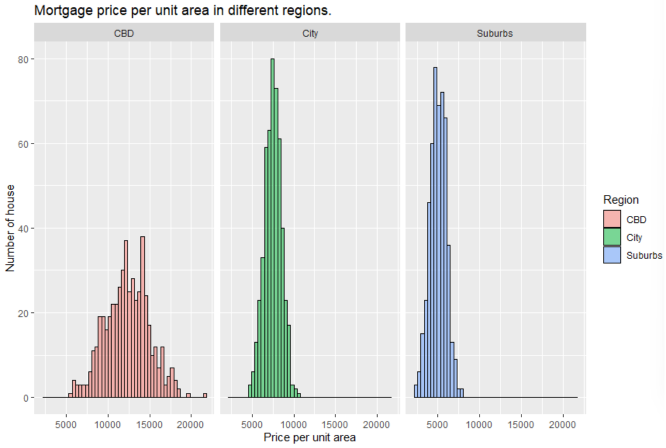


FIGURE 7.14
Use facets to plot the histogram of price per unit are of the house in different regions (subplots in columns).

```
# plot data
p <- ggplot(data=mortgage_price, aes(x=Size, y=Price))
p <- p + geom_point(aes(color=Region))
p + xlim(50, 200) + ylim(400000, 1200000) # first chart
p + coord_cartesian(xlim=c(50, 200), ylim = c(400000, 1200000)) #
    second chart
```

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. 7.15 and 7.16, respectively. The difference can be observed near the boundary.

7.5.6 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)
vec_size = list(vec_size_cbd = rnorm(500, 75, 10), vec_size_city =
    rnorm(500, 100, 15), vec_size_suburbs = rnorm(500, 150, 25))
```

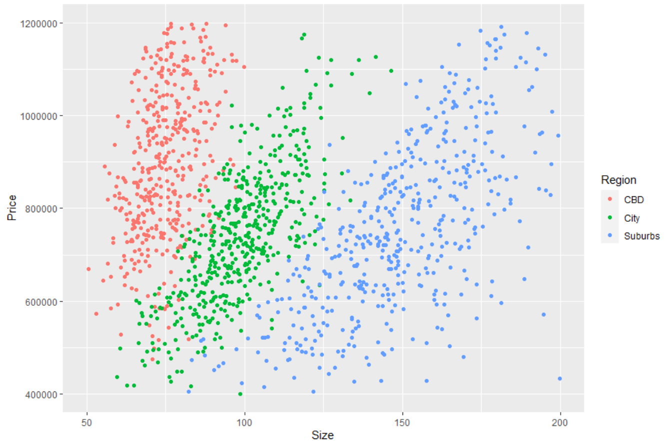


FIGURE 7.15
Add coordinates layer using `xlim()` and `ylim()`.

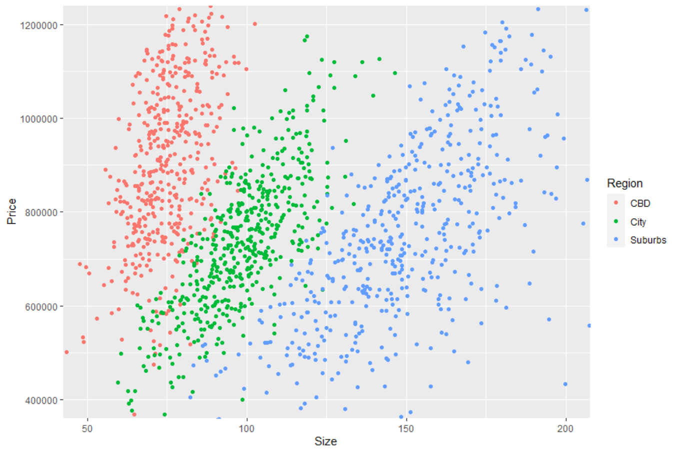


FIGURE 7.16
Add coordinates layer using `coord_cartesian()`.

```
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$vec_size_suburbs*rnorm
                   (500, 5000, 1000))
mortgage_price <- data.frame(Region = Region,
                          Size = unlist(vec_size),
                          Price = unlist(vec_price))
mortgage_price$Region <- as.factor(mortgage_price$Region)
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size
# plot data
p <- ggplot(data=mortgage_price, aes(x=Price.Unit))
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)
  )
```

The resulted chart is given in Fig. 7.17. Compare it with Fig. 7.12 to see the differences by applying `theme()` in the themes layer.

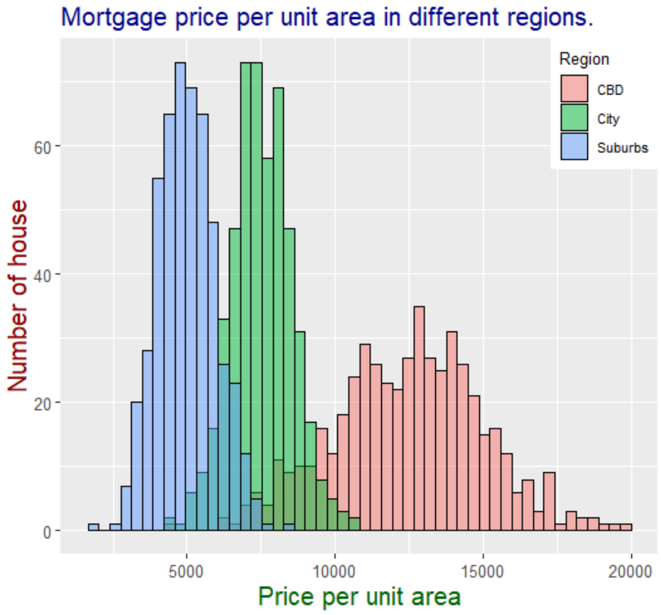


FIGURE 7.17
Mortgage price chart with theme.

8

R (Part II: Practice)

CONTENTS

8.1 Data Preparation 107
    8.1.1 Data Type Conversion 107
    8.1.2 Handling Missing Data 109
8.2 Connectivity with Data Sources 112

This chapter introduces workflows and advanced R skills that become handy in practical problems.

8.1 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, an AI engine, or for further statistics analysis. 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.

8.1.1 Data Type Conversion

It is important 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 if necessary convert data types using the following commands.

```
<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 shown by printing `my_factor` 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)
> 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, `S$6,125.50` in many occasions should be treated just as `6125.5`. 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.

8.1.2 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.

Flag Missing Data using `NA`

In R, `NA` is a special variable used to indicate a missing value. A general idea is to “flag” the missing data in the original data source, what format it may look like, using `NA` during or after the data importing. After that, use a special program in R to filter `NA` and deal with them separately. 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.

Locate and Filter `NA`

Notice that `NA` is treated as a logical data type in addition to `TURE` and `FALSE` in a logical expression. These 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
```

This applies to filtering. In filtering, when a variable of value `NA` is asserted with a criterion, the return is most likely `NA`. The filter often does not know how to deal with `NA`, hence it would simply return the rows with `NA` anyway. This can become inconvenient sometimes. An example is given below.

```
# create data frame
region <- rep(c("CBD","City","Suburbs"), each = 5)
vec_size <- list(vec_size_cbd = rnorm(5, 75, 10),
               vec_size_city = rnorm(5, 100, 15),
               vec_size_suburbs = rnorm(5, 150, 25))
vec_price <- list(vec_price_cbd = vec_size$vec_size_cbd*rnorm(5, 12500,
     2500),
               vec_price_city = vec_size$vec_size_city*rnorm(5, 7500,
                   1000),
               vec_price_suburbs = vec_size$vec_size_suburbs*rnorm(5,
                   5000, 1000))
vec_size <- unlist(vec_size)
vec_size[sample(1:15, 2)] = NA
vec_price <- unlist(vec_price)
vec_price[sample(1:15, 2)] = NA

mortgage_price <- data.frame(Region = region,
                          Size = vec_size,
                          Price = vec_price)
mortgage_price$Region <- as.factor(mortgage_price$Region)
mortgage_price$Price.Unit <- mortgage_price$Price / mortgage_price$Size
print(mortgage_price)
# find price > 750000
mortgage_price[mortgage_price$Price>750000,]
```

The return is as follows.

```
                 Region     Size    Price Price.Unit
vec_size_cbd1       CBD  73.47779 947130.9 12890.031
vec_size_cbd2       CBD        NA  678842.3        NA
vec_size_cbd3       CBD  85.06748 1261029.7 14823.875
vec_size_cbd4       CBD  92.35454       NA        NA
vec_size_cbd5       CBD  71.06649 1276168.6 17957.388
vec_size_city1     City  68.59874 491912.0  7170.861
vec_size_city2     City 118.39441 804794.5  6797.572
vec_size_city3     City        NA  534591.5        NA
vec_size_city4     City  95.57428 583044.7  6100.436
vec_size_city5     City  74.45356 468788.0  6296.382
vec_size_suburbs1 Suburbs 136.88432 939436.6 6862.997
vec_size_suburbs2 Suburbs 136.57799 810070.0 5931.190
vec_size_suburbs3 Suburbs 189.66586 561089.2 2958.303
vec_size_suburbs4 Suburbs 195.97476 913572.2 4661.683
vec_size_suburbs5 Suburbs 190.63082       NA        NA
```

```
                 Region     Size    Price Price.Unit
vec_size_cbd1       CBD  73.47779 947130.9 12890.031
vec_size_cbd3       CBD  85.06748 1261029.7 14823.875
NA                 <NA>        NA       NA        NA
vec_size_cbd5       CBD  71.06649 1276168.6 17957.388
vec_size_city2     City 118.39441 804794.5  6797.572
vec_size_suburbs1 Suburbs 136.88432 939436.6 6862.997
vec_size_suburbs2 Suburbs 136.57799 810070.0 5931.190
vec_size_suburbs4 Suburbs 195.97476 913572.2 4661.683
NA.1               <NA>        NA       NA        NA
```

In the above example, the intention of the program is to find all the houses with price larger than 750000. The program is able to filter out those houses cheaper than the threshold. However, there are two rows of `NA` returned, as explained earlier.

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.

8.2 Connectivity with Data Sources

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

9

Python (Part I: Basics)

CONTENTS

9.1 NumPy 113
9.2 SciPy 115
9.3 Matplotlib and Seaborn 116
  9.3.1 Matplotlib 116
  9.3.2 Seaborn 117
9.4 Pandas 119
  9.4.1 Data Importing 120
  9.4.2 Series and Data Frame 122

Python has become increasingly popular for data science in recent years. Many libraries and tools have been developed for Python to enhance its data analysis and visualization capabilities, including `numpy`, `scipy`, `scikit-learn`, `pandas`, `matplotlib`, `tensorflow`, and `pytorch`.

This chapter, together with subsequent chapters, introduces commonly used tools that data science adopts using Python. This part of the notebook is more application-driven, and only basic implementations are introduced. The theory supporting machine learning and artificial intelligence is not explored in depth.

It has become increasingly popular 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 programming languages. A detailed introduction to the installation and usage of Conda and Jupyter Notebook is not covered here. They are used when demonstrating examples in this chapter and subsequent chapters.

9.1 NumPy

When it comes to any sort of numerical computation, one of the most popular Python packages is NumPy. NumPy allows quick deployment of numerical vectors, matrices, and tensors, as well as associated efficient numerical calculations. It is the "MATLAB" equivalent 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
     format
```

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

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
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.

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
x > 5 # array([False, False, False, False, False, True, True, True,
    True])
x[x>5] # array([6, 7, 8, 9])
```

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.

9.2 SciPy

SciPy is a collection of libraries containing comprehensive algorithms widely used in scientific and technical calculations. Note that NumPy also has built-in basic and commonly used algorithms in the library, such as FFT. For algorithms not included in NumPy, there is a chance they can be found 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/*.

9.3 Matplotlib and Seaborn

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

9.3.1 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. 9.1.

```
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
```

```
plt.title("Fibonacci␣Series")
plt.xlabel("n")
plt.ylabel("f(n)")
```

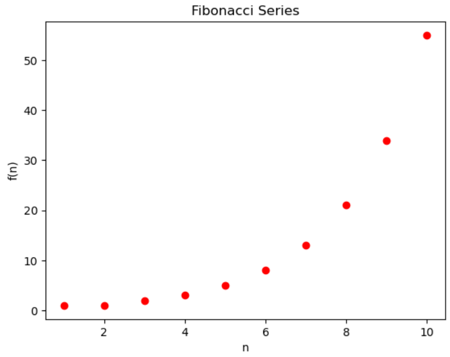


FIGURE 9.1
Plot Fibonacci series as scatter plot.

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.

9.3.2 Seaborn

Seaborn is another visualization library built on top of Matplotlib. It attempts to standardize plots with simple function calls. It sacrifices 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. 9.2.

```
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 categorical values) is given below. Note that the attribute whose values are put into categories is assigned to x in `seaborn.countplot()`. The result is given in Fig. 9.3.

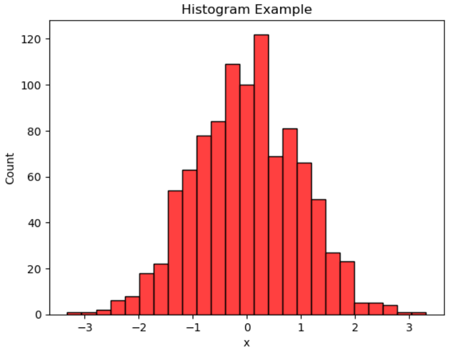


FIGURE 9.2
Histogram plot using Seaborn.

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

response = ["yes", "no", "no", "yes", "yes", "yes", "yes", "yes", "yes"
    , "no", "yes", "yes", "NA", "NA"]
plt.figure()
sns.countplot(x=response, color="blue")
plt.title("Count␣Plot␣Example")
plt.xlabel("response")
plt.ylabel("Count")
```

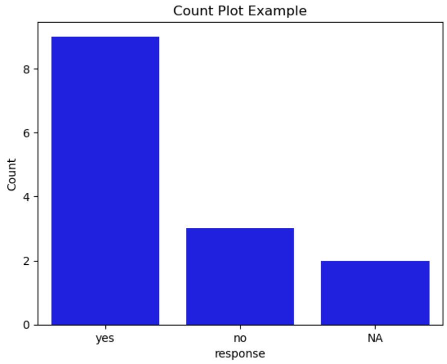


FIGURE 9.3
Count plot using Seaborn.

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

$(0\% - 25\%)$, $Q_1 < x < Q_2$ $(25\% - 50\%)$, $Q_2 < x < Q_3$ $(50\% - 75\%)$, 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. 9.4.

```
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")
```

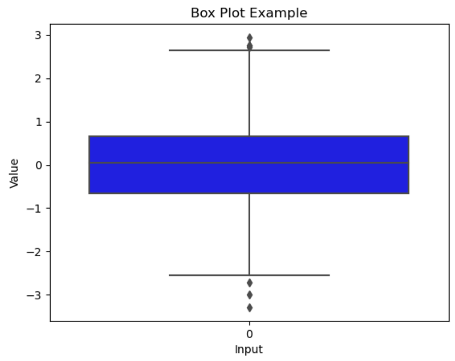


FIGURE 9.4
Box plot using Seaborn. The box gives IQR. The bars below and above the box give $Q_1 - 1.5 \times \text{IQR}$ and $Q_3 + 1.5 \times \text{IQR}$, respectively. The dots are outliers.

There are many more plot functions in Seaborn. For example, scatter plot `seaborn.scatterplot()` works similar with Matplotlib as shown by Fig. 9.1. 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.

9.4 Pandas

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__))
```

9.4.1 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.

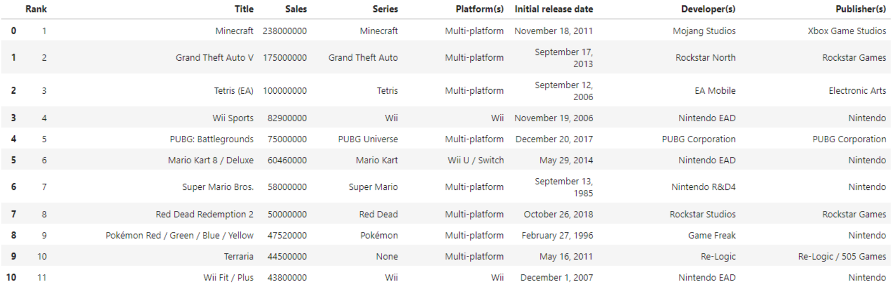
```
best_selling_games_df = pd.read_csv("best-selling-video-games.csv")
best_selling_games_df
```

which reads all the information in the CSV table into a data frame as shown by Fig. 9.5. It is possible to import only selected columns as follows.

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

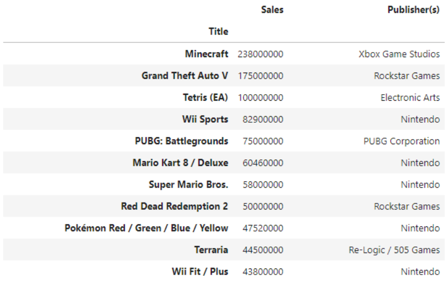
When no index column is specified, pandas will add an additional auto-incremental column and use it as the index column, as shown in Fig. 9.5 by the most-left column. When a column index is specified, pandas will use that column as index column. An example is given below. The result is shown in Fig. 9.6.

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



	Rank	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	Wii	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 9.5
The simplest data frame importing using pandas.



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

FIGURE 9.6
Specifying index column and reading only selected columns using pandas.

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.

```
# form sub data frame
best_selling_games_df = pd.read_csv("best-selling-video-games.csv")
sub_games_df = best_selling_games_df[["Title", "Sales", "Publisher(s)
    "]]
sub_games_df
# form single-column sub data frame
sub_games_df = best_selling_games_df[["Title"]]
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.

9.4.2 Series and Data Frame

10

Python (Part II: Python for Data Science)

CONTENTS

10.1 Quick Review 124
  10.1.1 AI Pipeline 124
  10.1.2 Data Preparation and Model Evaluation 124
  10.1.3 Commonly Seen ANN Use Cases 125
  10.1.4 Computer Vision 126
  10.1.5 Natural Language Processing 126
10.2 TensorFlow 126
  10.2.1 TensorFlow Basics 127
  10.2.2 Classification and Regression 127
  10.2.3 Computer Vision 131
  10.2.4 General Sequential Data Processing 132
  10.2.5 Natural Language Processing 132
  10.2.6 TensorFlow on Different Platforms 132
10.3 PyTorch 132
  10.3.1 PyTorch Basics 132
  10.3.2 Classification and Regression 132
  10.3.3 Computer Vision 132
  10.3.4 General Sequential Data Processing 132
  10.3.5 Natural Language Processing 132
  10.3.6 PyTorch on Different Platforms 132

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.

10.1 Quick Review

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

10.1.1 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.

10.1.2 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.

“sloppy

10.1.3 Commonly Seen ANN Use Cases

“nobreak

10.1.4 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

10.1.5 Natural Language Processing

“nobreak

10.2 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.

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.

10.2.1 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 "TensorFlow 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
```

10.2.2 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,
                         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:
                  ␣{}".format(self.num_training, 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,␣etc.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
              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:
                     print("Validation␣set␣is␣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,
              "activation": "relu"
       },
       {
```

```
                "type": "dense",
                "node": 64,
                "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.

10.2.3 Computer Vision

“nobreak

10.2.4 General Sequential Data Processing

“nobreak

10.2.5 Natural Language Processing

“nobreak

10.2.6 TensorFlow on Different Platforms

“nobreak

10.3 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.

10.3.1 PyTorch Basics

“nobreak

10.3.2 Classification and Regression

“nobreak

10.3.3 Computer Vision

“nobreak

10.3.4 General Sequential Data Processing

“nobreak

10.3.5 Natural Language Processing

“nobreak

10.3.6 PyTorch on Different Platforms

Part IV

Semantic Web

11

Semantic Web Basics

CONTENTS

11.1 Web of Data 137
  11.1.1 Web 1.0 and 2.0 138
  11.1.2 Web 3.0 139
  11.1.3 Semantic Web Vision 140
  11.1.4 Semantic Web Stack 140
  11.1.5 Semantic Web Limitations and Challenges 144
11.2 Ontology 144
  11.2.1 Philosophy Perspective 145
  11.2.2 Semantic Web Perspective 145
  11.2.3 Ontology Types and Categories 146
11.3 Logic 146
  11.3.1 Different Semantics from the Same Syntax 147
  11.3.2 Logic Framework 148
  11.3.3 Logical Expression 150
  11.3.4 Logical Equivalence 151
  11.3.5 Logical Reasoning 152

Ontology is the philosophical study of the nature of being, existence, or reality. It concerns “what is everything” and “how to define everything” in the context of inductive and deductive reasoning. It discusses how humans abstract and preserve knowledge.

Ontology inspires computer scientists regarding how information can be stored and exchanged efficiently using the internet. The solution is known as the semantic web, an internet-based knowledge base schema. The internet powered by the semantic web (together with other technologies) defines Web 3.0, a new-generation internet framework.

Note that there are subtle differences between “internet” and “Internet”. The lowercase “internet” refers to the technology that bridges machines to form a network of any size, while the uppercase “Internet” refers to the specific internet that links the entire world together. In this part of the notebook, however, they are used interchangeably.

11.1 Web of Data

The internet is a technology that enables information exchange among machines. With the power of the internet, users can obtain needed information from remote servers, databases, or knowledge bases.

In the early days, using the internet required professional skills. Nowadays, everyone can access the internet using a graphical-interface browser that can be easily found on a computer or mobile device. Public knowledge bases such as Wikipedia have made obtaining information much easier.

11.1.1 Web 1.0 and 2.0

Under the **Web 1.0** framework, which was popular in the early stage of internet development, information is stored on individual servers in a static manner. Users can browse the contents, but cannot edit them. It is essentially one-way data transmission. Authorities such as news companies provide the information, and users consume it. Examples of Web 1.0 implementations include news websites, static web gallery, etc.

One-way data transmission in Web 1.0 cannot meet the expectations of users who want to share their information with other users on the internet. Thanks to advances in information science and communication, under the **Web 2.0** framework users can interact with the internet bidirectionally. A user can search and filter data from the servers and even upload his own data and share it with others. The internet became far more powerful in terms of information exchange. The source of information does not necessarily come from the authority. The users are generators and consumers of information at the same time. Examples of Web 2.0 implementations include Blog, Twitter, YouTube and Weibo.

Web 2.0 is extremely popular and successful even to date. Yet, under the background of industry 4.0, big data and data-driven modeling of almost everything, there are still limitations and downsides to Web 2.0 that we would wish to improve.

One of the biggest challenges encountered with Web 2.0 is how to quickly locate desired information among the vast amount of irrelevant data. Conventionally in Web 2.0, keyword-based search engines are used. These search engines do not comprehend the contextual knowledge of the contents, and as a result may fail to return what a user truly expects. Users often need to manually pick up relevant information from the search results. Nowadays search engines are becoming smarter, and can sometimes pre-filter results. Nevertheless, it is still quite common that many returned results are useless to the user. Keyword-based engines also struggle with polysemy, synonyms, and implicit information from pictures in the search range, again due to the lack of understanding of the contents.

The problem here, however, is not caused by the searching engines alone. It is rather that the information stored on the internet does not come with its corresponding semantics in the first place. Today most of the information on the internet is stored in HTML format. HTML tells only the contents but not the meaning behind them. It is difficult for a machine to understand the meaning of the information from HTML corpus. This potentially makes it difficult for a machine to retrieve data efficiently. Even with the recent breakthrough in LLM enabling the machine to summarize articles, it is still unpractical for it to go through all the returned contents of a searching engine which sometimes contains hundreds of pages of information.

Structure determines functions. To solve the problem once for all, new data storage and sharing model needs to be introduced.

11.1.2 Web 3.0

The goal of **Web 3.0** is to allow more efficient information retrieval and sharing using the internet. Ideally, the search engine should truly understand the user's demand and return only the most relevant information summarized in a clear manner. If there is no readily available response on the internet, the search engine should derive the response based on existing information, i.e., it should be capable of simple reasoning.

For this purpose, there are at least the following two development trends:

- Let the LLM-based AI remember all the knowledge. The LLM should be smart enough to precisely capture what the user wants and get back to him with useful and accurate information. This leads to chatbots and copilots.
- Create a powerful and flexible NoSQL database. Make the information in the database readable for both humans and machines, and somehow integrate the semantics of the contents into the database. This leads to **semantic web**.

In practice, it is most recommended to use both LLM-based AI and semantic web simultaneously. This is because both LLM-based AI and semantic web have drawbacks when used alone.

- LLM-based AI
    - Time and computational load wise expensive to expand the knowledge base;
    - Lack the ability of reasoning;
    - A chance to produce misleading information;
    - Cannot be merged and migrated easily because the "knowledge" in the form of regression coefficients is not human or machine readable;
- Semantic web

  - Steep learning curve to use;
  - Not good at summarizing and formatting the response in a human-friendly manner.

Web 3.0 schema focuses on semantic web. The word "semantics" in this context specifically refers to the relations of objects and properties that can be used for machine-driven descriptive reasoning. See later sections for details. Semantic web is so important that it is often considered synonymous with Web 3.0, although the broader vision of Web 3.0 includes other features as well.

Another feature of Web 3.0 is distributed storage of information. Distributed storage has both advantages and disadvantages. On one hand, it provides higher resilience against data loss. On the other hand, it adds challenges to the searching and collecting of information during the querying. Technologies like InterPlanetary File System (IPFS), blockchain, and distributed databases are used to address these challenges.

This notebook discusses the technologies used in semantic web such as **resource description framework** (RDF) model, **RDF schema** (RDFS) and **web ontology language** (OWL). We will see how we can use semantic web to collect and organize information, and create a knowledge base for both humans, machines, and AI.

11.1.3 Semantic Web Vision

Semantic web database is "semantic" due to its data model. Semantic web data model not only stores objects and their properties, but also the relationships among objects and properties. The relationships, often represented by graphs, can be used for descriptive logic reasoning. The logic reasoning allows new information to be derived based on existing facts. In addition, the semantic web is more scalable, reusable, and reader-friendly for both humans and machines compared to AI methods.

Many research institutes and organizations have tried building semantic webs of different scales. An example is DBPedia (*dbpedia.org*). The goal of DBPedia is to create something like Wikipedia, but using semantic web.

11.1.4 Semantic Web Stack

A commonly seen semantic web stack looks like Fig. 11.1.

The layers and components in each layer are briefly introduced as follows.

- Web platform layer
  - Uniform resource identifier (URI) and/or internationalized resource identifier, often a string used to identify and trace an information resource.
  - Communication protocols, such as HTTP and HTTPS.

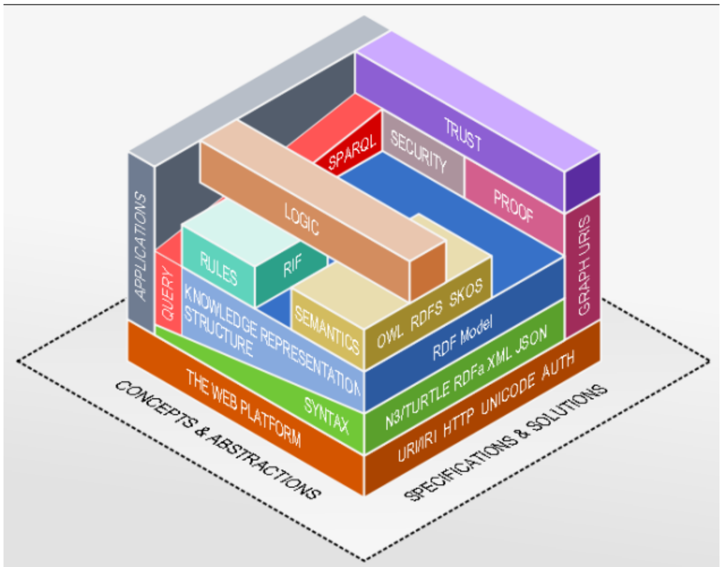

APPLICATIONS
SPARQL
SECURITY
TRUST
PROOF
GRAPH URIS
LOGIC
QUERY
RULES
RIF
KNOWLEDGE REPRESENTATION
STRUCTURE
SEMANTICS
OWL RDFS SKOS
RDF Model
SYNTAX
N3/TURTLE RDFa XML JSON
THE WEB PLATFORM
URI/IRI HTTP UNICODE AUTH
CONCEPTS & ABSTRACTIONS
SPECIFICATIONS & SOLUTIONS

FIGURE 11.1
Semantic web stack [2].

  - Text encoding methods, such as unicode, utf-8.
  - Authentication methods.

- Syntax
  - File formats to store information, such as RDF/XML and JSON-LD.
- Knowledge representation, semantics, and interfacing tools
  - RDF (by default) which uses triples to represent a graphical database.
  - RDFS and OWL which expand the capability of RDF model.
  - SPARQL which is the recommended language for semantic web query (SPARQL 1.0) and manipulation (SPARQL 1.1).
- Logic
  - Logic statements and reasoning that semantic web supports for query.
  - Rules that describes what query can be executed.

The knowledge and semantics are stored using the RDF/RDFS model. The RDF/RDFS is often further powered by OWL. RDF serves as the foundation with the basic structure. RDFS expands the capability of RDF by introducing new vocabularies such as “class”, “subclass”, “property”. And finally OWL enables a set of tools to define complex ontologies and create rigorous and complex semantics. As an analogy, think of RDF as the paper and pencil, RDFS the 24-color crayon set, and OWL the painting skills. Together they make a sophisticated and richly structured picture.

It is worth mentioning that many syntaxes such as RDF/XML can be used as markup languages to describe RDF/RDFS/OWL used in a semantic web. More details are introduced in later chapters.

SPARQL Protocol and RDF Query Language (SPARQL) is the recommended query language for querying and manipulating the semantic web. It allows users to search, retrieve, and modify information stored in the RDF model. The syntax of SPARQL is designed to look similar to SQL, and many commands in SPARQL have counterparts in SQL, such as `SELECT`, `WHERE`, `GROUP BY`, and `ORDER BY`. However, the backend technologies of a semantic web engine (known as triplestore or RDF management system) and a relational database management system differ largely. For example, triplestore relies heavily on graph theory and graph pattern mapping when searching through the semantic web.

An example of semantic web is given below.

Example: Semantic Web on Animals

The following examples demonstrates the use of RDF/RDFS model and OWL to create a small semantic web on animals.

RDF/RDFS Only

```
@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, RDF/RDFS is used 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.

RDF/RDFS with 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, OWL is used 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.

The above semantic web can be queried by SPARQL. An example is given below.

```
PREFIX ex: <http://example.org/>

SELECT ?mammal
WHERE {
  ?mammal rdf:type/rdfs:subClassOf* ex:Mammal .
}
```

There is a learning curve for SPARQL. Commercialized semantic web applications such as WolframAlpha (*wolframalpha.com*) often provide a "search bar" with some natural language processing capability which triggers SPARQL query according to the user's input. It is possible to use a powerful LLM-based chatbot to assist SPARQL query as well.

11.1.5 Semantic Web Limitations and Challenges

Though semantic web is powerful, building semantic web can be challenging and requires a lot of careful design and 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.

However, things may change due to the recent advent in internet-of-things (IoT, as defined in Industry 4.0) and LLM-based copilots. IoT devices generates large amount of data, which makes the base of building large-scale knowledge. Copilots are useful with converting data from other formats into RDF model.

11.2 Ontology

Ontology has very rich meanings from both philosophy and semantic web perspectives.

11.2.1 Philosophy Perspective

Knowledge is the overlapping part of truth and human beliefs, i.e., it is the truth that humans know of being the truth. **Ontology** is the methodology of storing and communicating knowledge.

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

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

The discussion of these questions can be traced back to 300 BC or even earlier. Aristotle defined a system to structure and reason knowledge. The famous Aristotelian logic "major premise + minor premise → conclusion" is a systematic way of reasoning new knowledge. Aristotelian logic is an important tool of ontology, and it inspires the proposition of the semantic web.

11.2.2 Semantic Web Perspective

In the context of the Semantic Web, ontology is a formal, machine-readable representation of the domain knowledge in a specific area. It serves as a shared vocabulary for describing and reasoning the knowledge within that domain. Notice that unlike natural language which can be ambiguous, semantic web shall be described clearly, precisely and consistently.

In practice, RDF/RDFS and OWL can be used to express the ontology. The followings vocabularies are defined in RDF/RDFS and OWL.

- Class. A class is an abstraction of objects sharing some similarities.
- Properties. A property defines a feature of a class. Triples are often used to describe a property. A triple follows the form of "subject + property + object".

- Relations. A relation describes class-to-class and property-to-property connections. Relations can be expressed by triples where both subject and object are classes, or by class and property hierarchies.
- Constraints. A constraint describes the rules enforced on a property.
- Instance. An instance is a realization of a class.

11.2.3 Ontology Types and Categories

Ideally in the vision of W3C, all ontology models should be linked together using URIs and form a global model just like the global Internet. In this ultimate global model, ontology is divided into layers. The higher the layer, the more general the knowledge. The lower the layer, the more specific the knowledge within a particular domain, application or task. An example is given in Fig. 11.2 on spacecraft control. Lower-layer ontology can inherit and modify knowledge from the upper-layer ontology. In practice, there might not be a clear boundary between two adjacent layers.

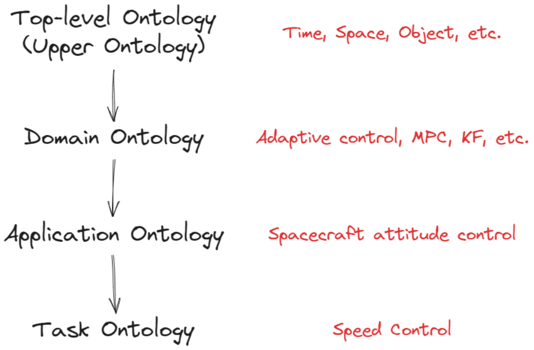

Top-level Ontology
(Upper Ontology)
Time, Space, Object, etc.
Domain Ontology
Adaptive control, MPC, KF, etc.
Application Ontology
Spacecraft attitude control
Task Ontology
Speed Control

FIGURE 11.2
A demonstrative example of ontology level using control engineering.

Other ways of leveling ontology include using the expressiveness. The "light-weight" ontology is informal, less semantic, and supports less comprehensive logic. The "heavy-weight" ontology, on the other hand, is formal, more semantic, and supports more comprehensive logic and reasoning up to first-order logic. The vocabulary also grows with the ontology complexity.

11.3 Logic

Humans are good at deriving new knowledge from existing knowledge. The systematic way of doing so is **logic**. The term "formal logic" rigorously defines logic reasoning methods and procedures to automate logic inference by machines.

Notice that logic and logical expression are by themselves very complex and rich concepts. It is not possible to include all the details about them in the scope of this notebook. Only a brief scratch is given.

11.3.1 Different Semantics from the Same Syntax

Given the same information, different people and/or machines can draw different semantics. Consider the following example. This is a piece of code written in Python.

```
ax = []
for i in range(20):
        if i<=1:
                ax.append(i)
        else:
                ax.append(ax[i-1]+ax[i-2])
```

Three interpreters are studied, the Python interpreter, the AI model, and the human cognition. Different interpreters draw different semantics from the same code above.

The Python interpreter is purely syntax-driven. When it executes a script, it follows the provided instructions without any broader comprehension or anticipation of what the code is doing. In the context of this example, the interpreter does not recognize that it is generating the famous Fibonacci sequence. It simply follows the rules of the script, calculating and outputting each number in the series as instructed.

On the other hand, an AI model like GPT-3 does possess a level of semantic understanding. Through training on massive data corpus especially codes, it can associate the above Python script with Fibonacci series. This understanding is not innate but rather a result of statistical inference and pattern matching from the training data. In other words, in the training data there are similar codes which is referred as "the code to generate Fibonacci series". When asked about the script, it can provide a summary or explanation based on its learned associations, including the name of the series and the purpose of each line of the code. It can even suggest improvement or fix bugs, so long as in the training data a more efficient realization of the code is provided.

Finally, there's the human level of semantics, which is by far most advanced and intuitive. Not only can a human understand the Python script and the concept of the Fibonacci series, they can also infer its broader mathematical properties, such as its relation to the golden ratio, and its general behavior. One good at mathematics can intuitively understand that the series will converge to the golden ratio not only when starting with 0 and 1 but with any two initial positive integers, even if he is not specifically taught that knowledge. This level of understanding is a combination of learned knowledge, pattern recognition, and the ability to extrapolate or generalize from existing information.

One of the goals of semantic web is to enable a machine to understand the semantics as much as possible, hopefully to gain the capability of logic reasoning just like a human.

11.3.2 Logic Framework

Different logic frameworks have different levels of complexity, hence different capabilities of expressing semantics. Commonly seen logic frameworks are briefly introduced as follows.

Propositional Logic

Propositional logic is the fundamental of logic reasoning. In propositional logic, knowledge is represented by either simple facts which are known as propositions, or facts connected with "AND", "OR", "NOT", "IF ... THEN ...", "IF AND ONLY IF" which are known as compound propositions.

First-Order Logic

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 $\wedge$, disjunction $\vee$ and negation $\neg$. A formal way of representing logic expressions are defined. For example, using the following to represent "all humans are mortal"

$$\forall x \, (\text{Human}(x) \rightarrow \text{Mortal}(x))$$

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 it is true that the item belongs to human, then it must be true that the item belongs to mortal".

Description Logic

Description logic (DL) is a subset of FOL. Derivation of DL from FOL is not under the scope of this notebook. Some key features of DL are summarized as follows.

- Define classes and subclasses.
- Define properties (roles) and associated domains and ranges.

- Define restrictions on classes and properties.
- Support universal and existential quantifiers.

Semantic web uses OWL to implement DL on computers. More details are given in Section 13.

Undecidability in FOL

We would surely want to introduce highly expressive and complex formalisms to the existing RDF/RDFS model. This motivates OWL, which enables DL in semantic web. More details of OWL is introduced later in Section 13.

Notice that DL instead of FOL is widely used in semantic web due to the fact that FOL is undecidable.

A logic is decidable if there is an algorithm that can determine whether any given statement in the logic is true or false. If no such algorithm exists for a logic, it is undecidable. G*ö*del's Incompleteness Theorems indicates that in any sufficiently powerful logical system such as FOL, there are statements that are true but cannot be proven within the system. FOL, therefore, is undecidable. An FOL reasoning algorithm may not terminate in finite time. The proof of this theorem can be found elsewhere and is not given in this notebook.

While FOL inference is not decidable, DL inference, on the other hand, is decidable. That is one of the reasons DL is preferred over FOL in semantic web.

Attributive language with complement (ALC) is one of the ways to describe a simple DL, and it forms an important subset of OWL. Understanding ALC is helpful with learning OWL in later sections. A brief introduction of ALC is given below.

"Concept" (corresponding with class in RDF/RDFS) and "role" (corresponding with property in RDF/RDFS) are introduced in ALC. Top concept (root class) and bottom concepts (leaf classes) are defined. Each property is associated with a range, which is basically a set of values that the property can take.

Constructors are used to describe the concepts, roles and ranges used in ALC. Conjunction, disjunction, negation, existential and universal quantifiers are supported. For example, let R be a role and C be a concept, and $\forall R.C$ means that all roles "R" must take values from concept "C". Likewise, $\exists R.C$ means that there is at least one role "R" whose value is taken from concept "C". Concept relations are defined. Commonly used concept relation constructors are inclusion (to describe subclass), equality (to assign class), union, intersection, complement, etc.

ALC uses **terminological knowledge statements** (define concepts and roles schema) and **assertional logic statements** (insert instances) to record knowledge. Examples are given below. Using terminological knowledge we can define a teacher as

```
Teacher ≡ Person ∧ ∃ HasStudent.Student ∧ ∃ Teaches.Lecture
```

which translates to "a Teacher is a Person, and it has at least one role HasStudent whose value is from Student, and has at lease one role Teaches whose value is from Lecture", where "Person", "Student", "Lecture" are concepts and "HasStudent", "Teaches" are roles. There are also teachers who teach only tutorials but not classes. To include them into the Teacher class, consider using

```
Teacher ≡ Person ∧ ∃ HasStudent.Student ∧ (∃ Teaches.Lecture ∨ ∃
    Teaches.Tutorial)
```

With terminological knowledge, ALC can enforce restrictions on concepts and rules flexibly. Using assertional knowledge, on the other hand, allows defining instances and subclasses as follows.

```
Teacher(Peter)
Lecture(Calculus)
Teaches(Peter, Calculus)
```

All the above ALC can be translated into OWL then implemented in the semantic web. More details are given in later sections.

11.3.3 Logical Expression

A logic is defined by

$$L := (S, \models)$$

where S is the set that contains all the statements of our interests, and $\models$ the entailment relation

$$\models \; = \; \models^1 \bigcup \models^2$$

where

$$\models^1 = \{(\Phi, \phi) | \Phi \subseteq S, \phi \in S, \Phi \to \phi\} \tag{11.1}$$

$$\models^2 = \left\{(\Phi, \Psi) \middle| \Phi, \Psi \subseteq S, \forall \psi \in \Psi, \Phi \models^1 \psi \right\} \tag{11.2}$$

In (11.1), ϕ is known as the logical consequence of Φ. If two logical assertions satisfy $\Phi \models \Psi$ and $\Psi \models \Phi$, then they are logically equivalent $\Phi \equiv \Psi$.

A logic statement is meaningful only if its interpretation I and formula F are clearly articulated. I and F are two very important terms in logic expression. **Logic interpretation** is a formal construct that defines the meaning of a symbol in a formal language. For example, in the ontology of people, an interpretation can map an instance symbol, such as "Alice", to an actual person in the real world, and class symbol "Person", to the concept of a human being, etc. **Logic formula**, on the other hand, is a statement formulated by string of symbols from a formal language. It is an assertion trying to represent some fact. For example, a formula can be "Alice hasSibling Bob".

The true or false of the formula depend not only by the formula itself, but also by the interpretation. In the earlier example "Alice hasSibling Bob", if "Alice" and "Bob" are indeed mapped to two siblings, and the relation "hasSibling" is as it literally represents, then it is true. Here is another example. Consider formula "10 is greater than 5". Intuitively, this is a universal truth. But from the interpretation and formula perspective, this also depends on the interpretation. If "10" and "5" are interpreted as numerical quantities and "is greater than" as the standard numerical greater-than relation, then it is true. However, if "10" and "5" are interpreted as amounts of debt and "is greater than" as "richer" (with less debt being richer), then it would be false under this interpretation.

The interpretations that make the formula true form the model of the formula denoted by $I(F)$ or $I \models F$, which read as "I models F" or "I satisfies F".

With the above, we can express FOL using logic expression. Here is an example

$$\forall X : \text{Child}(X) \rightarrow \text{lovesIcecream}(X)$$

which says "for all elements denoted by X, if X interpreted as Child holds true, then X interpreted as lovesIcrcream must also be true".

Multiple formulas can be grouped into a theory (T), which can be used interchangeably with F. A theory can be treated as a knowledge base.

11.3.4 Logical Equivalence

Logical equivalence has already been introduced earlier. Recall (11.2) where we say if $\Phi \models \Psi$ and $\Psi \models \Phi$, Φ and Ψ are logically equivalent denoted by $\Phi \equiv \Psi$. Here Φ and Ψ are sets of statements.

Consider a simplified case where Φ and Ψ each contains only one statement. Let the formula of the statements of Φ and Ψ be F and G respectively. The notations applied to Φ and Ψ applies to F and G similarly. For example, $F \models G$ means that under the same interpretation if F is true, G must also be true. If $F \models G$ and $G \models F$, F and G are logically equivalent denoted by $F \equiv G$.

There is a huge table of logical equivalence formulas. Commonly seen equivalence is given in Table 11.1. The proof of the formulas is not given here. The interpretation of conjunction $\wedge$, disjunction $\vee$ and negation $\neg$ are "AND", "OR", "NOT" respectively.

Canonical forms have been defined for logical expressions. There are at least 6 different canonical forms for a logical expression, namely conjunctive normal form, disjunctive normal form, prenex normal form, skolem normal form, negation normal form and clausal normal form. Canonical forms are not

TABLE 11.1
Numerical calculations.

Statement	Equivalent	Comment
$\neg\neg p$	p	Double negation law
$(p \wedge q)$	$(q \wedge p)$	Commutative law
$(p \vee q)$	$(q \vee p)$	Commutative law
$(p \wedge (q \wedge r))$	$((p \wedge q) \wedge r)$	Association law
$(p \vee (q \vee r))$	$((p \vee q) \vee r)$	Association law
$(p \wedge (q \vee r))$	$((p \wedge q) \vee (p \wedge r))$	Distributive law
$(p \vee (q \wedge r))$	$((p \vee q) \wedge (p \vee r))$	Distributive law
$(p \rightarrow q)$	$(\neg p \vee q)$	Conditional statements
$(p \rightarrow q)$	$(\neg q \rightarrow \neg p)$	Conditional statement
$(p \leftrightarrow q)$	$((p \rightarrow q) \wedge (q \rightarrow p))$	Conditional statement
$(\neg(p \wedge q))$	$(\neg p \vee \neg q)$	De Morgan's law
$(\neg(p \vee q))$	$(\neg p \wedge \neg q)$	De Morgan's law

necessarily the easiest and most intuitive forms of an expression (in fact it is quite the opposite), but somethings they are simple for further analysis and process, such as finding contradictions.

Notice that when deriving certain canonical forms from the original logical expression, information may get lost and they are not necessarily always equivalent, but they usually are.

11.3.5 Logical Reasoning

Logical reasoning is about proving true or false of a formula F given a theory T. The proof may not be unique, and sometimes it is difficult to find one.

Before talking about how knowledge is retrieved from the knowledge base using DL inference and reasoning, we need to discuss what to return if knowledge is not found. When open world assumption (OWA) is made, the knowledge base considers itself as underdevelopment, and it is open-minded to unknowns. While when closed world assumption (CWA) is made, the knowledge base considers itself as developed, and unknown means nonexistence.

Consider an example where it is asked "are Alice's children all males?", and in the database there are two children of Alice, both of which are male. Under OWA, the inference would return "maybe", as it does now know whether there are more children of Alice. While in CWA, the inference would return "yes", as it checks both children to be male, and it assumes they would be all the children Alice has. However, when Alice does have female child registered, both OWA and CWA inferences would return "no", because the female child contradicts the assertion regardless of the completeness of the database.

The law of contradiction is widely used in logic reasoning. For example, to prove $T \models F$, just find contradictions in $\{\neg F, T\}$. A commonly used way of looking for contradictions in $\{\neg F, T\}$ is to resolve the formula into something more structured, for example to one of its canonical forms. Commonly used canonical forms for contradiction check are clausal normal form and disjunctive normal form.

12

Resource Description Framework

CONTENTS

12.1 Uniform Resource Identifier 155
12.2 Resource Description Framework (RDF) 157
  12.2.1 Triple Representation 157
  12.2.2 Multi-valued Relation and Blank Node 158
  12.2.3 Lists 160
  12.2.4 Reification 160
  12.2.5 Converting RDB to RDF 162
12.3 RDF Schema 162
  12.3.1 RDF Versus RDF/RDFS 163
  12.3.2 RDFS Expanded Class and Properties 165
  12.3.3 Semantics inside RDF/RDFS 165
12.4 SPARQL Protocol and RDF Query Language (SPARQL) 166
  12.4.1 SPARQL for Basic Query 166
  12.4.2 SPARQL for Advanced Operations 169
  12.4.3 Default Graph and Named Graph 171
  12.4.4 SPARQL Programming Returns 171
  12.4.5 Underlying Data Structure of Triplestores 172

Semantic web uses URI to identify an existing interpretation, RDF/RDFS and OWL to store semantics, and SPARQL to query and manipulate the database. RDF and RDFS are discussed in this chapter.

12.1 Uniform Resource Identifier

Human uses symbols to represent objects in reality. The concept associated with a symbol is used to help define and interpret objects. An example of interpreting “Apple” is given in Fig. 12.1.

URI points to the interpretation of the symbols used in the semantic web. In practice, the resource is either the definition of the class/object/property, or the higher-level ontology (maybe also in the format of semantic web). For example, consider building a semantic web to describe an apple farm. Consider

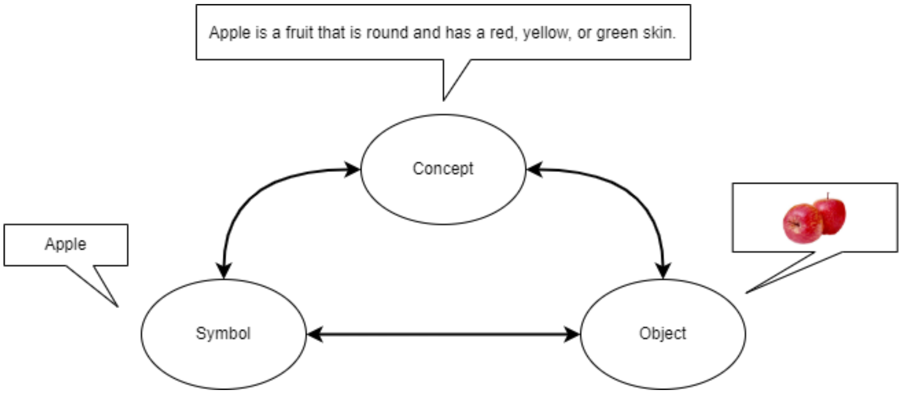

Apple is a fruit that is round and has a red, yellow, or green skin.
Concept
Apple
Symbol
Object

FIGURE 12.1
Semiotic triangle of Apple.

using `dbpedia.org/page/Apple` from DBpedia as the URI of interpretation of apple, which already gives a clear and rich definition of apple such as abstract, Wikipedia link, emoji, sugar level, etc. This saves the time of building the interpretation of apple from scratch.

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. If it looks similar with the Uniform Resource Locator (URL), that is because URL is considered as a subclass of URI.

```
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` specifies the user information, host, and port, separated by `@` and `:`, 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` 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 `?`.

- `fragment` identifies 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 at

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

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

12.2 Resource Description Framework (RDF)

RDF is the backbone data model of the semantic web. It stores data in the form of triples. RDF alone is not powerful enough to record DL. In practice, RDF usually works with RDFS which expand its vocabulary to include class, subclass, etc., and OWL which further expand its vocabulary and introduce DL features, to together form a comprehensive semantic web. RDF is the fundamental of RDFS and OWL.

Notice that RDF is not a syntax by itself. A markup language is needed to host the RDF framework. Commonly used markup languages are listed below. They can be translated from one to the other.

- 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.

Unless otherwise mentioned, Turtle is used in this notebook.

12.2.1 Triple Representation

RDF uses "subject-predicate(property)-object" triple to represent knowledge. A triple is corresponding with an edge in a directed graph, which is often used to visualize RDF.

For example, consider "Einstein was born in Ulm". In RDF, "Einstein" is the subject, and "Ulm" the object. The predicate is "has birthPlace", where "birthPlace" is a property assigned to "Einstein". The graph representation is given by Fig. 12.2. The RDF in Turtle is given as follows.

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

dbr:Albert_Einstein dbo:birthPlace dbr:Ulm .
```

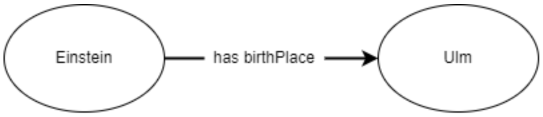


FIGURE 12.2
Graph representation of triple for knowledge “Einstein was born in Ulm”.

where `Albert_Einstein` and `Ulm` are defined in `dbpedia.org/resource/` as name and place, and `birthPlace` in `dbpedia.org/ontology/` as a property. To breakdown the Turtle in more details:

- `@prefix` 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:Ulm` is 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.

12.2.2 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. 12.3, 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. 12.4. In this example, blank node plays as an unnamed intermediate node that hosts a nested structure.

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

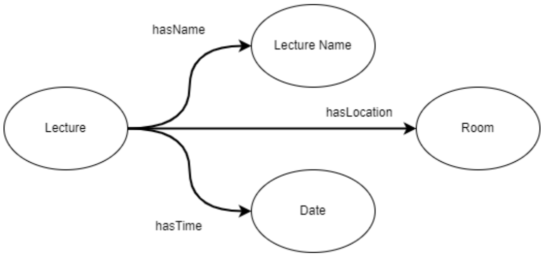

hasName
Lecture Name
Lecture
hasLocation
Room
Date
hasTime

FIGURE 12.3
An example that demonstrate when and where a lecture takes place.

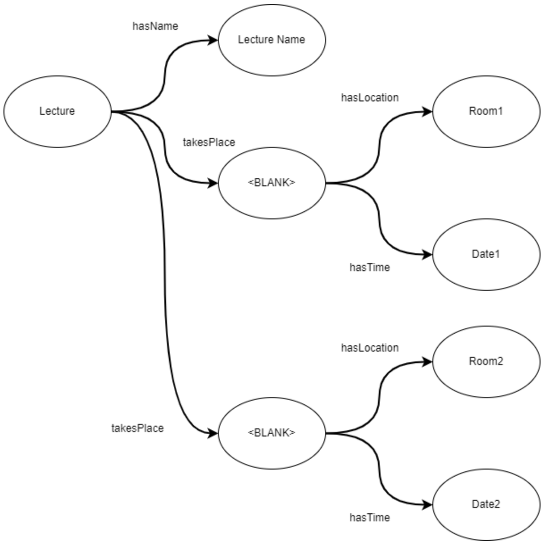

hasName
Lecture Name
Lecture
hasLocation
Room1
takesPlace
<BLANK>
Date1
hasTime
hasLocation
Room2
takesPlace
<BLANK>
Date2
hasTime

FIGURE 12.4
An example that demonstrate a lecture taking place at multiple locations and time slots using multi-valued relations and blank nodes.

```
<subject1> <predicate1> [
       <predicate2> <object2>;
       <predicate3> <object3>
] .
```

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

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

where `_:<blank-node-name>` is used to declare a blank node and assign it a name. For normal nodes URI takes the position of `_`. Bland nodes do not possess a global identifier, and `_:<blank-node-name>` is only a local identifier used in a single RDF script.

12.2.3 Lists

Lists help to make the code clean and readable. There are two types of lists, containers (open list, extendable, unordered or partially ordered) and collections (closed list, well-defined, ordered and fixed). Container is helpful to handle the situation given in Fig. 12.5. 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).

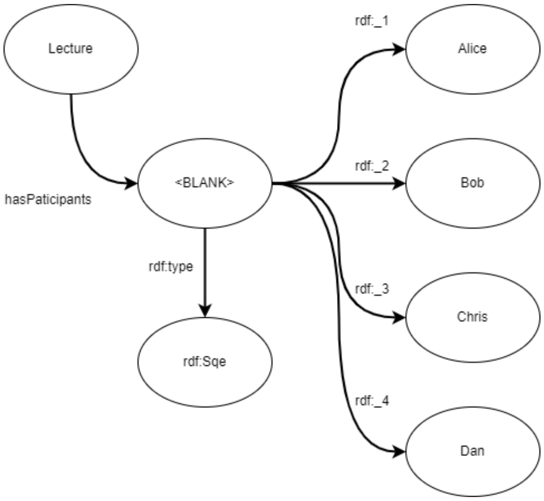


FIGURE 12.5
An example of a container.

The collection, on the other hand, defines a closed list as shown by Fig. 12.6. 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> <predicate> (<object1> <object2> <object3>) .
```

12.2.4 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”.

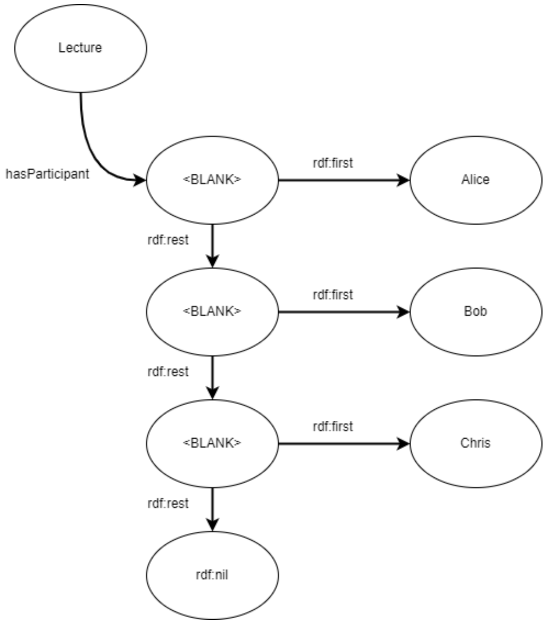

Lecture
hasParticipant
<BLANK>
rdf:first
Alice
rdf:rest
<BLANK>
rdf:first
Bob
rdf:rest
<BLANK>
rdf:first
Chris
rdf:rest
rdf:nil

FIGURE 12.6
An example of a collection.

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. 12.7.

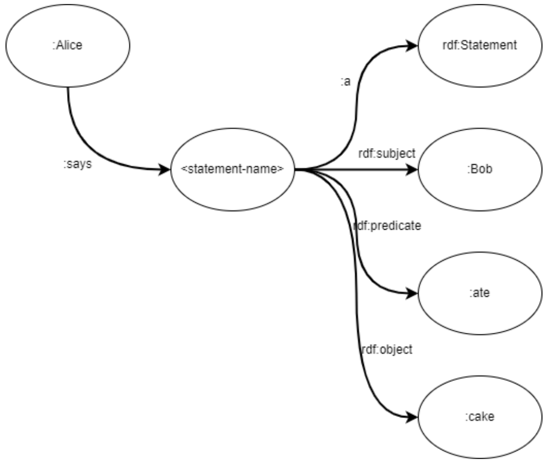

:Alice
:says
<statement-name>
:a
rdf:Statement
rdf:subject
:Bob
rdf:predicate
:ate
rdf:object
:cake

FIGURE 12.7
An example of RDF reification.

12.2.5 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, books, authors and publishers, respectively. Each table contains a few dozens of entries. The RDB can be converted to RDF as shown in Fig. 12.8. It can be realized very easily, simply by defining everything as nodes and stack triples for all relationships.

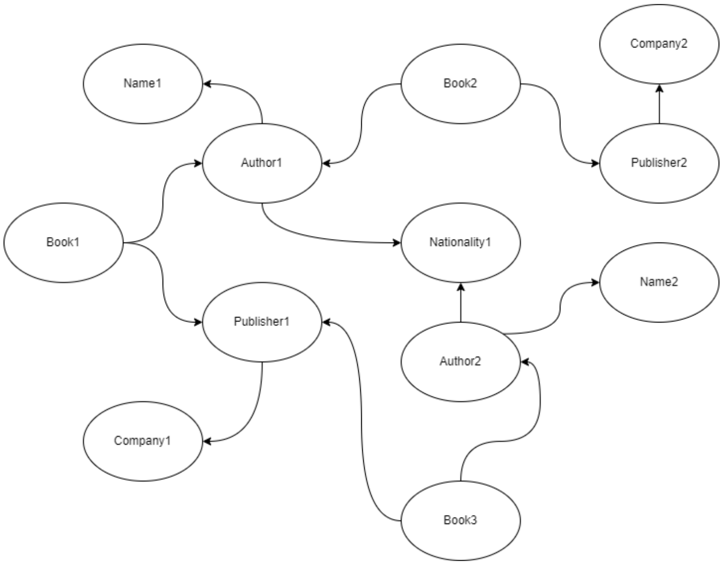


FIGURE 12.8
Semantic web of a few books, and their authors and publishers.

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 [8] for more details.

In summary, RDF defines the objects (nodes) and also the properties among objects (edges). This is essentially how RDF differs from a conventional key-value based NoSQL. RDF can be easily scaled up.

The underlying mechanisms 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. Though details are not given, a brief review of the mechanisms is given in later part of the notebook.

12.3 RDF Schema

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.

12.3.1 RDF Versus RDF/RDFS

RDF is flexible. It is so flexible that sometimes it becomes difficult to maintain consistency, let alone performing sophisticated reasoning from it. RDFS enforces 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/RDFS modeling more consistent and semantic than using RDF alone.

In the deeper insights, RDFS helps to add "ontology" to the RDF model by introducing the schema. It essentially integrate common understanding and domain knowledge to the information. For example, by creating a property "`:hasSpouse`" whose domain and range person, it demonstrates the common knowledge that a person can be married to another person.

The following example compares RDF and RDF/RDFS implementations on the same context. Consider statement "banana is yellow, apple is red, and orange is orange color". In a RDF implementation, this would look like the green-colored elements in Fig. 12.9. It is a disconnected graph. The semantic web failed to bridge the triples together and realize that all of them are discussing colors of fruits.

In a RDF/RDFS implementation, class/subclass and property are enforced in the RDF model, and each property must be associated with clearly defined domain and range. All classes eventually root to `rdfs:Resource`. This is shown by the green + red color in Fig. 12.9. 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 .
```

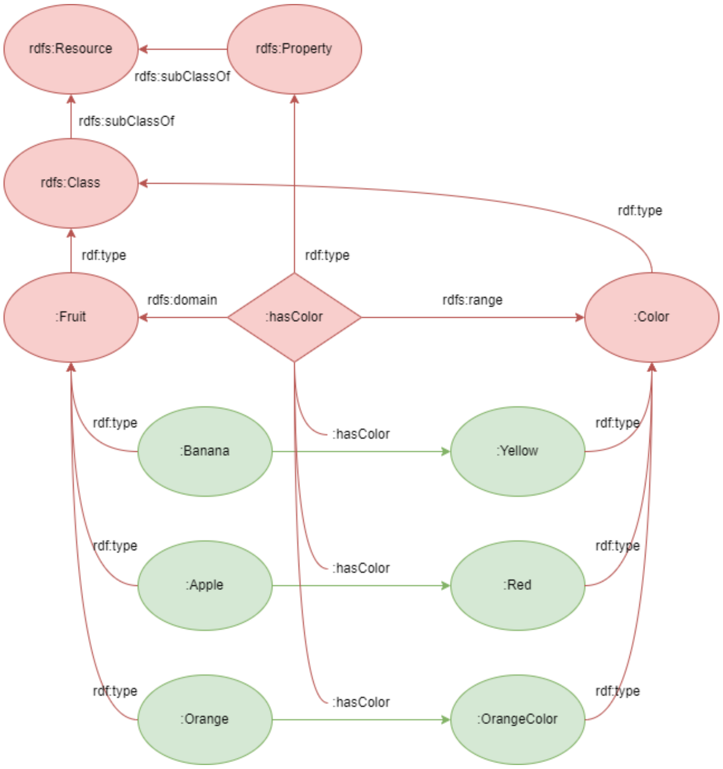

rdfs:Resource
rdfs:Property
rdfs:subClassOf
rdfs:subClassOf
rdfs:Class
rdf:type
rdf:type
rdf:type
:Fruit
rdfs:domain
:hasColor
rdfs:range
:Color
rdf:type
:Banana
:hasColor
:Yellow
rdf:type
rdf:type
:Apple
:hasColor
:Red
rdf:type
rdf:type
:Orange
:hasColor
:OrangeColor
rdf:type

FIGURE 12.9
Semantic web of fruits and their colors, with RDF implementation in green RDF/RDFS in red.

```
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 .
```

Notice that to build the RDF model shown by the green-colored elements in Fig. 12.9 (without RDFS), only the last 3 lines would be required. It can be seen from this example that RDF/RDFS uses more "complicated" structures 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.

12.3.2 RDFS Expanded Class and Properties

RDFS expands the vocabulary of RDF by introducing classes and properties hierarchy as well as domain and range for a property. As a brief summary, the following list shows some of the introduced concepts by RDFS.

- `rdfs:Resource`
- `rdfs:Class`
- `rdf:Property` (Notice that "Property" is already defined in RDF framework; RDFS enhanced its capability by introducing new mechanisms.)
- `rdfs:subClassOf`
- `rdfs:subPropertyOf`
- `rdfs:domain`
- `rdfs:range`

RDFS introduces additional powerful properties for a class. They help to make the RDF model more human-readable. The following is a short list of some of the new properties.

- `rdfs:seeAlso` points to a reference where a detailed explanation of the node can be found.
- `rdfs:isDefinedBy` defines the relation of a source 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/*.

12.3.3 Semantics inside RDF/RDFS

The semantics in RDF/RDFS are given by both the triples in the RDF model as well as the class and property hierarchies introduced by RDFS. Here are some examples of different types of hierarchies, and the semantics behind them.

- Class inheritance. If apple is a subclass of fruit, and fruit a subclass of plant, then apple must also be a subclass of a plant.
- Property domain and range. Let “hasColor” predicate be associated with domain “fruit” and range “color”. If an object “hasColor”, then the object must be a fruit, and corresponding subject in the triple must be a color.
- Property inheritance. Consider two predicates, “isMotherOf” and “isParentOf”. Both predicates have the same domain and range of “person”. Predicate “isMotherOf” is a sub property of “isParentOf”. In this case “A is the mother of B” leads to “A is the parent of B”.

The semantics allows some extent of reasoning, allowing “hidden information” to be derived.

12.4 SPARQL Protocol and RDF Query Language (SPARQL)

SPARQL is a widely used query and manipulation language for semantic web. It is SQL-like in the syntax, but its underlying mechanisms differ largely from how a relational database management system run SQL.

In the lately released SPARQL 1.1 standard, more operations such as advanced query and interfering are included, making it more powerful and capable than what SPARQL 1.0 standard described. SPARQL 1.1 is already supported by many triplestores.

Notice that SPARQL is not only a language, but also a protocol layer. The input and return have specific formats which are also defined by the SPARQL standard. Introducing SPARQL from a protocol perspective is not the focus of this chapter, hence it is not covered in details.

More details of SPARQL 1.1 can be found at *w3.org/TR/sparql11-query/*.

SPARQL is not the only language for semantic web query and manipulation. It is good at general tasks. However, when comes to specific tasks such as expressive querying, data validation and advanced reasoning, there might be better choices.

12.4.1 SPARQL for Basic Query

SPARQL offers flexible ways of querying data. There are different commands to trigger a query, each fulfilling a different purpose. For example,

- SELECT returns a tabular just like SQL.
- CONSTRUCT returns a new RDF graph based on the query result.
- ASK returns true and false of whether a query has a solution.
- DESCRIBE returns the schematic of a resource; this is useful when the structure of RDF data in the data source is unclear.

The basic syntax of SPARQL looks similar with SQL as shown below.

```
SELECT [?s ?p ?o]
[FROM rdfSource]
WHERE {
       [?s/s] [?p/p] [?o/o]
}
[ORDER BY (?s/?p/?o)]
[LIMIT limitNum]
[PFFSET offsetNum]
```

where `?variableName` denotes a variable, and without `?`, a constant. Recall the fruit example used in Section 12.3.1. The following is an example of querying that semantic web.

```
PREFIX ex: <http://www.example.com/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

ASK WHERE {
       ?fruit rdf:type ex:Fruit .
       ?fruit ex:hasColor ex:Yellow .
} # return yes

SELECT ?fruit
WHERE {
       ?fruit ex:hasColor ex:Yellow .
} # return a table with one element, ex:Banana

CONSTRUCT {?fruit ex:hasColor ?color}
WHERE {
       ?fruit rdf:type ex:Fruit .
       ?color rdf:type ex:Color .
       ?fruit ex:hasColor ?color .
       FILTER (?color = ex:Yellow || ?color = ex:Red)
} # returns 2 triples, banana has color yellow and apple has color red

DESCRIBE ?fruit
WHERE {
       ?fruit ex:hasColor ex:Yellow .
} # return triples related to ex:Banana
```

The “`FILTER`” keyword can be used to quantitatively filter a numerical or string-like variable. An example is given below. It is worth mentioning that only triples clauses need to end with a period “.”. The filter condition lead by `FILTER ()` does not require the period.

```
PREFIX ex: <http://example.org/>

SELECT ?person
WHERE {
       ?person ex:age ?age .
       FILTER (?age >= 30 && ?age <= 40)
}
```

Commonly used keywords and operators in FILTER clause include `&&` (and), `||` (or), `!` (not), as well as string functions `STR`, `LANG`, `CONTAINS`, `STRSTARTS`, `STRENDS`, `STRLEN`, `SUBSTR`, `REGEX` (regular expression matching), etc., and numeric functions `+`, `-`, `*`, `/`, `>`, `>=`, `<`, `<=`, `ABS`, `ROUND`, `CEIL`, `FLOOR`, etc., and comparison operators `=`, `!=`.

There are also other clauses such as `OPTIONAL` and `UNION`. When `OPTIONAL` clause is applied on a property as part of the `WHERE` clause, it allows both entities with the correct property values as well as entities without the property to pass the filter. An example is given below. RDF:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:alice
a foaf:Person ;
foaf:name "Alice" ;
foaf:mbox <mailto:alice@example.com> .

:bob
a foaf:Person ;
foaf:name "Bob" .
```

SPARQL:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?email
WHERE {
       ?person a foaf:Person ;
       foaf:name ?name .
       OPTIONAL { ?person foaf:mbox ?email . }
}
```

The above returns both names and Alice’s email. Though Bob does not have a property of `foaf:mbox`, his name is still included in the query result.

The `UNION` clause allows combining the returns of two queries together, given that the returns follow the same structure. An example is given below.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
       {
              ?person a foaf:Person .
              ?person foaf:name ?name .
       }
       UNION
       {
              ?organization a foaf:Organization .
              ?organization foaf:name ?name .
       }
}
```

which would return both people and company names in one go.

12.4.2 SPARQL for Advanced Operations

SPARQL 1.0 provides basic query functions. In the lately released SPARQL 1.1 standard, advanced query and triple manipulation are supported. They are introduced in this section.

Advanced Query

In the SELECT clause, it allows simple calculations on top of the returned result, and name it as a new column. For example,

```
SELECT ?x (?y*1.1 AS ?z)
WHERE {
       ?x ex:hasProperty ?y .
}
```

returns `?y*1.1` instead of `?y`, and furthermore rename it as `?z`.

SPARQL 1.1 enables aggregate functions. For example,

```
PREFIX ex: <http://www.example.com/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (Count(?fruit) AS ?numOfFruit)
WHERE {
       ?fruit rdf:type ex:Fruit .
}

SELECT (Count(DISTINCT ?fruit) AS ?numOfFruit)
WHERE {
       ?fruit rdf:type ex:Fruit .
}
```

counts the total number of fruits in the database. Notice that when using aggregate functions, new variable name (in this example, `?numOfFruit`) must be assigned. Otherwise, there will be an syntax error.

SPARQL 1.1 supports nested query. An example is given below.

```
PREFIX : <http://example.org/schema#>

SELECT ?book ?publisher
WHERE {
       # Find the publisher of "BookA"
       {
              SELECT ?publisher
              WHERE {
                     :BookA :publishedBy ?publisher .
              }
       }
       # Find other books published by the same publisher
       ?book :publishedBy ?publisher .
       FILTER(?book != :BookA)
}
```

Triple Manipulation

SPARQL provides operations to insert, edit, and delete elements in a semantic web as follows.

- INSERT inserts triples into a graph.
- DELETE deletes triples from a graph.

An example of creating a semantic web that contains fruits and their colors is given below.

```
PREFIX ex: <http://www.example.com/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

INSERT DATA {
       # Define resources
       ex:Banana rdf:type ex:Fruit .
       ex:Apple rdf:type ex:Fruit .
       ex:Pear rdf:type ex:Fruit .
       ex:Yellow rdf:type ex:Color .
       ex:Red rdf:type ex:Color .
       ex:Green rdf:type ex:Color .

       # Define properties
       ex:hasColor rdf:type rdf:Property ;
       rdfs:domain ex:Fruit ;
       rdfs:range ex:Color .

       # Link resources with properties
       ex:Banana ex:hasColor ex:Yellow .
       ex:Apple ex:hasColor ex:Red .
       ex:Pear ex:hasColor ex:Green .
}
```

12.4.3 Default Graph and Named Graph

There can be multiple semantic webs in a triplestore, each semantic web corresponding with a graph name. When querying, data can be retrieved across multiple semantic webs. If graph name is not specified in the SPARQL command in a insert command, the default graph of the database is used.

An example of inserting and querying a specific graph is given below.

```
INSERT DATA {
       GRAPH <http://example.org/mygraph> {
              <http://example.org/book1> <http://purl.org/dc/elements
                  /1.1/title> "A new book" .
       }
       GRAPH <http://example.org/myothergraph> {
              <http://example.org/book2> <http://purl.org/dc/elements
                  /1.1/title> "Another book" .
       }
}

SELECT ?book ?title WHERE {
       GRAPH <http://example.org/mygraph> {
              ?book <http://purl.org/dc/elements/1.1/title> ?title .
       }
}
```

The semantic web name (graph name) is given in the form of a URI, and can be used as a reference resources in other semantic webs.

12.4.4 SPARQL Programming Returns

SPARQL is an HTML based protocol. The returned result of a SPARQL query in the protocol layer can be specified in the HTML header. There are several return types defined in the standard. When SELECT or ASK are used, the return types can be:

- XML
- JSON
- TSV (table-separated values, similar to CSV)

When CONSTRUCT or DESCRIBE are used, the return is an RDF that can be in

- RDF/XML

- Turtle
- N-Triples
- JSON-LD

and a few more. The decoding of the return can be done using variety of tools and packages, and they are not discussed here.

12.4.5 Underlying Data Structure of Triplestores

RDF is a data model or framework that stores data in the form of triples. The benefit of doing so is to enable DL reasoning and hence build semantics into the data.

When comes to the underlying data structure that actually processes the data in the computer memory and runs the RDF functions (such as SPARQL query based on graph pattern matching) efficiently, each triplestore may have its preferences. Some triplestores may develop dedicated graph database engines to store data and process queries. Others may utilize existing SQL or NoSQL database structures and engines and build an RDF interface on top of them.

The mechanisms of these different engines and data structures are not discussed in details here. Only a brief introduction is given as follows.

Table-Based Data Structure

An intuitive way of storing triples is to use a structured table that contains three columns: subject, predicate (property), and object. To save some space, all subjects, predicates and objects can be encoded into integers, where there are additional translation tables that can be used to decode the data. RDF should support multiple graphs. To enable multiple graphs, an additional column indicating graph ID needs to be added to all the tables mentioned above.

In this case, a SPARQL query needs to be converted into SQL (introducing self-join) then performed on the table. One of the major problems of this data structure is that it is too computationally expensive when the query is complicated.

To reduce self-join in the query, table size needs to be made small. The big structured tables need to be split into multiple small tables via aggregation. Due to the nature of triples, there will be many "NULL" in the tables. Implementing multi-value property, etc., can also be complicated. Overall, it is just too difficult to design the optimal RDB schematics, let along creating a database of that schematics.

To systematically and automatically split big tables into small tables without warring too much about the schematics, one idea is to destruct the table to the very ground level. Each property becomes a small table that contains its subjects and objects. The problem of this approach, however, is that it becomes time consuming to loop over all the tables when the RDF model is large. The performance of the architecture will be bad, and things such as inserting will become expensive.

NoSQL-Based Data Structure

Structured table and RDB approaches are rarely used in commercialized triplestores due to the expensive computational cost. The most widely used approaches nowadays are NoSQL based.

Two commonly seen NoSQL data structures are vertically partitioned tables and hexastores. These structures help to either reduce or speed up joins, making the computation more efficient than the RDB based methods.

Dedicated Graph Database

Some triplestores use dedicated graph database engines designed to natively store and process graph-structured data. These engines are optimized for the kinds of operations that are common in RDF and SPARQL, such as complex graph traversals and pattern matching.

13

Web Ontology Language

CONTENTS

13.1 RDF/RDFS Limitations 175
13.2 OWL Vision 177
13.3 OWL Basic Syntax 178
13.4 OWL Advanced Syntax 179
13.5 Semantic Web with Rules 183

By adding schema to RDF, RDFS already boosts the capability and adds ontology to the RDF model. As demonstrated in the previous chapter, RDF/RDFS is able to process simple logic reasoning. However, RDF/RDFS alone cannot handle DL. OWL is, therefore, proposed to enable DL in the semantic web.

Notice that it is OWL the abbreviation of web ontology language, not WOL, for historical contexts reasons.

13.1 RDF/RDFS Limitations

Several examples are given to demonstrate the limitations of RDF/RDFS.

RDFS brings schema to the RDF model by introducing class and property hierarchy. For example, in the schematic design of the model we can create "animal eats food", where "eat" is a property with domain "animal" and range "food". We can then add instances to animal and food classes. This is demonstrated in Fig. 13.1. The SPARQL to create such an RDF model is given below.

```
PREFIX ex: <http://www.example.com/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

INSERT DATA {
      # Define resources
      ex:Human rdf:type ex:Animal .
      ex:Cow rdf:type ex:Animal .
```

```
       ex:Vegetable rdfs:subClassOf ex:Food .
       ex:Meat rdfs:subClassOf ex:Food .
       ex:Cabbage rdf:type ex:Vegetable .
       ex:Ribeye rdf:type ex:Meat .

       # Define properties
       ex:eat rdf:type rdf:Property ;
       rdfs:domain ex:Animal ;
       rdfs:range ex:Food .

       # Define triples
       ex:Human ex:eat ex:Meat .
       ex:Human ex:eat ex:Cabbage .
       ex:Cow ex:eat ex:Cabbage .
}
```

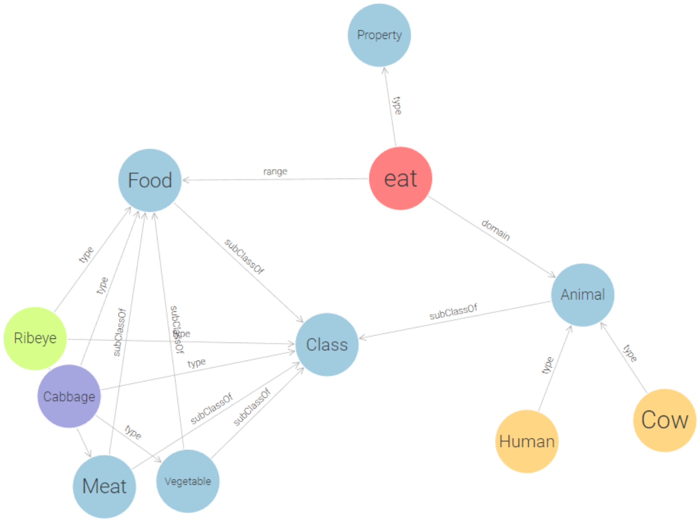

Property
type
Food
range
eat
domain
subClassOf
Animal
type
subClassOf
type
subClassOf
subClassOf
type
Ribeye
type
Class
subClassOf
type
Cabbage
subClassOf
subClassOf
type
type
type
Cow
Human
Meat
Vegetable

FIGURE 13.1
An RDF model that demonstrates “animal eats food”.

From Fig. 13.1, under OWA when querying what animals eat what food, the following would be returned:

- Human eats rib-eye.
- Human eats cabbage.
- Cow eats rib-eye (this is not what we expect).
- Cow eats cabbage.

Since it is not explicitly denied that cow cannot eat meat, under OWA, semantic web thinks that there is the possibility of cows eating meat. RDF/RDFS cannot exclude meat from the menu of a cow, or to say it in general, RDF/RDFS cannot add restrictions to a property.

A walk around is to define two properties, "eatMeat" and "eatVegetable", each with their associated ranges. Let human have both properties while cow have only "eatVegetable" property. However, this complicates the connection of symbols and destroys the schema of the model, essentially making null the effort of RDFS.

Similarly, consider an example where "hasVisualAcuity" is used to describe human's clarity of vision. A person usually have two visual acuity values for the two eyes. In RDF, that means a person should have two and only two "hasVisualAcuity" property. This cannot be enforced by RDF/RDFS.

Consider another example where "man is a subclass of human" and "woman is a subclass of human". With only RDF/RDFS, however, there is no way to add restrictions that says "a person cannot be man and woman at the same time", i.e., to disjoint man class and woman class.

RDF/RDFS does not support class combinations. For example, for existing classes "Car", "Motorcycle", "Bicycle", "Ship", "Plane", RDF/RDFS cannot create a new class "AllTranportationTool" to automatically include every instance from the above classes. In general, with RDF/RDFS alone, it is not possible to define a class which is a union/intersect/complement of other classes.

In semantic web stack Fig. 11.1, one layer above RDF/RDFS, OWL is proposed to solve the above problems.

13.2 OWL Vision

OWL essentially introduces DL inference to the semantic web, making it more expressive and capable of complicated reasoning. With that in mind, OWL provides additional features to enforce and enhance the schema of an RDF model, thus addressing the problems mentioned in Section 13.1. It allows adding relations and property constraints to the RDF model. The following summarizes the main features introduced by OWL.

- Allow the disjoint of sub classes (an instant cannot be man and woman at the same time).
- Allow enforcing the number of attributes of each property type (one can have only one social security number, but can have many mail addresses)
- Allow enforcing the range of the values an attribute can take (the age of a person must be a positive integer).

- Offer more expressive class definitions including union, intersection, complement, etc.
- Enlarge the vocabulary sets on top of RDF/RDFS.

As of this writing, OWL 2 is the latest version of OWL recommended by W3C. It makes the following assumptions:

- OWA. In this context, the knowledge base is considered open, and absence of information must not be valued as negative.
- An instance can have non unique names, i.e., multiple syntax can be mapped to the same instance via interpretation.

In practice, OWL comes in different flavors, including OWL Lite, OWL DL and OWL Full, just to name a few. OWL 2 supports different choices of syntax, including RDF-syntax, XML-syntax, functional-syntax, Manchester-syntax, and Turtle.

13.3 OWL Basic Syntax

Unless otherwise mentioned, Turtle is used when introducing OWL syntax. Classes, individuals, and properties are already introduced in RDF/RDFS. OWL also defines these concepts, each with enriched features. It is worth mentioning that OWL defines two special classes, `owl:Thing` and `owl:Nothing`, corresponding with the top class (universal set) and bottom class (empty set) in DL, respectively.

Class and Subclass

Defining a class from the root `owl:Class` using OWL is given below.

```
:Fruit a owl:Class ;
```

where `a` can be replaced by `rdf:type`. To define an individual via class membership, simply use

```
:Apple a :Fruit .
```

which is equivalent of saying `Fruit(Apple)` in DL. It is also possible to define an individual without a named class as follows, which indicates that the named class will be added in a later stage. In this case, `owl:NamedIndividual` is used.

```
:ANewThing a owl:NamedIndividual .
```

Subclass can be defined similarly as follows.

```
:Fruit a owl:Class ;
       rdfs:subClassOf :Food .
:Meat a owl:Class ;
       rdfs:subClassOf :Food .
:Food a owl:Class .
```

Notice that it was impossible to enforce class disjoint using RDF/RDFS alone. This is made possible in OWL as follows.

```
:Fruit owl:disjointWith :Meat .
```

Or alternatively,

```
[] a owl:AllDisjointClasses ;
       owl:members
       (      :Fruit
              :Meat
              :SoftDrink
              :Beer ) .
```

which is a shortcut when disjoint is claimed on multiple classes. Similarly, class equivalence can be defined using `owl:equivalentWith` keyword.

Property

There are two types of properties defined in OWL, namely the object property which links to another resource URL (another object), and datatype property which links to a literal. Examples are given below.

```
:hasColor a owl:ObjectProperty ;
       rdfs:domain :Fruit ;
       rdfs:range :Color .
:hasShelfLife a owl:DatatypeProperty ;
       rdfs:domain :Fruit ;
       rdfs:range xsd:integer .
```

where notice that XML schema definition (XSD) defines many data types and sub data types, including `xsd:integer`, `xsd:string`, `xsd:decimal`, `xsd:boolean`, `xsd:date` (a calendar date), `xsd:time` (time in a day), `xsd:dateTime`, `xsd:double` (64-bit floating number), `xsd:float` (32-bit floating number), `xsd:anyURI`, `xsd:language`, etc.

13.4 OWL Advanced Syntax

Equivalent and Different Individuals

In RDF/RDFS, different individuals are distinguished by their names, and each individual can have one and only one name (it is technically possible for an individual to have no name, but it is not often a good practice). In OWL, it is possible to either map multiple names to the same individual, or to emphasize that two names are pointing to different individuals (this is sometimes necessary due to the OWA). Examples are given below.

```
:Computer a owl:Class .
:HarryPc a :Computer .
:PotterPc a :Computer ;
       owl:sameAs :HarryPc .
:DumbledorePc a :Computer ;
       owl:differentFrom :HarryPc .
```

To state that individuals are different, alternatively use the following

```
[] a owl:AllDifferent ;
       owl:distinctMembers
       ( :HarryPc,
         :DumbledorePc,
         :HermionePc,
         :HagridPc    ) .
```

Closed Class

It is possible to define an enumerate class, whose instances are taken from individuals. An example is given below.

```
:Weekdays a owl:Class ;
       owl:oneOf
       (      :Monday
              :Tuesday
              :Wednesday
              :Thursday
              :Friday ) .
```

which indicates that any individuals to be defined under class “Weekdays” must be one of the 5 individuals “Monday”, “Tuesday”, etc. Do not confuse `owl:oneOf` with `owl:unionOf`, as the former defines a class as an enumerate of individuals, and the later defines a class as an enumerate of subclasses.

Class Constructors

Intersection, union and complement are the commonly seen class (set) constructors. They can be defined as follows.

```
:MyFavourateBook a owl:Class .
:HisFavourateBook a owl:Class .
:OurFavourateBook a Class ;
       owl:intersectionOf (:MyFavourateBook :HisFavourateBook) .
:Wine a owl:Class ;
       owl:equivalentClass [
              owl:unionOf (
                     :DryWine
```

```
                    :MediumWine
                    :SweetWine
             )
      ] .
:Plant a owl:Class ;
      rdfs:subClassOf [
             owl:complementOf :Animal
      ] .
```

Property Restrictions

It has been introduced earlier that the type of the property can be specified by using either `owl:ObjectProperty` (specify range as a class) or `owl:DatatypeProperty` (specify range as a datatype such as string, decimal, integer, etc). In the context of OWL, more restrictions can be enforced to properties, including the number of appearance of each property, and the values each property can take. Details are given below.

The syntax of adding different property restrictions may differ slightly. A general syntax is given by

```
:<Class> rdfs:subClassOf [
   rdf:type owl:Restriction ;
   owl:onProperty :<PropertyName> ;
   owl:<RestrictionType> <RestrictionDetails>
] .
```

where the blank node is used for property restrictions.

For example, to indicate that the value of a property must be taken from a class, use

```
:<Class> owl:restriction [
      owl:onProperty :<PropertyName> ;
             owl:allValuesFrom :<RestrictionClass>
] .
```

And to indicate that among all the property values of a property, at least one of them must take values from a class, replace `owl:allValuesFrom` with `owl:someValuesFrom`.

To indicate that a property must take specific value, use `owl:hasValue` at the restriction type, and put the value as `<RestrictionDetails>`, whether an individual or a datatype value.

To restrict the number of a property, use `owl:maxCardinality`, `owl:minCardinality` or `owl:cardinality`, and put the maximum, minimum or fixed number as `<RestrictionDetails>`.

It is possible to define a class from property restrictions, essentially saying "the class is defined as a collection of individuals that satisfy the following property restrictions". To do that, use `rdfs:subClassOf` with property restrictions as given in the example below.

```
:MyBook a owl:Class ;
       rdfs:subClassOf [
              a owl:Restriction ;
              owl:onProperty :hasOwner ;
                     owl:hasValue :Me
       ] .
```

where `:Me` is a predefined individual that maps to myself. To put it in words, a book is considered an instance of `:MyBook` if it has a `:hasOwner` property with the value `:Me`.

Property Hierarchy

Properties can have hierarchy like classes. Properties hierarchy can be realized using `rdfs:subPropertyOf` in RDF/RDFS framework. OWL further enriches that idea, by introducing new concepts such as `owl:inverseOf`. Examples below are used to demonstrate property hierarchy.

To define property hierarchy, use

```
:isOwnerOf a owl:ObjectProerty ;
       rdfs:subPropertyOf :isMasterOf ;
       rdfs:domain :Person ;
       rdfs:range :Book .
:isOwnedBy a owl:ObjectProperty ;
       owl:inverseOf :isOwnerOf ;
       rdfs:domain :Book ;
       rdfs:range :Person .
```

Notice that when defining a inverse property, it is not necessary, but still good practice, to point out the domain and range. This is for better readability and easier error checking.

OWL further defines the following features of a property.

- `owl:TransitiveProperty`: if `Property(a,b)` and `Property(b,c)`, then `Property(a,c)`. An example is “isAncestorOf”.
- `owl:SymmetricProperty`: if `Property(a,b)`, then `Property(b,a)`. An example is “isColleagueWith”; there is also `owl:AsymmetricProperty`, which indicates that if `Property(a,b)`, it is not possible that `Property(b,a)`.
- `owl:FunctionalProperty`: if `Property(a,b)` and `Property(a,c)`, then `b=c`. An example is `hasMother`.
- `owl:InverseFunctionalProperty`: if `Property(a,c)` and `Property(b,c)`, then `a=b`. Examples are `isMotherOf`, `hasId` (assuming ID is unique for each individual).

Restrictions `owl:disjointWith` and `owl:AllDisjointClasses` can be used to claim disjoint of classes, it is possible to declare disjunctive properties using `owl:propertyDisjointWith` and `owl:AllDisjointProperties` as follows. Disjunctive properties cannot take the same value. For example,

```
:hasParent a owl:ObjectProperty ;
       rdfs:domain :Person ;
       rdfs:range :Person .
[] a owl:AllDisjointProperties ;
       owl:members ( :hasParent :hasChild :hasSibling ) .
```

In OWL, top and bottom classes are defined, corresponding to the universal set and empty set, respectively. Similar ideas apply to properties. Object and datatype properties each has its top and bottom properties.

It is possible to declare a negative assertion as a property to state that a relation does not hold for two individuals. An example is given below.

```
[] a owl:negativePropertyAssertion ;
       owl:sourceIndividual :SherlockHolmes ;
       owl:assertionProperty :isMurderOf ;
       owl:targetIndividual :Watson .
```

which claims that it is not true that “SherlockHolmes” has the property “isMurderOf” with the range “Watson”. Notice that negative assertion is supported only at individual-to-individual level. It is currently not possible to claim that Sherlock Holmes is not a murder of any victims, i.e, `:Watson` cannot be replaced by a class such as `:Person`. In FOL, this would have been possible. But currently OWL does not adopt FOL for a good reason (FOL can be undecidable, and too computationally expensive).

General Role Inclusion

If we define “B is the father of A” and “C is the brother of B”, then naturally, “C is the uncle of A” can be derived. This role chain can be realized via OWL general role inclusion. However, this adds uncertainty to the reasoning, and may cause the model to be undecidable. This is one of the reasons why different tiers of OWL have been proposed. More features usually mean more computations and higher risks of undecidable results, and the user needs to decide which tier to use.

And do notice that due to the Gödel’s incompleteness theorems, there is no sophisticated enough system or knowledge base that can use finite input to derive all the knowledge in a complete (every statement can be proved true or false) and consistent (no contradiction) way.

Using role chain, we can define `isUncleOf` as follows.

```
:isUncleOf a owl:ObjectProperty ;
       owl:PropertyChainAxiom ( :hasFather :hasBrother ) .
```

It is not recommended ot use role chain in a semantic web.

13.5 Semantic Web with Rules

Rules (rule-based systems, also known as rule-based engines) are used to express logic beyond DL, i.e., beyond what RDF/RDFS and OWL can describe. From this perspective, one can think of rules-based semantic web as a further enhancement beyond RDF/RDFS and OWL.

The basic syntax of a rule is simply

```
IF A THEN B
```

where `A` and `B` are premises and conclusion respectively. There are many variations, including logical rules (FOL rules, for example), procedural rules, and logic programming rules.

A FOL rule often looks like the following:

$$A \rightarrow B$$

or

$$A_1 \wedge \ldots \wedge A_n \rightarrow B$$

They can be equivalently converted to

$$\neg A \vee B$$

and

$$\neg A_1 \vee \ldots \vee \neg A_n \vee B$$

respectively.

Notice that implementing FOL rules (by using ALC just like we did for DL) in general would cause undecidability. However, it is possible to implement a subset of FOL rules while keeping the semantic web decidable. There are syntaxes to do that such as Declarative Logic Programming Language (also known as Datalog). By implementing the (decidable subset of) FOL rules, the expressiveness of the semantic web can be boosted.

There are semantic web triplestores that support the implementation of rules. Rules add expressiveness and meantime complexity and additional computational load to the semantic web.

Semantic Web Rule Language (SWRL) is based on the combination of parts of OWL and Datalog. The motivation of SWRL is to implement Datalog rules to the OWL ontology. Rule Interchange Format (RIF) is a collection of languages, rules, datatypes and frameworks recommended by W3C that tries to standardize rule-based semantic web description.

14

Semantic Web Practice

CONTENTS

14.1 Ontological Engineering 185
14.2 Ontology Design 186
  14.2.1 General Tasks 186
  14.2.2 Ontology Design Basics 187
  14.2.3 Semantic Web Design for Enterprise 188
14.3 Linked Data Engineering 189
  14.3.1 Web of Data 189
  14.3.2 Semantic Search in Semantic Web 191
14.4 Triplestore 192
14.5 Example: Semantic Web for Home Assets 193
  14.5.1 Define Classes Hierarchy 194
  14.5.2 Define Properties Hierarchy 195
  14.5.3 Add OWL 195
  14.5.4 Data Retrieval Examples 195
14.6 Reference: Commonly Used Namespace 195

This chapter studies the design, development and deployment of ontology and semantic web model. Both the methodologies and the tools are concerned.

14.1 Ontological Engineering

Different ontologies (class, property, hierarchy, interpretation, logic, etc.) can be used to describe the identical knowledge. This is known as the "problem of semantic gap". It is challenging (maybe impossible) to find the optimal and consistent way of representing the knowledge using ontology and semantic web.

Ontological engineering studies the systematic ways to design the ontology and the semantic web for a specific domain, application or task (recall Fig. 11.2). It has at least the following research interests:

- Ontology design concerns with the methods to systematically design, develop, and develop ontology models.
- Ontology mapping concerns with the methods to efficiently compare different ontology models.
- Ontology merging concerns with the methods to efficiently combine ontology models.
- Ontology learning concerns with the automatic learning of new knowledge by an existing ontology model, when new data sets are provided.

14.2 Ontology Design

Ontology design describes all activities necessary for the construction of an ontology model. The goal is to design ontology models efficiently, consistently, and sometimes distributively (collaboratively) if required. Notice that a good ontology model is not built in one go. It is often iteratively improved.

14.2.1 General Tasks

Design, develop and deploy ontology model for a sophisticated system can be time and manpower consuming. It is important to manage each step during the entire procedure to ensure healthy development of the system. This includes

- Scheduling: identify tasks and problems to be solved by the semantic web; plan and arrange resources, time, manpower and money ahead.
- Control: guarantee correct execution of tasks and problems to be solved.
- Quality assurance: guarantee all steps are done correctly, including using the correct software, and everything is documented in details.

In pre-development stage, environment study needs to be carried out. This is mainly to identify what software to use to host the semantic web, and what interface/API shall the semantic have, and what applications would call the API to talk to the semantic web. A feasibility study is also necessary. For example, we need to consider whether it makes sense to develop a semantic web for the application, and whether the semantic web design is realizable.

In development stage, domain expert needs to come in to build domain knowledge in a conceptual model. Knowledge engineer or data scientist then formalize the conceptual model into a computable (formal) model, then into ontology representation language.

Finally, in the post-development phase, pipeline needs to be designed to maintain, update and scale up and down the ontology model. In the case where the ontology model needs to be migrated into different platforms, used by unplanned applications, or merged with other models, necessary changes need to be made to the model. In the case when the knowledge in a model needs to be exported, knowledge recycle needs to be supported.

There are many ontology support activities. These activities need to be carried out during the different stages of ontology development. Below is a list of some of these activities.

- Knowledge acquisition. Interview experts in the field and learn domain knowledge from them. This is referred as ontology learning. This is often done manually by the knowledge engineer or data scientist in the beginning stage. It is also possible to develop tools to automatically gather information and transfer it into ontology models, and even merge it with existing models.
- Technical evaluation. A domain expert checks the developing ontology model occasionally to make sure everything is correct.
- Integration and merging. This refers to the case where a big (scaled-up) ontology model can be built from a small existing ontology model.
- Alignment. Where there are multiple ontology models describing the same physical thing, alignment needs to be made to ensure knowledge consistency.
- Documentation and version management.

14.2.2 Ontology Design Basics

Designing comprehensive ontology model and associated semantic web requires professional skills from knowledge engineers and data scientists. In this section, a basic method is introduced for tutorial purpose. The method introduced in this section, like many other ontology design methods, involves the iterative designing and refining the model.

If there is an expert from the domain of interest, discuss with him each step during the design.

Determine the Scope

As a first step, decide what knowledge should be included in the semantic web. It is important to draw a clear boundary between what the ontology should include, and what should not.

Decide what the ontology would be used for, who would use it, and what kind of information the user needs to query from the ontology model. Think of a list of competence questions, i.e., the questions that the semantic web would be asked in its practical usage.

Notice that the scope of the ontology model, especially the competence questions may change during the development of the semantic web. Try to leave some margin and look at the bigger picture.

Look for existing ontology model of similar scope, and consider reusing them instead of starting from scratch whenever possible. This should same the cost. In addition to the semantic web itself, the interface, add-on tools, etc., can also be reused.

Determine the Vocabularies: Symbols and Interpretations

As the first step to design the schema of the semantic web, it is often a good idea to brainstorm the symbols and interpretations the semantic web would want to include as part of the knowledge base.

For example, to build a semantic web for fruits, visit a local store and list down all the fruits on sale on a piece of paper, as well as the food that goes well with the fruits. This would often serve as a good starting point.

If the domain of interest already has a database (not necessarily semantic web), let the existing data stored in the database to inspire the ontology design. The data usually reflects the features that people concerns with the most, and they shall probably be covered in the semantic web as well.

Design Hierarchy and Associate Properties

Once we have a big picture of what symbols and terms to be included in the semantic web, go through the vocabularies and categorize them into classes, relations and properties. Design class and property hierarchies accordingly.

Furthermore, associate properties with classes, i.e., decide the domain and range of properties as well as the constraints.

Populate Classes with Individuals

Finally, once the schema is ready, populate the classes with individuals, and assign properties to these individuals. The individuals may come from an existing database, in which case information conversion is required. A program that automates the information conversion would become handy when the amount of data is huge.

Iterative Development

Notice that the above processes shall be carried out iteratively to improve the ontology model and the semantic web.

14.2.3 Semantic Web Design for Enterprise

The basic semantic web design method introduced in Section 14.2.2 is not suitable for large size enterprise tier semantic webs, in the later case of which a more formal workflow is often required. Continuous inputs from domain experts and knowledge engineers are expected.

An demonstration of the workflow is given in Fig. 14.1. In the workflow, domain experts and knowledge engineers get different involvement at different stages of the workflow. The workflow is iteratively executed for many times, each time with a different focus. For example, in an early stage of the design, an iteration may focus on the lexicon of the semantic web, while in an late stage, on OWL.

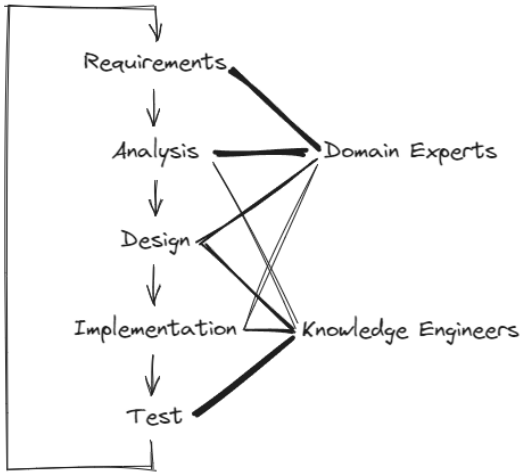


FIGURE 14.1
Semantic web design workflow.

There are existing ontology "templates" available in the community. These templates are essentially reusable ontology models one can learn from and adapt to his application as a starting point. An repository of such templates is given in

```
http://ontologydesignpatterns.org/
```

14.3 Linked Data Engineering

Linked Data is a set of design principles for sharing machine-readable interlinked data on the internet. Linked Data engineering, as its name suggests, is the practice of creating, managing and sharing machine-readable (and also human readable, in most occasions) data on the web.

14.3.1 Web of Data

Consider the conventional way of retrieving data from a web server as shown in Fig. 14.2. The address of the server, in this case a URL, is used to identify the server. The local machine request for services using the web API defined by the server. HTTP/HTTPS protocol is used to shake hand and transmit the data between the server and the local machine. Behind the screen, the server retrieves the data either from other web servers or from its internal database.

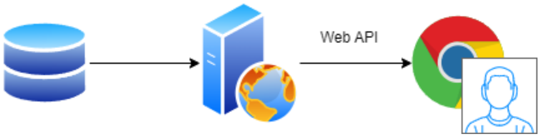


FIGURE 14.2
Linked Data example: web browsing.

A web server is like an isolated data island. Different web servers may apply different web APIs. The web API decides what services the server provides as well as how the local machine can interact with the server. When a server changes its web APIs, all applications linked to the server also need to change the associated interfaces.

In addition, since the servers are isolated both physically and logically, it is difficult for the local machine to retrieve, compare and process semantically relevant data across machines. This is certainly a drawback as the data retrieving and processing would have been more efficient and reliable if the data is linked together from the semantic perspective.

Linked Data tries to address the above issues by standardize the way data is stored and shared. Some important principles include:

- Use (HTTP) URIs as the names for information pieces. URIs are standardized and both human and machine readable. When HTTP URIs are used, people can conveniently access the information pieces and look up things.
- Use RDF to store information, and support SPARQL for querying information. This adds semantics, flexibility and scalability to the information.
- In each knowledge piece, include links (in the form of URIs) to other knowledge pieces. This allows linking information pieces together, though they might be stored physically distributively.

As an example, the following is the URI to HTML page of "William Shakespeare" on DBpedia.

```
https://dbpedia.org/page/William_Shakespeare
```

The RDF that backs up this web page can be downloaded from

```
https://dbpedia.org/data/William_Shakespeare.rdf
```

Both the above links are accessible by the public. Inside the RDF are not only the introduction to Shakespeare (for example, his name in more than 10 languages) but also many links to other relevant resources such as figures of him on Wikipedia. This RDF is a good example of practicing linked data in storing information.

The result of Linked Data is the "Web of Data". We have been using DBpedia as an example of RDF database. In fact, DBpedia is not an isolated database, but an important component of the existing web of data project, where it links to other RDF databases such as friend-of-a-friend (FOAF) and many more. URIs that link to these databases can be found everywhere in the RDFs of DBpedia.

The web of data is growing. "The Linked Open Data Cloud" website *lod-cloud.net* gives an overview of some of the databases. A screenshot is given in Fig. 14.3. DBpedia sits in the center of this figure, as it was one of the earliest and most comprehensive knowledge base.

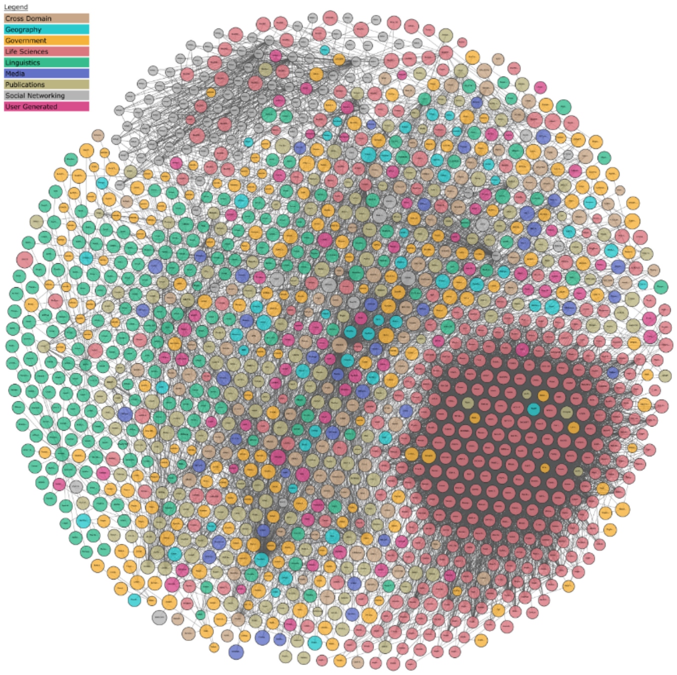

Legend
Cross Domain
Geography
Government
Life Sciences
Linguistics
Media
Publications
Social Networking
User Generated

FIGURE 14.3
The linked open data cloud from *lod-cloud.net*.

14.3.2 Semantic Search in Semantic Web

Consider the following query: "tell me something about Armstrong who landed the eagle on the moon". Conventionally, the search engine would go through websites, documents, database and other resources, looking for keywords "Armstrong", "landed", "eagle", "moon". It will return the sentences or paragraphs that hit most of the matching. This might work. However, it has some drawbacks such as being weak to typo and synonyms.

Semantic search refers to the search of content by meaning, not by keywords. For example, in the context of the increasingly popular LLM, semantic search can be realized using a "encoder" (a component in the Transformer) to translate the query to the semantic space. The paragraphs whose semantic space images are close to the query are returned.

In the context of semantic web, semantic search refers to the locating and retrieval of information from the semantic web efficiently based on the query. It is semantic in the sense that the triplestore is able to use logic reasoning during the searching.

In this example, with the semantic web, the search engine interprets "Armstrong" not just as a plain word that spells "a-r-m-s-t-r-o-n-g", but as a person, "Neil Armstrong", with his birthday, birthplace, nationality, etc., all available in one place from his URI (in the case of DBpedia, *https://dbpedia.org/data/Neil_Armstrong.rdf*). The search engine will understand that "landing the eagle on the moon" refers to "Apollo 11 mission" as this event is also well defined and has a URI called "*dbr:Moon_landing*" (notice that *dbr* refers to an HTTP URI defined elsewhere).

Since the semantic web links all the information pieces together, we could have also searched "tell me something about the first person landed on the moon" to retrieve everything about "Neil Armstrong" all the same, as he can be traced both from his name and from the activities he participated.

14.4 Triplestore

Triplestore is the database engine of the RDF model.

When it comes to relational database, there are many choices of DBMS such as Microsoft SQL Server, Oracle DBMS, MySQL and MariaDB. Similarly, there are many choices of triplestores for semantic web. A list of widely used triplestores is given below, just to name a few.

- GraphDB: a commercialized enterprise-tier semantic graph database management system compliant with W3C standards. It is famous for its performance and inference capabilities. It also provides free-tier for learning and for small projects, with limited capability.
- Apache Jena: an open-source Java framework for building semantic web

and linked data applications. It has RDF APIs that can read and process RDF and SPARQL written in XML, Turble, JSON-LD and N-Triples.

- Virtuoso: a multi-model DBMS for both RDB and NoSQL databases such as RDF. It is famous for its scalability and standards compliance.
- AllegroGraph: a closed source triplestore which is designed to store RDF triples. It also operates as a document store designed for storing, retrieving and managing document-oriented information, in JSON-LD format. AllegroGraph is currently in use in commercial projects and a US Department of Defense project. It offers both free-tier license and enterprise-tier license.

More triplestores can be found at W3C, where a list of triplestores is maintained [1, 3]. As of this writing, there are 50 triplestores registered at W3C.

For the demonstrations given in this notebook, GraphDB is used, unless otherwise mentioned. A full instruction on using GraphDB, including applying for access and installation of the software, can be found at *graphdb.ontotext.com*.

Notice that RDF/RDFS/OWL/SPARQL APIs can be enabled using packages or libraries in many programming languages. For example, in Python, there are several packages available for semantic web operations, including `rdflib`, `Owlready2`, `SPARQLWrapper`, etc. Some of these packages have "lite" triplestore engine built-in which provides some SPARQL features for in-memory operations. They do not have all the features of a triplestore. However, they can connect to a triplestore API, in which case they serve as Python-to-triplestore interfaces.

14.5 Example: Semantic Web for Home Assets

As an example, we are creating a semantic web for a the assets in a household. Here "assets" refer to the electrical products, furniture, LEGOs, and other non-consumable, relatively static elements.

We will start with defining classes to divide everything into large groups, including electrical product, furniture, toy, etc. Under each class, sub-classes are defined, such as TV, computer, game console under electrical product, bed, chair, sofa, lamp under furniture, and LEGO under toy. Lastly, we will define instances under each sub-class. For example, bedroom TV and living room TV under TV, Nintendo SWITCH under game console, living room sofa under sofa, study computer, living room computer, TV attached computer under computer, etc.

We will then use RDFS to enforce schema to the RDF model as follows.

Consider electrical product class for example. All elements in this class shall have a property called "hasBrand", which maps them to a pre-defined "electricalBrand" class, inside which are commonly seen electrical brands such as Boche, Siemens, Nintendo, Sony, Google, etc. The electrical product shall also have a "hasPrice" property, "warrantyExpiresAt" property, etc. The similar concept applies to all other produces including furniture, etc.

Finally, use OWL to setup some limits of the properties. For example, for electrical products, the price is usually between 0 to 5000 dollars, etc.

The result should be something like Fig. 14.4, after visualization.

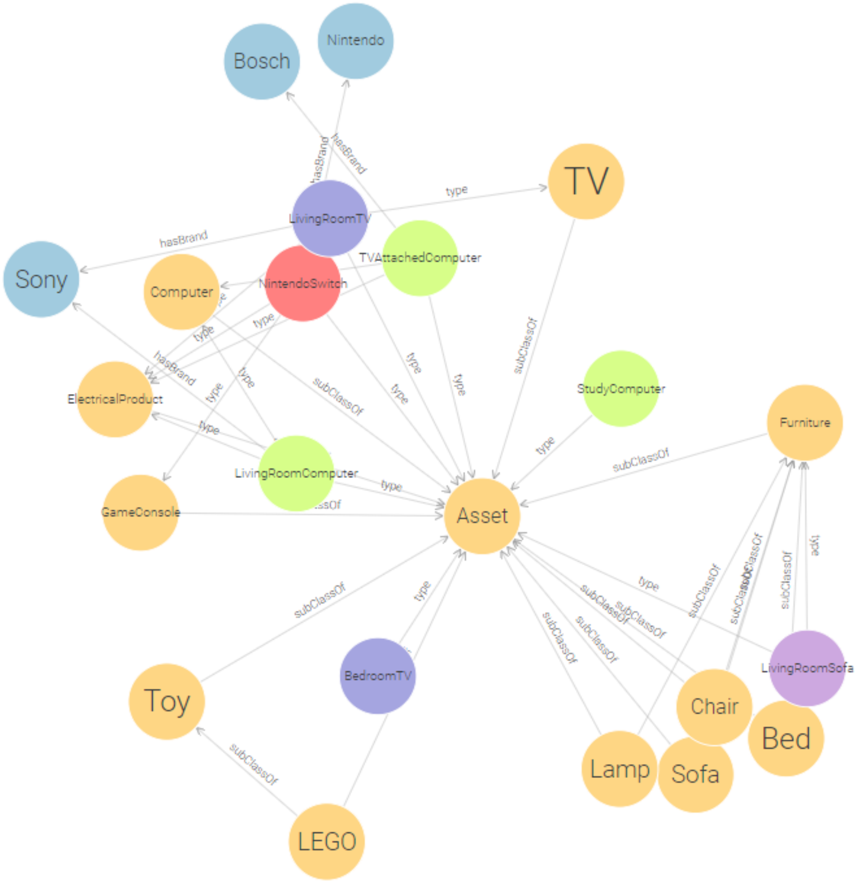

Bosch
Nintendo
hasBrand
hasBrand
TV
type
LivingRoomTV
hasBrand
Sony
Computer
NintendoSwitch
TVAttachedComputer
type
type
type
hasBrand
type
subClassOf
type
type
type
subClassOf
ElectricalProduct
type
StudyComputer
Furniture
type
subClassOf
LivingRoomComputer
type
GameConsole
Asset
subClassOf
type
type
subClassOf
subClassOf
subClassOf
subClassOf
subClassOf
subClassOf
subClassOf
type
BedroomTV
LivingRoomSofa
Toy
Chair
Bed
subClassOf
Lamp
Sofa
LEGO

FIGURE 14.4
An example of an RDF model in GraphDB that describes house assets. This is only a demonstration graph and the information inside is artificial and not true.

TABLE 14.1
Commonly used name spaces in RDF models. URI is neglected since they can be easily found online.

Namespace	Description
`rdf`	RDF syntax.
`rdfs`	RDFS syntax.
`xsd`	XML syntax.
`foaf`	Friend-of-a-friend. It describes people, their activities and relations to other people and object.

“sloppy

14.5.1 Define Classes Hierarchy

“nobreak

14.5.2 Define Properties Hierarchy

“nobreak

14.5.3 Add OWL

“nobreak

14.5.4 Data Retrieval Examples

“nobreak

14.6 Reference: Commonly Used Namespace

Commonly used built-in name spaces are summarized in Table 14.1.

To search URI for a name space, use *prefix.cc*.

Bibliography

[1] Category:triplestore.

[2] The common layered semantic web technology stack.

[3] Largetriplestores.

[4] Package index.

[5] The R project for statistical computing.

[6] Rstudio.

[7] Using r language with anaconda.

[8] Franck Michel, Johan Montagnat, and Catherine Faron Zucker. *A survey of RDB to RDF translation approaches and tools.* PhD thesis, I3S, 2014.