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

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) are released, people from photocopy shops would start selling notebooks photocopies that they claim to be from the top scorers of the exam. Much curious as I was about what these notebooks look like, never have I expected myself to actually learn anything from them, mainly for the following three reasons.

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

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

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

However, 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 (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 much time on other irrelevant but still important and interesting contents.

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

—

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meaning that there is no guarantee for the statement and knowledge in this notebook to be absolutely correct as it is not peer reviewed. **Do NOT cite this notebook in your academic research paper or book!** Of course, if you find anything helpful with your research, please trace back to the origin of the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

This notebook is suitable as:

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

This notebook is NOT suitable as:

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

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

—

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

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

Preface

This notebook introduces probability, statistics, data science and engineering. They are the "must-have" ability in most, if not all, academic and industrial projects.

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 statistics hypothesis and draw conclusions and predictions.

In Part III of the notebook, commonly used software and toolkits for statistics 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. Notice that Artificial Intelligence (AI) has become increasingly popular in recent years for data analysis. There is a separate notebook on introducing AI. In this notebook, the tools to process data using AI is only briefly introduced, again focusing only on the usage of software, not theory.

As a bonus, in Part IV, semantic web, the database framework defined under Web 3.0, is introduced. Semantic web does not necessarily contribute to solving a specific problem using a specific statistics theory. It is rather a concept or organizing and exchanging data efficiently.

Key references of this notebook are summarized as follows. Notice that these materials are so very widely used here and there in the entire notebook that it becomes improbable to address them each time they are used.

For probability and statistics parts:

- Spiegel, Murray, John Schiller, and Alu Srinivasan. *Probability and statistics.* 2020.

- Dekking, Frederik Michel, et al., *A Modern Introduction to Probability and Statistics: Understanding why and how.* Vol. 488. London: Springer, 2005.

For data science part:

- Kirill Eremenko, *R Programming A-Z: R For Data Science With Real Exercises*, Udemy Course.

- Lakshmanan, Valliappa, Martin Görner, and Ryan Gillard. Practical Machine Learning for Computer Vision. " O'Reilly Media, Inc.", 2021.

- Jose Portilla, *Complete TensorFlow 2 and Keras Deep Learning Bootcamp*, Udemy

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

List of Figures

List of Tables

Part I

Probability

1

Random Variables and Experiments

CONTENTS

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

1.1 Randomness and Stochasticity

People use *random* and *stochastic* to describe a variable or a model whose corresponding measurement or outcome are not precisely predictable.

By saying *randomness*, we often refer 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 which can be described as the statistic property 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.

By saying *stochasticity*, we often refer 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 notebook mainly studies random variables and its impact on data science. Stochastic process and control of stochastic systems are introduced elsewhere in control system related notebooks.

1.1.1 Random Experiments

"Experiment" is one of the most important activities in science and engineering. Very often, experiments are used to verify a theory. In these cases, experiments are carefully designed so that its outcome is 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, there are other experiments where we do not have full control over their results 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 experiments 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 obtained from the result efficiently.

1.1.2 Sample Space

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

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 *noncountably infinite sample space*. Examples are given below.

- Finite sample space. Randomly pick 5 balls out of a bag that contains 50 red balls and 50 green balls. The number of red balls in the picked 5 balls forms a finite sample space.
- Countably infinite sample space. The number of products a new machine can manufacture before it is broken forms a countably infinite sample space.
- Noncountably infinite sample space. The precise time consumption it takes for a PC to start forms a noncountably infinite sample space.

Countable Infinity VS Uncontable 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 an axis, on an axis interval, 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 space and countably infinite sample space are also known as *discrete sample space*, whereas noncountably infinite sample space, *nondiscrete sample space*.

1.1.3 Events

An event is a subset A of the sample space S, i.e., it is a portion of possible outcomes. An event may or may not occur depending on the outcome of an experiment. In the special case where A has only one element, the event is also called an *elementary event*, or a *sample*. 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 the *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.

1.2.1 Classical and Empirical Probability

There are different ways to calculate or estimate the probability of an event. In the *classical approach* where we know the total number of outcomes n, all of which have a equal chance of happening, and there are h ways for the event to occur, then the probability of the event is h/p. Of course, this method applies to only some finite sample spaces.

In the *frequency approach*, we can repeat the experiment n times where n is a large number, and record the number of instants where the event happened as h. The *empirical probability* of the event is obtained by calculating h/p which should converge to the true probability as n goes large.

1.2.2 Axioms

Both the classical approach and the frequency approach have some drawbacks. It is often difficult to define "equal chance" and "large number" in the two approaches respectively. Therefore, *axiomatic approach* is introduced as follows.

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 $P(A_i)$ the probability of the 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, $P(A_1 \cup A_2) = P(A_1) + P(A_2)$.

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, $P(A_1 \cup A_2) = P(A_1) + P(A_2) - P(A_1 \cap A_2)$.

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 (elementary events are always mutually exclusive) $A_i, i = 1, ..., n$, 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.

1.2.3 Conditional Probability

Assume 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. Think of this definition as S replaced by A since A has been confirmed 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.

In the special case where $P(B|A) = P(B)$, or equivalently $P(A \cap B) = P(A)P(B)$, the two events A and B are said to be *independent events*.

1.3 Permutation and Combination

In classical probability, calculating the cardinality of event A and sample space S is the key to solving the problem. *Permutation* and *combination* are widely used for such calculations.

Suppose that there are n distinct objects, and we would like to select r objects from them and put them into a sequence. The permutation

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

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

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

where $n(n-1)(n-2)\cdots\times 1$ is often denoted as $n!$. Therefore, (1.1) becomes

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

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

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

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

2

Random Variables and Distributions

CONTENTS

The definition of random variable has been introduced in the previous chapter. This chapter digs deeper into the different types and properties of random variables, as well as how we describe a random variable.

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 Variables

If a random variable X takes only discrete values $x_1, x_2, ...$, it is called a *discrete random variable*. The probability of X taking a particular value x is denoted by $P(X = x)$ or simply $f(x) = P(X)$ if no ambiguity. In this case, $P(X = x)$ and $f(x)$ are called 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*. It is easy to prove that $F(x)$ is nondecreasing, and $\lim_{x\to-\infty} F(x) = 0$, $\lim_{x\to\infty} F(x) = 1$. Also, $F(x)$ "jumps" at each $P(X = x_k) > 0$ and it is continuous from the right.

2.1.2 Continuous Random Variables

A random variable X may also take continuous values in many applications. For example, let X denote the time consumption to finish a task, which can be any value above 0 hours.

In this case, the chance for X to take a precise value, say for the student to finish his homework using precisely 25 minutes 13 seconds and 750 milliseconds, is very small (in fact, zero). The probability function $P(X = x)$ becomes pointless. The cumulative distribution function $F(x) = P(X \leq x)$ still makes sense, as it essentially calculates the probability of X given by not a precise value but within a range.

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

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

Therefore,

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

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 taken as a special case of continuous random variables with impulse PDF.

2.2 Joint Distributions

The joint distribution refers to the distribution of multiple random variables. It is especially useful when these variables are correlated, in which case the joint probability function or PDF can be used to describe the correlation. For example, in a classic system identification problem the unknown system parameters are correlated with the measurements and their correlations are described by the joint PDF. Hence, we can estimate the system parameters using the measurements.

2.2.1 Joint Probability

Consider two random variables X and Y. In the case of discrete variables, 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, joint PDF of X and Y is defined such that

$$
\int_{x=a}^{b} \int_{y=c}^{d} f(x,y)dxdy = P\left(a < X < b, c < y < d\right) \tag{2.1}
$$

Integrate (2.1) w.r.t an axis gives the PDF of the other variable. For example,

$$
f_X(x) = \int_{y=-\infty}^{\infty} f(x,y) \tag{2.2}
$$

which becomes a function of a single variable, x. Here $f_X(x)$ is the PDF of X, completely ignoring the other variable Y.

We can define cumulative distribution function for joint distribution likewise as follows.

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

2.2.2 Conditional Distributions

As introduced earlier, (2.2) gives the PDF of X by completely ignoring Y. In other words, it is the best guess of X (in the form of PDF) when there is no measurement of Y.

Conditional distribution, on the other hand, try to find the best guess of X when Y is measured. For example, from (2.1), let Y be measured by a fixed value y, and we would like to calculate $f_{X|Y}(x|Y = y)$, i.e., the PDF of X given the known information $Y = y$. 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.

$$
f_{X|Y}(x|y) = \frac{f(x,y)}{f_Y(y)} \tag{2.3}
$$

where $f_Y(y)$ is obtained using (2.2). Equation (2.3) is given as an analytical equation of both x and y. Just substitute the measured value of y into (2.3) to get a function of x alone. Since (2.3) itself is a PDF of X, its integration w.r.t x is certainly 1. This can be easily verified as follows.

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

for any y. Notice that given a particular value of y, $f_Y(y)$ is a constant value independent of x, hence can be taken out from 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, (2.3) 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.

2.2.3 Parameter Estimation with Conditional Distribution

Equation (2.3) is widely used in parameter estimation. 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.

Without measurement x, the estimate of θ is given by

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

where $f_\theta(\theta)$ is obtained using (2.2). This is known as the *priori estimation* of θ.

Given measurement x, the *posteriori estimation* of θ can be obtained as follows. From (2.3),

$$\begin{aligned}
f_{\theta|X}(\theta|x) &= \frac{f_{X|\theta}(x|\theta)f_\theta(\theta)}{f_X(x)} \\
\hat{\theta} &= \int_{-\infty}^{\infty} f_{\theta|X}(\theta|x)d\theta
\end{aligned} \tag{2.4}$$

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*, in this case a constant value given x.

Equation (2.4) can be memorized as follows.

$$\text{posteriori} = \frac{\text{likelihood} \times \text{priori}}{\text{evidence}} \tag{2.5}$$

Terms *priori* and *prior*, *posteriori* and *posterior* can be used interchangeably.

2.2.4 Geometric Probability

Geometric probability is an extension of the classical probability to continuous random variables. Similar with classical probability where it is prerequisite that all elementary events share the same probability, in geometric probability it is assumed that all random variables (usually 2 or 3 variables in total) are uniformly distributed. The probability of an event happening then 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 does not show up.

Calculate the chance of the two people successfully meeting up.

Draw the sample space in a 2-D plot as shown in Fig. 2.1, 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.

2.3 Probability of Derived Variables

Let X be a random variable with PDF $f_X(x)$. Let U be another random variable which is a function of X via $U = \phi(X)$. The PDF of U can be calculated as follows. For simplicity, assume that $U = \phi(X)$ is a injective function (one-to-one function), and $X = \phi^{-1}(U) = \psi(U)$. In that case, the

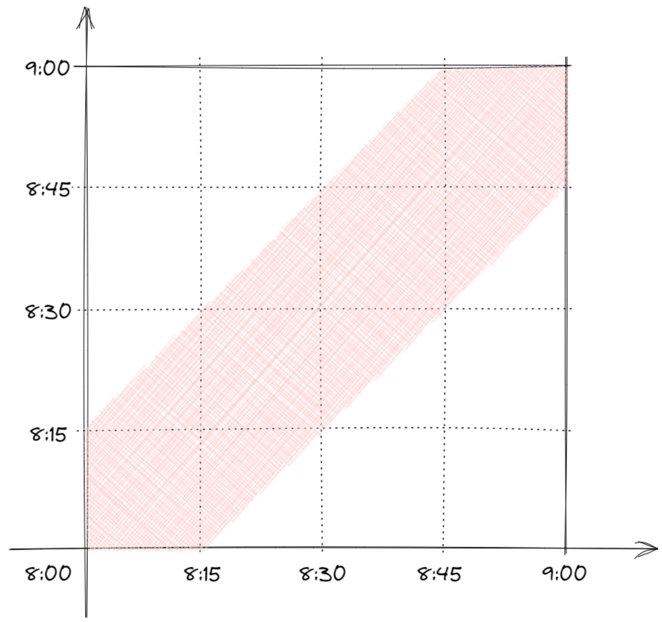

FIGURE 2.1
Sample space of two people arriving at part from 8:00 AM to 9:00 AM.

PDF of U, $g(u)$, can be obtained as follows.

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

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

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

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.

$$g(u) = \int_{-\infty}^{\infty} f(x, u - x)dx \tag{2.6}$$

In the special case where X and Y are independent, $f(x, y) = f_X(x)f_Y(y)$, and (2.6) 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.

3

Measures of Distributions

CONTENTS

Given the probability function or PDF of a random variable, a lot of information can be extracted. Commonly seen measures of a distribution are introduced in this chapter.

3.1 Expectation

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.

$$\mathrm{E}(X) = \sum_{i=1}^{n} x_i P(x_i) \tag{3.1}$$

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 (3.1). For continuous random variable, it is

$$\mathrm{E}(X) = \int_{-\infty}^{\infty} x f(x) dx \tag{3.2}$$

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 (3.1) and (3.2). Furthermore, if X, Y are independent, recall $f(x,y) = f_X(x)f_Y(y)$,

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

3.2 Variance and Standard Deviation

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

$$\begin{aligned} \mathrm{Var}(X) &= \mathrm{E}\left((X-\mathrm{E}(X))^2\right) && (3.3) \\ &= \mathrm{E}\left(X^2 - 2X\mathrm{E}(X) + \mathrm{E}(X)^2\right) \\ &= \mathrm{E}\left(X^2\right) - \mathrm{E}(X)^2 && (3.4) \\ \mathrm{Std}(X) &= \sqrt{\mathrm{Var}(X)} \end{aligned}$$

Variance and standard deviation are sometimes denoted as σ^2 and σ respectively. Notice that (3.4) 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 (3.4) the variance is

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

Think of X as an estimation of a parameter θ. If the estimation is non biased, $E(X) = \theta$. Therefore, from the above equation, for a non-biased estimation, the variance of the estimates (left) equals to the MSE of the estimate (right) with sample numbers approaching infinity.

Some features of variance calculation are given below.

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

For independent random variables X and Y,

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

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

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

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.

3.3 Moments

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

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

By simply saying *moment* without specifying c, it is by default that $c = 0$. 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.

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

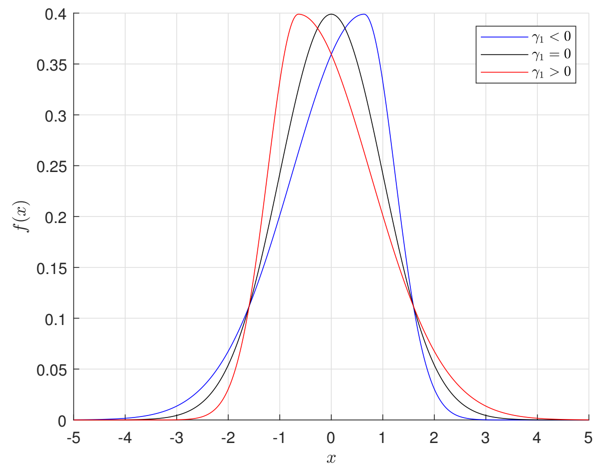

FIGURE 3.1
Demonstration of PDF with different skewness.

where

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

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

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. 3.1.

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. 3.2.

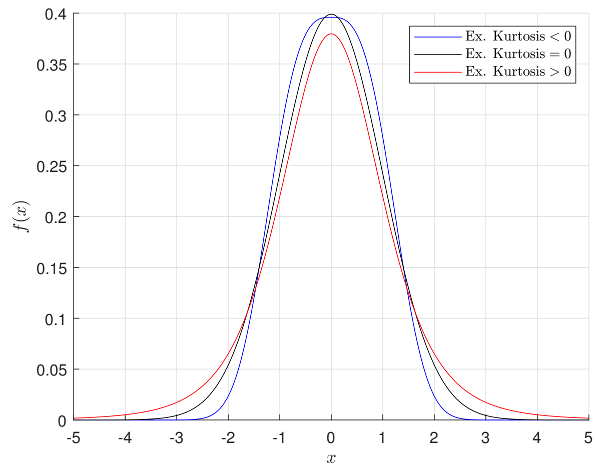

FIGURE 3.2
Demonstration of PDF with different excess kurtosis.

3.4 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.

3.4.1 One Variable

The PDF of one variable can be derived from the joint distribution using (2.2). It is straight forward to get the expectation and variance for that variable as follows.

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

3.4.2 Covariance

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

$$\begin{aligned}\mathrm{Cov}(X,Y) &= \mathrm{E}\left((x-\mathrm{E}(X))(y-\mathrm{E}(Y))\right) &(3.5)\\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\mathrm{E}(X))(y-\mathrm{E}(Y))f(x,y)dxdy &(3.6)\end{aligned}$$

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 (3.6)

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

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 as follows. Notice that lemma (3.7) is used in the proof.

Lemma

For two random variables X and Y,

$$\mathrm{E}(XY)^2 \leq \mathrm{E}(X^2)\mathrm{E}(Y^2) \tag{3.7}$$

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 (3.7) on (3.5), (3.3) 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,

$$-\sigma_X\sigma_Y \leq \sigma_{XY} \leq \sigma_X\sigma_Y \tag{3.8}$$

3.4.3 Correlation

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 (3.8), apparently $-1 \leq \rho \leq 1$. When two variables are uncorrelated or independent, $\rho = 0$. If $\rho = 1$, it is called *perfect positive correlation*. This happens usually because the two variables are positively linearly depended, for example, $X = 2Y$ or $X = Y + 1$. If $\rho = -1$, it is called *perfect negative correlation*, and the similar idea applies.

3.5 Important Theorems

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

3.5.1 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,

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

3.5.2 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.

4

Special Distributions

CONTENTS

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

4.1 Bernoulli and Binomial

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 $0 \le p \le 1$, $0 \le q \le 1$ 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, $\frac{q-p}{\sqrt{pq}}$ and $\frac{1-6pq}{pq}$, 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

$$\begin{aligned} f(x) &= P(X=x) \\ &= \begin{pmatrix} n \\ x \end{pmatrix} p^x (1-p)^{n-x} \\ &= \frac{n!}{x!\,(n-x)!} p^x (1-p)^{n-x} \end{aligned} \tag{4.1}$$

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 distribution are np, npq, $\frac{q-p}{\sqrt{npq}}$ and $\frac{1-6pq}{npq}$, respectively.

Binomial distribution is naturally a verification of CLT on Bernoulli trail. According CLT, when n is large, binomial distribution should approach 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, ..., each with probability p_1, p_2, ..., 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

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

Do distinguish binomial distribution with *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

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

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.

4.2 Normal Distribution

Normal distribution, also known as Gaussian distribution, is a continuous distribution. It is one of the most widely used assumption of 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.

4.2.1 Single 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 $Z = \dfrac{X-\mu}{\sigma}$, 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.

4.2.2 Multivariate Normal Distribution

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

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

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.

4.3 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. 4.1.

Notice that The sum of two Poisson distribution with shape parameters λ_1 and λ_2 is also Poisson distribution with $\lambda = \lambda_1 + \lambda_2$.

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. 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 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. 4.2.

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 associated with the number of trails. Therefore, the number of failure of the machine in a year can be approximately described by binomial distribution with $x = 31536000$. The underlying assumption is that the machine can either pass or fail every second, and it can fail only once per second.

Another example of Poisson distribution is the number of visitors of a

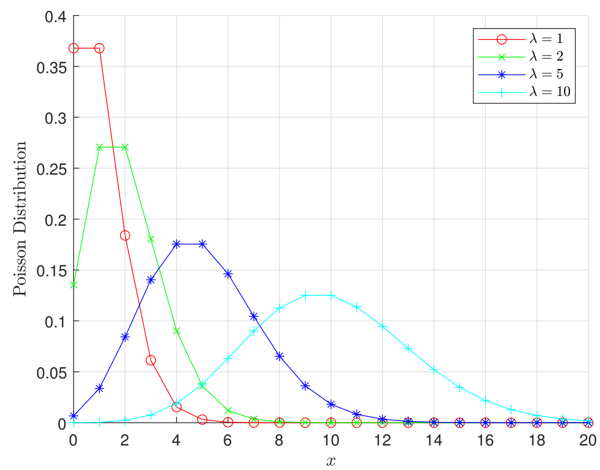

FIGURE 4.1
Poisson distribution with different λ.

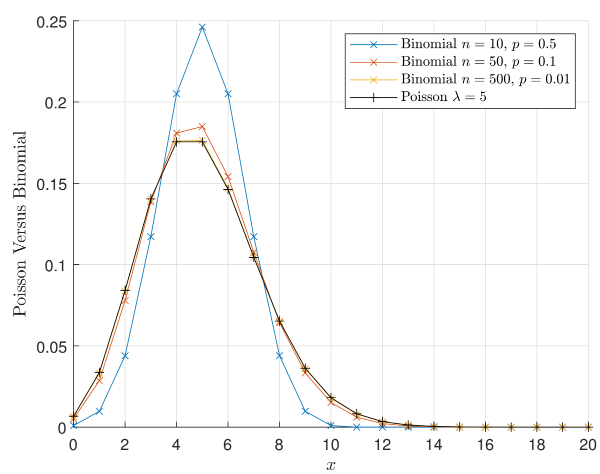

FIGURE 4.2
Poisson distribution approximation using binomial distribution.

website in a given period of time. Each internet user has a small probability p to visit the shopping site. 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.

4.4 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 a special case of the gamma distribution which will be introduced in more details later. Since it is an important widely used distribution, exponential distribution is introduced here, right after the normal and the Poisson distributions.

The PDF is the exponential distribution is given by

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where λ is the shape parameter. Exponential distributions with different choice of λ is plotted in Fig. 4.3.

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

4.5 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

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

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

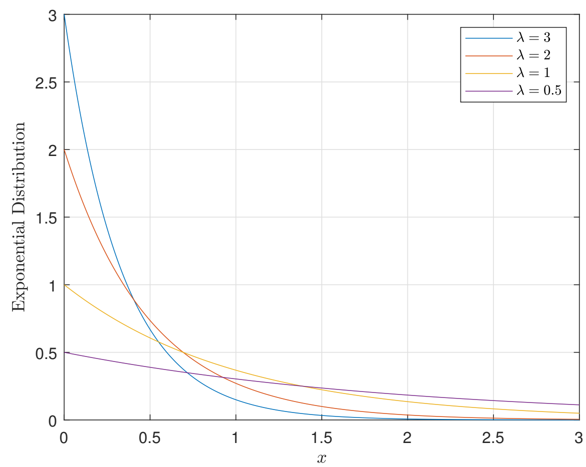

FIGURE 4.3
Exponential distributions with different λ.

4.6 Cauchy Distribution

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

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

and it is plotted in Fig. 4.4.

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.

4.7 Gamma, Chi-Squared and Beta Distributions

The Γ, χ^2 and β distributions are introduced as follows.

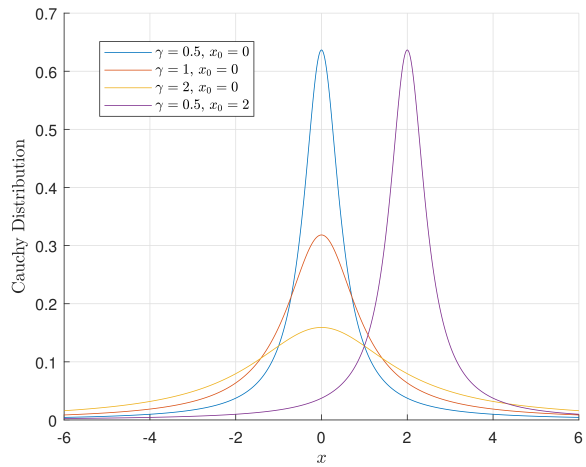

FIGURE 4.4
Cauchy Distribution.

4.7.1 The Γ Distribution

Gamma distribution is a continuous distribution with the following PDF

$$f(x) = \begin{cases} \dfrac{\beta^\alpha}{\Gamma(\alpha)} x^{k-1} e^{-\beta x} & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

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 $\alpha > 0$. Gamma distribution is often used to model the interval of two events in the Poisson process.

The plot of the PDF of Gamma distribution is given in Fig. 4.5.

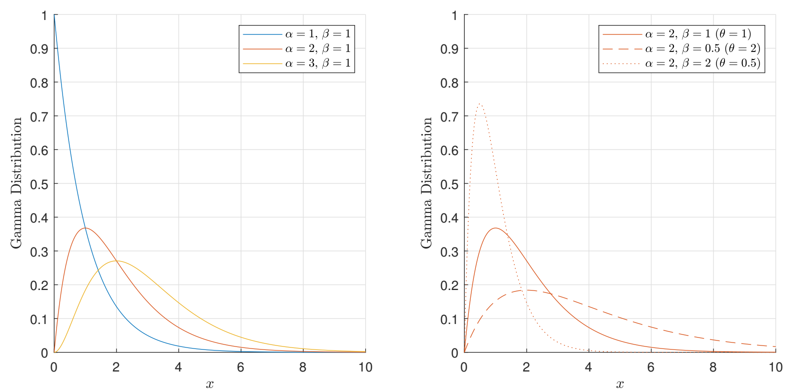

FIGURE 4.5
Gamma Distribution.

A Little Bit about Gamma Function

Gamma function is given by

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

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

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

Therefore, $\Gamma(n) = \Gamma(n-1)\Gamma(n-2)\ldots \times 3 \times 2 \times 1 \times \Gamma(1) = (\alpha - 1)!$ 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.

4.7.2 The χ^2 Distribution

The χ^2 estimation is a special case of Gamma distribution. In (4.3), let $\alpha = \frac{r}{2}$ and $\beta = \frac{1}{2}$ to get the PDF of χ^2 distribution as follows.

$$f(x) = \begin{cases} \frac{1}{\Gamma(r/2)2^{r/2}} x^{r/2-1} e^{-x/2} & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

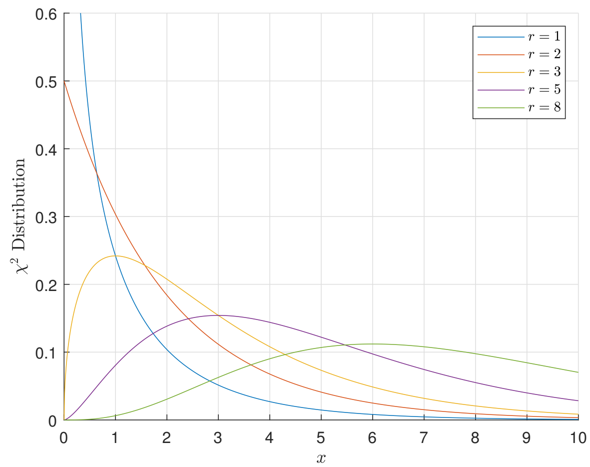

FIGURE 4.6
The χ^2 Distribution.

Equation (4.4) 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. 4.6.

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.

4.7.3 The 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 (4.3). Denote $X = \frac{X_1}{X_1 + X_2}$. In this case, X would be a random variable following β distribution whose PDF can be derived from (4.3) and it is given by

$$f(x) = \begin{cases} \frac{\Gamma(\alpha_1+\alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)} x^{\alpha_1-1}(1-x)^{\alpha_2-1} & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

In some literatures, notation (a, b) or (α, β) are used to denote (α_1, α_2) in (4.5).

4.8 Student's t-Distribution

The zero-mean t-distribution PDF is given by

$$f(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\sigma\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{x^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}} \tag{4.6}$$

where ν and σ are known as the shape and scale parameters respectively. When $\nu \to \infty$, (4.6) reduces to normal distribution.

Student's 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

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

then X follows t-distribution (4.6).

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 (4.6) are 0, $\frac{\nu}{\nu-2}$ $(\nu > 2)$, 0 $(\nu > 3)$ and $\frac{6}{\nu-4}$ $(\nu > 4)$ respectively.

4.9 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

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

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

The PDF of the F-distribution is given by

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

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

Part II

Statistics

5

Sampling

CONTENTS

Statistics is very widely used in science, engineering and social science. From a broad view, any attempt trying to estimate an unknown value (for example, a parameter of a system, a property of the population) using its direct or indirect measurements (for example, outputs of the system, surveys conducted on randomly selected samples from the population) can be counted as an application of statistics.

Due to the measurement error, limitation of sample size, etc., we might not be able to get the accurate value of the unknown. Unavoidably, there is error between the estimates and the true value of the parameter. Statistics studies the methodologies to find the most probable estimate of the parameter by efficiently using the measurements and samples. It also quantitatively calculate the confidence level to the conclusions it draws.

Statistics starts from obtaining the measurements and samples, and using them to model the unknown parameter or population. Different sampling and modeling methods are introduced in this chapter.

5.1 Problem Formulation

Consider the following two problems.

1. There is an unknown parameter in the system. The parameter can be measured either directly or indirectly, subject to measurement noise. The problem is to estimate the most probable value or range of the parameter.

2. There is the population, and it is impossible to interview each element to get an aggregated feature of the population. However, it is possible to survey a small part of the population. The problem is to select samples from the population, conduct the survey, and using the survey results to

estimate the interested aggregated feature or a probable range for that feature.

These two problems seem to differ largely, but they are closely related in the mathematical insights, and can be formulated as the same (or very similar) problem.

Let “measurements” and “samples” be used interchangeably. Denote the samples as x_i, $i = 1, \ldots, m$, where m is the total number of samples. Denote the parameter to be estimated by θ. In both problems, the objective is to obtain an estimation of θ, $\hat{\theta}$, using the samples x_i.

5.2 Sampling Methods

Sampling methods can be a concern of Problem 2 introduced earlier in Section 5.1. When selecting samples from the population, it is critical to ensure that the selection is done randomly, and all elements in the population have a equal probability to be selected.

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

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

Sometimes it is interesting to compare the differences of the two methods, especially when the population is finite. An obvious difference is that by using sampling with replacement all the samples can be considered as “independent”: the previously selected samples would not affect the aggregated feature of the future samples. From this point of view, using sampling with replacement is equivalent of sampling from infinite population by thinking that the population is duplicated infinite times and each sample is done in an independent duplication.

While by using sampling without replacement, previous samples may change the distributions in the elements in the population, thus making the samples relevant.

When the population size is much larger than the sample size, the choice of the two sampling methods may not make any visible difference in the conclusion.

For further demonstration of the differences between sampling with and without replacement, consider the following example. A set of N random variables are generated from a Gaussian distribution as the population. Sample the population M times using sampling with/without replacement. Calculate the sampled mean and compare it with the mean of the entire population.

In the first example, let $N = 100$ and $M = 500$. Figures 5.1 and 5.2 gives

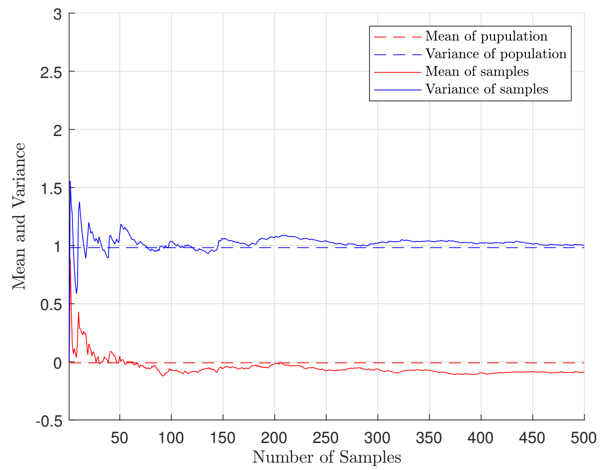

FIGURE 5.1
Sample with replacement, population size $N = 100$, sample size $0 < M \leq 500$.

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. 5.2, after number of samples exceeding 100, the entire population has been sampled, and thus the sampling stops. This explains why its mean and variance stop fluctuating and converge to the population mean and variance respectively.

In practice, however, the population size is often orders of magnitudes larger than the number of samples. In the second example, let $N = 10000$ and $M = 500$. The corresponding figures are given in Figs. 5.3 and 5.4. There is no obvious differences of the two figures.

5.3 Models of Population

The features of the population is often unknown, or at least not known precisely. It is common to make some preliminary assumptions to the distribution of the population using past experience.

For example, let X be a feature of the population. It could be, for example, the heights of all teenagers in a city. We can make an assumption that X follows Gaussian distribution with mean μ and standard deviation σ, and each

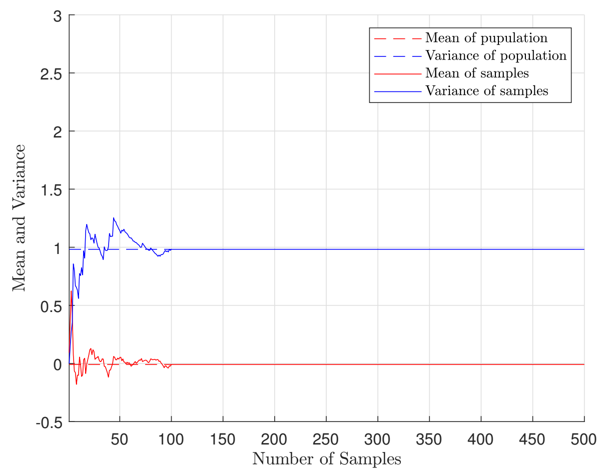

FIGURE 5.2
Sample without replacement, population size $N = 100$, sample size $0 < M \leq 500$.

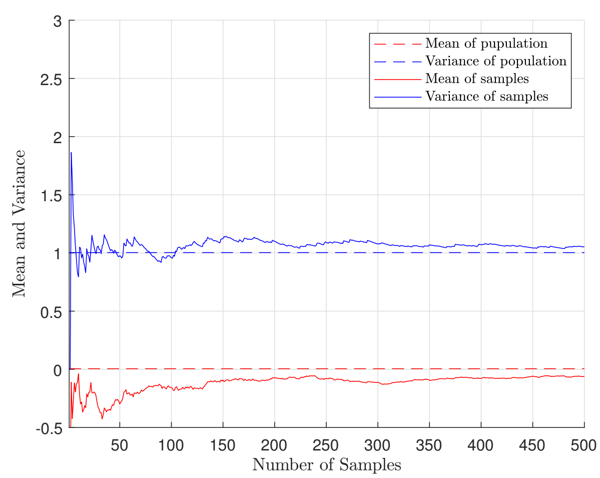

FIGURE 5.3
Sample with replacement, population size $N = 10000$, sample size $0 < M \leq 500$.

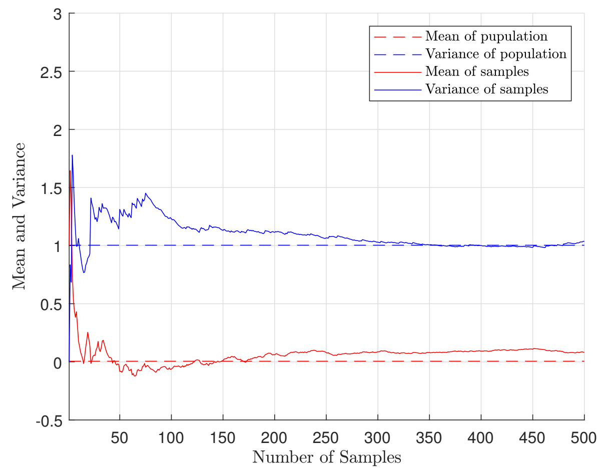

FIGURE 5.4
Sample without replacement,population size $N = 10000$, sample size $0 < M \leq 500$.

element in the population, X_i, can be taken as a random variable generated from $f(x)$.

The preliminary assumptions of the population can be helpful in using the samples to model the population more efficiently. Of course, if the samples deny the assumption in the hypothesis test, the assumptions can be rejected and new models can be proposed.

6

Estimation of Population Statistics

CONTENTS

The statistics of the population, such as its mean, variance, etc., can be estimated using the samples. For more insights, we can assume a distribution for the population, and consequently using the samples to estimate the parameters in the distribution. Commonly seen methods are introduced in this section.

6.1 Method of Moments

The statistics of the samples can in some extent reflect the system. The most intuitive assumption that one may make is that the samples share the same statistical features with the population. The larger size of the sample set, the more likely this statement is true. This statement can be proved correct for some of the statistical properties such as the mean.

6.1.1 Mean and Variance

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

$$\bar{X} = \frac{1}{m}\sum_{i=1}^{m} X_i \tag{6.1}$$

$$S^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(X_i - \bar{X}\right)^2 \tag{6.2}$$

Equations (6.1) and (6.2) are often used to estimate the mean and variance of the population, respectively. Note the denominator $m-1$ instead of m in (6.2). This is to ensure that there is no static bias in the estimation of the population variance using the sample variance when the samples are chosen without replacement (this is often the case when the population size is extremely large).

The proof that (6.2) is an unbiased estimation of the population variance is given as follows. Let the true mean and variance (not the sample mean and variance) of X_i be denoted by $E[X_i] = \mu$ and $\mathrm{Var}(X_i) = \sigma^2$ respectively. The objective is to prove $E\left[S^2\right] = \sigma^2$.

From (6.2),

$$\begin{aligned}
E\left[S^2\right] &= E\left[\frac{1}{m-1}\sum_{i=1}^{m}\left(X_i - \bar{X}\right)^2\right] \\
&= \frac{1}{m-1}E\left[\sum_{i=1}^{m}\left(X_i^2 - 2X_i\bar{X} + \bar{X}^2\right)\right] \qquad (6.3) \\
&= \frac{1}{m-1}E\left[\sum_{i=1}^{m}\left(X_i^2 - \bar{X}^2\right)\right] \\
&= \frac{1}{m-1}\sum_{i=1}^{m}E\left[X_i^2\right] - \frac{1}{m(m-1)}E\left[\left(\sum_{i=1}^{m}X_i\right)^2\right] \qquad (6.4) \\
&= \frac{1}{m-1}\sum_{i=1}^{m}\left(\mathrm{Var}(X_i) + E[X_i]^2\right) \\
&\quad - \frac{1}{m(m-1)}\left(\mathrm{Var}\left(\sum_{i=1}^{m}X_i\right) + E\left[\sum_{i=1}^{m}X_i\right]^2\right) \\
&= \frac{m}{m-1}\left(\sigma^2 + \mu^2\right) - \frac{1}{m(m-1)}\left(m\sigma^2 + m^2\mu^2\right) \\
&= \frac{1}{m-1}\left(m\sigma^2 + m\mu^2 - \sigma^2 - m\mu^2\right) \\
&= \sigma^2
\end{aligned}$$

where notice that in (6.3) $\sum_{i=1}^{m} X_i = \sum_{i=1}^{m} \bar{X}$ and in (6.4) $E\left[X^2\right] = \mathrm{Var}(X) + E[X]^2$ for any random variable X from (3.4).

This use of $m-1$ in (6.2) is essentially due to the fact that $\bar{X}$ is also obtained from the samples. Imagine a scenario where the mean of the population μ is known beforehand. In that case, we could have used

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

as the sample variance to estimate the population variance σ^2, which also gives a non-biased estimation.

6.1.2 Other Moments

Recall from Section 3.3 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 of moments are 4th order or lower. Notice that bias might be introduced when using central of moments and the central is obtained from the samples, just like illustrated earlier for the sample variance.

6.2 Maximum Likelihood and Maximum a Posteriori Estimations

Assume that the distribution of the population follows PDF $f(x;\theta)$ where θ is the collection of parameters in the distribution. For example, if $f(x)$ follows normal distribution, $\theta = [\mu, \sigma^2]$ is the collection of the mean and variance of the normal distribution. In the case where the population is discrete and finite, it is assumed that the population are populated from $f(x)$

The objective is to estimate θ given samples $X_1, \ldots, X_m$.

6.2.1 Maximum Likelihood Estimation

In MLE, we assume that there is no priori knowledge of the possible value of θ. The estimation is to "guess" θ that maximizes $P(x|\theta)$, hence the name, maximum likelihood.

There are many ways to solve an MLE problem. In the special cases where $f(x)$ 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

$$J = -\sum_{i=1}^{m} \ln f(X_i) \tag{6.5}$$

and the solution is given by

$$\hat{\theta} = \arg\min_{x} J$$

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

6.2.2 Maximum a Posteriori Estimation

Maximum a Posteriori (MAP) estimation is also known as the Bayesian estimation. Unlike MLE which maximizes $P(x|\theta)$ using (2.4) and (2.5). To summarize, MLP finds such θ that maximizes the following posteriori probability

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}$$

or in PDF form,

$$g(\theta) = \frac{f(x)f_\theta(\theta)}{f_X(x)} \tag{6.6}$$

where $f_\theta(\theta)$ and $f_X(x)$ is the ground truth distribution of θ and x, respectively. Notice that $f_X(x)$ is not affected by the guess of θ. Hence, maximizing (6.6) is essentially maximizing $f(x)f_\theta(\theta)$.

A general way of solving MAP is given below. Modify the cost function of MLE in (6.5) as follows to consider the priori knowledge $f_\theta(\theta)$

$$J = -\sum_{i=1}^{m} \ln f(X_i) - \ln f_\theta(\theta) \tag{6.7}$$

and (6.7) can be solved likewise.

6.2.3 Relation of MLE and MAP

MLE and MAP differ in the objective function. MLE maximizes $f(x)$ while MAP maximizes $f(x)f_\theta(\theta)$. The difference lies on the priori knowledge of $f_\theta(\theta)$, i.e., how θ is distributed without taking into account the samples. MAP requires the priori knowledge of $f_\theta(\theta)$ while MLE does not.

When $f_\theta(\theta)$ is unknown, MLE is the preferred choice. MLE can also be taken as a special case of MAP where $f_\theta(\theta)$ is constant, i.e., in the priori knowledge, θ is distributed evenly for all its possible values. When $f_\theta(\theta)$ is known, MAP can be used.

In some literatures, $f_\theta(\theta)$ is considered as part of the MLE cost function as given by (6.7). Though it is called an MLE in those literatures, it is effectively an MAP estimator.

6.3 Comparison of Different Estimation Methods

With the same set of samples, different methods (methods of moments, MLE and MAP with different likelihood functions, etc.) will give different results. There is no universally best way to estimate the statistical features of the population. It is important to choose the appropriate estimation method case by case depending on the problem.

With that been said, there are benchmarks to evaluate the performance of an estimation method. One of them is bias, which has been briefly mentioned in the previous section. Other benchmarks include efficiency, robustness, etc. Commonly seen ones are introduced in this section.

6.3.1 Unbiased Estimation

Recall that we use samples to estimate the statistics parameters of the population. When distributions are used to model the population, the samples are used to estimate the parameters in the distributions. We have been taken it as granted that:

- The estimation can have error;
- The expectation of the estimation error should be zero.

The above implies that if we repeat the estimation many times (each time using new samples) and aggregate the results, the positive and negative error would cancel out, eventually giving us the estimates very close to the true parameters values. So long as we have enough samples, the error can be reduced to be as small as we need. When the sample size approaches infinity, the estimation error should approach zero.

When the expectation of the estimation error is zero, the estimation is said to be *unbiased*. This is often considered a good feature of the estimation. However, this is not always the case. It is difficult to make the estimation unbiased in reality. For those estimators claimed to be unbiased, rigorous mathematical proof is often required.

There are several factors that may contribute to the bias:

- Samples are not selected randomly. This is often not as much of a concern in science and engineering as compared with social science.
- Measurement noise is skewed. We often assume normal distribution for the measurement noise of sensors, which is non-skewed. However, normal distribution is only an approximation to the reality. The actual measurement noise may be skewed and not zero-mean.
- The estimator is biased. Depending on the configurations of an estimator such as the noise assumption in the MLE, the estimator itself can be biased.

6.3.2 Mean Squared Error

Consider unbiased estimation. The magnitude of the estimation error can be evaluated using MSE and 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.

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

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

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

for unbiased estimator 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, $\text{Var}(\hat{\theta})$, interchangeably with MSE as an evaluation of the performance of the estimation. In most literatures, $\text{Var}(\hat{\theta})$ is used, implying that the result is theoretical and derived under the assumption of "infinite runs".

The smaller $\text{Var}(\hat{\theta})$, the more precise the estimation. There is a limitation to how small $\text{Var}(\hat{\theta})$ 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.

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 $\text{Var}(\hat{\theta})$ and Fisher Information Matrix $I(\theta)$ for a scalar unbiased estimator.

$$\text{Var}(\hat{\theta}) \geq I(\theta)^{-1} \tag{6.8}$$

which is known as the *Cramér–Rao lower bound* (CRLB) of the estimator. From (6.8), 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 is said to be *optimal*.

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.

6.4 Interval Estimation

The estimation obtained from empirical data may bias from its 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 is corresponding with a probability.

The purpose of interval estimation is to obtain the probability of a CI or find the CI for a specific probability given the samples.

6.4.1 Estimate the Mean of a Normal Distribution

As an example, consider estimating the CI of a normal distribution using samples taken from that distribution. Notice that this is only one of the many scenarios of interval estimation.

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. Let μ^*, ${\sigma^2}^*$ be the mean and variance estimate of the distribution respectively. The objective is to calculate the CI of μ using $\bar{X}$, ${\sigma^2}^*$ and m.

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

$$\begin{aligned}\mathrm{Var}[\mu - \mu^*] &= E\left[\left(\mu - \frac{1}{m}\sum_{i=1}^{m} X_i\right)^2\right] \\ &= \frac{1}{m^2}E\left[\sum_{i=1}^{m}(\mu - X_i)^2\right] &(6.9) \\ &= \frac{1}{m}\sigma^2 &(6.10)\end{aligned}$$

Though (6.10) 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, (6.10) needs to be reformulated to include μ^* and σ^{2*}.

Denote $\tilde{\mu} = \mu - \mu^*$. Note that

$$\mathrm{Var}[\mu - \mu^*] = E\left[(\tilde{\mu} - E[\tilde{\mu}])^2\right] = E[\tilde{\mu}^2]$$

because $E[\tilde{\mu}] = E[\mu] - E[\mu^*] = 0$. Rewrite (6.9) as follows.

$$\begin{aligned}\mathrm{Var}[\mu - \mu^*] &= E[\tilde{\mu}^2] \\ &= \frac{1}{m^2}E\left[\sum_{i=1}^{m}(\mu^* + \tilde{\mu} - X_i)^2\right] \\ &= \frac{1}{m^2}E\left[\sum_{i=1}^{m}(\mu^* - X_i)^2\right] + \frac{2}{m^2}E\left[\sum_{i=1}^{m}(\mu^* - X_i)\,\tilde{\mu}\right] + \frac{1}{m^2}E\left[\sum_{i=1}^{m}\tilde{\mu}^2\right] \\ &= \frac{m-1}{m^2}\sigma^{2*} + \frac{1}{m}E[\tilde{\mu}^2] &(6.11)\end{aligned}$$

Solving (6.11) for $E[\tilde{\mu}^2]$ gives

$$\mathrm{Var}[\mu - \mu^*] = \frac{1}{m}\sigma^{2*} \tag{6.12}$$

Equations (6.10) and (6.12) look similar except that σ^2 is replaced by σ^{2*}. Notice that (6.12) gives the variance. For the standard deviation,

$$\mathrm{Std}(\mu - \mu^*) = \frac{1}{\sqrt{m}}\sqrt{\sigma^{2*}} = \frac{1}{\sqrt{m}}\sigma^* \tag{6.13}$$

where σ^* is the estimate of the standard deviation using the samples. Given the confidence, CI can be determined using (6.13) as

$$\left[\mu^* - \frac{u_\alpha}{\sqrt{m}}\sigma^*, \mu^* + \frac{u_\alpha}{\sqrt{m}}\sigma^*\right] \tag{6.14}$$

where u_α is a gain determined by the desired confidence, i.e., $P\left(\hat{\theta}_{\min} \leq \theta \leq \hat{\theta}_{\max}\right)$.

The gain u_α can be derived from the CDF of the noise assumption. The higher P, the larger u_α. 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 general, u_α can be derived from the CDF of the normal distribution as follows.

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

or equivalently

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

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

6.4.2 General Interval Estimation

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.

7

Statistical Hypothesis Testing

CONTENTS

A statistical hypothesis test is a method of statistical inference used to decide whether the empirical data sufficiently support a particular hypothesis.

7.1 Problem Formulation

Simple motivating examples are given to illustrate the hypothesis testing problem.

7.1.1 Motivating Examples

Consider tossing a coin. The result follows Bernoulli distribution with the chance of head p and tail $1 - p$. For a normally designed coin, 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 $p = 0.5$ 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: event 1 can happen quite often, event 2 very rare, and event 3 seemingly never happen in practice. On the other hand, if event 3 indeed happens, 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.

Consider another example of estimating the average monthly income of the population in a city. The purpose is to evaluate whether the average income is at least $3,000. It is too costly to interview all the $n = 1000000$ citizens in the city. Therefore, only 1% of the population, $m = 10000$ people, are randomly selected and interviewed.

In this example, the hypothesis is $H_0 : \bar{X} \geq 3000$ where $\bar{X}$ is the mean of income of the entire population. Its alternative hypothesis is $H_1 : \bar{X} < 3000$. 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 such as variance. An example is given below. In the first coin tossing example, the coin manufacture has claimed that the probability of head p follows a normal distribution of $\mathcal{N}(0.5, 0.01^2)$. We can use hypothesis testing to verify whether $H_0 : \sigma^2 = 0.01^2$ or $H_1 : \sigma^2 > 0.01^2$.

7.1.2 Null Hypothesis and Alternative Hypothesis

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 its name 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. From this perspective, 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.

7.1.3 Simple Hypothesis and Compound Hypothesis

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

In the earlier motivating examples, " when tossing a the coin the probability to get head $p = 0.5$" is a simple hypothesis, while "average income for people in the city $\bar{X} \geq 3000$" a compound hypothesis.

7.1.4 Acceptance Region and Rejection Region

It is very likely that among the populations, some samples fulfill the hypothesis while others are not. The result of a hypothesis test, therefore, is random and depends on the set samples that happen to be chosen.

Let $X_1, \ldots, X_m$ be the m samples taken in the trail. Define

$$A = \{(x_1, \ldots, X_m) | H_0 \text{ is accepted}\} \tag{7.1}$$

$$R = \{(x_1, \ldots, X_m) | H_0 \text{ is rejected}\} \tag{7.2}$$

as the *acceptance* and *rejection regions* respectively. Notice that the rejection region is also known as the critical region. In practice, "H_0 is accepted", "H_0 is rejected" must be made more specific in a mathematical manner. For example, in the hypothesis test of the mean value of the population, (7.1) and (7.2) become

$$\begin{aligned} A &= \left\{(x_1, \ldots, X_m) \middle| \bar{X} \geq C\right\} \\ R &= \left\{(x_1, \ldots, X_m) \middle| \bar{X} < C\right\} \end{aligned}$$

where C is a threshold set for the hypothesis testing. Obviously, the value of C determines the cardinality of A and R, hence the chance of the hypothesis being rejected. For example, if X_i represents the income of a person and the null hypothesis proposes the minimum average income of the population, the lower C the more likely that the null hypothesis would be accepted.

7.1.5 Power Function

The probability of the null hypothesis being rejected relates to the acceptance and rejection region. The *power function* is used to describe the probability that the samples reject the null hypothesis H_0 as denoted below.

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

where θ is the true value of the parameter under testing, and Φ the hypothesis testing method.

When θ is such that H_0 is true, we would like $\beta_\Phi(\theta)$ to be small. On the contrary, when θ is such that H_0 is false, we would like $\beta_\Phi(\theta)$ to be large. 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 H_0 is true. In this case, the power function describes the probability of the hypothesis test making a *Type I error*: reject a true null hypothesis. The probability is described by

$$\alpha_{1\Phi}(\theta) = \begin{cases} \beta_\Phi(\theta) & H_0 \text{ is true} \\ 0 & H_0 \text{ is false} \end{cases} \tag{7.3}$$

On the other hand, when H_0 is false, the power function describes (the compensation of) the probability of the hypothesis test making a *Type II error*: fail to reject a false null hypothesis. The probability is described by

$$\alpha_{2\Phi}(\theta) = \begin{cases} 0 & H_0 \text{ is true} \\ 1 - \beta_\Phi(\theta) & H_0 \text{ is false} \end{cases} \tag{7.4}$$

Ideally, a good hypothesis test shall ensure that both (7.3) and (7.4) are small. However, notice that there is a trade-off. An very small (7.3), for example by acting extremely conservative and not rejecting any hypothesis regardless of the samples, may lead to a very large (7.4), and vise versa.

In a common practice, we would define a *significance level* α for (7.3). The target to minimize (7.4) while keeping (7.3) 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 $\alpha_{1\Phi}(\theta) \leq \alpha$ when H_0 is true, their $\alpha_{2\Phi}(\theta)$ when H_0 is false decides which one is efficient. The lower $\alpha_{2\Phi}(\theta)$ (or equivalently the higher the power function $\beta_\Phi(\theta)$) when H_0 is false, the better.

Among all the hypothesis testing methods that meet $\alpha_{1\Phi}(\theta) \leq \alpha$, if there is a hypothesis testing method Φ that gives the largest $\beta_\Phi(\theta)$ for any parameter θ that denies H_0, 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 require a lot of statistics expertise.

7.2 Commonly Seen Hypothesis Tests

Hypothesis test to verify the mean of a normal distribution has been briefly discussed in the motivating example in earlier sections. In this section, more commonly seen hypothesis test methods are introduced.

The most commonly seen hypothesis test is the comparison of the mean of a normal distribution with a constant. For this problem, the results and derived in details. For the rest hypothesis test problems, only the problem formulation is introduced.

7.2.1 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 σ^{*2} given by (6.2) wherever the variance is used. For simplicity, only σ^2 is used consistently in the remaining of this discussion.

Two null hypothesis are considered:

1. H_0: $\mu \geq \mu_0$ ($\mu \leq \mu_0$); H_1: $\mu < \mu_0$ ($\mu > \mu_0$)
2. H_0: $\mu = \mu_0$; H_1: $\mu \neq \mu_0$

Consider scenario 1 where 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 $\mu^* = \frac{1}{m}\sum_{i=1}^m X_i$, and let

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \mu^* \geq C \\ \text{reject } H_0 & \mu^* < C \end{cases} \tag{7.5}$$

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

Recall from (6.10) and (6.12) that given the samples mean μ^* and the variance σ^2, the true value of μ follows a normal distribution centered at μ^* and with standard deviation $\frac{1}{\sqrt{m}}\sqrt{\sigma^2}$. The distribution of μ is given by the red line in Fig. 7.1. Notice that Fig. 7.1 is shifted by μ^* in the horizontal axis so that it is centered at zero.

Draw a vertical bar at $\mu - \mu^* = C'$ in Fig. 7.1 so that the area of the red zone equals to α and green zone $1 - \alpha$. From (6.14),

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

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

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \mu_0 - \mu^* \leq C' \\ \text{reject } H_0 & \mu_0 - \mu^* > C' \end{cases} \tag{7.6}$$

The hypothesis test (7.6) 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. 7.1. 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

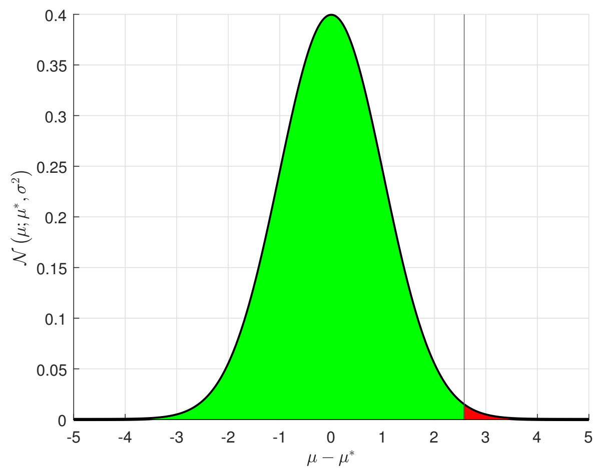

FIGURE 7.1
Distribution of μ as a function of μ^* and σ^2.

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 (7.5) and (7.6) gives

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

and (7.5) becomes

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \mu^* \geq \mu_0 - \frac{u_\alpha}{\sqrt{m}}\sigma \\ \text{reject } H_0 & \mu^* < \mu_0 - \frac{u_\alpha}{\sqrt{m}}\sigma \end{cases}$$

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

Consider scenario 2 where 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. 7.1 at both tails. The hypothesis test is given by

$$\Phi : \begin{cases} \text{fail to reject } H_0 & \mu_0 - \frac{u_\alpha}{\sqrt{m}}\sigma \leq \mu^* \leq \mu_0 + \frac{u_\alpha}{\sqrt{m}}\sigma \\ \text{reject } H_0 & \text{otherwise} \end{cases}$$

where notice that

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

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.

7.2.2 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 null hypothesis

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

where σ_0^2 is a given constant.

7.2.3 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: $\mu_x - \mu_y < \mu^*$
2. H_0: $\mu_x = \mu_y$; H_1: $\mu_x \neq \mu_y$

where μ_0 is a given constant.

7.2.4 Exponential Distribution

As introduced earlier in Section 4.4, 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 (4.2). In practice, it can be m sample products' lifespan. The expectation of (4.2) is $\dfrac{1}{\lambda}$. 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.

7.2.5 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.

7.2.6 Poisson Distribution

8

Bayesian Methods

CONTENTS

Part III

Tools

9

R (Part I: Basics)

CONTENTS

This chapter and the sequential chapters introduce R language, a widely used programming language for data mining and visualization.

This chapter focuses on the fundamental setup and basic operations such as the installation of R and RStudio, basic data types, vector and data frame manipulation, and finally basic visualization methods. Chapter 10 introduces advanced data analysis skills and tools. Finally, Chapter 11 put things into practice to solve actual problems.

9.1 R and RStudio Installation

R is a programming language for statistical computing and visualization. It is widely used among statisticians and data miners for developing statistical software and carrying out data analysis. R is free and can be downloaded from [4], from which more details about R can also be found.

RStudio, also known as *Posit*, is an IDE widely used for R programming and testing. RStudio IDE is open-source and free of charge for personal use. It can be downloaded from [5].

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

9.2 R Packages Management

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

9.2.1 Manage Packages with Built-in Functions

There are a number of default packages that come along with R and they do not need separate installation. Only third-party packages need additional installation. To install or remove a package, use

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

respectively. For example,

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

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. Alternatively, use prefix `<package>::<function>`, `<package>::<dataframe>` to call the functions and data frames defined in a package without loading it. The later approach can become inconvenient when the function or data frame is used frequently.

To unload a package, use

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

For example,

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

9.2.2 Manage Packages with Third-Party Packages

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 them is given as comments in the above demo.

9.2.3 Manage Packages with RStudio IDE

RStudio provides a graphical interface to manage packages as shown in Fig. 9.1. A package can be loaded or unloaded simply by checking and unchecking the package.

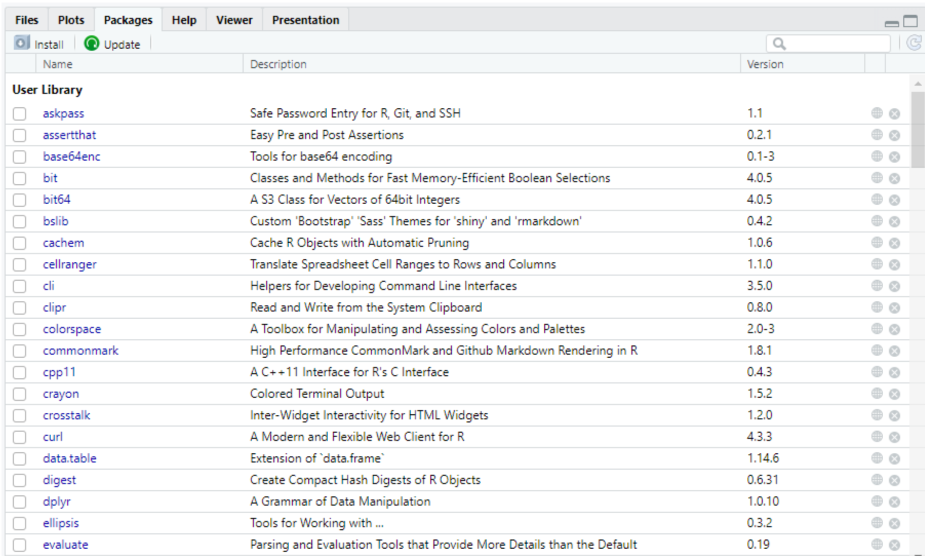

FIGURE 9.1
RStudio's graphical interface for package management.

9.3 Basic Syntax

This section introduces the basic syntax and commonly used data types of R programming.

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]
```

TABLE 9.1
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`	An double float value.
complex	`z <- 3+2i`	A complex value.
character	`a <- "a"`	A character or a string.
logical	`q <- T`	A boolean value. Use `T`, `TRUE` and `F`, `FALSE` to represent true and false respectively.

```
# 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)
```

9.3.1 Data Types

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

Examples of assigning variables and checking their types are given as follows. Notice that `>` is the prompt indicating that the commands are executed in a console.

```
> 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
```

TABLE 9.2
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.

```
> 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 9.2, 9.3, 9.4, 9.5 and 9.6.

9.3.2 Conditionals and Loops

The `if` statement syntax is given as follows.

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

TABLE 9.3
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 9.4
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. The index starts from 1 instead of 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 9.5
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 9.6
Statistics related functions.

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

```
        <command>
} else{
        <command>
}
```

An example of using `if` statement is givne below.

```
> x <- rnorm(1)
> if(x > 0){
        +   y <- x
        + } else if(x < 0){
        +   y <- -x
        + } else{
        +   y <- 0
        + }
> print(x)
[1] -1.981445
> print(y)
[1] 1.981445
```

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
```

9.3.3 User-Defined Functions

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>)]
}
```

9.4 Vector and Matrix

Vector and matrix are the fundamental forms using which R stores and processes data.

9.4.1 Vector

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 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"
```

It is possible to assign names to each element as shown 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
```

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>]` or `<vector>["<name>"]`. Notice that the first element in a vector has the index of 1 instead of 0. The index can be an integer, or an integer vector. 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"
```

It is also possible to address an element by its name

```
> 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
```

Vectorization operation significantly speeds up the calculation comparing with looping over elements, hence is quite commonly seen in high-layer languages such as R and Python. Most numerical calculations, including `+`, `-`, `*`, `/`, `%/%`, `%%`, `^`, support vectorization operation. 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.

9.4.2 Matrix

Similar with vector, all elements in a matrix must have the consistent type. A demonstration of matrix indexing in R is shown in Fig. 9.2. Let the matrix be named `A`. The elements in the matrix can be accessed by the name of the table followed by the index coordinates. For example, in the figure, `A[1,1]` refers 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.

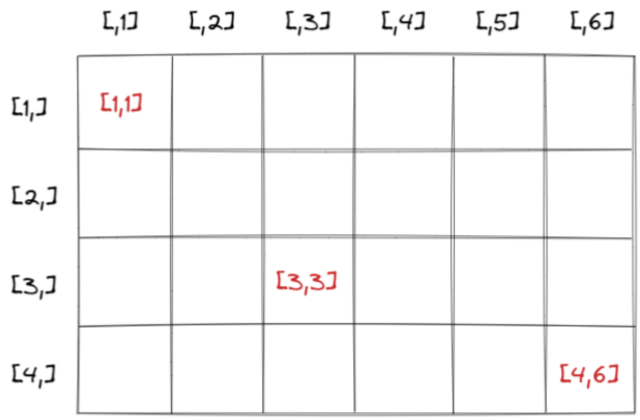

FIGURE 9.2
A demonstration of a matrix in R.

A matrix can be created from scratch by stacking rows as follows. First, create rows in the matrix. Then, use `rbind()` to bind rows.

```
# 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
```

As illustrated earlier that `names(<vector>)` can be used to view and assign names to elements in a vector, likewise `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.

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

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

To get consistent results, when segmenting matrix to get a single row or column vector, deliberately ask R to not drop the matrix dimensions. This can be done as follows. By doing this, the names assigned to columns and rows preserve.

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

9.4.3 Matrix Visualization Using `matplot()`

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

To demonstrate `matplot()`, consider the following example.

```
professor <- c(1130, 1026, 893, 922, 776)
student <- c(2, 14, 24, 49, 46)
citation <- rbind(professor, student)
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. 9.3.

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

```
# generate table
```

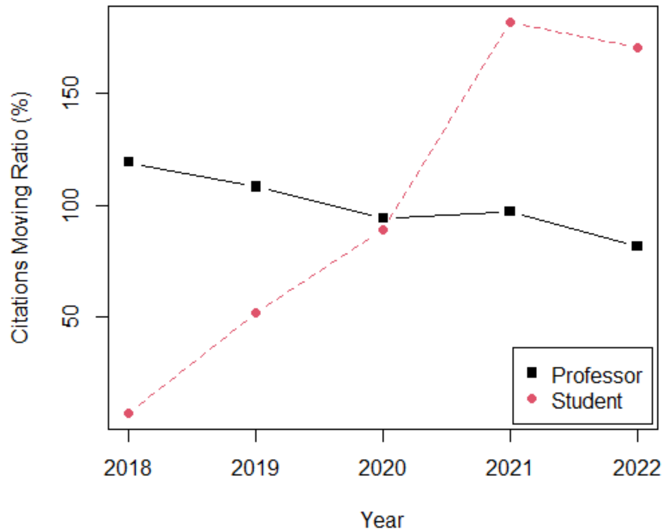

FIGURE 9.3
A demonstration of using `matplot` to plot trends.

```
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
```

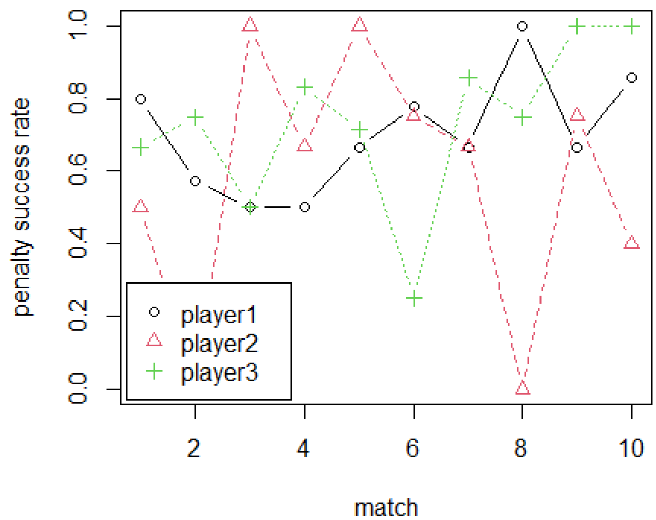

FIGURE 9.4
Plot of penalty success rate of the 3 players in 10 matches.

```
        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. 9.4 and 9.5.

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

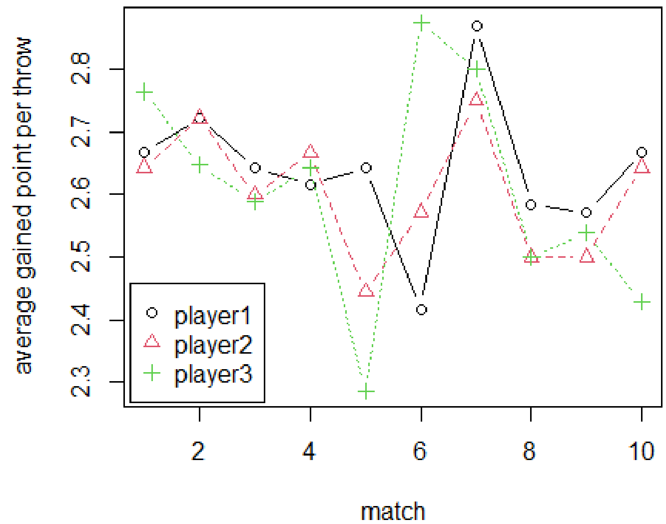

FIGURE 9.5
Plot of average point gained per throw attempt for the 3 players in 10 matches.

9.5 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.

9.5.1 Import Data into Data Frame

Data frame can be created from scratch by concatenating columns just like matrix. 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 and data concatenating will be introduced in a later section.

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

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

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

The following commands import a specified CSV file.

TABLE 9.7
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.

```
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.

9.5.2 Access Data in Data Frame

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 data frame do not have names.

Table 9.7 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
```

9.5.3 Filter Data from Data Frame

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

An example is given below.

```
> library(datasets)
```

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

9.5.4 Create Data Frames

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.

9.6 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. 9.6. 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 9.7 and 9.8 can be obtained.

```
library(ggplot2)
rm(list=ls())
# create data frame
vec_region <- rep(c("CBD","City","Suburbs"),each = 100)
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)
```

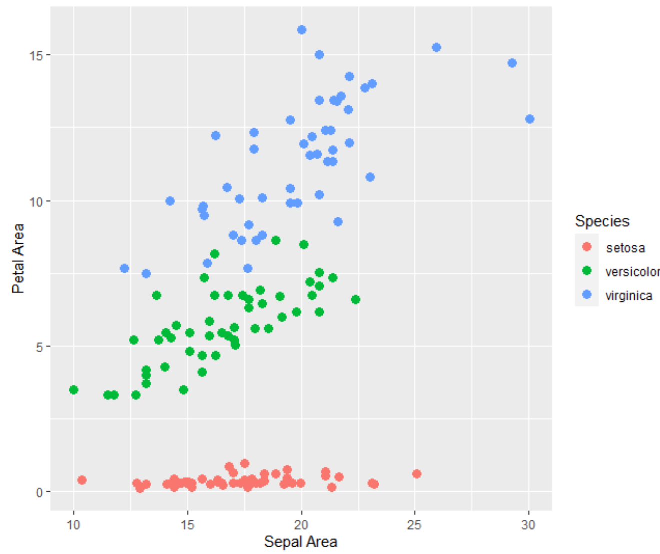

FIGURE 9.6
A demonstration of `qplot`.

```
# 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")
```

9.7 Advanced Data Visualizations Using `ggplot()`

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

9.7.1 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.

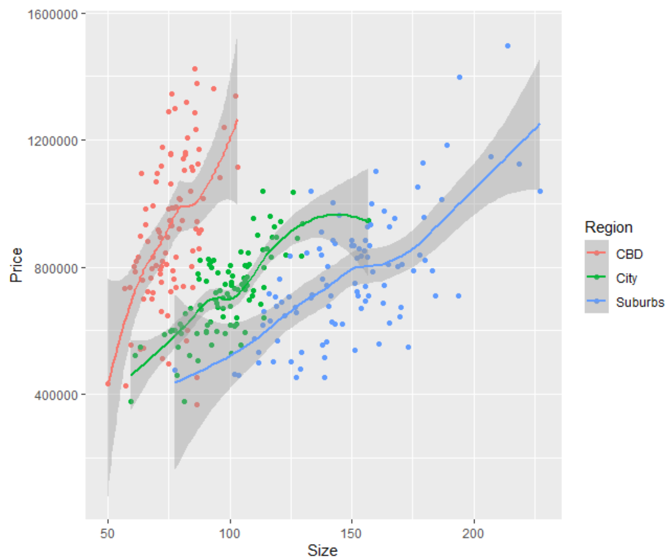

FIGURE 9.7
A demonstration of `qplot` on mortgage price data frame.

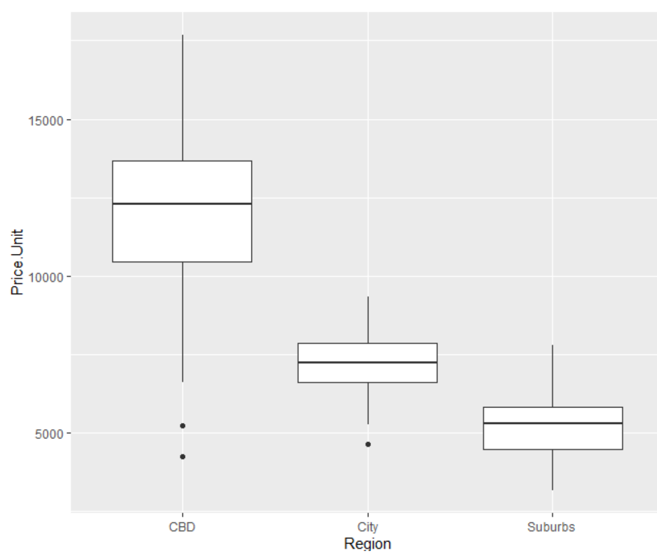

FIGURE 9.8
A second demonstration of `qplot` on mortgage price data frame.

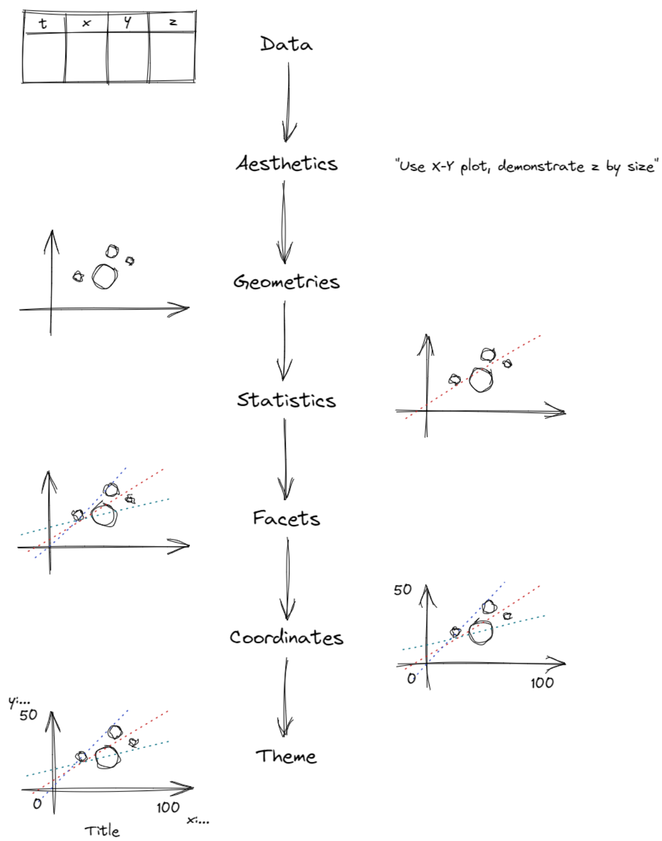

FIGURE 9.9
Multiple layers in chart design.

- 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. 9.9 is given to illustrate the different layers in a chart.

TABLE 9.8
Commonly used commands for data frame exploration.

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

9.7.2 Data, Aesthetics and Geometries Layers

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

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

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

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

```
p <- ggplot(data=<df>, aes(x=<column>, y=<column>, color=<column>, size
    =<column>))
p + geom_point()
p + geom_line() # another selection of geometries choice
p + geom_line() + geom_point() # overlying both geometries choices
p + geom_point(aes(size=<another-column>)) # overwrite aesthetics
    settings.
```

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

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

9.7.3 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 9.8 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
# plot
p <- ggplot(data=mortgage_price, aes(x=Region, y=Price.Unit, color=
    Region)) + ggtitle("Mortgage uprice per area VS region") + xlab("
    Region") + ylab("Price per area")
p + geom_boxplot() + geom_jitter(aes(size=Size, color=Region), alpha
    =0.25)
```

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

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

```
library(ggplot2)
# generate data
t <- 1:500
var1 <- 1.5*t + rnorm(500, 0, 100)
var2 <- 0.5*t + rnorm(500, 200, 10) + t^1.3*rnorm(500, 0, 0.1)
df <- data.frame(t=t, x=var1, y=var2)
# plot data
p <- ggplot(data=df) +
ggtitle("Plot of x and y VS t.") +
```

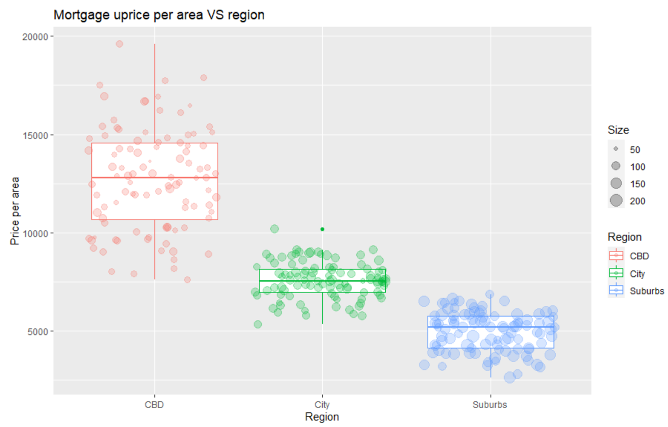

FIGURE 9.10
An example of box plot of the mortgage price data frame using `ggplot()` and `geom_boxplot()`.

```
xlab("t") +
ylab("x and y") +
geom_point(aes(x=t, y=x), color="blue", shape=1, size=1.5) +
geom_smooth(aes(x=t, y=x), color="blue") +
geom_point(aes(x=t, y=y), color="red", shape=2, size=1.5) +
geom_smooth(aes(x=t, y=y), color="red")
p
```

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

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

9.7.4 Facets Layers

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

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

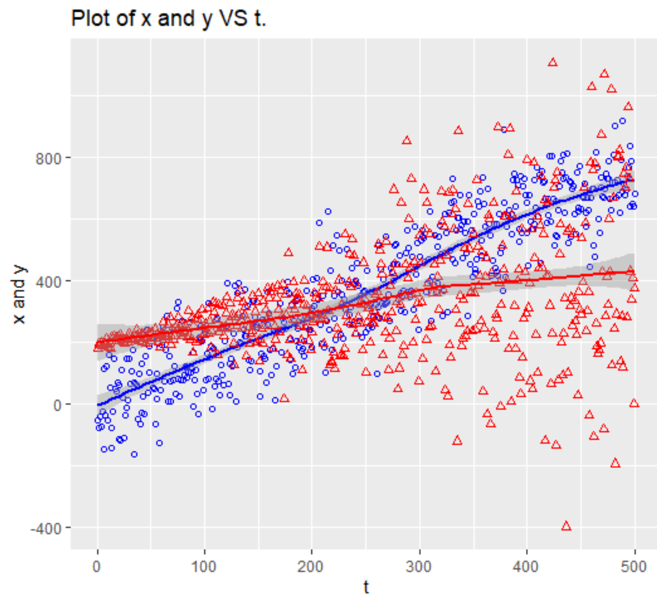

FIGURE 9.11
An example of using `geom_smooth()` for scatter point fitting.

TABLE 9.9
Functions that fit smooth lines to scatter points.

Function	Description
`loess()`	Non-parametric method for fitting a smooth line to a 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.

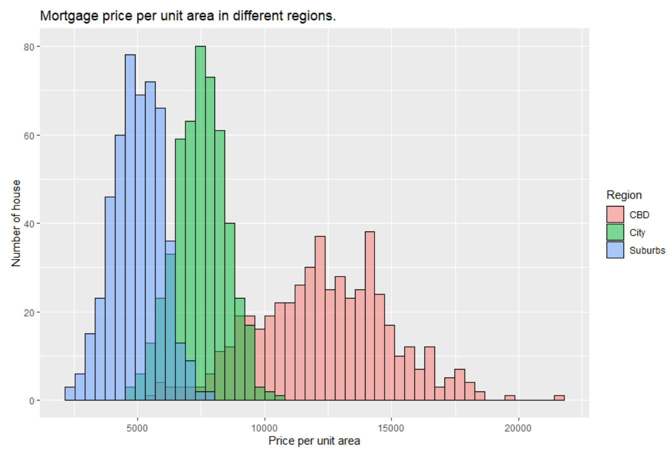

FIGURE 9.12
An example of histogram plot of house price per unit area in different regions in a single plot.

```
               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
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))
```

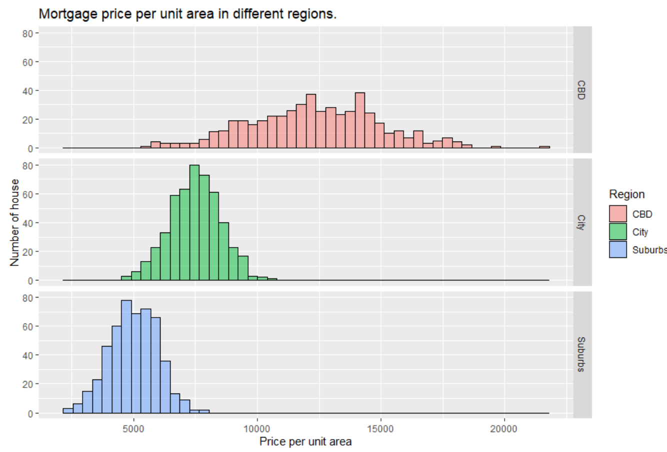

FIGURE 9.13
Use facets to plot the histogram of price per unit are of the house in different regions (subplots in rows).

```
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
```

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

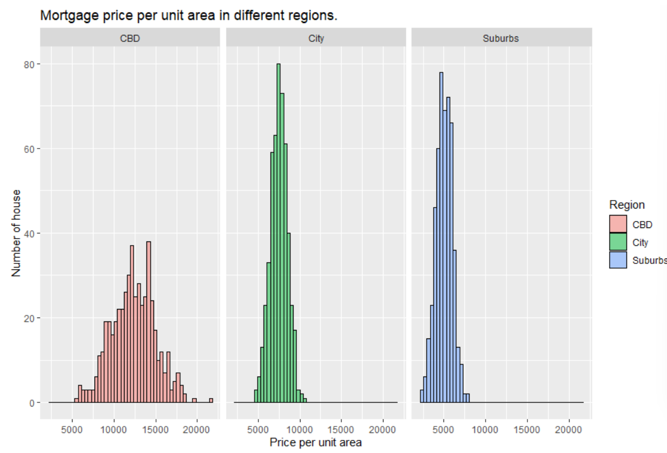

FIGURE 9.14
Use facets to plot the histogram of price per unit are of the house in different regions (subplots in columns).

9.7.5 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
# 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
```

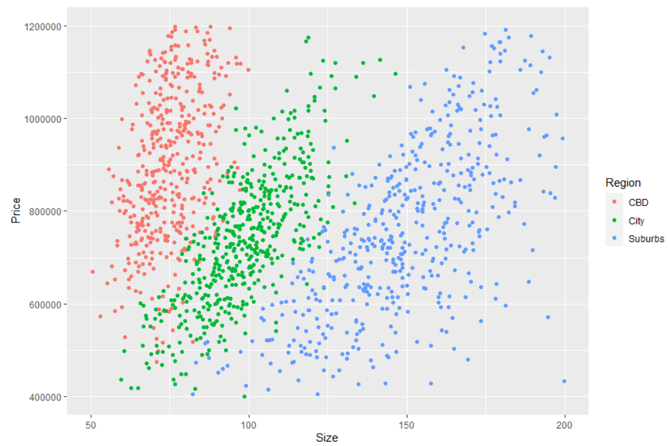

FIGURE 9.15
Add coordinates layer using `xlim()` and `ylim()`.

```
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. 9.15 and 9.16, respectively. The difference can be observed near the boundary.

9.7.6 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))
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))
```

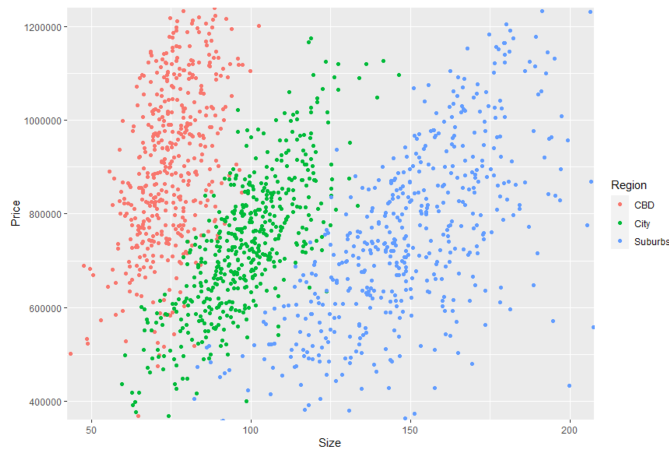

FIGURE 9.16
Add coordinates layer using `coord_cartesian()`.

```
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. 9.17. Compare it with Fig. 9.12 to see the differences by applying `theme()` in the themes layer.

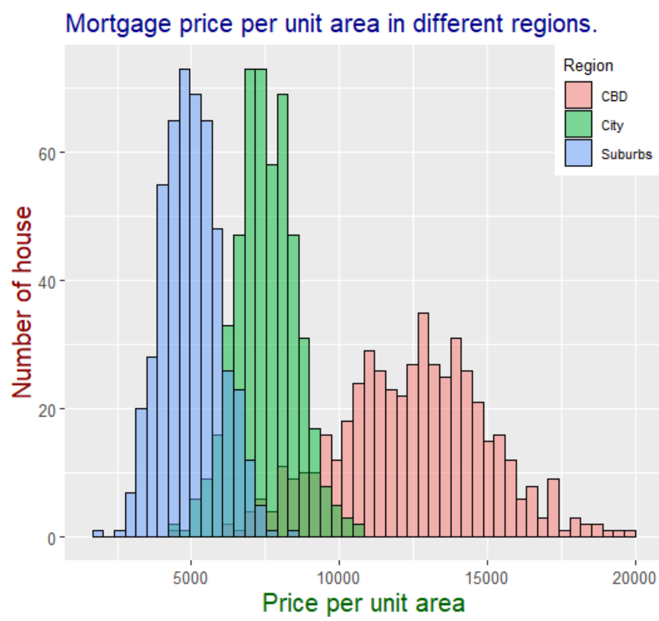

FIGURE 9.17
Mortgage price chart with theme.

10

R (Part II: Advanced)

CONTENTS

This chapter introduces advanced skills of using R language, including data preparation techniques, the use of list, etc.

10.1 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.

10.1.1 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.

10.1.2 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 intension 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.

10.2 Connectivity with Data Sources

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

11

R (Part III: Practice)

CONTENTS

12

Python (Part I: Basics)

CONTENTS

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

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

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

12.1 NumPy

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

Details of NumPy can be found at *numpy.org*.

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

```
import numpy as np

# create numpy array from python list
x = np.array([1, 2, 3, 4, 5]) # 1d
x = np.array([[1, 2], [3, 4]]) #2d

# create numpy array/matrix using built-in functions
x = np.arange(0, 5) # array([0, 1, 2, 3, 4])
x = np.linspace(0, 5, 3) # array([0, 2.5, 5])
x = np.zeros(5) # 1d zero vector
x = np.zeros([5, 5]) # 2d zero matrix
x = np.ones(5)
x = np.ones([5, 5])
x = np.eye(5)

# create random vector/matrix
np.random.seed(1) # set seed; optional
x = np.random.rand(5, 5) # uniform distribution [0, 1)
x = np.random.randn(5, 5) # standard normal distribution N(0, 1)

# reshape
x = np.random.randn(16)
y = x.reshape(4, 4)
y = x.reshape(1, 16) # return is always 2d matrix format, not 1d vector
     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.

12.2 SciPy

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

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

12.3 Matplotlib and Seaborn

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

12.3.1 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. 12.1.

```
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)")
```

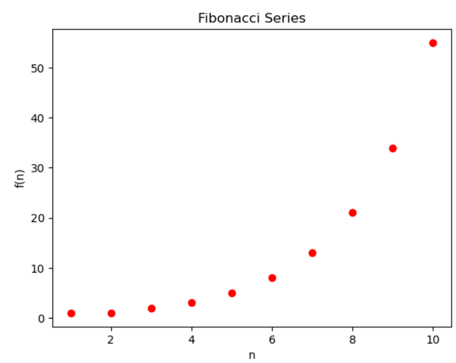

FIGURE 12.1
Plot Fibonacci series as scatter plot.

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.

12.3.2 Seaborn

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

An example of using Seaborn to plot a histogram is given below. The result is given in Fig. 12.2.

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

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

An example of using Seaborn to plot a count plot for discrete values (usually category values) is given below. Notice that the attribute whose values are put into categories is assigned to x of `seaborn.countplot()`. The result is given in Fig. 12.3.

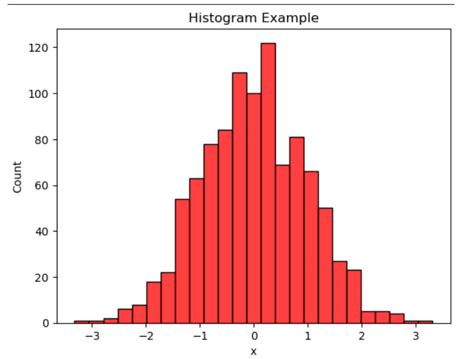

FIGURE 12.2
Histogram plot using Seaborn.

```
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")
```

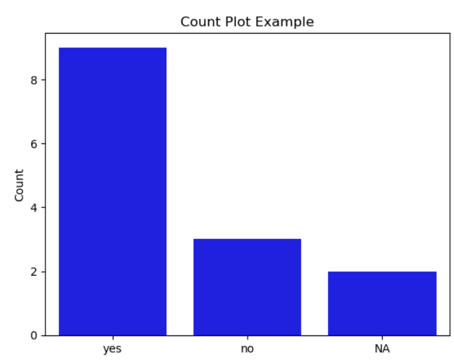

FIGURE 12.3
Count plot using Seaborn.

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

(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. 12.4.

```
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")
```

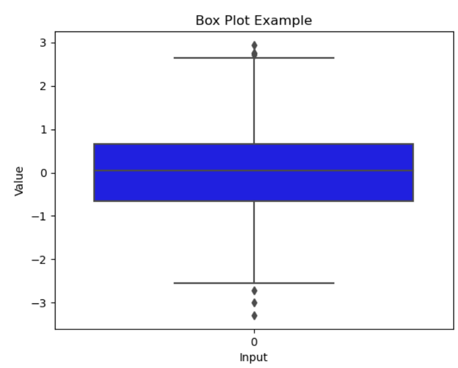

FIGURE 12.4
Box plot using Seaborn. The box gives IQR. The bars below and above the box give $Q_1 - 1.5 \times \text{IQR}$ and $Q_3 + 1.5 \times \text{IQR}$, respectively. The dots are outliers.

There are many more plot functions in Seaborn. For example, scatter plot `seaborn.scatterplot()` works similar with Matplotlib as shown by Fig. 12.1. The marker size and color can be adjusted to reflect different parameters, just like in R language. Pairplot `seaborn.pairplot()` generates a matrix of plots to demonstrate associations of columns in a data frame.

12.4 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__))
```

12.4.1 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. 12.5. 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. 12.5 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. 12.6.

```
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 12.5
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 12.6
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.

12.4.2 Series and Data Frame

13

Python (Part II: Advanced)

CONTENTS

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

There are many ANN engines for Python. Among all, TensorFlow and PyTorch are very popular and powerful generic-purpose ANN engines. They both cover a large range of supervised, reinforcement and unsupervised learning applications including classification, regression, pattern recognition, computer vision, natural language processing, clustering, abnormality detection, and many more. They both offer variety of tools to quickly and flexibly design and deploy different types of AI models such as conventional dense networks, CNN models, RNN models, and many more. Both of them can be used to train, evaluate and run networks. Both of them provide server solutions, cloud solutions and edge computing solutions. TensorFlow and PyTorch are introduced in this chapter.

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

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

13.1 Quick Review

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

13.1.1 AI Pipeline

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

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

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

13.1.2 Data Preparation and Model Evaluation

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

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

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

- Data preprocessing:
  - Data cleaning. This is a critical step that greatly influences the model's performance. It includes cleaning the data, handling missing values, and dealing with outliers. Often, domain knowledge plays a significant role in these steps. In practice, some of the above procedures are done using AI engine built-in functions, while others are done using other packages such as pandas.
  - Feature engineering. This involves creating new features from the existing data that might help improve the model's performance. Feature selection is also crucial to reduce overfitting and improve the model's interpretability. Again, domain knowledge can be very beneficial here.
- Model design, evaluation and tuning:
  - Model selection. Choosing the right model or architecture for the problem at hand requires a solid understanding of the strengths and weaknesses of different models.
  - Hyperparameter tuning. While there are systematic approaches like grid search or random search, often, practical experience and intuition play a significant role in choosing the right hyperparameters.
  - Model evaluation and analysis. Evaluating a model goes beyond looking at a single metric. It involves understanding the model's errors, checking its performance on different subsets of data, and considering aspects like fairness and interpretability.

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

13.1.3 Commonly Seen ANN Use Cases

“nobreak

13.1.4 Computer Vision

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

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

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

The commonly seen objectives of CV include:

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

13.1.5 Natural Language Processing

“nobreak

13.2 TensorFlow

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

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

13.2.1 TensorFlow Basics

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

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

tf.test.is_gpu_available()
```

where `.is_gpu_available()` tests whether GPU is enabled on the machine.

The well-known `numpy` package defines "NumPy arrays" to store vectors, matrices and tensors. Package `tensorflow` also defines counterparts "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
```

13.2.2 Classification and Regression

Classification

Regression

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

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

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

```
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import seaborn as sns

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

```
                        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.

13.2.3 Computer Vision

“nobreak

13.2.4 General Sequential Data Processing

“nobreak

13.2.5 Natural Language Processing

“nobreak

13.2.6 TensorFlow on Different Platforms

“nobreak

13.3 PyTorch

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

13.3.1 PyTorch Basics

“nobreak

13.3.2 Classification and Regression

“nobreak

13.3.3 Computer Vision

“nobreak

13.3.4 General Sequential Data Processing

“nobreak

13.3.5 Natural Language Processing

“nobreak

13.3.6 PyTorch on Different Platforms

Part IV

Semantic Web

14

Semantic Web Basics

CONTENTS

Ontology is the philosophical study of the nature of being, existence, or reality. It concerns about "what is everything" and "how to define it" in the context of inductive and deductive reasoning. It discusses how we abstract and preserve knowledge for the generations to come.

Ontology inspires people in computer science about how we can store and exchange information efficiently using the internet. The solution is called the semantic web, an internet-based knowledge base schema. The internet powered by semantic web (together with other technologies) defines Web 3.0, a new-generation internet framework.

Notice that there are subtle differences between "internet" and "Internet". The lower case "internet" refers to the technology that bridges machines to form a network of any size, and the upper case "Internet" refers to the specific internet that links the entire world together. In this part of the notebook, however, they are used interchangeably.

14.1 Web of Data

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

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

14.1.1 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. A user can browse the contents, but he cannot edit them. It is essentially a one-way data transmission. The "authority" such as a news company provides the information, and the 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 expectation from the users who want to share their information to other users on the internet. Thanks to the advent 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 that we encounter with Web 2.0 is how to quickly locate the information we want among the vast amount of irrelevant data. Conventionally in Web 2.0, keyword-based searching engines are used. These searching engines do not comprehend the contextual knowledge of the contents, and as a result they may fail to return what a user truly expects. The user often needs to further manually pick up relevant information from the searching results. Nowadays searching engines are becoming smarter, and they can sometimes pre-filter the results. Nevertheless, it is still quite common that many returned results are useless to the user. Keyword-based engines also struggle with polysemous, synonyms and implicit information from pictures in the searching 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.

14.1.2 Web 3.0

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

14.1.3 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, semantic web is more scalable, reusable and reader-friendly for both humans and machines comparing with 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.

14.1.4 Semantic Web Stack

A commonly seen semantic web stack looks like Fig. 14.1.

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

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

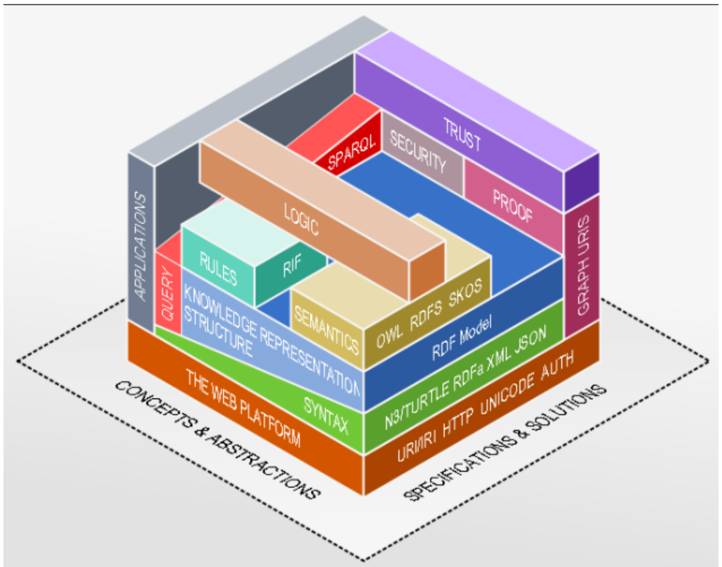

FIGURE 14.1
Semantic web stack [2].

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

- Syntax
  - File formats to store information, such as and RDF/XML;

- Knowledge representation, semantics, and interfacing tools
  - RDF model by default, which uses triples to represent a graphical database;
  - RDFs and OWL which expand the capability of RDF model;
  - SPARQL which is the default 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 syntax such as RDF/XML can be used as the 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 a user to search, retrieve, and modify information stored in the RDF model. The syntax of SPARQL is designed to look similar with 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. The example is provided by *ChatGPT-4*.

RDF/RDFS

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .

ex:Animal rdf:type rdfs:Class .

ex:Mammal rdf:type rdfs:Class ;
   rdfs:subClassOf ex:Animal .

ex:Reptile rdf:type rdfs:Class ;
   rdfs:subClassOf ex:Animal .

ex:hasLegs rdf:type rdf:Property ;
   rdfs:domain ex:Animal ;
   rdfs:range rdfs:Literal .

ex:Dog rdf:type rdfs:Class ;
   rdfs:subClassOf ex:Mammal .
```

```
ex:Lizard rdf:type rdfs:Class ;
    rdfs:subClassOf ex:Reptile .

ex:Max rdf:type ex:Dog ;
    ex:hasLegs 4 .

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

In this example, 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.

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.

14.1.5 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.

14.2 Ontology

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

14.2.1 Philosophy Perspective

Knowledge is the overlapping part of ground 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.

14.2.2 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.

14.2.3 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 single 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. 14.2. In practice, there might not be a clear boundary between two adjacent layers.

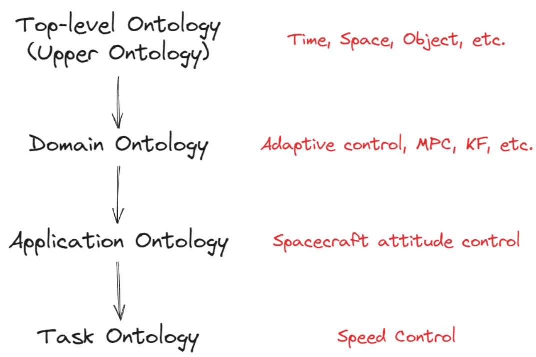

FIGURE 14.2
A demonstrative example of ontology level using control engineering.

Lower-layer ontology can inherit and modify knowledge from the upper-layer ontology. 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.

14.3 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 form a research area by themselves and it is not possible to include all the details in the scope of this notebook. Only a brief scratch is given.

14.3.1 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 syntax.

```
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, but it is quite different from that of a human. 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 "tho 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 similar but 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.

14.3.2 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"

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

where CLASS(x) is equivalent of a proposition "x belongs to CLASS", which can be either true or false. The above proposition says "for any item, if that item belongs to human, then that item belongs to mortal".

Description Logic

Description logic (DL) is a subset of the first-order logic. 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 16.

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 16. 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 (valid or invalid). If no such algorithm exists for a logic, it is undecidable. Gödel'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 formula can be 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 (insert instances) statements 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.

14.3.3 Logical Expression

A logic is defined by

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

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

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

where

$$\models^1 \quad = \quad \{(\Phi, \phi) | \Phi \subseteq S, \phi \in S, \Phi \rightarrow \phi\} \tag{14.1}$$

$$\models^2 \quad = \quad \left\{(\Phi, \Psi) \middle| \Phi, \Psi \subseteq S, \forall \psi \in \Psi, \Phi \models^1 \psi\right\} \tag{14.2}$$

In (14.1), 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. 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. 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.

14.3.4 Logical Equivalence

Logical equivalence has already been introduced in an earlier section. Recall (14.2) 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. The proof of the formulas is not given here. Commonly seen equivalence is given in Table 14.1. 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 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.

TABLE 14.1
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

14.3.5 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.

15

Resource Description Framework

CONTENTS

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.

15.1 Uniform Resource Identifier

Human use 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. 15.1.

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,

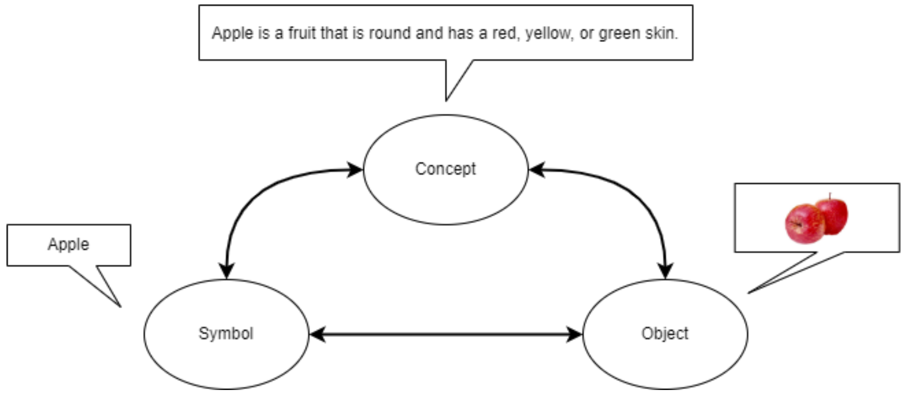

FIGURE 15.1
Semiotic triangle of Apple.

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 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 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.

15.2 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.

15.2.1 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. 15.2. The RDF in Turtle is given as follows.

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

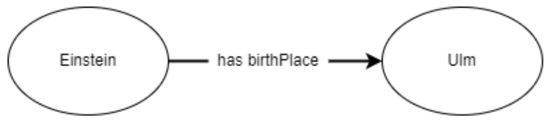

FIGURE 15.2
Graph representation of triple for knowledge "Einstein was born in Ulm".

```
dbr:Albert_Einstein dbo:birthPlace dbr:Ulm .
```

where `Albert_Einstein` and `Ulm` are defined in `dbpedia.org/resource/` as name and place, and `birthPlace` in `dbpedia.org/ontology/` as 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.

15.2.2 Multi-valued Relation and Blank Node

It is possible to use multi-valued relations and blank nodes to enforce combining of information. Consider an example given in Fig. 15.3, 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. 15.4.

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

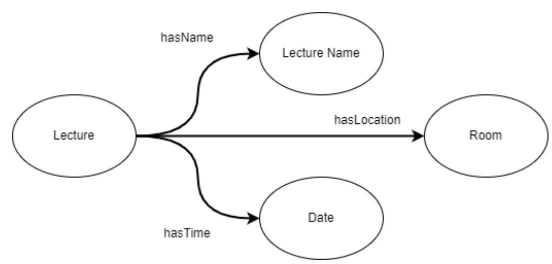

FIGURE 15.3
An example that demonstrate when and where a lecture takes place.

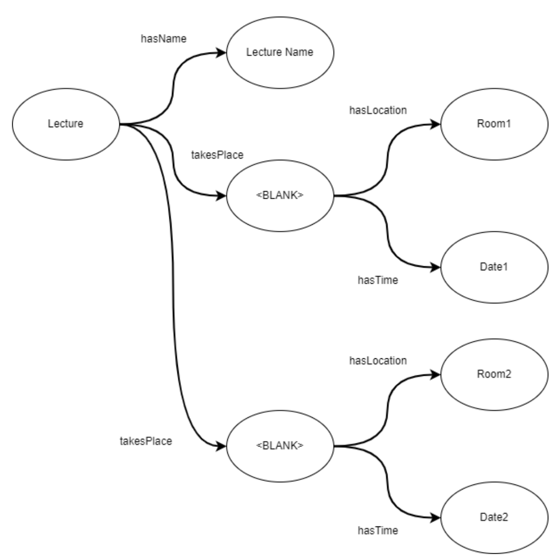

FIGURE 15.4
An example that demonstrate a lecture taking place at multiple locations and time slots using multi-valued relations and blank nodes.

```
<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.

15.2.3 Lists

Lists help to make the code clean and readable. There are two types of lists, the container (open list, extendable) and the collections (closed list, fixed). Container is helpful to handle the situation given in Fig. 15.5. Notice that

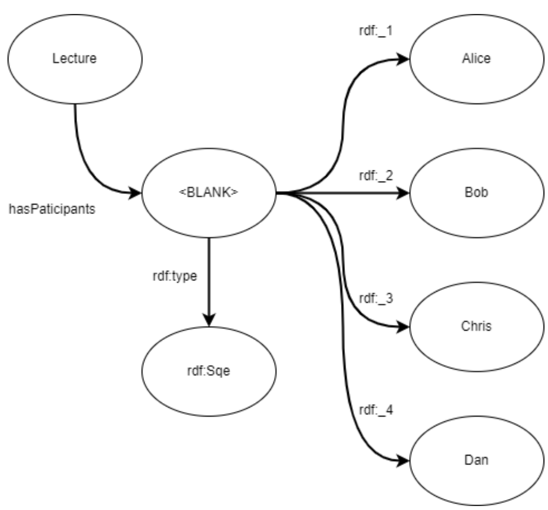

FIGURE 15.5
An example of a container.

the blank node has a type `rdf:Seq`. This tells that the items stored in the container follows an ordered set. There are other container types, such as `rdf:Bag` (unordered set) and `rdf:Alt` (alternatives of elements; only one element is relevant for the application).

The collection, on the other hand, defines a closed list as shown by Fig. 15.6. In Turtle, this can be done by using nested `[]` iteratively. A short cut

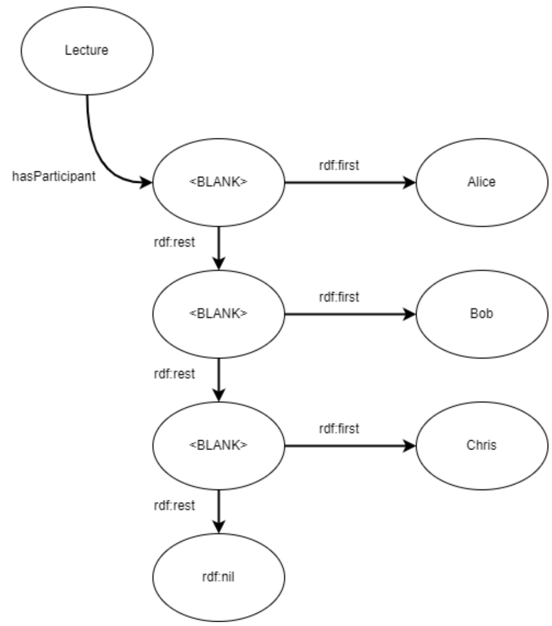

FIGURE 15.6
An example of a collection.

is to use `()`, with the items in the collection listed down in the bracket as follows.

```
<subject> <predicate> (<object1> <object2> <object3>) .
```

15.2.4 Reification

RDF permits interleaving of statements, i.e., to make a statement about another statement. For example, consider "Alice says that Bob ate the cake". This example contains nested statement, where "Bob ate the cake" is a statement, and "Alice says . . . " is a statement on that statement. RDF reification follows Fig. 15.7.

15.2.5 Converting RDB to RDF

For small-scale relational database, it is possible to convert it to semantic web RDF manually. For example, consider an RDB for all the books in a study. There are three tables in the database, books, authors and publishers, respectively. Each table contains a few dozens of entries. The RDB can be converted to RDF as shown in Fig. 15.8. It can be realized very easily, simply by defining everything as nodes and stack triples for all relationships.

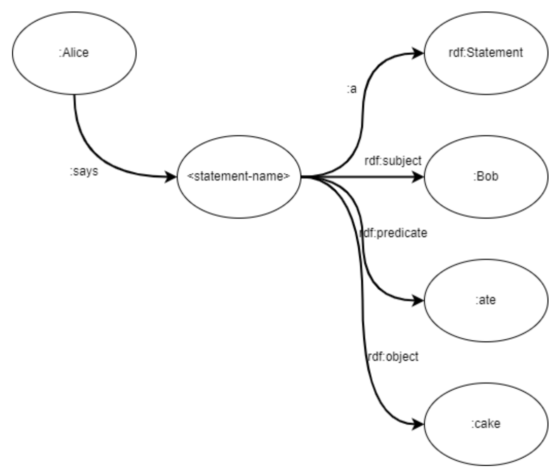

FIGURE 15.7
An example of RDF reification.

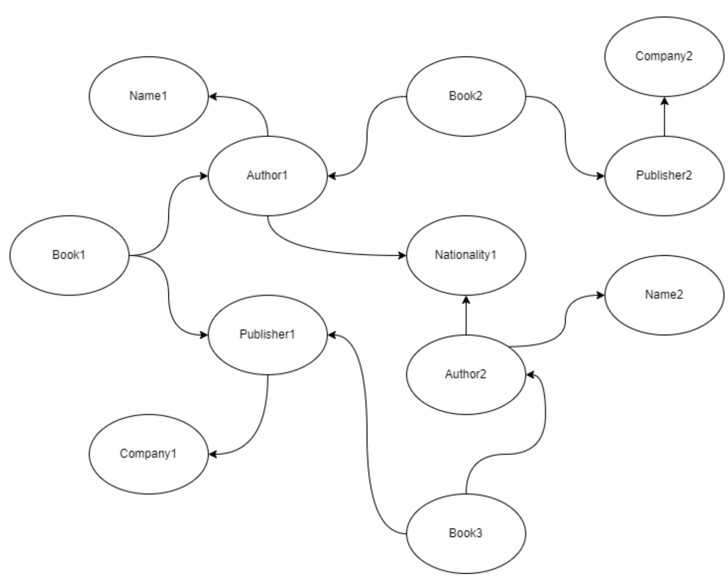

FIGURE 15.8
Semantic web of a few books, and their authors and publishers.

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 [6] for more details.

15.2.6 Conclusion

In summary, RDF, different from a plain XML or JSON which can also store independent classes, defines not only the objects (nodes) themselves but also the relationships among objects. This is essentially how RDF differs from a conventional key-value based NoSQL. RDF can be easily scaled up as nodes with the same name naturally merge together.

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

15.3 RDF Schema (RDFS)

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

15.3.1 RDFS Motivation

RDF is flexible. It is so flexible that sometimes it becomes difficult to maintain consistency of the RDF model, let alone performing sophisticated reasoning from it. RDFS, also known as RDF vocabulary description language, enforce schema to the RDF model by adding more "meta information" which builds more connections among the nodes.

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

15.3.2 RDF Versus RDF/RDFS via an Example

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. 15.9. 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.

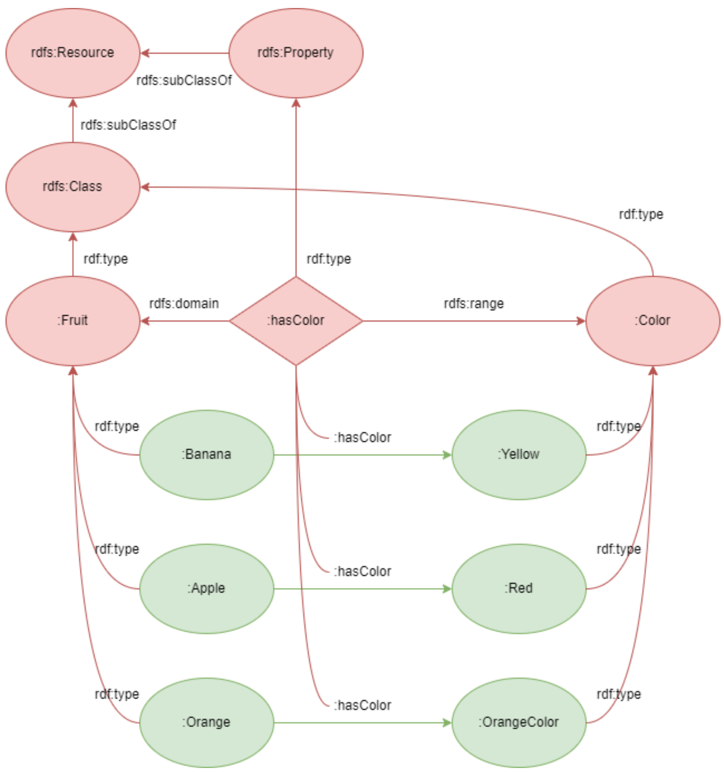

FIGURE 15.9
Semantic web of fruits and their colors, with RDF implementation in green RDF/RDFS in red.

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. 15.9. 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 .

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. 15.9 (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.

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

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

15.3.4 Semantics inside RDF/RDFS

The semantics in RDF/RDFS 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.

15.4 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.

15.4.1 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 15.3.2. 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.

15.4.2 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 .
}
```

15.4.3 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.

15.4.4 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.

15.4.5 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.

16

Web Ontology Language

CONTENTS

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.

16.1 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. 16.1. 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 .
}
```

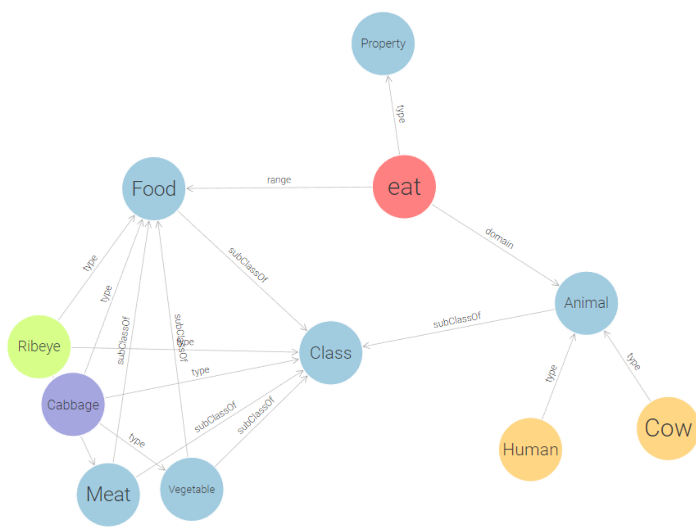

FIGURE 16.1
An RDF model that demonstrates "animal eats food".

From Fig. 16.1, 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. 14.1, one layer above RDF/RDFS, OWL is proposed to solve the above problems.

16.2 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 16.1. 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.

16.3 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.

16.4 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.

16.5 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:

$$A \to B$$

or

$$A_1 \wedge \ldots \wedge A_n \to B$$

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.

17

Semantic Web Practice

CONTENTS

This chapter studies the design, development and deployment of ontology and semantic web model. Both the methodologies and the tools are concerned.

17.1 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. 14.2). 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.

17.2 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.

17.2.1 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.

17.2.2 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.

17.2.3 Semantic Web Design for Enterprise

The basic semantic web design method introduced in Section 17.2.2 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. 17.1. In the workflow,

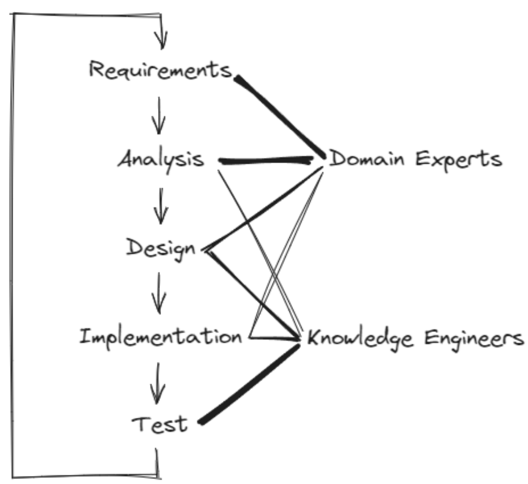

FIGURE 17.1
Semantic web design workflow.

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.

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/
```

17.3 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.

17.3.1 Web of Data

Consider the conventional way of retrieving data from a web server as shown in Fig. 17.2. 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 17.2
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. 17.3. DBpedia sits in the center of this figure, as it was one of the earliest and most comprehensive knowledge base.

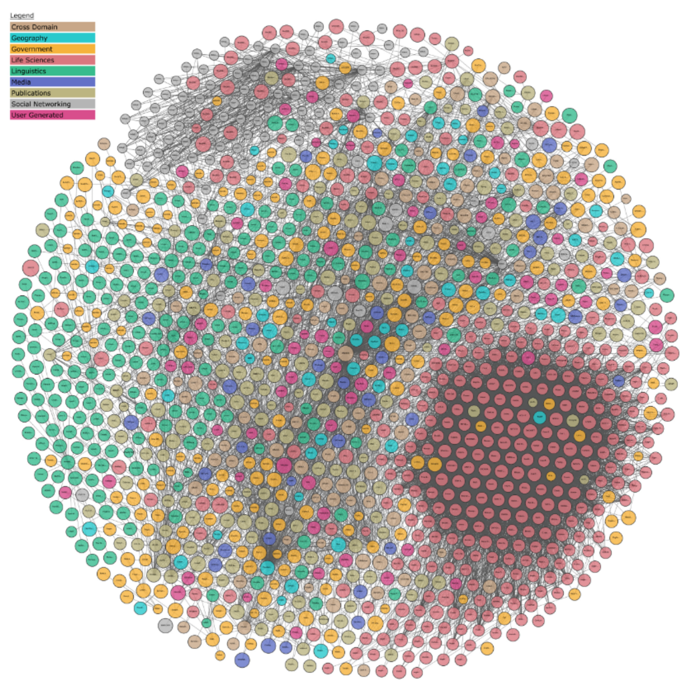

FIGURE 17.3
The linked open data cloud from *lod-cloud.net*.

17.3.2 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.

17.4 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.

17.5 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. 17.4, after visualization.

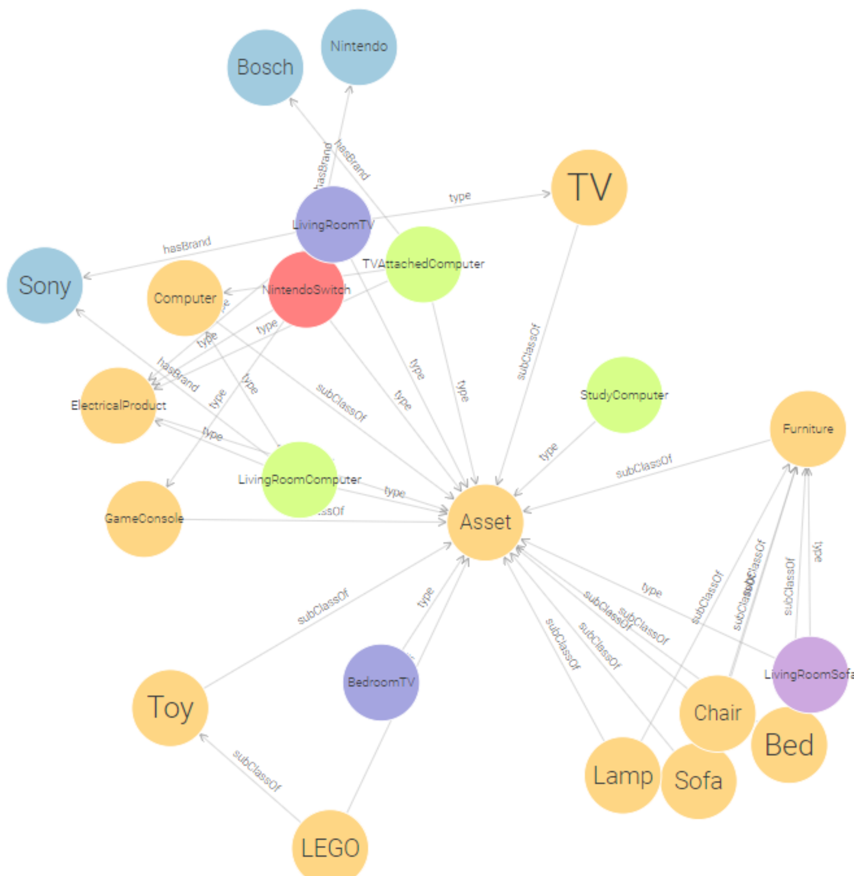

FIGURE 17.4
An example of an RDF model in GraphDB that describes house assets. This is only a demonstration graph and the information inside is artificial and not true.

TABLE 17.1
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.

17.5.1 Define Classes Hierarchy

“nobreak

17.5.2 Define Properties Hierarchy

“nobreak

17.5.3 Add OWL

“nobreak

17.5.4 Data Retrieval Examples

“nobreak

17.6 Reference: Commonly Used Namespace

Commonly used built-in name spaces are summarized in Table 17.1.

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] The R project for statistical computing.

[5] Rstudio.

[6] Franck Michel, Johan Montagnat, and Catherine Faron Zucker. *A survey of RDB to RDF translation approaches and tools*. PhD thesis, I3S, 2014.