(BSFUI +BNFT t %BOJFMB 8JUUFO t
5SFWPS)BTUJF 3PCFSU
5JCTIJSBOJ t +POBUIBO 5BZMPS

"O *OUSPEVDUJPO UP 4UBUJTUJDBM -FBSOJOH XJUI

"QQMJDBUJPOT JO 1ZUIPO

To our parents:

Alison and Michael James

Chiara Nappi and Edward

Witten Valerie and Patrick

Hastie Vera and Sami Tibshirani

John and Brenda Taylor

and to our families:

Michael, Daniel, and Catherine

Tessa, Theo, Otto, and Ari

Samantha, Timothy, and Lynda

Charlie, Ryan, Julie, and

Cheryl Lee-Ann and Isobel

Preface

Statistical learning refers to a set of tools for *making sense of complex datasets*. In recent years, we have seen a staggering increase in the scale and scope of data collection across virtually all areas of science and industry. As a result, statistical learning has become a critical toolkit for anyone who wishes to understand data — and as more and more of today's jobs involve data, this means that statistical learning is fast becoming a critical toolkit for *everyone*.

One of the first books on statistical learning — *The Elements of Statistical Learning* (ESL, by Hastie, Tibshirani, and Friedman) — was published in 2001, with a second edition in 2009. ESL has become a popular text not only in statistics but also in related fields. One of the reasons for ESL's popularity is its relatively accessible style. But ESL is best-suited for individuals with advanced training in the mathematical sciences.

An Introduction to Statistical Learning, With Applications in R (ISLR) — first published in 2013, with a second edition in 2021 — arose from the clear need for a broader and less technical treatment of the key topics in statistical learning. In addition to a review of linear regression, ISLR covers many of today's most important statistical and machine learning approaches, including resampling, sparse methods for classification and regression, generalized additive models, tree-based methods, support vector machines, deep learning, survival analysis, clustering, and multiple testing.

Since it was published in 2013, ISLR has become a mainstay of undergraduate and graduate classrooms worldwide, as well as an important reference book for data scientists. One of the keys to its success has been that, beginning with Chapter 2, each chapter contains an R lab illustrating how to implement the statistical learning methods seen in that chapter, providing the reader with valuable hands-on experience.

However, in recent years Python has become an increasingly popular lan guage for data science, and there has been increasing demand for a Python

based alternative to ISLR. Hence, this book, *An Introduction to Statistical Learning, With Applications in Python* (ISLP), covers the same materials as ISLR but with labs implemented in Python — a feat accomplished by the addition of a new co-author, Jonathan Taylor. Several of the labs make use of the ISLP Python package, which we have written to facilitate carrying out the statistical learning methods covered in each chapter in Python. These labs will be useful both for Python novices, as well as experienced users.

The intention behind ISLP (and ISLR) is to concentrate more on the applications of the methods and less on the mathematical details, so it is appropriate for advanced undergraduates or master's students in statistics or related quantitative felds, or for individuals in other disciplines who wish to use statistical learning tools to analyze their data. It can be used as a textbook for a course spanning two semesters.

We are grateful to these readers for providing valuable comments on the frst edition of ISLR: Pallavi Basu, Alexandra Chouldechova, Patrick Dana her, Will Fithian, Luella Fu, Sam Gross, Max Grazier G'Sell, Courtney Paulson, Xinghao Qiao, Elisa Sheng, Noah Simon, Kean Ming Tan, Xin Lu Tan. We thank these readers for helpful input on the second edition of ISLR: Alan Agresti, Iain Carmichael, Yiqun Chen, Erin Craig, Daisy Ding, Lucy Gao, Ismael Lemhadri, Bryan Martin, Anna Neufeld, Geof Tims, Carsten Voelkmann, Steve Yadlowsky, and James Zou. We are immensely grateful to Balasubramanian "Naras" Narasimhan for his assistance on both ISLR and ISLP.

It has been an honor and a privilege for us to see the considerable impact that ISLR has had on the way in which statistical learning is practiced, both in and out of the academic setting. We hope that this new Python edition will continue to give today's and tomorrow's applied statisticians and data scientists the tools they need for success in a data-driven world.

It's tough to make predictions, especially about the future. -Yogi

Berra

Contents

1

Introduction

An Overview of Statistical Learning

Statistical learning refers to a vast set of tools for *understanding data*. These tools can be classifed as *supervised* or *unsupervised*. Broadly speaking, supervised statistical learning involves building a statistical model for pre dicting, or estimating, an *output* based on one or more *inputs*. Problems of this nature occur in felds as diverse as business, medicine, astrophysics, and public policy. With unsupervised statistical learning, there are inputs but no supervising output; nevertheless we can learn relationships and struc ture from such data. To provide an illustration of some applications of statistical learning, we briefy discuss three real-world data sets that are considered in this book.

Wage Data

In this application (which we refer to as the Wage data set throughout this book), we examine a number of factors that relate to wages for a group of men from the Atlantic region of the United States. In particular, we wish to understand the association between an employee's age and education, as well as the calendar year, on his wage. Consider, for example, the left-hand panel of Figure 1.1, which displays wage versus age for each of the individu als in the data set. There is evidence that wage increases with age but then decreases again after approximately age 60. The blue line, which provides an estimate of the average wage for a given age, makes this trend clearer. Given an employee's age, we can use this curve to *predict* his wage. However, it is also clear from Figure 1.1 that there is a signifcant amount of vari ability associated with this average value, and so age alone is unlikely to provide an accurate prediction of a particular man's wage.

G. James et al., *An Introduction to Statistical Learning*, Springer Texts in Statistics,
https://doi.org/10.1007/978-3-031-38747-0_1

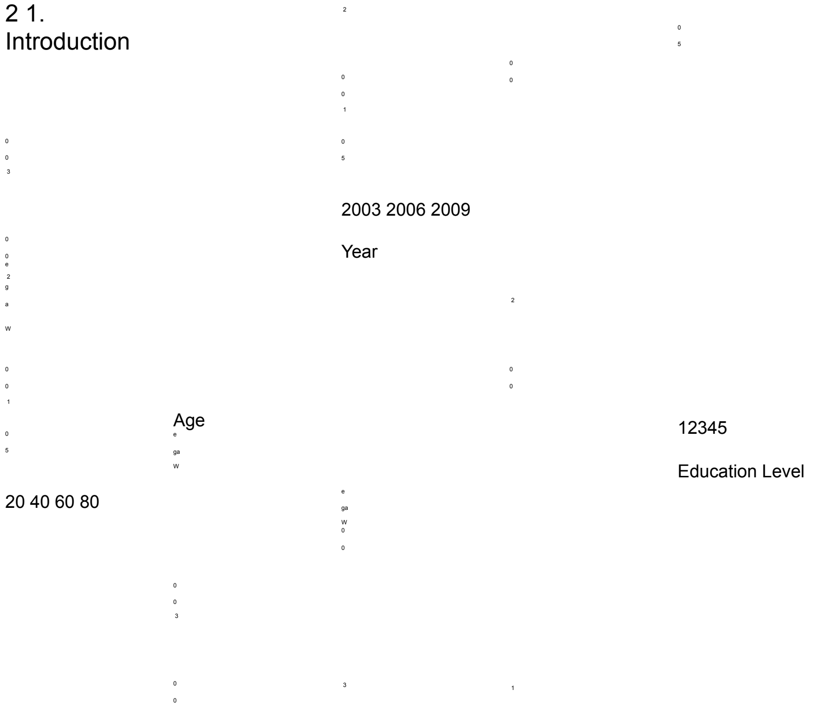


FIGURE 1.1. Wage *data, which contains income survey information for men from the central Atlantic region of the United States.* Left: wage *as a function of* age. *On average,* wage *increases with* age *until about* 60 *years of age, at which point it begins to decline.* Center: wage *as a function of* year. *There is a slow but steady increase of approximately* $10,000 *in the average* wage *between* 2003 *and* 2009. Right: *Boxplots displaying* wage *as a function of* education*, with* 1 *indicating the lowest level (no high school diploma) and* 5 *the highest level (an advanced graduate degree). On average,* wage *increases with the level of education.*

We also have information regarding each employee's education level and the year in which the wage was earned. The center and right-hand panels of Figure 1.1, which display wage as a function of both year and education, indicate that both of these factors are associated with wage. Wages increase by approximately $10,000, in a roughly linear (or straight-line) fashion, between 2003 and 2009, though this rise is very slight relative to the vari ability in the data. Wages are also typically greater for individuals with higher education levels: men with the lowest education level (1) tend to have substantially lower wages than those with the highest education level (5). Clearly, the most accurate prediction of a given man's wage will be obtained by combining his age, his education, and the year. In Chapter 3, we discuss linear regression, which can be used to predict wage from this data set. Ideally, we should predict wage in a way that accounts for the non-linear relationship between wage and age. In Chapter 7, we discuss a class of approaches for addressing this problem.

Stock Market Data

The Wage data involves predicting a *continuous* or *quantitative* output value. This is often referred to as a *regression* problem. However, in certain cases we may instead wish to predict a non-numerical value—that is, a *categorical* or *qualitative* output. For example, in Chapter 4 we examine a stock market data set that contains the daily movements in the Standard & Poor's 500 (S&P) stock index over a 5-year period between 2001 and 2005. We refer to this as the Smarket data. The goal is to predict whether the index will *increase* or *decrease* on a given day, using the past 5 days' percentage changes in the index. Here the statistical learning problem does not involve predicting a numerical value. Instead it involves predicting whether a given day's stock market performance will fall into the Up bucket or the Down bucket. This is known as a *classifcation* problem. A model that could accurately predict the direction in which the market will move would be very useful!

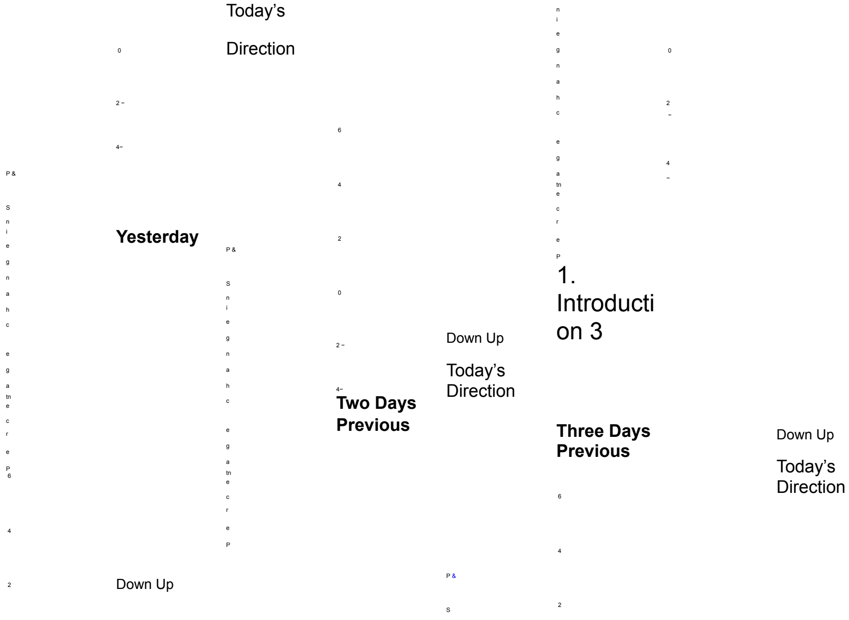


FIGURE 1.2. Left: *Boxplots of the previous day's percentage change in the S&P index for the days for which the market increased or decreased, obtained from the* Smarket *data.* Center and Right: *Same as left panel, but the percentage changes for 2 and 3 days previous are shown.*

The left-hand panel of Figure 1.2 displays two boxplots of the previous day's percentage changes in the stock index: one for the 648 days for which the market increased on the subsequent day, and one for the 602 days for which the market decreased. The two plots look almost identical, suggest ing that there is no simple strategy for using yesterday's movement in the S&P to predict today's returns. The remaining panels, which display box plots for the percentage changes 2 and 3 days previous to today, similarly indicate little association between past and present returns. Of course, this lack of pattern is to be expected: in the presence of strong correlations be tween successive days' returns, one could adopt a simple trading strategy to generate profts from the market. Nevertheless, in Chapter 4, we explore these data using several diferent statistical learning methods. Interestingly, there are hints of some weak trends in the data that suggest that, at least for this 5-year period, it is possible to correctly predict the direction of movement in the market approximately 60% of the time (Figure 1.3).

Gene Expression Data

The previous two applications illustrate data sets with both input and output variables. However, another important class of problems involves situations in which we only observe input variables, with no corresponding output. For example, in a marketing setting, we might have demographic information for a number of current or potential customers. We may wish to understand which types of customers are similar to each other by grouping individuals according to their observed characteristics. This is known as a

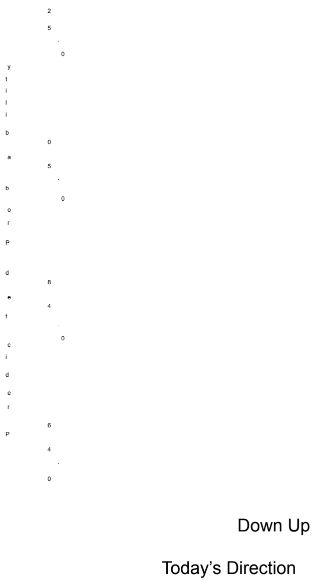


FIGURE 1.3. *We ft a quadratic discriminant analysis model to the subset of the* Smarket *data corresponding to the 2001–2004 time period, and predicted the probability of a stock market decrease using the 2005 data. On average, the predicted probability of decrease is higher for the days in which the market does decrease. Based on these results, we are able to correctly predict the direction of*

movement in the market 60% of the time.

clustering problem. Unlike in the previous examples, here we are not trying to predict an output variable.

We devote Chapter 12 to a discussion of statistical learning methods for problems in which no natural output variable is available. We consider the NCI60 data set, which consists of 6,830 gene expression measurements for each of 64 cancer cell lines. Instead of predicting a particular output variable, we are interested in determining whether there are groups, or clusters, among the cell lines based on their gene expression measurements. This is a difficult question to address, in part because there are thousands of gene expression measurements per cell line, making it hard to visualize the data.

The left-hand panel of Figure 1.4 addresses this problem by representing each of the 64 cell lines using just two numbers, Z_1 and Z_2. These are the first two *principal components* of the data, which summarize the 6,830 expression measurements for each cell line down to two numbers or *dimensions*. While it is likely that this dimension reduction has resulted in some loss of information, it is now possible to visually examine the data for evidence of clustering. Deciding on the number of clusters is often a difficult problem. But the left-hand panel of Figure 1.4 suggests at least four groups of cell lines, which we have represented using separate colors.

In this particular data set, it turns out that the cell lines correspond to 14 different types of cancer. (However, this information was not used to create the left-hand panel of Figure 1.4.) The right-hand panel of Figure 1.4 is identical to the left-hand panel, except that the 14 cancer types are shown using distinct colored symbols. There is clear evidence that cell lines with the same cancer type tend to be located near each other in this two-dimensional representation. In addition, even though the cancer information was not used to produce the left-hand panel, the clustering obtained does bear some resemblance to some of the actual cancer types observed in the right-hand panel. This provides some independent verification of the accuracy of our clustering analysis.

FIGURE 1.4. Left: *Representation of the* NCI60 *gene expression data set in a two-dimensional space, Z_1 and Z_2. Each point corresponds to one of the* 64 *cell lines. There appear to be four groups of cell lines, which we have represented using diferent colors.* Right: *Same as left panel except that we have represented each of the* 14 *diferent types of cancer using a diferent colored symbol. Cell lines corresponding to the same cancer type tend to be nearby in the two-dimensional space.*

A Brief History of Statistical Learning

Though the term *statistical learning* is fairly new, many of the concepts that underlie the feld were developed long ago. At the beginning of the nine teenth century, the method of *least squares* was developed, implementing the earliest form of what is now known as *linear regression*. The approach was frst successfully applied to problems in astronomy. Linear regression is used for predicting quantitative values, such as an individual's salary. In order to predict qualitative values, such as whether a patient survives or dies, or whether the stock market increases or decreases, *linear discrim inant analysis* was proposed in 1936. In the 1940s, various authors put forth an alternative approach, *logistic regression*. In the early 1970s, the term *generalized linear model* was developed to describe an entire class of statistical learning methods that include both linear and logistic regression as special cases.

By the end of the 1970s, many more techniques for learning from data were available. However, they were almost exclusively *linear* methods because fitting *non-linear* relationships was computationally difficult at the time. By the 1980s, computing technology had finally improved sufficiently that non-linear methods were no longer computationally prohibitive. In the mid 1980s, *classification and regression trees* were developed, followed shortly by *generalized additive models*. *Neural networks* gained popularity in the 1980s, and *support vector machines* arose in the 1990s.

Since that time, statistical learning has emerged as a new subfield in statistics, focused on supervised and unsupervised modeling and prediction. In recent years, progress in statistical learning has been marked by the increasing availability of powerful and relatively user-friendly software, such as the popular and freely available Python system. This has the potential to continue the transformation of the field from a set of techniques used and developed by statisticians and computer scientists to an essential toolkit for a much broader community.

This Book

The Elements of Statistical Learning (ESL) by Hastie, Tibshirani, and Friedman was first published in 2001. Since that time, it has become an important reference on the fundamentals of statistical machine learning. Its success derives from its comprehensive and detailed treatment of many important topics in statistical learning, as well as the fact that (relative to many upper-level statistics textbooks) it is accessible to a wide audience. However, the greatest factor behind the success of ESL has been its topical nature. At the time of its publication, interest in the field of statistical learning was starting to explode. ESL provided one of the first accessible and comprehensive introductions to the topic.

Since ESL was first published, the field of statistical learning has continued to flourish. The field's expansion has taken two forms. The most obvious growth has involved the development of new and improved statistical learning approaches aimed at answering a range of scientific questions across a number of fields. However, the field of statistical learning has also expanded its audience. In the 1990s, increases in computational power generated a surge of interest in the field from non-statisticians who were eager to use cutting-edge statistical tools to analyze their data. Unfortunately, the highly technical nature of these approaches meant that the user community remained primarily restricted to experts in statistics, computer science, and related fields with the training (and time) to understand and implement them.

In recent years, new and improved software packages have significantly eased the implementation burden for many statistical learning methods. At the same time, there has been growing recognition across a number of fields, from business to health care to genetics to the social sciences and beyond, that statistical learning is a powerful tool with important practical applications. As a result, the field has moved from one of primarily academic interest to a mainstream discipline, with an enormous potential audience. This trend will surely continue with the increasing availability of enormous quantities of data and the software to analyze it.

The purpose of *An Introduction to Statistical Learning* (ISL) is to facilitate the transition of statistical learning from an academic to a mainstream field. ISL is not intended to replace ESL, which is a far more comprehensive text both in terms of the number of approaches considered and the depth to which they are explored. We consider ESL to be an important companion for professionals (with graduate degrees in statistics, machine learning, or related fields) who need to understand the technical details behind statistical learning approaches. However, the community of users of statistical learning techniques has expanded to include individuals with a wider range of interests and backgrounds. Therefore, there is a place for a less technical and more accessible version of ESL.

In teaching these topics over the years, we have discovered that they are of interest to master's and PhD students in fields as disparate as business administration, biology, and computer science, as well as to quantitatively oriented upper-division undergraduates. It is important for this diverse group to be able to understand the models, intuitions, and strengths and weaknesses of the various approaches. But for this audience, many of the technical details behind statistical learning methods, such as optimization algorithms and theoretical properties, are not of primary interest. We believe that these students do not need a deep understanding of these aspects in order to become informed users of the various methodologies, and in order to contribute to their chosen fields through the use of statistical learning tools.

ISL is based on the following four premises.

1. *Many statistical learning methods are relevant and useful in a wide range of academic and non-academic disciplines, beyond just the statistical sciences.* We believe that many contemporary statistical learning procedures should, and will, become as widely available and used as is currently the case for classical methods such as linear regression. As a result, rather than attempting to consider every possible approach (an impossible task), we have concentrated on presenting the methods that we believe are most widely applicable.

2. *Statistical learning should not be viewed as a series of black boxes.* No single approach will perform well in all possible applications. Without understanding all of the cogs inside the box, or the interaction between those cogs, it is impossible to select the best box. Hence, we have attempted to carefully describe the model, intuition, assumptions, and trade-offs behind each of the methods that we consider.

3. *While it is important to know what job is performed by each cog, it is not necessary to have the skills to construct the machine inside the box!* Thus, we have minimized discussion of technical details related to fitting procedures and theoretical properties. We assume that the reader is comfortable with basic mathematical concepts, but we do not assume a graduate degree in the mathematical sciences. For instance, we have almost completely avoided the

use of matrix algebra, and it is possible to understand the entire book without a detailed knowledge of matrices and vectors.

4. *We presume that the reader is interested in applying statistical learn ing methods to real-world problems.* In order to facilitate this, as well as to motivate the techniques discussed, we have devoted a section within each chapter to computer labs. In each lab, we walk the reader through a realistic application of the methods considered in that chap ter. When we have taught this material in our courses, we have al located roughly one-third of classroom time to working through the labs, and we have found them to be extremely useful. Many of the less computationally-oriented students who were initially intimidated by the labs got the hang of things over the course of the quarter or semester. This book originally appeared (2013, second edition 2021)

with computer labs written in the R language. Since then, there has been increasing demand for Python implementations of the important techniques in statistical learning. Consequently, this version has labs in Python. There are a rapidly growing number of Python pack ages available, and by examination of the imports at the beginning of each lab, readers will see that we have carefully selected and used the most appropriate. We have also supplied some additional code and functionality in our package ISLP. However, the labs in ISL are self contained, and can be skipped if the reader wishes to use a different software package or does not wish to apply the methods discussed to real-world problems.

Who Should Read This Book?

This book is intended for anyone who is interested in using modern statistical methods for modeling and prediction from data. This group includes scientists, engineers, data analysts, data scientists, and quants, but also less technical individuals with degrees in non-quantitative fields such as the so cial sciences or business. We expect that the reader will have had at least one elementary course in statistics. Background in linear regression is also useful, though not required, since we review the key concepts behind linear regression in Chapter 3. The mathematical level of this book is modest, and a detailed knowledge of matrix operations is not required. This book provides an introduction to Python. Previous exposure to a programming language, such as MATLAB or R, is useful but not required.

The first edition of this textbook has been used to teach master’s and PhD students in business, economics, computer science, biology, earth sci ences, psychology, and many other areas of the physical and social sciences. It has also been used to teach advanced undergraduates who have already taken a course on linear regression. In the context of a more mathemat ically rigorous course in which ESL serves as the primary textbook, ISL could be used as a supplementary text for teaching computational aspects of the various approaches.

Notation and Simple Matrix Algebra

Choosing notation for a textbook is always a difficult task. For the most part we adopt the same notational conventions as ESL.

We will use n to represent the number of distinct data points, or observa tions, in our sample. We will let p denote the number of variables that are available for use in making predictions. For example, the Wage data set con sists of 11 variables for 3,000 people, so we have n = 3,000 observations and p = 11 variables (such as year, age, race, and more). Note that throughout this book, we indicate variable names using colored font: Variable Name.

In some examples, p might be quite large, such as on the order of thousands or even millions; this situation arises quite often, for example, in the analysis of modern biological data or web-based advertising data.

In general, we will let x_{ij} represent the value of the jth variable for the ith observation, where i = 1, 2,...,n and j = 1, 2,...,p. Throughout this book, i will be used to index the samples or observations (from 1 to n) and j will be used to index the variables (from 1 to p). We let X denote an n × p matrix whose (i, j)th element is x_{ij}. That is,

$x_{12} \ldots x_{1p}$ $x_{n2} \ldots x_{np}$

x_{21} $x_{22} \ldots$ □

X = x_{2p} ·· ·· . . □□□

□ □ .

□□□ ·· ··. x_{n1}

□x_{11} .

For readers who are unfamiliar with matrices, it is useful to visualize X as a spreadsheet of numbers with n rows and p columns.

At times we will be interested in the rows of X, which we write as x_1, x_2,...,x_n. Here x_i is a vector of length p, containing the p variable measurements for the ith observation. That is,

□□□ □

□x_{i1}

□□□

$x^{i2..}$. □ . (1.1)

x_i =

□ x_{ip}

(Vectors are by default represented as columns.) For example, for the Wage data, x_i is a vector of length 11, consisting of year, age, race, and other values for the ith individual. At other times we will instead be interested in the columns of X, which we write as x_1, x_2,..., x_p. Each is a vector of

length n. That is,

□□□

□x_{1j}

$x^{2j..}$.

x_j =

□ x_{nj}

$$.$$

For example, for the Wage data, x_1 contains the n = 3,000 values for year. Using this notation, the matrix X can be written as

$$\mathbf{X} = \left(\mathbf{x}_1 \; \mathbf{x}_2 \; \cdots \; \mathbf{x}_p \right),$$

or

$$\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}.$$

The ^{T} notation denotes the *transpose* of a matrix or vector. So, for example,

$$\mathbf{X}^T = \begin{pmatrix} x_{11} & x_{21} & \ldots & x_{n1} \\ x_{12} & x_{22} & \ldots & x_{n2} \\ \vdots & \vdots & & \vdots \\ x_{1p} & x_{2p} & \ldots & x_{np} \end{pmatrix},$$

while

$$x_i^T = \left(x_{i1} \; x_{i2} \; \cdots \; x_{ip} \right).$$

We use y_i to denote the ith observation of the variable on which we wish to make predictions, such as wage. Hence, we write the set of all n observations in vector form as

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

Then our observed data consists of $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where each x_i is a vector of length p. (If p = 1, then x_i is simply a scalar.) In this text, a vector of length n will always be denoted in *lower case bold*; e.g.

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}.$$

However, vectors that are not of length n (such as feature vectors of length p, as in (1.1)) will be denoted in *lower case normal font*, e.g. a. Scalars will also be denoted in *lower case normal font*, e.g. a. In the rare cases in which these two uses for lower case normal font lead to ambiguity, we will clarify which use is intended. Matrices will be denoted using *bold capitals*, such as A. Random variables will be denoted using *capital normal font*, e.g. *A*, regardless of their dimensions.

Occasionally we will want to indicate the dimension of a particular object. To indicate that an object is a scalar, we will use the notation $a \in$ R. To indicate that it is a vector of length *k*, we will use $a \in$ R^{k} (or a $\in$ R^{n} if it is of length *n*). We will indicate that an object is an *r* × *s* matrix using A $\in$ R$^{r\times s}$.

We have avoided using matrix algebra whenever possible. However, in a few instances it becomes too cumbersome to avoid it entirely. In these rare instances it is important to understand the concept of multiplying two matrices. Suppose that A $\in$ R$^{r\times d}$ and B $\in$ R$^{d\times s}$. Then the product of A and B is denoted AB. The (*i, j*)th element of AB is computed by multiplying each element of the *i*th row of A by the corresponding element

of the *j*th column of B. That is, (AB)$_{ij\,=}$)$^{d}_{k=1}$ $a_{ik}b_{kj}$. As an example, consider

+ * 5 6 7 8 .

Then

A = * 1 2 3 4 and B = +

AB = * 1 2 3 4 8 + + * 5 6 7 = * 1 × 5+2 × 7 1 × 6+2 × 8 3 × 5+4 × 7 3 × 6+4 × 8 + = * 19 22 43 50 + .

Note that this operation produces an *r* × *s* matrix. It is only possible to compute AB if the number of columns of A is the same as the number of rows of B.

Organization of This Book

Chapter 2 introduces the basic terminology and concepts behind statisti cal learning. This chapter also presents the *K-nearest neighbor* classifer, a very simple method that works surprisingly well on many problems. Chap ters 3 and 4 cover classical linear methods for regression and classifcation. In particular, Chapter 3 reviews *linear regression*, the fundamental start ing point for all regression methods. In Chapter 4 we discuss two of the most important classical classifcation methods, *logistic regression* and *lin ear discriminant analysis*.

A central problem in all statistical learning situations involves choosing the best method for a given application. Hence, in Chapter 5 we intro duce *cross-validation* and the *bootstrap*, which can be used to estimate the accuracy of a number of diferent methods in order to choose the best one.

Much of the recent research in statistical learning has concentrated on non-linear methods. However, linear methods often have advantages over their non-linear competitors in terms of interpretability and sometimes also accuracy. Hence, in Chapter 6 we consider a host of linear methods, both classical and more modern, which ofer potential improvements over stan dard linear regression. These include *stepwise selection*, *ridge regression*, *principal components regression*, and the *lasso*.

The remaining chapters move into the world of non-linear statistical learning. We first introduce in Chapter 7 a number of non-linear methods that work well for problems with a single input variable. We then show how these methods can be used to fit non-linear *additive* models for which there is more than one input. In Chapter 8, we investigate *tree*-based methods, including *bagging*, *boosting*, and *random forests*. *Support vector machines*, a set of approaches for performing both linear and non-linear classification, are discussed in Chapter 9. We cover *deep learning*, an approach for non-linear regression and classification that has received a lot of attention in recent years, in Chapter 10. Chapter 11 explores *survival analysis*, a regression approach that is specialized to the setting in which the output variable is *censored*, i.e. not fully observed.

In Chapter 12, we consider the *unsupervised* setting in which we have input variables but no output variable. In particular, we present *principal components analysis*, *K-means clustering*, and *hierarchical clustering*. Finally, in Chapter 13 we cover the very important topic of multiple hypothesis testing.

At the end of each chapter, we present one or more Python lab sections in which we systematically work through applications of the various methods discussed in that chapter. These labs demonstrate the strengths and weaknesses of the various approaches, and also provide a useful reference for the syntax required to implement the various methods. The reader may choose to work through the labs at their own pace, or the labs may be the focus of group sessions as part of a classroom environment. Within each Python lab, we present the results that we obtained when we performed the lab at the time of writing this book. However, new versions of Python are continuously released, and over time, the packages called in the labs will be updated. Therefore, in the future, it is possible that the results shown in the lab sections may no longer correspond precisely to the results

Name	Description
Auto	Gas mileage, horsepower, and other information for cars.
Bikeshare	Hourly usage of a bike sharing program in Washington, DC.
Boston	Housing values and other information about Boston census tracts.
BrainCancer	Survival times for patients diagnosed with brain cancer.
Caravan	Information about individuals offered caravan insurance.
Carseats	Information about car seat sales in 400 stores.
College	Demographic characteristics, tuition, and more for USA colleges.
Credit	Information about credit card debt for 400 customers.
Default	Customer default records for a credit card company.
Fund	Returns of 2,000 hedge fund managers over 50 months.
Hitters	Records and salaries for baseball players.
Khan	Gene expression measurements for four cancer types.
NCI60	Gene expression measurements for 64 cancer cell lines.
NYSE	Returns, volatility, and volume for the New York Stock Exchange.
OJ	Sales information for Citrus Hill and Minute Maid orange juice.
Portfolio	Past values of financial assets, for use in portfolio allocation.
Publication	Time to publication for 244 clinical trials.
Smarket	Daily percentage returns for S&P 500 over a 5-year period.
USArrests	Crime statistics per 100,000 residents in 50 states of USA.
Wage	Income survey data for men in central Atlantic region of USA.
Weekly	1,089 weekly stock market returns for 21 years.

TABLE 1.1. *A list of data sets needed to perform the labs and exercises in this textbook. All data sets are available in the* ISLP *package, with the exception of* USArrests*, which is part of the base* R *distribution, but accessible from* Python.

obtained by the reader who performs the labs. As necessary, we will post updates to the labs on the book website.

We use the symbol to denote sections or exercises that contain more challenging concepts. These can be easily skipped by readers who do not wish to delve as deeply into the material, or who lack the mathematical background.

Data Sets Used in Labs and Exercises

In this textbook, we illustrate statistical learning methods using applications from marketing, finance, biology, and other areas. The ISLP package contains a number of data sets that are required in order to perform the labs and exercises associated with this book. One other data set is part of the base R distribution (the USArrests data), and we show how to access it from Python in Section 12.5.1. Table 1.1 contains a summary of the data sets required to perform the labs and exercises. A couple of these data sets are also available as text files on the book website, for use in Chapter 2.

Book Website

The website for this book is located at

www.statlearning.com

It contains a number of resources, including the Python package associated with this book, and some additional data sets.

Acknowledgements

A few of the plots in this book were taken from ESL: Figures 6.7, 8.3, and 12.14. All other plots were produced for the R version of ISL, except for Figure 13.10 which differs because of the Python software supporting the plot.

2

Statistical Learning

2.1 What Is Statistical Learning?

In order to motivate our study of statistical learning, we begin with a simple example. Suppose that we are statistical consultants hired by a client to investigate the association between advertising and sales of a particular product. The Advertising data set consists of the sales of that product in 200 diferent markets, along with advertising budgets for the product in each of those markets for three diferent media: TV, radio, and newspaper. The data are displayed in Figure 2.1. It is not possible for our client to directly increase sales of the product. On the other hand, they can control the advertising expenditure in each of the three media. Therefore, if we determine that there is an association between advertising and sales, then we can instruct our client to adjust advertising budgets, thereby indirectly increasing sales. In other words, our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets.

In this setting, the advertising budgets are *input variables* while sales input variable is an *output variable*. The input variables are typically denoted using the symbol X, with a subscript to distinguish budget. The inputs them. So X_1 might be the TV budget, X_2 output variable the radio budget, and X_3 the newspaper

go by diferent names, such as *predictors*, *independent variables*, *features*, predictor or sometimes just *variables*. The output variable—in this case, sales—is often called the *response* or *dependent variable*, and is typically denoted using the symbol Y . Throughout this book, we will use all of these terms interchangeably. More generally, suppose that we observe a quantitative response Y and p diferent predictors, X_1, X_2,...,X_p. We assume that there is some relationship between Y and $X = (X_1, X_2,...,X_p)$, which can be written in the very general form

$Y = f(X) +$ ". (2.1)

independent variable
feature
variable
response
dependent variable

G. James et al., *An Introduction to Statistical Learning*, Springer Texts in Statistics,
https://doi.org/10.1007/978-3-031-38747-0_2

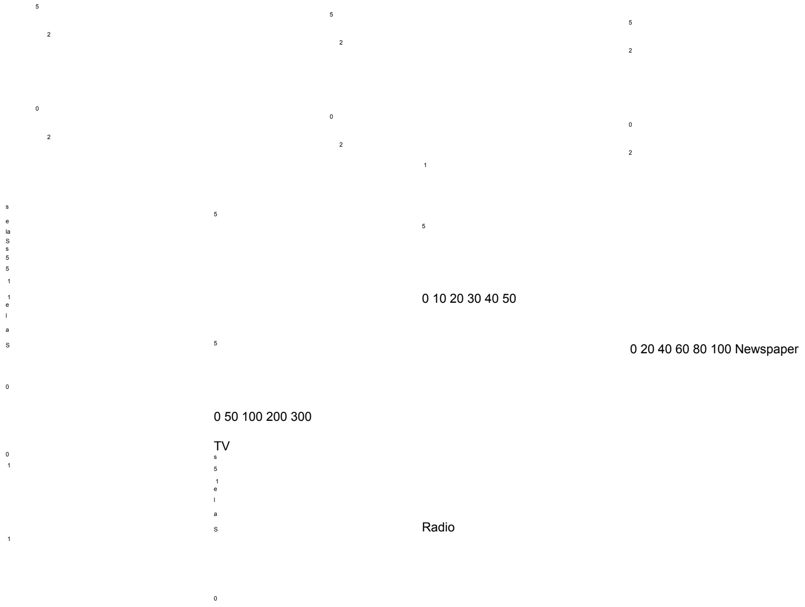
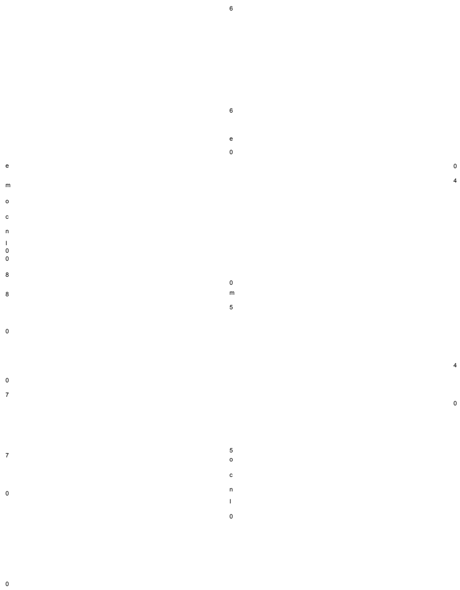


FIGURE 2.1. *The* Advertising *data set. The plot displays* sales*, in thousands of units, as a function of* TV*,* radio*, and* newspaper *budgets, in thousands of dollars, for* 200 *diferent markets. In each plot we show the simple least squares ft of* sales *to that variable, as described in Chapter* 3*. In other words, each blue line represents a simple model that can be used to predict* sales *using* TV*,* radio*, and* newspaper*, respectively.*

Here *f* is some fxed but unknown function of $X_1,...,X_p$, and " is a random *error term*, which is independent of *X* and has mean zero. In this formulation, *f* represents the *systematic* information that *X* provides about *Y* .

error term

systematic



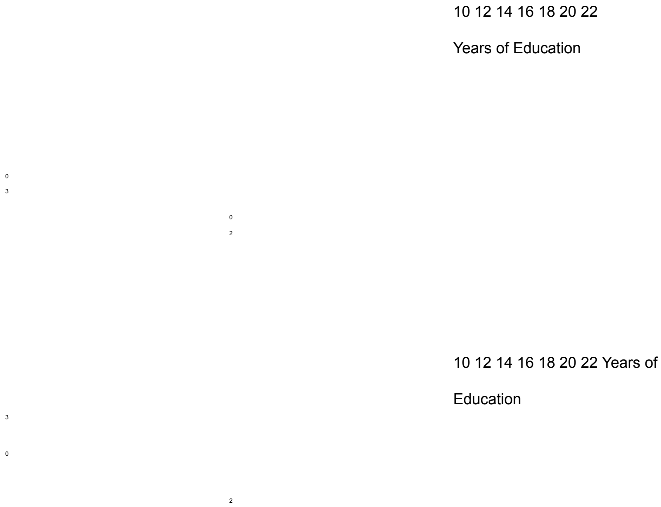


FIGURE 2.2. *The* Income *data set.* Left: *The red dots are the observed values of* income *(in thousands of dollars) and* years of education *for* 30 *individuals.* Right: *The blue curve represents the true underlying relationship between* income *and* years of education*, which is generally unknown (but is known in this case because the data were simulated). The black lines represent the error associated with each observation. Note that some errors are positive (if an observation lies above the blue curve) and some are negative (if an observation lies below the curve). Overall, these errors have approximately mean zero.*

As another example, consider the left-hand panel of Figure 2.2, a plot of income versus years of education for 30 individuals in the Income data set. The plot suggests that one might be able to predict income using years of education. However, the function *f* that connects the input variable to the output variable is in general unknown. In this situation one must estimate *f* based on the observed points. Since Income is a simulated data set, *f* is known and is shown by the blue curve in the right-hand panel of Figure 2.2. The vertical lines represent the error terms ″. We note that some of the 30 observations lie above the blue curve and some lie below it; overall, the errors have approximately mean zero.

In general, the function *f* may involve more than one input variable. In Figure 2.3 we plot income as a function of years of education and seniority. Here *f* is a two-dimensional surface that must be estimated based on the observed data.

In essence, statistical learning refers to a set of approaches for estimating *f*. In this chapter we outline some of the key theoretical concepts that arise in estimating *f*, as well as tools for evaluating the estimates obtained.

2.1.1 Why Estimate f?

There are two main reasons that we may wish to estimate *f*: *prediction* and *inference*. We discuss each in turn.

Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. In this setting, since the error term averages to zero, we can predict Y using

$$\hat{Y} = \hat{f}(X), \quad (2.2)$$

where $\hat{f}$ represents our estimate for f, and $\hat{Y}$ represents the resulting prediction for Y. In this setting, $\hat{f}$ is often treated as a *black box*, in the sense that one is not typically concerned with the exact form of $\hat{f}$, provided that it yields accurate predictions for Y.

As an example, suppose that $X_1,...,X_p$ are characteristics of a patient's blood sample that can be easily measured in a lab, and Y is a variable encoding the patient's risk for a severe adverse reaction to a particular drug. It is natural to seek to predict Y using X, since we can then avoid giving the drug in question to patients who are at high risk of an adverse reaction—that is, patients for whom the estimate of Y is high.

The accuracy of $\hat{Y}$ as a prediction for Y depends on two quantities, which we will call the *reducible error* and the *irreducible error*. In general, $\hat{f}$ will not be a perfect estimate for f, and this inaccuracy will introduce some error. This error is *reducible* because we can potentially improve the accuracy of $\hat{f}$ by using the most appropriate statistical learning technique to estimate f. However, even if it were possible to form a perfect estimate for f, so that our estimated response took the form $\hat{Y} = f(X)$, our prediction would still have some error in it! This is because Y is also a function of ″, which, by defnition, cannot be predicted using X. Therefore, variability associated with ″ also afects the accuracy of our predictions. This is known as the *irreducible* error, because no matter how well we estimate f, we cannot reduce the error introduced by ″. Why is the irreducible error larger than zero? The quantity ″ may con tain unmeasured variables that are useful in predicting Y: since we don't

reducible
error
irreducible error

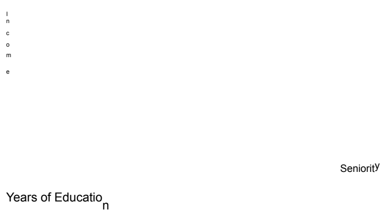


FIGURE 2.3. *The plot displays* income *as a function of* years of education *and* seniority *in the* Income *data set. The blue surface represents the true underlying*

relationship between income *and* years of education *and* seniority*, which is known since the data are simulated. The red dots indicate the observed values of these quantities for* 30 *individuals.*

measure them, *f* cannot use them for its prediction. The quantity ″ may also contain unmeasurable variation. For example, the risk of an adverse reaction might vary for a given patient on a given day, depending on manufacturing variation in the drug itself or the patient's general feeling of well-being on that day.

Consider a given estimate $\hat{f}$ and a set of predictors *X*, which yields the prediction $\hat{Y} = \hat{f}(X)$. Assume for a moment that both $\hat{f}$ and *X* are fxed, so that the only variability comes from ″. Then, it is easy to show that

$$E(Y - \hat{Y})^2 = E[f(X) + ″ - \hat{f}(X)]^2$$

$$= [f(X) - \hat{f}(X)]^2 + \mathrm{Var}(″) \;, -. / \qquad , (2.3)$$

, -. / Reducible Irreducible

where $E(Y - \hat{Y})^2$ represents the average, or *expected value*, of the squared diference between the predicted and actual value of *Y* , and Var(″)

expected value

repre sents the *variance* associated with the error term ″.

variance

The focus of this book is on techniques for estimating *f* with the aim of minimizing the reducible error. It is important to keep in mind that the irreducible error will always provide an upper bound on the accuracy of our prediction for *Y* . This bound is almost always unknown in practice.

Inference

We are often interested in understanding the association between *Y* and $X_1,...,X_p$. In this situation we wish to estimate *f*, but our goal is not necessarily to make predictions for *Y* . Now $\hat{f}$ cannot be treated as a black box, because we need to know its exact form. In this setting, one may be interested in answering the following questions:

- *Which predictors are associated with the response?* It is often the case that only a small fraction of the available predictors are substantially associated with *Y* . Identifying the few *important* predictors among a large set of possible variables can be extremely useful, depending on the application.

- *What is the relationship between the response and each predictor?* Some predictors may have a positive relationship with *Y* , in the sense that larger values of the predictor are associated with larger values of *Y* . Other predictors may have the opposite relationship. Depending on the complexity of *f*, the relationship between the response and a given predictor may also depend on the values of the other predictors.

- *Can the relationship between Y and each predictor be adequately sum marized using a linear equation, or is the relationship more compli cated?* Historically, most methods for estimating *f* have taken a linear form. In some situations, such an assumption is reasonable or even de sirable. But often the true relationship is more complicated, in which case a linear model may not provide an accurate representation of the relationship between the input and output variables.

In this book, we will see a number of examples that fall into the prediction setting, the inference setting, or a combination of the two. For instance, consider a company that is interested in conducting a direct-marketing campaign. The goal is to identify individuals who are likely to respond positively to a mailing, based on observations of demographic variables measured on each individual. In this case, the demographic variables serve as predictors, and response to the marketing cam paign (either positive or negative) serves as the outcome. The company is not interested in obtaining a deep understanding of the relationships be tween each individual predictor and the response; instead, the company simply wants to accurately predict the response using the predictors. This is an example of modeling for prediction.

In contrast, consider the Advertising data illustrated in Figure 2.1. One may be interested in answering questions such as:

– *Which media are associated with sales?*

– *Which media generate the biggest boost in sales?* or

– *How large of an increase in sales is associated with a given increase in TV advertising?*

This situation falls into the inference paradigm. Another example involves modeling the brand of a product that a customer might purchase based on variables such as price, store location, discount levels, competition price, and so forth. In this situation one might really be most interested in the association between each variable and the probability of purchase. For in stance, *to what extent is the product's price associated with sales?* This is an example of modeling for inference.

Finally, some modeling could be conducted both for prediction and inference. For example, in a real estate setting, one may seek to relate values of homes to inputs such as crime rate, zoning, distance from a river, air quality, schools, income level of community, size of houses, and so forth. In this case one might be interested in the association between each individ ual input variable and housing price—for instance, *how much extra will a house be worth if it has a view of the river?* This is an inference problem. Alternatively, one may simply be interested in predicting the value of a home given its characteristics: *is this house under- or over-valued?* This is a prediction problem.

Depending on whether our ultimate goal is prediction, inference, or a combination of the two, diferent methods for estimating *f* may be ap propriate. For example, *linear models* allow for relatively simple and interpretable inference, but

linear model

may not yield as accurate predictions as some other approaches. In contrast, some of the highly non-linear approaches that we discuss in the later chapters of this book can potentially provide quite accurate predictions for Y , but this comes at the expense of a less interpretable model for which inference is more challenging.

2.1.2 How Do We Estimate f?

Throughout this book, we explore many linear and non-linear approaches for estimating *f*. However, these methods generally share certain charac teristics. We provide an overview of these shared characteristics in this section. We will always assume that we have observed a set of *n* diferent data points. For example in Figure 2.2 we observed n = 30 data points. These observations are called the *training data* because we will use these training

data observations to train, or teach, our method how to estimate *f*. Let x_{ij} represent the value of the *j*th predictor, or input, for observation *i*, where $i = 1, 2,...,n$ and $j = 1, 2,...,p$. Correspondingly, let y_i represent the response variable for the *i*th observation. Then our training data consist of $\{(x_1, y_1),(x_2, y_2),...,(x_n, y_n)\}$ where $x_i = (x_{i1}, x_{i2},...,x_{ip})^T$.

Our goal is to apply a statistical learning method to the training data in order to estimate the unknown function *f*. In other words, we want to fnd a function $\hat{f}$ such that $Y \approx \hat{f}(X)$ for any observation (X, Y). Broadly speaking, most statistical learning methods for this task can be character ized as either *parametric* or *non-parametric*. We now briefy discuss these parametric two types of approaches.

Parametric Methods

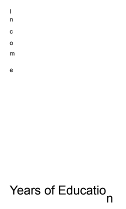
Parametric methods involve a two-step model-based approach.

1. First, we make an assumption about the functional form, or shape, of *f*. For example, one very simple assumption is that *f* is linear in *X*:

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p. \quad (2.4)$$

This is a *linear model*, which will be discussed extensively in Chap ter 3. Once we have assumed that *f* is linear, the problem of estimat ing *f* is greatly simplifed. Instead of having to estimate an entirely arbitrary *p*-dimensional function $f(X)$, one only needs to estimate the p + 1 coefcients $\beta_0, \beta_1,..., \beta_p$.

non parametric



Seniority

FIGURE 2.4. *A linear model ft by least squares to the* Income *data from Figure 2.3. The observations are shown in red, and the yellow plane indicates the least squares ft to the data.*

2. After a model has been selected, we need a procedure that uses the training data to *ft* or *train* the model. In the case of the linear model ft train (2.4), we need to estimate the parameters $\beta_0, \beta_1, ..., \beta_p$. That is, we want to fnd values of these parameters such that

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p.$$

The most common approach to ftting the model (2.4) is referred to as *(ordinary)*

least squares

least squares, which we discuss in Chapter 3. However,

least squares is one of many possible ways to ft the linear model. In Chapter 6, we discuss other approaches for estimating the parameters in (2.4).

The model-based approach just described is referred to as *parametric*; it reduces the problem of estimating f down to one of estimating a set of parameters. Assuming a parametric form for f simplifes the problem of estimating f because it is generally much easier to estimate a set of pa rameters, such as $\beta_0, \beta_1, ..., \beta_p$ in the linear model (2.4), than it is to ft an entirely arbitrary function f. The potential disadvantage of a paramet ric approach is that the model we choose will usually not match the true unknown form of f. If the chosen model is too far from the true f, then our estimate will be poor. We can try to address this problem by choos ing *fexible* models that can ft many diferent

fexible

possible functional forms for f. But in general, ftting a more fexible model requires estimating a

greater number of parameters. These more complex models can lead to a phenomenon known as *overftting* the data, which essentially means they

overftting

follow the errors, or *noise*, too closely. These issues are discussed

noise

through- out this book. Figure 2.4 shows an example of the parametric approach applied to the Income data from Figure 2.3. We have ft a linear model of the form

$$\text{income} \approx \beta_0 + \beta_1 \times \text{education} + \beta_2 \times \text{seniority}.$$

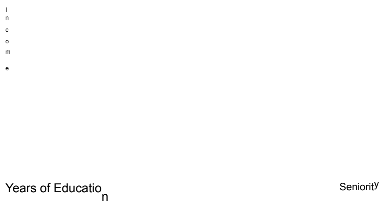


FIGURE 2.5. *A smooth thin-plate spline fit to the* Income *data from Figure 2.3 is shown in yellow; the observations are displayed in red. Splines are discussed in Chapter 7.*

Since we have assumed a linear relationship between the response and the two predictors, the entire fitting problem reduces to estimating β_0, β_1, and β_2, which we do using least squares linear regression. Comparing Figure 2.3 to Figure 2.4, we can see that the linear fit given in Figure 2.4 is not quite right: the true f has some curvature that is not captured in the linear fit. However, the linear fit still appears to do a reasonable job of capturing the positive relationship between years of education and income, as well as the slightly less positive relationship between seniority and income. It may be that with such a small number of observations, this is the best we can do.

Non-Parametric Methods

Non-parametric methods do not make explicit assumptions about the func tional form of f. Instead they seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly. Such approaches can have a major advantage over parametric approaches: by avoiding the assumption of a particular functional form for f, they have the potential to accurately fit a wider range of possible shapes for f. Any parametric approach brings with it the possibility that the functional form used to estimate f is very different from the true f, in which case the resulting model will not fit the data well. In contrast, non-parametric approaches completely avoid this danger, since essentially no assumption about the form of f is made. But non-parametric approaches do suffer from a major disadvantage: since they do not reduce the problem of estimating f to a small number of parameters, a very large number of observations (far more than is typically needed for a parametric approach) is required in order to obtain an accurate estimate for f.

An example of a non-parametric approach to fitting the Income data is shown in Figure 2.5. A *thin-plate spline* is used to estimate f. This approach does not impose any pre-specified model on f. It instead attempts

thin-plate spline

FIGURE 2.6. *A rough thin-plate spline fit to the* Income *data from Figure 2.3. This fit makes zero errors on the training data.*

to produce an estimate for *f* that is as close as possible to the observed data, subject to the fit—that is, the yellow surface in Figure 2.5—being *smooth*. In this case, the non-parametric fit has produced a remarkably ac curate estimate of the true *f* shown in Figure 2.3. In order to fit a thin-plate spline, the data analyst must select a level of smoothness. Figure 2.6 shows the same thin-plate spline fit using a lower level of smoothness, allowing for a rougher fit. The resulting estimate fits the observed data perfectly! However, the spline fit shown in Figure 2.6 is far more variable than the true function *f*, from Figure 2.3. This is an example of overfitting the data, which we discussed previously. It is an undesirable situation because the fit obtained will not yield accurate estimates of the response on new observations that were not part of the original training data set. We dis cuss methods for choosing the *correct* amount of smoothness in Chapter 5. Splines are discussed in Chapter 7.

As we have seen, there are advantages and disadvantages to parametric and non-parametric methods for statistical learning. We explore both types of methods throughout this book.

2.1.3 The Trade-Off Between Prediction Accuracy and Model Interpretability

Of the many methods that we examine in this book, some are less flexible, or more restrictive, in the sense that they can produce just a relatively small range of shapes to estimate *f*. For example, linear regression is a relatively inflexible approach, because it can only generate linear functions such as the lines shown in Figure 2.1 or the plane shown in Figure 2.4. Other methods, such as the thin plate splines shown in Figures 2.5 and 2.6, are considerably more fexible because they can generate a much wider range of possible shapes to estimate *f*.

Subset

Selection

Trees

y t il b a t e r pr e t n I

Bagging, Boosting

Support Vector Machines
Deep Learning

w o L

Lasso

g i H

Low High Flexibility

Least Squares

Generalized Additive Models

FIGURE 2.7. *A representation of the tradeof between fexibility and inter pretability, using diferent statistical learning methods. In general, as the fexibility of a method increases, its interpretability decreases.*

One might reasonably ask the following question: *why would we ever choose to use a more restrictive method instead of a very fexible approach?* There are several reasons that we might prefer a more restrictive model. If we are mainly interested in inference, then restrictive models are much more interpretable. For instance, when inference is the goal, the linear model may be a good choice since it will be quite easy to understand the relationship between Y and X_1, X_2,...,X_p. In contrast, very fexible approaches, such as the splines discussed in Chapter 7 and displayed in Figures 2.5 and 2.6, and the boosting methods discussed in Chapter 8, can lead to such complicated estimates of *f* that it is difcult to understand how any individual predictor is associated with the response.

Figure 2.7 provides an illustration of the trade-of between fexibility and interpretability for some of the methods that we cover in this book. Least squares linear regression, discussed in Chapter 3, is relatively infexible but is

lasso

quite interpretable. The *lasso*, discussed in Chapter 6, relies upon the linear model (2.4) but uses an alternative ftting procedure for estimating the coefcients β_0, β_1,..., β_p. The new procedure is more restrictive in es timating the coefcients, and sets a number of them to exactly zero. Hence in this sense the lasso is a less fexible approach than linear regression. It is also more interpretable than linear regression, because in the fnal model the response variable will only be related to a small subset of the predictors—namely, those with nonzero coefcient estimates. *Generalized additive models* (GAMs), discussed in Chapter 7, instead extend the lin-

generalized

ear model (2.4) to allow for certain non-linear relationships. Consequently, GAMs are more fexible than linear regression. They are also somewhat less interpretable than linear regression, modeled using a curve. Finally, because the relationship between each additive model predictor and the response is now

fully non-linear methods such as *bagging*, *boosting*, *support vector machines* bagging boosting with non-linear kernels, and *neural networks* (deep learning), discussed in Chapters 8, 9, and 10, are highly fexible machine approaches that are harder to interpret.

support vector

We have established that when inference is the goal, there are clear ad vantages to using simple and relatively infexible statistical learning meth ods. In some settings, however, we are only interested in prediction, and the interpretability of the predictive model is simply not of interest. For instance, if we seek to develop an algorithm to predict the price of a stock, our sole requirement for the algorithm is that it predict accurately— interpretability is not a concern. In this setting, we might expect that it will be best to use the most fexible model available. Surprisingly, this is not always the case! We will often obtain more accurate predictions using a less fexible method. This phenomenon, which may seem counterintu itive at frst glance, has to do with the potential for overftting in highly fexible methods. We saw an example of overftting in Figure 2.6. We will discuss this very important concept further in Section 2.2 and throughout this book.

2.1.4 Supervised Versus Unsupervised Learning

Most statistical learning problems fall into one of two categories: *supervised* supervised or *unsupervised*. The examples that we have discussed so far in this chap-

unsupervised

ter all fall into the supervised learning domain. For each observation of the predictor measurement(s) x_i, $i = 1,...,n$ there is an associated response measurement y_i. We wish to ft a model that relates the response to the predictors, with the aim of accurately predicting the response for future observations (prediction) or better understanding the relationship between the response and the predictors (inference). Many classical statistical learn ing methods such as linear regression and *logistic regression* (Chapter 4), as logistic well as more modern approaches such as GAM, boosting, and support vec- regression tor machines, operate in the supervised learning domain. The vast majority of this book is devoted to this setting.

By contrast, unsupervised learning describes the somewhat more chal lenging situation in which for every observation $i = 1,...,n$, we observe a vector of measurements x_i but no associated response y_i. It is not pos sible to ft a linear regression model, since there is no response variable to predict. In this setting, we are in some sense working blind; the sit uation is referred to as *unsupervised* because we lack a response vari able that can supervise our analysis. What sort of statistical analysis is possible? We can seek to understand the relationships between the variables or between the observations. One statistical learning tool that we may use in this setting is *cluster analysis*, or clustering. The goal of cluster analysis cluster analysis is to ascertain, on the basis of $x_1,...,x_n$, whether the observations fall into relatively distinct groups. For example, in a market segmentation study we might observe multiple characteristics (variables) for potential customers, such as zip code, family income, and shopping habits. We might believe that the customers fall into diferent groups, such as big spenders versus low spenders. If the information about each customer's spending patterns were available, then a supervised analysis would be possible. However, this information is not available—that is, we do not know whether each poten tial customer is a big spender or not. In this setting, we can try to cluster the customers on the basis of the variables measured, in order to identify distinct groups of potential customers. Identifying such groups can be of interest because it might be that the groups difer with respect to some property of interest, such as spending habits.

FIGURE 2.8. *A clustering data set involving three groups. Each group is shown using a diferent colored symbol.* Left: *The three groups are well-separated. In this setting, a clustering approach should successfully identify the three groups.* Right: *There is some overlap among the groups. Now the clustering task is more challenging.*

Figure 2.8 provides a simple illustration of the clustering problem. We have plotted 150 observations with measurements on two variables, X_1 and X_2. Each observation corresponds to one of three distinct groups. For illustrative purposes, we have plotted the members of each group using diferent colors and symbols. However, in practice the group memberships are unknown, and the goal is to determine the group to which each obser vation belongs. In the left-hand panel of Figure 2.8, this is a relatively easy task because the groups are well-separated. By contrast, the right-hand panel illustrates a more challenging setting in which there is some overlap between the groups. A clustering method could not be expected to assign all of the overlapping points to their correct group (blue, green, or orange).

In the examples shown in Figure 2.8, there are only two variables, and so one can simply visually inspect the scatterplots of the observations in order to identify clusters. However, in practice, we often encounter data sets that contain many more than two variables. In this case, we cannot easily plot the observations. For instance, if there are p variables in our data set, then $p(p - 1)/2$ distinct scatterplots can be made, and visual inspection is simply not a viable way to identify clusters. For this reason, automated clustering methods are important. We discuss clustering and other unsupervised learning approaches in Chapter 12.

Many problems fall naturally into the supervised or unsupervised learning paradigms. However, sometimes the question of whether an analysis should be considered supervised or unsupervised is less clear-cut. For instance, suppose that we have a set of n observations. For m of the observations, where $m<n$, we have both predictor measurements and a response measurement. For the remaining $n - m$ observations, we have predictor measurements but no response measurement. Such a scenario can arise if the predictors can be measured relatively cheaply but the corresponding responses are much more expensive to collect. We refer to this setting as a *semi-supervised learning* problem. In this setting, we wish to use a statistical learning method that can incorporate the m observations for which response measurements are available as well as the $n - m$ observations for which they are not. Although this is an interesting topic, it is beyond the scope of this book.

semi

supervised learning

2.1.5 Regression Versus Classifcation Problems

Variables can be characterized as either *quantitative* or *qualitative* (also known as *categorical*). Quantitative variables take on numerical values. Examples include a person's age, height, or income, the value of a house, and the price of a stock. In contrast, qualitative variables take on values in one of K diferent *classes*, or categories. Examples of qualitative variables include a person's marital status (married or not), the brand of product purchased (brand A, B, or C), whether a person defaults on a debt (yes or no), or a cancer diagnosis (Acute Myelogenous Leukemia, Acute Lymphoblastic Leukemia, or No Leukemia). We tend to refer to problems with a quantitative response as *regression* problems, while those involving a qualitative response are often referred to as *classifcation* problems. However, the distinction is not always that crisp. Least squares linear regression (Chapter 3) is used with a quantitative response, whereas

quantitative

qualitative

categorical

class

regression

classifcation

logistic regression (Chapter 4) is typically used with a qualitative (two-class, or *binary*) response. Thus, despite its name, logistic regression is a classification method. But since it estimates class probabilities, it can be thought of as a regression method as well. Some statistical methods, such as *K*-nearest neighbors (Chapters 2 and 4) and boosting (Chapter 8), can be used in the case of either quantitative or qualitative responses.

binary

We tend to select statistical learning methods on the basis of whether the response is quantitative or qualitative; i.e. we might use linear regression when quantitative and logistic regression when qualitative. However, whether the *predictors* are qualitative or quantitative is generally considered less important. Most of the statistical learning methods discussed in this book can be applied regardless of the predictor variable type, provided that any qualitative predictors are properly *coded* before the analysis is performed. This is discussed in Chapter 3.

2.2 Assessing Model Accuracy

One of the key aims of this book is to introduce the reader to a wide range of statistical learning methods that extend far beyond the standard linear regression approach. Why is it necessary to introduce so many different statistical learning approaches, rather than just a single *best* method? *There is no free lunch in statistics:* no one method dominates all others over all possible data sets. On a particular data set, one specific method may work

best, but some other method may work better on a similar but different data set. Hence it is an important task to decide for any given set of data which method produces the best results. Selecting the best approach can be one of the most challenging parts of performing statistical learning in practice.

In this section, we discuss some of the most important concepts that arise in selecting a statistical learning procedure for a specific data set. As the book progresses, we will explain how the concepts presented here can be applied in practice.

2.2.1 Measuring the Quality of Fit

In order to evaluate the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions actually match the observed data. That is, we need to quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation. In the regression setting, the most commonly-used measure is the *mean squared error* (MSE), given by

mean squared error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2, \quad (2.5)$$

where $\hat{f}(x_i)$ is the prediction that $\hat{f}$ gives for the *i*th observation. The MSE will be small if the predicted responses are very close to the true responses, and will be large if for some of the observations, the predicted and true responses difer substantially.

The MSE in (2.5) is computed using the training data that was used to ft the model, and so should more accurately be referred to as the *training MSE*. But in general, we do not really care how well the method works training MSE on the training data. Rather, *we are interested in the accuracy of the pre dictions that we obtain when we apply our method to previously unseen test data*. Why is this what we care about? test data Suppose that we are interested in developing an algorithm to predict a stock's price based on previous stock returns. We can train the method using stock returns from the past 6 months. But we don't really care how well our method predicts last week's stock price. We instead care about how well it will predict tomorrow's price or next month's price. On a similar note, suppose that we have clinical measurements (e.g. weight, blood pressure, height, age, family history of disease) for a number of patients, as well as information about whether each patient has diabetes. We can use these patients to train a statistical learn ing method to predict risk of diabetes based on clinical measurements. In practice, we want this method to accurately predict diabetes risk for *future patients* based on their clinical measurements. We are not very interested in whether or not the method accurately predicts diabetes risk for patients used to train the model, since we already know which of those patients have diabetes.

To state it more mathematically, suppose that we ft our statistical learn ing method on our training observations $\{(x_1, y_1),(x_2, y_2),\ldots,(x_n, y_n)\}$, and we obtain the estimate $\hat{f}$. We can then compute $\hat{f}(x_1), \hat{f}(x_2),\ldots, \hat{f}(x_n)$.

o r r E 5 . d 1 8

Y

5 . 2 2 1

e r a u q S 0 . 6 1 n a e M 4

0 . 2 0

r 1

5

0

X

2

0

0

2 5 10 20 Flexibility

0 20 40 60 80 100

FIGURE 2.9. Left: *Data simulated from f, shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fts (blue and green curves).* Right: *Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fts shown in the left-hand panel.*

If these are approximately equal to y_1, y_2,...,y_n, then the training MSE given by (2.5) is small. However, we are really not interested in whether $\hat{f}(x_i) \approx y_i$; instead, we want to know whether $\hat{f}(x_0)$ is approximately equal to y_0, where (x_0, y_0) is a *previously unseen test observation not used to train the statistical learning method*. We want to choose the method that gives the lowest *test MSE*, as opposed to the lowest training MSE. In other words, test MSE if we had a large number of test observations, we could compute

$$\mathrm{Ave}(y_0 - \hat{f}(x_0))^2, \quad (2.6)$$

the average squared prediction error for these test observations (x_0, y_0). We'd like to select the model for which this quantity is as small as possible. How can we go about trying to select a method that minimizes the test MSE? In some settings, we may have a test data set available—that is, we may have access to a set of observations that were not used to train the statistical learning method. We can then simply evaluate (2.6) on the test observations, and select the learning method for which the test MSE is smallest. But what if no test observations are available? In that case, one might imagine simply selecting a statistical learning method that minimizes the training MSE (2.5). This seems like it might be a sensible approach, since the training MSE and the test MSE appear to be closely related. Unfortunately, there is a fundamental problem with this strategy: there is no guarantee that the method with the lowest training MSE will also have the lowest test MSE. Roughly speaking, the problem is that many statistical methods specifcally estimate coefcients so as to minimize the training set MSE. For these methods, the training set MSE can be quite small, but the test MSE is often much larger.

Figure 2.9 illustrates this phenomenon on a simple example. In the left hand panel of Figure 2.9, we have generated observations from (2.1) with

the true *f* given by the black curve. The orange, blue and green curves illus trate three possible estimates for *f* obtained using methods with increasing levels of fexibility. The orange line is the linear regression ft, which is rela tively infexible.

The blue and green curves were produced using *smoothing splines*, discussed in Chapter 7, with different levels of smoothness. It is smoothing spline clear that as the level of flexibility increases, the curves fit the observed data more closely. The green curve is the most flexible and matches the data very well; however, we observe that it fits the true *f* (shown in black) poorly because it is too wiggly. By adjusting the level of flexibility of the smoothing spline fit, we can produce many different fits to this data.

We now move on to the right-hand panel of Figure 2.9. The grey curve displays the average training MSE as a function of flexibility, or more formally the *degrees of freedom*, for a number of smoothing splines. The degrees of freedom degrees of freedom is a quantity that summarizes the flexibility of a curve; it is discussed more fully in Chapter 7. The orange, blue and green squares indicate the MSEs associated with the corresponding curves in the left hand panel. A more restricted and hence smoother curve has fewer degrees of freedom than a wiggly curve—note that in Figure 2.9, linear regression is at the most restrictive end, with two degrees of freedom. The training MSE declines monotonically as flexibility increases. In this example the true *f* is non-linear, and so the orange linear fit is not flexible enough to estimate *f* well. The green curve has the lowest training MSE of all three methods, since it corresponds to the most flexible of the three curves fit in the left-hand panel.

In this example, we know the true function *f*, and so we can also compute the test MSE over a very large test set, as a function of flexibility. (Of course, in general *f* is unknown, so this will not be possible.) The test MSE is displayed using the red curve in the right-hand panel of Figure 2.9. As with the training MSE, the test MSE initially declines as the level of flex ibility increases. However, at some point the test MSE levels off and then starts to increase again. Consequently, the orange and green curves both have high test MSE. The blue curve minimizes the test MSE, which should not be surprising given that visually it appears to estimate *f* the best in the left-hand panel of Figure 2.9. The horizontal dashed line indicates Var(″), the irreducible error in (2.3), which corresponds to the lowest achievable test MSE among all possible methods. Hence, the smoothing spline represented by the blue curve is close to optimal.

In the right-hand panel of Figure 2.9, as the flexibility of the statistical learning method increases, we observe a monotone decrease in the training MSE and a *U-shape* in the test MSE. This is a fundamental property of statistical learning that holds regardless of the particular data set at hand and regardless of the statistical method being used. As model flexibility increases, the training MSE will decrease, but the test MSE may not. When a given method yields a small training MSE but a large test MSE, we are said to be *overfitting* the data. This happens because our statistical learning procedure is working too hard to find patterns in the training data, and may be picking up some patterns that are just caused by random chance rather than by true properties of the unknown function *f*. When we overfit the training data, the test MSE will be very large because the supposed

8

Y

2

1

0

1

5

·

2

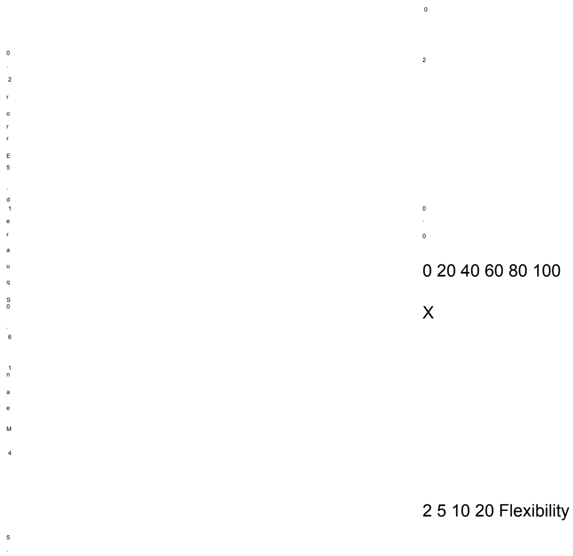


FIGURE 2.10. *Details are as in Figure 2.9, using a diferent true f that is much closer to linear. In this setting, linear regression provides a very good ft to the data.*

patterns that the method found in the training data simply don't exist in the test data. Note that regardless of whether or not overftting has occurred, we almost always expect the training MSE to be smaller than the test MSE because most statistical learning methods either directly or indirectly seek to minimize the training MSE. Overftting refers specifcally to the case in which a less fexible model would have yielded a smaller test MSE.

Figure 2.10 provides another example in which the true *f* is approxi mately linear. Again we observe that the training MSE decreases mono tonically as the model fexibility increases, and that there is a U-shape in the test MSE. However, because the truth is close to linear, the test MSE only decreases slightly before increasing again, so that the orange least squares ft is substantially better than the highly fexible green curve. Fi nally, Figure 2.11 displays an example in which *f* is highly non-linear. The training and test MSE curves still exhibit the same general patterns, but now there is a rapid decrease in both curves before the test MSE starts to increase slowly.

In practice, one can usually compute the training MSE with relative ease, but estimating the test MSE is considerably more difcult because usually no test data are available. As the previous three examples illustrate, the fexibility level corresponding to the model with the minimal test MSE can vary considerably among data sets. Throughout this book, we discuss a variety of approaches that can be used in practice to estimate this minimum point. One important method is *cross-validation* (Chapter 5), which is a cross method for estimating the test MSE using the training data. validation

2.2.2 The Bias-Variance Trade-Of

The U-shape observed in the test MSE curves (Figures 2.9–2.11) turns out to be the result of two competing properties of statistical learning methods.

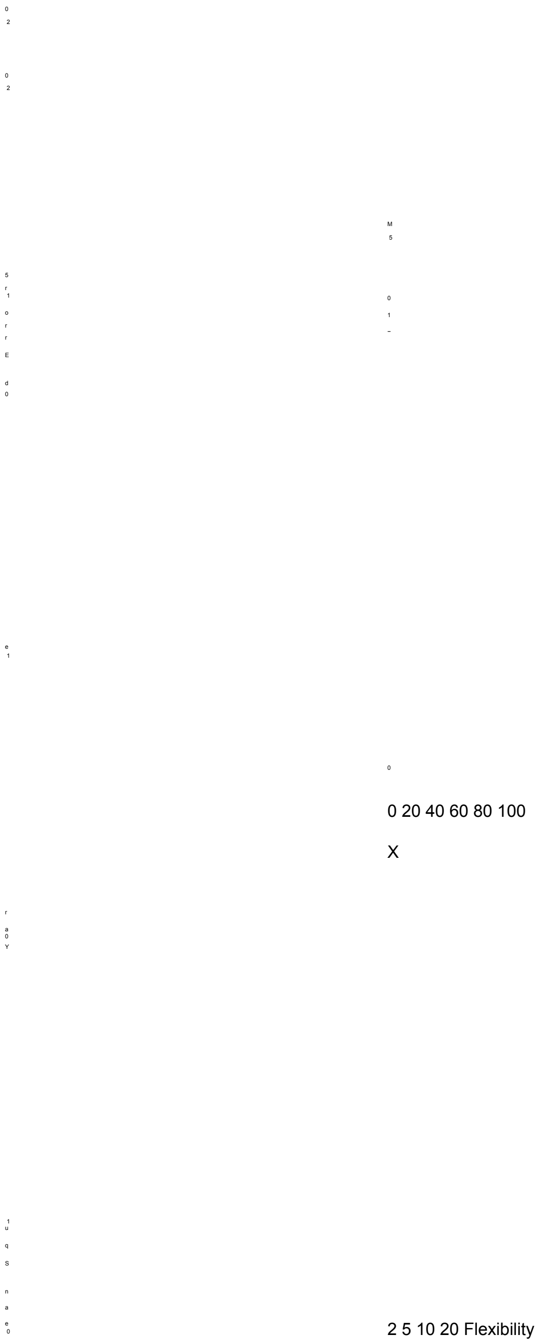


FIGURE 2.11. *Details are as in Figure 2.9, using a diferent f that is far from linear. In this setting, linear regression provides a very poor ft to the data.*

Though the mathematical proof is beyond the scope of this book, it is possible to show that the expected test MSE, for a given value x_0, can always be decomposed into the sum of three fundamental quantities: the *variance* of $\hat{f}(x_0)$, the squared *bias* of $\hat{f}(x_0)$ and the variance of the error variance

bias terms ″. That is,

1

E

$y_0 - \hat{f}(x_0)2^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}($

$\hat{f}(x_0))]^2 + \text{Var}(″).$ (2.7)

1

Here the notation *E*

$y_0 - \hat{f}(x_0)2^2$ defnes the *expected test MSE* at x_0, expected

and refers to the average test MSE that we would obtain if we repeatedly test MSE estimated f using a large number of training sets, and tested each at x_0. The

1

overall expected test MSE can be computed by averaging *E* over all

possible values of x_0 in the test set.

$y_0 - \hat{f}(x_0)2_2$

Equation 2.7 tells us that in order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves *low variance* and *low bias*. Note that variance is inherently a nonnegative quantity, and squared bias is also nonnegative. Hence, we see that the expected test MSE can never lie below Var(″), the irreducible error from (2.3).

What do we mean by the *variance* and *bias* of a statistical learning method? *Variance* refers to the amount by which $\hat{f}$ would change if we estimated it using a diferent training data set. Since the training data are used to ft the statistical learning method, diferent training data sets will result in a diferent $\hat{f}$. But ideally the estimate for f should not vary too much between training sets. However, if a method has high variance then small changes in the training data can result in large changes in $\hat{f}$. In general, more fexible statistical methods have higher variance. Consider the green and orange curves in Figure 2.9. The fexible green curve is following the observations very closely. It has high variance because changing any one of these data points may cause the estimate $\hat{f}$ to change considerably.

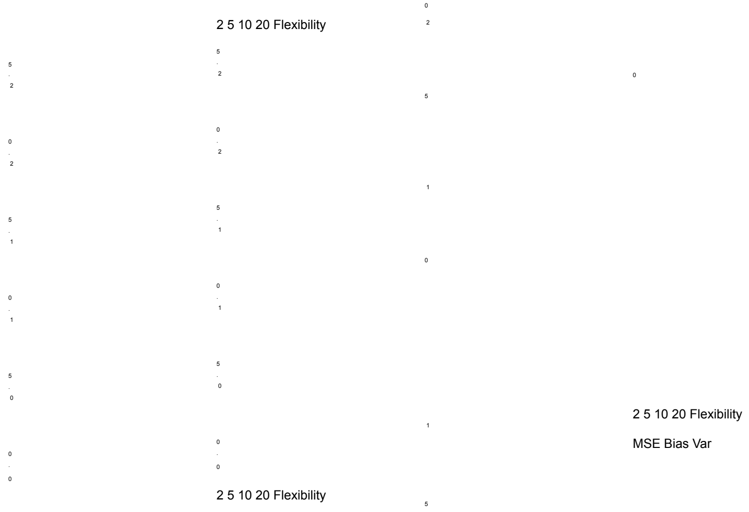


FIGURE 2.12. *Squared bias (blue curve), variance (orange curve), Var(!) (dashed line), and test MSE (red curve) for the three data sets in Figures 2.9–2.11. The vertical dotted line indicates the fexibility level corresponding to the smallest test MSE.*

In contrast, the orange least squares line is relatively infexible and has low variance, because moving any single observation will likely cause only a small shift in the position of the line.

On the other hand, *bias* refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. For example, linear regression assumes that there is a linear relationship between Y and X_1, X_2,...,X_p. It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing lin ear regression will undoubtedly result in some bias in the estimate of f. In Figure 2.11, the true f is substantially non-linear, so no matter how many training observations we are given, it will not be possible to produce an accurate estimate using linear regression. In other words, linear regression results in high bias in this example. However, in Figure 2.10 the true f is very close to linear, and so given enough data, it should be possible for linear regression to produce an accurate estimate. Generally, more fexible methods result in less bias.

As a general rule, as we use more fexible methods, the variance will increase and the bias will decrease. The relative rate of change of these two quantities determines whether the test MSE increases or decreases. As we increase the fexibility of a class of methods, the bias tends to initially decrease faster than the variance increases. Consequently, the expected test MSE declines. However, at some point increasing fexibility has little impact on the bias but starts to signifcantly increase the variance. When this happens the test MSE increases. Note that we observed this pattern of decreasing test MSE followed by increasing test MSE in the right-hand panels of Figures 2.9–2.11.

The three plots in Figure 2.12 illustrate Equation 2.7 for the examples in Figures 2.9–2.11. In each case the blue solid curve represents the squared bias, for diferent levels of fexibility, while the orange curve corresponds to the variance. The horizontal dashed line represents Var(″), the irreducible error. Finally, the red curve, corresponding to the test set MSE, is the sum

of these three quantities. In all three cases, the variance increases and the bias decreases as the method's fexibility increases. However, the fexibility level corresponding to the optimal test MSE difers considerably among the three data sets, because the squared bias and variance change at different rates in each of the data sets. In the left-hand panel of Figure 2.12, the bias initially decreases rapidly, resulting in an initial sharp decrease in the expected test MSE. On the other hand, in the center panel of Figure 2.12 the true *f* is close to linear, so there is only a small decrease in bias as flexibility increases, and the test MSE only declines slightly before increasing rapidly as the variance increases. Finally, in the right-hand panel of Figure 2.12, as flexibility increases, there is a dramatic decline in bias because the true *f* is very non-linear. There is also very little increase in variance as flexibility increases. Consequently, the test MSE declines substantially before experiencing a small increase as model flexibility increases.

The relationship between bias, variance, and test set MSE given in Equation 2.7 and displayed in Figure 2.12 is referred to as the *bias-variance trade-off*. Good test set performance of a statistical learning method requires low variance as well as low squared bias. This is referred to as a trade-off because it is easy to obtain a method with extremely low bias but high variance (for instance, by drawing a curve that passes through every single training observation) or a method with very low variance but high bias (by fitting a horizontal line to the data). The challenge lies in finding a method for which both the variance and the squared bias are low. This trade-off is one of the most important recurring themes in this book.

bias-variance trade-off

In a real-life situation in which *f* is unobserved, it is generally not possible to explicitly compute the test MSE, bias, or variance for a statistical learning method. Nevertheless, one should always keep the bias-variance trade-off in mind. In this book we explore methods that are extremely flexible and hence can essentially eliminate bias. However, this does not guarantee that they will outperform a much simpler method such as linear regression. To take an extreme example, suppose that the true *f* is linear. In this situation linear regression will have no bias, making it very hard for a more flexible method to compete. In contrast, if the true *f* is highly non-linear and we have an ample number of training observations, then we may do better using a highly flexible approach, as in Figure 2.11. In Chapter 5 we discuss cross-validation, which is a way to estimate the test MSE using the training data.

2.2.3 The Classification Setting

Thus far, our discussion of model accuracy has been focused on the regression setting. But many of the concepts that we have encountered, such as the bias-variance trade-off, transfer over to the classification setting with only some modifications due to the fact that y_i is no longer quantitative. Suppose that we seek to estimate *f* on the basis of training observations $\{(x_1, y_1),...,(x_n, y_n)\}$, where now $y_1,...,y_n$ are qualitative. The most common approach for quantifying the accuracy of our estimate $\hat{f}$ is the training *error rate*, the proportion of mistakes that are made if we apply

error rate

our estimate $\hat{f}$ to the training observations:

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i). \qquad (2.8)$$

Here $\hat{y}_i$ is the predicted class label for the ith observation using $\hat{f}$. And $I(y_i \%= \hat{y}_i)$ is an *indicator variable* that equals 1 if $y_i \%= \hat{y}_i$ and zero if $y_i = \hat{y}_i$. indicator variable If $I(y_i \%= \hat{y}_i)$=0 then the ith observation was classified correctly by our classification method; otherwise it was misclassified. Hence Equation 2.8 computes the fraction of incorrect classifications.

Equation 2.8 is referred to as the *training error* rate because it is com- training error puted based on the data that was used to train our classifier. As in the regression setting, we are most interested in the error rates that result from applying our classifier to test observations that were not used in training. The *test error* rate

test error

associated with a set of test observations of the form (x_0, y_0) is given

by Ave $(I(y_0 = \hat{}\ \%\ y_0))$, (2.9)

where $\hat{y}_0$ is the predicted class label that results from applying the classifier to the test observation with predictor x_0. A *good* classifier is one for which the test error (2.9) is smallest.

The Bayes Classifier

It is possible to show (though the proof is outside of the scope of this book) that the test error rate given in (2.9) is minimized, on average, by a very simple classifier that *assigns each observation to the most likely class, given its predictor values*. In other words, we should simply assign a test observation with predictor vector x_0 to the class j for which

$$\Pr(Y = j|X = x_0) \quad (2.10)$$

is largest. Note that (2.10) is a *conditional probability*: it is the probability conditional probability that $Y = j$, given the observed predictor vector x_0. This very simple clas sifer is called the *Bayes classifier*. In a two-class problem where there are Bayes classifier only two possible response values, say *class 1* or *class 2*, the Bayes classifier corresponds to predicting class one if $\Pr(Y = 1|X = x_0) > 0.5$, and class two otherwise.

Figure 2.13 provides an example using a simulated data set in a two dimensional space consisting of predictors X_1 and X_2. The orange and blue circles correspond to training observations that belong to two different classes. For each value of X_1 and X_2, there is a different probability of the response being orange or blue. Since this is simulated data, we know how the data were generated and we can calculate the conditional probabilities for each value of X_1 and X_2. The orange shaded region reflects the set of points for which $\Pr(Y =$ orange$|X)$ is greater than 50 %, while the blue shaded region indicates the set of points for which the probability is below 50 %. The purple dashed line represents the points where the probability is exactly 50 %. This is called the *Bayes decision boundary*. The Bayes Bayes classifier's prediction is determined by the Bayes decision boundary; an observation that falls on the orange side

of the boundary will be assigned

decision boundary

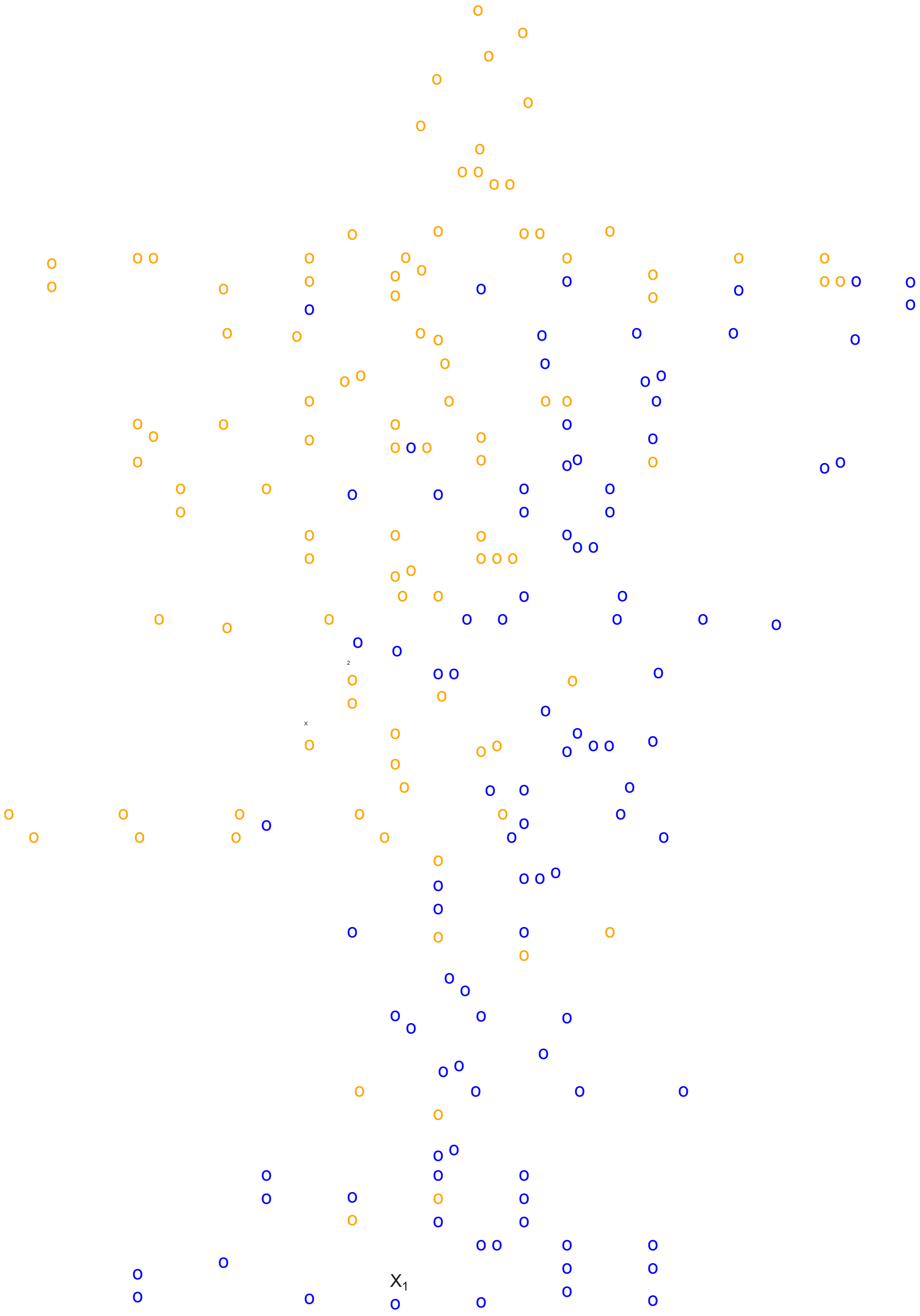


FIGURE 2.13. *A simulated data set consisting of* 100 *observations in each of two groups, indicated in blue and in orange. The purple dashed line represents the*

Bayes decision boundary. The orange background grid indicates the region in which a test observation will be assigned to the orange class, and the blue background grid indicates the region in which a test observation will be assigned to the blue class.

to the orange class, and similarly an observation on the blue side of the boundary will be assigned to the blue class.

The Bayes classifier produces the lowest possible test error rate, called the *Bayes error rate*. Since the Bayes classifier will always choose the class Bayes error rate for which (2.10) is largest, the error rate will be $1-\max_j \Pr(Y = j|X = x_0)$ at $X = x_0$. In general, the overall Bayes error rate is given by

$$1 - E_{*} \qquad \max_{j} \Pr(Y = j|X)^{+} \quad , (2.11)$$

where the expectation averages the probability over all possible values of X. For our simulated data, the Bayes error rate is 0.133. It is greater than zero, because the classes overlap in the true population, which implies that $\max_j \Pr(Y = j|X = x_0) < 1$ for some values of x_0. The Bayes error rate is analogous to the irreducible error, discussed earlier.

K-Nearest Neighbors

In theory we would always like to predict qualitative responses using the Bayes classifier. But for real data, we do not know the conditional distri bution of Y given X, and so computing the Bayes classifier is impossi ble. Therefore, the Bayes classifier serves as an unattainable gold standard against which to compare other methods. Many approaches attempt to estimate the conditional distribution of Y given X, and then classify a given observation to the class with highest *estimated* probability. One such method is the *K-nearest neighbors* (KNN) classifier. Given a positive in- *K*-nearest

neighbors

teger K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0, represented by N_0. It then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j:

$$\Pr(Y = j|X = x_0) = {}^{1}K0_{\,i\in N_0} \qquad I(y_i = j). \quad (2.12)$$

Finally, KNN classifies the test observation x_0 to the class with the largest probability from (2.12).

Figure 2.14 provides an illustrative example of the KNN approach. In the left-hand panel, we have plotted a small training data set consisting of six blue and six orange observations. Our goal is to make a prediction for the point labeled by the black cross. Suppose that we choose $K = 3$. Then KNN will first identify the three observations that are closest to the cross. This neighborhood is shown as a circle. It consists of two blue points and one orange point, resulting in estimated probabilities of 2/3 for the blue class and 1/3 for the orange class. Hence KNN will predict that the black cross belongs to the blue class. In the right-hand panel of Figure 2.14 we have applied the KNN approach with $K = 3$ at all of the possible values for X_1 and X_2, and have drawn in the corresponding KNN decision boundary.

Despite the fact that it is a very simple approach, KNN can often produce classifiers that are surprisingly close to the optimal Bayes classifier. Figure 2.15 displays the KNN decision boundary, using $K = 10$, when applied to the larger simulated data set from Figure 2.13. Notice that even though the true distribution is not known by the KNN classifier, the KNN decision boundary is very close to that of the Bayes classifier. The test error rate using KNN is 0.1363, which is close to the Bayes error rate of 0.1304.

The choice of K has a drastic effect on the KNN classifier obtained. Figure 2.16 displays two KNN fits to the simulated data from Figure 2.13, using $K = 1$ and $K = 100$. When $K = 1$, the decision boundary is overly flexible and finds patterns in the data that don't correspond to the Bayes decision boundary. This corresponds to a classifier that has low bias but very high variance. As K grows, the method becomes less flexible and produces a decision boundary that is close to linear. This corresponds to a low-variance but high-bias classifier. On this simulated data set, neither $K = 1$ nor $K = 100$ give good predictions: they have test error rates of 0.1695 and 0.1925, respectively.

Just as in the regression setting, there is not a strong relationship between the training error rate and the test error rate. With $K = 1$, the KNN training error rate is 0, but the test error rate may be quite high. In general, as we use more flexible classification methods, the training error rate will decline but the test error rate may not. In Figure 2.17, we have plotted the KNN test and training errors as a function of $1/K$. As $1/K$ increases, the method becomes more flexible. As in the regression setting, the training error rate consistently declines as the flexibility increases. However, the test error exhibits a characteristic U-shape, declining at first (with a minimum at approximately $K = 10$) before increasing again when the method becomes excessively flexible and overfits.

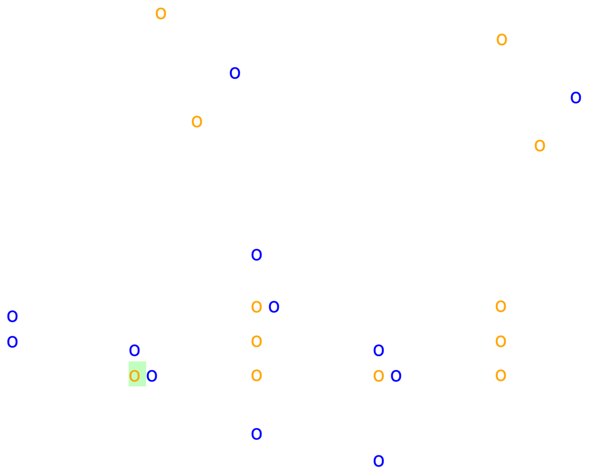


FIGURE 2.14. *The KNN approach, using* $K = 3$, *is illustrated in a simple situation*

with six blue observations and six orange observations. Left: *a test observation at which a predicted class label is desired is shown as a black cross. The three closest points to the test observation are identifed, and it is predicted that the test observation belongs to the most commonly-occurring class, in this case blue.* Right: *The KNN decision boundary for this example is shown in black. The blue grid indicates the region in which a test observation will be assigned to the blue class, and the orange grid indicates the region in which it will be assigned to the orange class.*

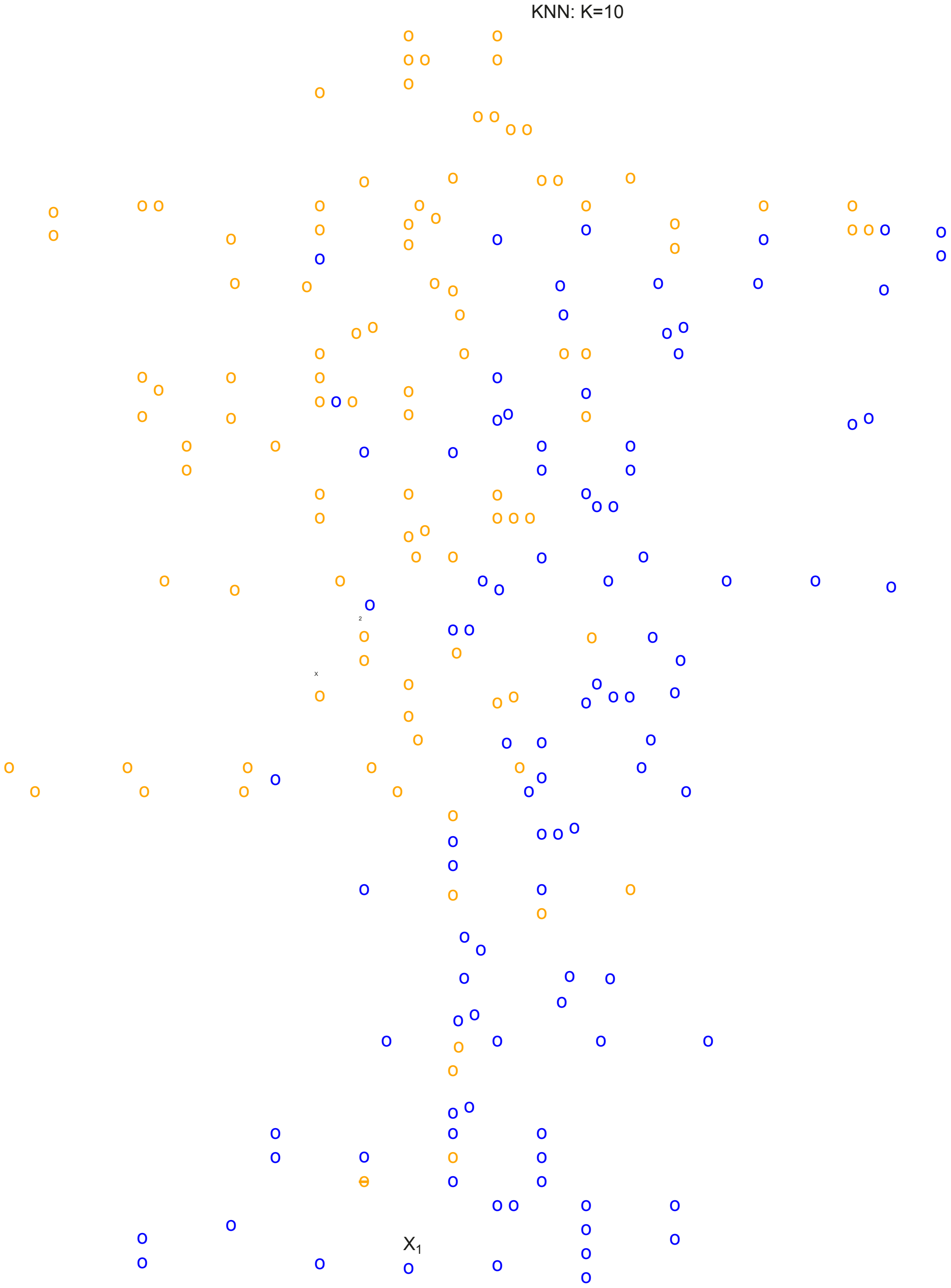


FIGURE 2.15. *The black curve indicates the KNN decision boundary on the data from Figure 2.13, using $K = 10$. The Bayes decision boundary is shown as a purple dashed line. The KNN and Bayes decision boundaries are very similar.*

In both the regression and classification settings, choosing the correct level of flexibility is critical to the success of any statistical learning method. The bias-variance tradeoff, and the resulting U-shape in the test error, can make this a difficult task. In Chapter 5, we return to this topic and discuss

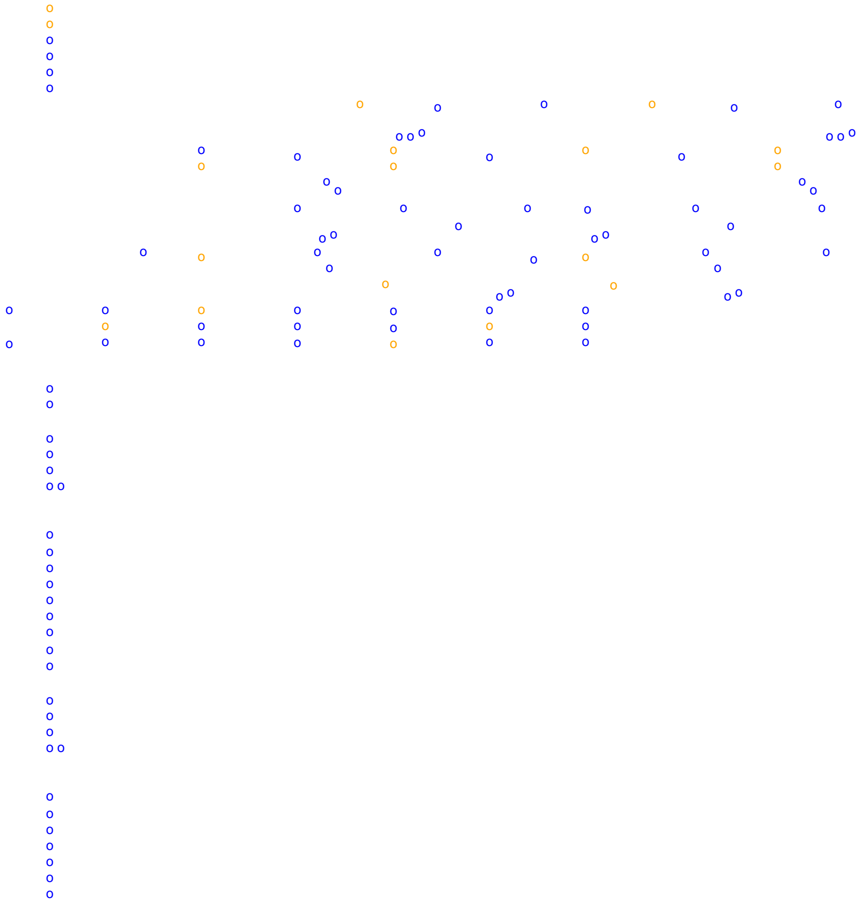


FIGURE 2.16. *A comparison of the KNN decision boundaries (solid black curves) obtained using K* = 1 *and K* = 100 *on the data from Figure 2.13. With K* = 1*, the decision boundary is overly fexible, while with K* = 100 *it is not sufciently fexible. The Bayes decision boundary is shown as a purple dashed line.*

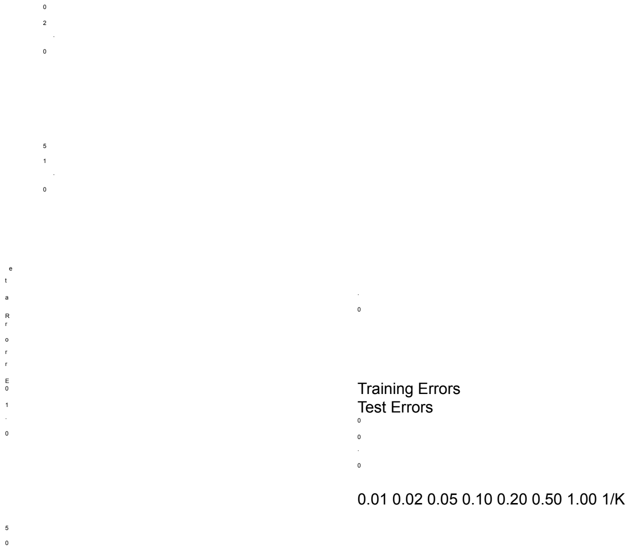


FIGURE 2.17. *The KNN training error rate (blue, 200 observations) and test error*

rate (orange, 5,000 observations) on the data from Figure 2.13, as the level of fexibility (assessed using 1/K on the log scale) increases, or equivalently as the number of neighbors K decreases. The black dashed line indicates the Bayes error rate. The jumpiness of the curves is due to the small size of the training data set.

various methods for estimating test error rates and thereby choosing the optimal level of fexibility for a given statistical learning method.

In [1]:

2.3 Lab: Introduction to Python

2.3.1 Getting Started

To run the labs in this book, you will need two things:

1. An installation of Python3, which is the specific version of Python used in the labs.

2. Access to Jupyter, a very popular Python interface that runs code through a file called a *notebook*. notebook You can download and install Python3 by following the instructions avail able at anaconda.com.

There are a number of ways to get access to

Jupyter. Here are just a few: 1. Using

Google's Colaboratory service:

colab.research.google.com/. 2. Using

JupyterHub, available at jupyter.org/hub.

3. Using your own jupyter installation. Installation instructions are available at jupyter.org/install.

Please see the Python resources page on the book website statlearning.com for up-to-date information about getting Python and Jupyter working on your computer.
You will need to install the ISLP package, which provides access to the datasets and custom-built functions that we provide.
Inside a macOS or Linux terminal type pip install ISLP; this also installs most other packages needed in the labs. The Python resources page has a link to the ISLP documentation website.
To run this lab, download the file Ch2-statlearn-lab.ipynb from the Python resources page. Now run the following code at the command line: jupyter lab Ch2-statlearn-lab.ipynb.
If you're using Windows, you can use the start menu to access anaconda, and follow the links. For example, to install ISLP and run this lab, you can run the same code above in an anaconda shell.

2.3.2 Basic Commands

In this lab, we will introduce some simple Python commands. For more resources about Python in general, readers may want to consult the tutorial at docs.python.org/3/tutorial/.

Like most programming languages, Python

function

uses *functions* to perform operations. To run a function called fun, we type fun(input1,input2), where the inputs (or *arguments*) input1 and input2 tell Python how to run the function. A function can have any number of inputs. For example, the

argument

print() function outputs a text representation of all of its arguments to

print()

the console.

```
print('fit a model with', 11, 'variables')
```

```
fit a model with 11 variables
```

The following command will provide information about the print() func tion.

```
print?
```

Adding two integers in Python is pretty intuitive.

```
3+5
```

In [2]: In [3]:

Out[3]: 8

string

In Python, textual data is handled using *strings*. For instance, "hello" and 'hello' are strings. We can concatenate them using the addition + symbol.

In [4]:

```
"hello" + " " + "world"
```

Out[4]: 'hello world'

A string is actually a type of *sequence*: this is a generic term for an ordered

sequence

list. The three most important types of sequences are lists, tuples, and strings. We introduce lists now.

The following command instructs Python to join together the numbers 3, 4, and 5, and to save them as a *list* named x. When we type x, it gives us

list

back the list.

In [5]:

```
x = [3, 4, 5]
x
```

Out[5]: [3, 4, 5]

Note that we used the brackets [] to construct this list.

We will often want to add two sets of numbers together. It is reasonable to try the following code, though it will not produce the desired results.

In [6]:

```
y = [4, 9, 7]
x+y
```

Out[6]: [3, 4, 5, 4, 9, 7]

The result may appear slightly counterintuitive: why did Python not add the entries of the lists element-by-element? In Python, lists hold *arbitrary* objects, and are added using *concatenation*. In fact, concatenation is the concatenat ion behavior that we saw earlier when we entered "hello" + " " + "world".

This example refects the fact that Python is a general-purpose programming language. Much of Python's data-specifc functionality comes from other packages, notably numpy and pandas. In the next section, we will introduce the numpy package. See docs.scipy.org/doc/numpy/user/quickstart.html for more information about numpy.

In [7]: In [8]:

In [9]:

2.3.3 Introduction to Numerical Python

As mentioned earlier, this book makes use of functionality that is contained in the numpy *library*, or *package*. A package is a collection of modules that numpy package are not necessarily included in the base Python distribution. The name numpy is an abbreviation for *numerical Python*.

To access numpy, we must frst import it. import

```
import numpy as np
```

In the previous line, we named the numpy module *module* np; an abbreviation for easier referencing. In numpy, an *array* is a generic term for a multidimensional set of numbers. array We use the np.array() function to defne x and y, which are one-dimensional np.array() arrays, i.e. vectors.

```
x = np.array([3, 4, 5])
y = np.array([4, 9, 7])
```

Note that if you forgot to run the import numpy as np command earlier, then you will encounter an error in calling the np.array() function in the previous line. The syntax np.array() indicates that the function being called is part of the numpy package, which we have abbreviated as np.

Since x and y have been defned using np.array(), we get a sensible result when we add them together. Compare this to our results in the previous section, when we tried to add two lists without using numpy.

```
x+y
```

```
Out[9]: array([ 7, 13, 12])
```

In numpy, matrices are typically represented as two-dimensional arrays, and vectors as one-dimensional arrays.[1] We can create a two-dimensional array as follows.

In [10]:

```
x = np.array([[1, 2], [3, 4]])
x
```

```
Out[10]: array([[1, 2],
               [3, 4]])
```

The object x has several *attributes*, or associated objects. To access an

attribute

attribute of x, we type x.attribute, where we replace attribute with the

ndim

name of the attribute. For instance, we can access the ndim attribute of x as follows.

In [11]:

```
x.ndim
```

```
Out[11]: 2
```

The output indicates that x is a two-dimensional array. Similarly, x.dtype is the *data*

data type

type attribute of the object x. This indicates that x is comprised of 64-bit integers:

```
x.dtype
```

In [12]:

```
Out[12]: dtype('int64')
```

Why is x comprised of integers? This is because we created x by passing in exclusively integers to the np.array() function. If we had passed in any decimals, then we would have obtained an array of *foating point numbers* foating point (i.e. real-valued numbers).

In [13]:

```
np.array([[1, 2], [3.0, 4]]).dtype
```

```
Out[13]: dtype('float64')
```

Typing fun? will cause Python to display documentation associated with the function fun, if it exists. We can try this for np.array().

In [14]: In [15]:

```
np.array?
```

This documentation indicates that we could create a foating point array by passing a dtype argument into np.array(). dtype

```
np.array([[1, 2], [3, 4]], float).dtype
```

[1] While it is also possible to create matrices using np.matrix(), we will use np.array() throughout the labs in this book.

Out[15]: dtype('float64')

The array x is two-dimensional. We can fnd out the number of rows and columns by looking at its shape attribute.

shape

In [16]:

```
x.shape
```

Out[16]: (2, 2)

method

A *method* is a function that is associated with an object. For instance, given an array x, the expression x.sum() sums all of its elements, using the sum() method for arrays. The call x.sum() automatically provides x as the frst argument to its sum() method.

.sum()

In [17]:

```
x = np.array([1, 2, 3, 4])
x.sum()
```

Out[17]: 10

We could also sum the elements of x by passing in x as an argument to the np.sum() function.

np.sum()

In [18]:

```
x = np.array([1, 2, 3, 4])
np.sum(x)
```

Out[18]: 10

As another example, the reshape() method returns a new array with the same elements as x, but a diferent shape. We do this by passing in a tuple in our call to reshape(), in this case (2, 3). This tuple specifes that we would like to create a two-dimensional array with 2 rows and 3 columns.[2] In what follows, the \n character creates a *new line*.

.reshape()

tuple

In [19]: In [20]:

```
x = np.array([1, 2, 3, 4, 5, 6])
print('beginning x:\n', x)
x_reshape = x.reshape((2, 3))
print('reshaped x:\n', x_reshape)
```

```
beginning x:
[1 2 3 4 5 6]
reshaped x:
[[1 2 3]
[4 5 6]]
```

The previous output reveals that numpy arrays are specifed as a sequence of *rows*. This is called *row-major ordering*, as opposed to *column-major ordering*. Python (and hence numpy) uses 0-based indexing. This means that to access the top left element of x_reshape, we type in x_reshape[0,0].

```
x_reshape[0, 0]
```

Out[20]: 1

Similarly, x_reshape[1,2] yields the element in the second row and the third column of x_reshape.

In [21]:

```
x_reshape[1, 2]
```

Out[21]: 6

Similarly, x[2] yields the third entry of x.

Now, let's modify the top left element of x_reshape. To our surprise, we discover that the frst element of x has been modifed as well!

In [22]:

```
print('x before we modify x_reshape:\n',
print('x_reshape before we modify x_res
0] = 5
print('x_reshape after we modify its top l
print('x after we modify top left element (
```

Out[22]:

```
x before we modify x_reshape:
 [1 2 3 4 5 6]
x_reshape before we modify x_reshape:
 [[1 2 3]
 [4 5 6]]
x_reshape after we modify its top left element:
 [[5 2 3]
```

```
[4 5 6]]
x after we modify top left element of x_reshape:
[5 2 3 4 5 6]
```

Modifying x_reshape also modifed x because the two objects occupy the same space in memory.

We just saw that we can modify an element of an array. Can we also mod ify a tuple? It turns out that we cannot — and trying to do so introduces an *exception*, or error. exception

In [23]: In [24]:

```
my_tuple = (3, 4, 5)
my_tuple[0] = 2
```

```
TypeError: 'tuple' object does not support item
assignment
```

[2] Like lists, tuples represent a sequence of objects. Why do we need more than one way to create a sequence? There are a few diferences between tuples and lists, but perhaps the most important is that elements of a tuple cannot be modifed, whereas elements of a list can be.

We now briefy mention some attributes of arrays that will come in handy. An array's shape attribute contains its dimension; this is always a tuple. The ndim attribute yields the number of dimensions, and T provides its transpose.

```
x_reshape.shape, x_reshape.ndim, x_reshape.T
```

```
Out[24]: ((2, 3),
          2,
          array([[5, 4],
                 [2, 5],
                 [3, 6]]))
```

Notice that the three individual outputs (2,3), 2, and array([[5, 4],[2, 5], [3,6]]) are themselves output as a tuple.

We will often want to apply functions to arrays. For instance, we can compute the square root of the entries using the np.sqrt() function: np.sqrt()

```
In [25]: np.sqrt(x)
```

```
Out[25]: array([2.24, 1.41, 1.73, 2., 2.24, 2.45])
```

We can also square the elements:

```
In [26]: x**2
```

```
Out[26]: array([25, 4, 9, 16, 25, 36])
```

We can compute the square roots using the same notation, raising to the power of 1/2 instead of 2.

```
In [27]: x**0.5
```

```
Out[27]: array([2.24, 1.41, 1.73, 2., 2.24, 2.45])
```

Throughout this book, we will often want to generate random data. The np.random.normal() function generates a vector of random normal variables.

np.random.

We can learn more about this function by looking at the help page, via a call normal() to np.random.normal?. The frst line of the help page reads normal(loc=0.0, scale=1.0, size=None). This *signature* line tells us that the function's arguments are loc, scale, and size. These are

signature

keyword

keyword arguments, which means that when they are passed into the function, they can be referred to by name (in any order).[3] By default, this function will generate random normal variable(s) with mean (loc) 0 and standard deviation (scale) 1; fur thermore, a single random variable will be generated unless the argument to size is changed.

We now generate 50 independent random variables from a $N(0, 1)$ dis tribution.

```
In [28]: x = np.random.normal(size=50)
```

```
x
```

Out[28]:

```
array([-1.19, 0.41, 0.9 , -0.44, -0.9 , -0.38, 0.13, 1.87, -0.35, 1.16, 0.79, -0.97, -1.21,
       0.06, -1.62, -0.6 ,
       -0.77, -2.12, 0.38, -1.22, -0.06, -1.97, -1.74, -0.56,
         1.7 , -0.95, 0.56, 0.35, 0.87, 0.88, -1.66, -0.32,
       -0.3 , -1.36, 0.92, -0.31, 1.28, -1.94, 1.07, 0.07,
        0.79, -0.46, 2.19, -0.27, -0.64, 0.85, 0.13, 0.46,
       -0.09, 0.7 ])
```

We create an array y by adding an independent *N*(50, 1) random variable to each element of x.

In [29]: In [30]:

```
y = x + np.random.normal(loc=50, scale=1, size
```

The np.corrcoef() function computes the correlation matrix between x and np.corrcoef() y. The of-diagonal elements give the correlation between x and y.

```
np.corrcoef(x, y)
```

Out[30]:

```
array([[1. , 0.69],
       [0.69, 1. ]])
```

If you're following along in your own Jupyter notebook, then you probably noticed that you got a diferent set of results when you ran the past few commands. In particular, each time we call np.random.normal(), we will get a diferent answer, as shown in the following example.

In [31]:

```
print(np.random.normal(scale=5, size=2
print(np.random.normal(scale=5, size=2
```

Out[31]:

```
[4.28 2.59]
[4.62 -2.54]
```

In order to ensure that our code provides exactly the same results each time it is run, we can set a *random seed* using the np.random.default_rng() function. This function takes an arbitrary, user-specifed integer argument. If we set a random seed before generating random data, then re-running our code will yield the same results. The object rng has essentially all the random number generating methods found in np.random. Hence, to generate normal data we use rng.normal().

random seed

np.random. default_rng()

In [32]:

```
rng = np.random.default_rng(1303)
print(rng.normal(scale=5, size=2))
rng2 = np.random.default_rng(1303)
print(rng2.normal(scale=5, size=2))
```

[3] Python also uses *positional* arguments. Positional arguments do not need to use a keyword. To see an example, type in np.sum?. We see that a is a positional argument, i.e. this function assumes that the frst unnamed argument that it receives is the array to be summed. By contrast, axis and dtype are keyword arguments: the position in which these arguments are entered into np.sum() does not matter.

Out[32]:
```
[4.09 -1.07 ]
 [4.09 -1.07 ]
```

Throughout the labs in this book, we use np.random.default_rng() when ever we perform calculations involving random quantities within numpy. In principle, this should enable the reader to exactly reproduce the stated results. However, as new versions of numpy become available, it is possible that some small discrepancies may occur between the output in the labs and the output from numpy.

The np.mean(), np.var(), and np.std() functions can be used to compute np.mean() available as methods on the arrays.

In [33]:

the mean, variance, and standard deviation of arrays. These functions are also

```
rng = np.random.default_rnp.var() np.std()
y = rng.standard_normal(1
np.mean(y), y.mean()
```

Out[33]:
```
(-0.11, -0.11)
```

In [34]:
```
np.var(y), y.var(), np.mean((y - y.mean()
```

Out[34]:
```
(2.72, 2.72, 2.72)
```

Notice that by default np.var() divides by the sample size n rather than $n - 1$; see the ddof argument in np.var?.

In [35]:
```
np.sqrt(np.var(y)), np.std(y)
```

Out[35]:
```
(1.65, 1.65)
```

The np.mean(), np.var(), and np.std() functions can also be applied to the rows and columns of a matrix. To see this, we construct a 10 × 3 matrix of $N(0, 1)$ random variables, and consider computing its row sums.

In [36]:
```
X = rng.standard_normal((10, 3))
X
```

Out[36]:
```
array([[ 0.23, -0.35, -0.28], [-0.67,
          -1.06, -0.39],
          [ 0.48, -0.24, 0.96],
          [-0.2 , 0.02, 1.55],
          [ 0.55, -0.51, -0.18],
          [ 0.54, 1.94, -0.27],
          [-0.24, 1. , -0.89],
          [-0.29, 0.88, 0.58],
          [ 0.09, 0.67, -2.83],
          [ 1.02, -0.96, -1.67]])
```

Since arrays are row-major ordered, the frst axis, i.e. axis=0, refers to its rows. We pass this argument into the mean() method for the object X. .mean()

In [37]:

```
X.mean(axis=0)
```

Out[37]: array([0.15, 0.14, -0.34]) The following

yields the same result.

In [38]:

```
X.mean(0)
```

Out[38]: array([0.15, 0.14, -0.34])

2.3.4 Graphics

matplotlib

In Python, common practice is to use the library matplotlib for graphics. However, since Python was not written with data analysis in mind, the no tion of plotting is not intrinsic to the language. We will use the subplots() function from matplotlib.pyplot to create a figure and the axes onto which we plot our data. For many more examples of how to make plots in Python, readers are encouraged to visit matplotlib.org/stable/gallery/.

In matplotlib, a plot consists of a *figure* and one or more *axes*. You can figure axes think of the figure as the blank canvas upon which one or more plots will be displayed: it is the entire plotting window. The *axes* contain important information about each plot, such as its *x*- and *y*-axis labels, title, and more. (Note that in matplotlib, the word *axes* is not the plural of *axis*: a plot's *axes* contains much more information than just the *x*-axis and the *y*-axis.)

We begin by importing the subplots() function from matplotlib. We subplots() use this function throughout when creating figures. The function returns a tuple of length two: a figure object as well as the relevant axes object. We will typically pass figsize as a keyword argument. Having created our axes, we attempt our first plot using its plot() method. To learn more .plot() about it, type ax.plot?.

In [39]: In [40]:

```
from matplotlib.pyplot import subplots
fig, ax = subplots(figsize=(8, 8))
x = rng.standard_normal(100)
y = rng.standard_normal(100)
ax.plot(x, y);
```

We pause here to note that we have *unpacked* the tuple of length two returned by subplots() into the two distinct variables fig and ax. Unpacking is typically preferred to the following equivalent but slightly more verbose code:

```
output = subplots(figsize=(8, 8))
fig = output[0]
ax = output[1]
```

We see that our earlier cell produced a line plot, which is the default. To create a scatterplot, we provide an additional argument to ax.plot(), indicating that circles should be displayed.

In [41]: In [42]: In [43]:

```
fig, ax = subplots(figsize=(8, 8))
ax.plot(x, y, 'o');
```

Diferent values of this additional argument can be used to produce diferent colored lines as well as diferent linestyles.

As an alternative, we could use the ax.scatter() function to create a .scatter() scatterplot.

```
fig, ax = subplots(figsize=(8, 8))
ax.scatter(x, y, marker='o');
```

Notice that in the code blocks above, we have ended the last line with a semicolon. This prevents ax.plot(x, y) from printing text to the notebook. However, it does not prevent a plot from being produced. If we omit the trailing semi-colon, then we obtain the following output:

```
fig, ax = subplots(figsize=(8, 8))
ax.scatter(x, y, marker='o')
```

```
Out[43]: <matplotlib.collections.PathCollection at 0x7fb3d9c8f310 >
         Figure(432x288)
```

In what follows, we will use trailing semicolons whenever the text that would be output is not germane to the discussion at hand.

To label our plot, we make use of the set_xlabel(), set_ylabel(), and .set_xlabel() .set_ylabel() set_title() methods of ax.

```
fig, ax = subplots(figsize=(8, 8))
ax.scatter(x, y, marker='o')
ax.set_xlabel("this is the x-axis")
ax.set_ylabel("this is the y-axis")
ax.set_title("Plot of X vs Y");
```

In [44]: .set_title()

Having access to the fgure object fig itself means that we can go in and change some aspects and then redisplay it. Here, we change the size from (8, 8) to (12, 3).

```
fig.set_size_inches(12,3)
fig
```

Occasionally we will want to create several plots within a fgure. This can be achieved by passing additional arguments to subplots(). Below, we create a 2 × 3 grid of plots in a fgure of size determined by the figsize argument. In such situations, there is often a relationship between the axes in the plots. For example, all plots may have a common *x*-axis. The subplots() function can automatically handle this situation when passed the keyword argument sharex=True. The axes object below is an array pointing to diferent plots in the fgure.

In [45]:

```
fig, axes = subplots(nrows=2,
                     ncols=3,
                     figsize=(15
```

We now produce a scatter plot with 'o' in the second column of the frst row and a scatter plot with '+' in the third column of the second row.

Type subplots? to learn more about subplots(). To save the output of fig, we call its savefig()

.savefig()

method. The argument dpi is the dots per inch, used to determine how large the fgure will be in pixels.

In [46]:

```
fig.savefig("Figure.png", dpi=400)
fig.savefig("Figure.pdf", dpi=200);
```

In [47]: In [48]:

We can continue to modify fig using step-by-step updates; for example, we can modify the range of the *x*-axis, re-save the fgure, and even re-display it.

```
axes[0,1].set_xlim([-1,1])
fig.savefig("Figure_updated.jpg")
fig
```

We now create some more sophisticated plots.

.contour()

The ax.contour() method produces a *contour plot* in order to represent

contour plot

three-dimensional data, similar to a topographical map. It takes three arguments: • A vector of x values (the frst dimension),

• A vector of y values (the second dimension), and

• A matrix whose elements correspond to the z value (the third dimen sion) for each pair of (x,y) coordinates.

In [49]: In [50]:

In [51]:

```
axes[0,1].plot(x, y, 'o')
axes[1,2].scatter(x, y, marker='+')
fig
```

To create x and y, we'll use the command

np.linspace()

np.linspace(a, b, n), which returns a vector of n numbers starting at a and ending at b.

```
fig, ax = subplots(figsize=(8, 8))
x = np.linspace(-np.pi, np.pi, 50)
y=x
f = np.multiply.outer(np.cos(y), 1 / (1 + x**2))
ax.contour(x, y, f);
```

We can increase the resolution by adding more levels to the image.

```
fig, ax = subplots(figsize=(8, 8))
ax.contour(x, y, f, levels=45);
```

To fne-tune the output of the ax.contour() function, take a look at the help fle by typing ?plt.contour.

The ax.imshow() method is similar to ax.contour(), except that it produces a color-coded plot whose colors depend on the z value. This is known as a *heatmap*, and is sometimes used to plot temperature in weather forecasts.

.imshow()

heatmap

```
fig, ax = subplots(figsize=(8, 8))
ax.imshow(f);
```

Notation

As seen above, the function np.linspace() can be used to create a sequence of numbers.

In [52]:

```
seq1 = np.linspace(0, 10, 11)
seq1
```

2.3.5 Sequences and Slice

Out[52]: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9., 10.])

The function np.arange() returns a sequence of numbers spaced out by step. If step is not specifed, then a default value of 1 is used. Let's create a sequence that starts at 0 and ends at 10.

np.arange()

In [53]:

```
seq2 = np.arange(0, 10)
seq2
```

Out[53]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

Why isn't 10 output above? This has to do with *slice* notation in Python. Slice notation is used to index sequences such as lists, tuples and arrays. Suppose we want to retrieve the fourth through sixth (inclusive) entries of a string. We obtain a slice of the string using the indexing notation [3:6].

slice

In [54]:

```
"hello world"[3:6]
```

Out[54]: 'lo '

In the code block above, the notation 3:6 is shorthand for slice(3,6) when used inside [].

In [55]:

```
"hello world"[slice(3,6)]
```

Out[55]: 'lo '

You might have expected slice(3,6) to output the fourth through seventh characters in the text string (recalling that Python begins its indexing at zero), but instead it output the fourth through sixth. This also explains why the earlier np.arange(0, 10) command output only the integers from 0 to 9. See the documentation slice? for useful options in creating slices.

2.3.6 Indexing Data

To begin, we create a two-dimensional numpy array.

In [56]:

```
A = np.array(np.arange(16)).reshape((4
A
```

Out[56]:
```
array([[ 0, 1, 2, 3],
       [ 4, 5, 6, 7],
       [ 8, 9, 10, 11],
       [12, 13, 14, 15]])
```

Typing A[1,2] retrieves the element corresponding to the second row and third column. (As usual, Python indexes from 0.)

```
A[1,2]
```

In [57]:

Out[57]: 6

The frst number after the open-bracket symbol [refers to the row, and the second number refers to the column.

Indexing Rows, Columns, and Submatrices

To select multiple rows at a time, we can pass in a list specifying our selection. For instance, [1,3] will retrieve the second and fourth rows:

In [58]:

```
A[[1,3]]
```

Out[58]:
```
array([[ 4, 5, 6, 7],
       [12, 13, 14, 15]])
```

To select the frst and third columns, we pass in [0,2] as the second ar gument in the square brackets. In this case we need to supply the frst argument : which selects all rows.

In [59]:

```
A[:,[0,2]]
```

Out[59]:
```
array([[ 0, 2],
       [ 4, 6],
       [ 8, 10],
       [12, 14]])
```

Now, suppose that we want to select the submatrix made up of the second and fourth rows as well as the frst and third columns. This is where index ing gets slightly tricky. It is natural to try to use lists to retrieve the rows and columns:

In [60]:

```
A[[1,3],[0,2]]
```

Out[60]: array([4, 14])

Oops — what happened? We got a one-dimensional array of length two identical to

In [61]:

```
np.array([A[1,0],A[3,2]])
```

Out[61]: array([4, 14])

Similarly, the following code fails to extract the submatrix comprised of the second and fourth rows and the frst, third, and fourth columns:

In [62]:

```
A[[1,3],[0,2,3]]
```

```
IndexError: shape mismatch: indexing arrays could
not be broadcast together with shapes (2,) (3,)
```

We can see what has gone wrong here. When supplied with two indexing lists, the numpy interpretation is that these provide pairs of *i, j* indices for a series of entries. That is why the pair of lists must have the same length. However, that was not our intent, since we are looking for a submatrix.

One easy way to do this is as follows. We frst create a submatrix by subsetting the rows of A, and then on the fy we make a further submatrix by subsetting its columns.

```
A[[1,3]][:,[0,2]]
```

In [63]:

Out[63]:

```
array([[ 4, 6],
       [12, 14]])
```

There are more efcient ways of achieving the same result.

The *convenience function* np.ix_() allows us to extract a submatrix using lists, by creating an intermediate *mesh* object.

convenience function np.ix_() mesh

In [64]:

```
idx = np.ix_([1,3],[0,2,3])
A[idx]
```

Out[64]:

```
array([[ 4, 6, 7],
       [12, 14, 15]])
```

Alternatively, we can subset matrices efciently using slices. The slice 1:4:2 captures the second and fourth items of a sequence, while the slice 0:3:2 captures the frst and third items (the third element in a slice sequence is the step size).

In [65]:

```
A[1:4:2,0:3:2]
```

Out[65]:

```
array([[ 4, 6],
       [12, 14]])
```

Why are we able to retrieve a submatrix directly using slices but not using lists? Its because they are diferent Python types, and are treated diferently by numpy. Slices can be used to extract objects from arbitrary sequences, such as strings, lists, and tuples, while the use of lists for indexing is more limited.

Boolean Indexing

In numpy, a *Boolean* is a type that equals either True or False (also rep-

Boolean

resented as 1 and 0, respectively). The next line creates a vector of 0's, represented as Booleans, of length equal to the frst dimension of A.

In [66]:

```
keep_rows = np.zeros(A.shape[0], bool)
keep_rows
```

Out[66]: array([False, False, False, False]) We now set

two of the elements to True.

In [67]:

```
keep_rows[[1,3]] = True
keep_rows
```

Out[67]: array([False, True, False, True])

Note that the elements of keep_rows, when viewed as integers, are the same as the values of np.array([0,1,0,1]). Below, we use == to verify their equal ity. When applied to two arrays, the == operation is applied elementwise.

In [68]:

```
np.all(keep_rows == np.array([0,1,0,1]))
```

Out[68]: True

(Here, the function np.all() has checked whether all entries of an array np.all() are

np.any()

True. A similar function, np.any(), can be used to check whether any entries of an array are True.) However, even though np.array([0,1,0,1]) and keep_rows are equal ac cording to ==, they index diferent sets of rows! The former retrieves the frst, second, frst, and second rows of A.

In [69]:

```
A[np.array([0,1,0,1])]
```

Out[69]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7],
       [0, 1, 2, 3],
       [4, 5, 6, 7]])
```

By contrast, keep_rows retrieves only the second and fourth rows of A — i.e. the rows for which the Boolean equals TRUE.

In [70]:

```
A[keep_rows]
```

Out[70]:
```
array([[ 4, 5, 6, 7],
       [12, 13, 14, 15]])
```

This example shows that Booleans and integers are treated diferently by numpy.

We again make use of the np.ix_() function to create a mesh containing the second and fourth rows, and the frst, third, and fourth columns. This time, we apply the function to Booleans, rather than lists.

In [71]:

```
keep_cols = np.zeros(A.shape[1], bool)
keep_cols[[0, 2, 3]] = True
idx_bool = np.ix_(keep_rows, keep_cols
```

```
A[idx_bool]
```

Out[71]:

```
array([[ 4, 6, 7],
       [12, 14, 15]])
```

We can also mix a list with an array of Booleans in the arguments to np.ix_():

In [72]:

```
idx_mixed = np.ix_([1,3], keep_cols)
A[idx_mixed]
```

Out[72]:

```
array([[ 4, 6, 7],
       [12, 14, 15]])
```

For more details on indexing in numpy, readers are referred to the numpy tutorial mentioned earlier.

2.3.7 Loading Data

Data sets often contain diferent types of data, and may have names as sociated with the rows or columns. For these reasons, they typically are best accommodated using a *data frame*. We can think of a data frame as a sequence of arrays of identical length; these are the columns. Entries in the diferent arrays can be combined to form a row. The pandas library can be used to create and work with data frame objects.

data frame

Reading in a Data Set

The frst step of most analyses involves importing a data set into Python. Before attempting to load a data set, we must make sure that Python knows where to fnd the fle containing it. If the fle is in the same location as this notebook fle, then we are all set. Otherwise, the command os.chdir() can be used to *change directory*. (You will need to call import os before calling os.chdir().)

os.chdir()

In [73]: In [74]:

We will begin by reading in Auto.csv, available on the book website. This is a

In [75]:

comma-separated fle, and can be read in using pd.read_csv(): pd.read_csv()

```
import pandas as pd
Auto = pd.read_csv('Auto.csv')
Auto
```

The book website also has a whitespace-delimited version of this data, called Auto.data. This can be read in as follows:

```
Auto = pd.read_csv('Auto.data', delim_whitespa
```

Both Auto.csv and Auto.data are simply text fles. Before loading data into Python, it is a good idea to view it using a text editor or other software, such as Microsoft Excel.

We now take a look at the column of Auto corresponding to the variable horsepower:

```
Auto['horsepower']
```

Out[75]:

```
0 130.0
 1 165.0
 2 150.0
 3 150.0
 4 140.0
           ...
 392 86.00
 393 52.00
 394 84.00
 395 79.00
 396 82.00
 Name: horsepower, Length: 397, dtype: object
```

We see that the dtype of this column is object. It turns out that all values of the horsepower column were interpreted as strings when reading in the data. We can fnd out why by looking at the unique values.

In [76]:

```
np.unique(Auto['horsepower'])
```

To save space, we have omitted the output of the previous code block. We see the culprit is the value ?, which is being used to encode missing values. To fx the problem, we must provide pd.read_csv() with an argument called na_values. Now, each instance of ? in the fle is replaced with the value np.nan, which means *not a number*:

```
Auto = pd.read_csv('Auto.data',
                   na_values=['?'],
                   delim_whitespace=True)
Auto['horsepower'].sum()
```

In [77]:

Out[77]: 40952.0

The Auto.shape attribute tells us that the data has 397 observations, or rows, and nine variables, or columns.

In [78]:

```
Auto.shape
```

Out[78]: (397, 9)

There are various ways to deal with missing data. In this case, since only fve of the rows contain missing observations, we choose to use the Auto.dropna() method to simply remove these rows. .dropna()

```
In [79]:
Auto_new = Auto.dropna()
Auto_new.shape
Out[79]: (392, 9)
```

Basics of Selecting Rows and Columns

We can use Auto.columns to check the variable names.

```
In [80]:
Auto = Auto_new # overwrite the previo
Auto.columns
Out[80]: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
               'year', 'origin', 'name'],
              dtype='object')
```

Accessing the rows and columns of a data frame is similar, but not identical, to accessing the rows and columns of an array. Recall that the first argument to the [] method is always applied to the rows of the array. Similarly, passing in a slice to the [] method creates a data frame whose *rows* are determined by the slice:

```
In [81]:
Auto[:3]
Out[81]: mpg cylinders displacement horsepower weight ... 0 18.0 8 307.0 130.0 3504.0 ... 1
 15.0 8 350.0 165.0 3693.0 ... 2 18.0 8 318.0 150.0 3436.0 ...
```

Similarly, an array of Booleans can be used to subset the rows:

```
In [82]: In [83]:
```

However, if we pass in a list of strings to the [] method, then we obtain a data frame containing the corresponding set of *columns*.

```
Auto[['mpg', 'horsepower']]
```

```
idx_80 = Auto['year'] > 80
Auto[idx_80]
Out[83]: mpg horsepower
 0 18.0 130.0
 1 15.0 165.0
 2 18.0 150.0
 3 16.0 150.0
 4 17.0 140.0
  ... ... ...
 392 27.0 86.0
 393 44.0 52.0
 394 32.0 84.0
 395 28.0 79.0
 396 31.0 82.0
 392 rows x 2 columns
```

Since we did not specify an *index* column when we loaded our data frame, the rows are labeled using integers 0 to 396.

In [84]:

```
Auto.index
```

Out[84]:
```
Int64Index([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...
            387, 388, 389, 390, 391, 392, 393, 394, 395, 396],
      dtype='int64', length=392)
```

We can use the set_index() method to re-name the rows using the contents .set_index() of Auto['name'].

In [85]:

```
Auto_re = Auto.set_index('name')
Auto_re
```

Out[85]:
```
mpg cylinders displacement ... name

chevrolet chevelle malibu 18.0 8 307.0 ... buick skylark 32 15.0 8 350.0 ... plymouth
                   satellite 18.0 8 318.0 ... amc rebel sst 16.0 8 304.0 ...
```

In [86]:

```
Auto_re.columns
```

Out[86]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin'],
      dtype='object')
```

We see that the column 'name' is no longer there.
Now that the index has been set to name, we can access rows of the data frame by name using the loc[] method of Auto: .loc[]

```
rows = ['amc rebel sst', 'ford torino']
Auto_re.loc[rows]
```

In [87]:

58 2. Statistical Learning

Out[87]:
```
mpg cylinders displacement horsepower ... name
amc rebel sst 16.0 8 304.0 150.0 ... ford torino 17.0 8 302.0 140.0 ...
```

As an alternative to using the index name, we could retrieve the 4th and 5th rows of Auto using the iloc[] method: .iloc[]

In [88]: In [89]:

```
Auto_re.iloc[:,[0,2,3]]
```

We can extract the 4th and 5th rows, as well as the 1st, 3rd and 4th columns, using a single call to iloc[]:

In [90]:

```
Auto_re.iloc[[3,4]]
```

```
Auto_re.iloc[[3,4],[0,2,3]]
```

We can also use it to retrieve the 1st, 3rd and and 4th columns of Auto_re:

Out[90]:
```
mpg displacement horsepower
          name
amc rebel sst 16.0 304.0 150.0
  ford torino 17.0 302.0 140.0
```

Index entries need not be unique: there are several cars in the data frame named ford galaxie 500.

In [91]:

```
Auto_re.loc['ford galaxie 500', ['mpg', 'or
```

Out[91]:

```
mpg origin
                 name
ford galaxie 500 15.0 1
ford galaxie 500 14.0 1
ford galaxie 500 14.0 1
```

More on Selecting Rows and Columns

Suppose now that we want to create a data frame consisting of the weight and origin of the subset of cars with year greater than 80 — i.e. those built after 1980. To do this, we frst create a Boolean array that indexes the rows. The loc[] method allows for Boolean entries as well as strings:

In [92]: In [93]:

```
idx_80 = Auto_re['year'] > 80
Auto_re.loc[idx_80, ['weight', 'origin']]
```

To do this more concisely, we can use an anonymous function called a lambda: lambda

```
Auto_re.loc[lambda df: df['year'] > 80, ['weig
```

The lambda call creates a function that takes a single argument, here df, and returns df['year']>80. Since it is created inside the loc[] method for

In [94]: In [95]:

In [96]: In [97]:

the dataframe Auto_re, that dataframe will be the argument supplied. As another example of using a lambda, suppose that we want all cars built after 1980 that achieve greater than 30 miles per gallon:

```
Auto_re.loc[lambda df: (df['year'] > 80) & (df['mpg'] > 30), ['
            ]
```

The symbol & computes an element-wise *and* operation. As another ex ample, suppose that we want to retrieve all Ford and Datsun cars with displacement less than 300. We check whether each name entry contains either the string ford or datsun using the str.contains() method of the .str. contains() index attribute of of the dataframe:

```
Auto_re.loc[lambda df: (df['displacement'] <
                                                         &
                                                         |
               ['weight', 'origin']
             ]
```

Here, the symbol | computes an element-wise *or* operation.

In summary, a powerful set of operations is available to index the rows and columns of data frames. For integer based queries, use the iloc[] method. For string and Boolean selections, use the loc[] method. For functional queries that filter rows, use the loc[] method with a function (typically a lambda) in the rows argument.

2.3.8 For Loops

A for loop is a standard tool in many for languages that repeatedly evaluates some chunk of code while varying different values inside the code. For example, suppose we loop over elements of a list and compute their sum.

```
total = 0
for value in [3,2,19]:
    total += value
print('Total is: {0}'.format(total))
```

```
Total is: 24
```

The indented code beneath the line with the for statement is run for each value in the sequence specified in the for statement. The loop ends either when the cell ends or when code is indented at the same level as the original for statement. We see that the final line above which prints the total is executed only once after the for loop has terminated. Loops can be nested by additional indentation.

```
total = 0
for value in [2,3,19]:
    for weight in [3, 2, 1]:
        total += value * weight
print('Total is: {0}'.format(total))
```

```
Total is: 144
```

In [98]: In [99]:

Above, we summed over each combination of value and weight. We also took advantage of the *increment* notation in Python: the increment expression a += b is equivalent to a=a+b. Besides being a convenient notation, this can save time in computationally heavy tasks in which the intermediate value of a+b need not be explicitly created.

Perhaps a more common task would be to sum over (value, weight) pairs. For instance, to compute the average value of a random variable that takes on possible values 2, 3 or 19 with probability 0.2, 0.3, 0.5 respectively we would compute the weighted sum. Tasks such as this can often be accomplished using the zip() function that loops over a sequence of tuples. zip()

```
total = 0
for value, weight in zip([2,3,19],
                                        [0.2,0.3,0
    total += weight * value
print('Weighted average is: {0}'.format(total)
```

```
Weighted average is: 10.8
```

String Formatting

In the code chunk above we also printed a string displaying the total. However, the object total is an integer and not a string. Inserting the value of something into a string is a common task, made simple using some of the powerful string formatting tools in Python. Many data cleaning tasks involve manipulating and programmatically producing strings.

For example we may want to loop over the columns of a data frame and print the percent missing in each column. Let’s create a data frame D with columns in which 20% of the entries are missing i.e. set to np.nan. np.nan We’ll create the values in D from a normal distribution with mean 0 and variance 1 using rng.standard_normal() and then overwrite some random entries using rng.choice().

```
rng = np.random.default_rng(1)
A = rng.standard_normal((127, 5))
M = rng.choice([0, np.nan], p=[0.8,0.2], size=A.shape) A +=
D = pd.DataFrame(A, columns=['food',
                                                   'bar',
                                                   'pickle',
                                                   'snack',
                                                   'popcorn'])
D[:3]
```

Out[99]:
```
food  bar  pickle  snack  popcorn  0  0.345584  0.821618  0.330437
-1.303157  NaN  1  NaN  -0.536953 0.581118 0.364572 0.294132 2
NaN 0.546713 NaN -0.162910 -0.482119
```

In [100]:

```
for col in D.columns:
    template = 'Column "{0}" has {1:.2
    print(template.format(col,
          np.isnan(D[col]).mean()))
```

In [101]: In [102]:

In [103]: In [104]:

In [105]:

```
Column "food" has 16.54% missing values
Column "bar" has 25.98% missing values
Column "pickle" has 29.13% missing values
Column "snack" has 21.26% missing values
Column "popcorn" has 22.83% missing values
```

We see that the template.format() method expects two arguments {0} and {1:.2%}, and the latter includes some formatting information. In particular, it specifies that the second argument should be expressed as a percent with two decimal digits.

The reference docs.python.org/3/library/string.html includes many helpful and more complex examples.

2.3.9 Additional Graphical and Numerical Summaries

We can use the ax.plot() or ax.scatter() functions to display the quantitative variables. However, simply typing the variable names will produce an error message, because Python does not know to look in the Auto data set for those variables.

```
fig, ax = subplots(figsize=(8, 8))
ax.plot(horsepower , mpg, 'o');
```

```
NameError: name 'horsepower' is not defined
```

We can address this by accessing the columns directly:

```
fig, ax = subplots(figsize=(8, 8))
ax.plot(Auto['horsepower'], Auto['mpg'], 'o');
```

.plot()

Alternatively, we can use the plot() method with the call Auto.plot(). Using this method, the variables can be accessed by name. The plot methods of a data frame return a familiar object: an axes. We can use it to update the plot as we did previously:

```
ax = Auto.plot.scatter('horsepower', 'mpg');
ax.set_title('Horsepower vs. MPG')
```

If we want to save the figure that contains a given axes, we can find the relevant figure by accessing the figure attribute:

```
fig = ax.figure
fig.savefig('horsepower_mpg.png');
```

We can further instruct the data frame to plot to a particular axes object. In this case the corresponding plot() method will return the modified axes we passed in as an argument. Note that when we request a one-dimensional grid of plots, the object axes is similarly one-dimensional. We place our scatter plot in the middle plot of a row of three plots within a fgure.

```
fig, axes = subplots(ncols=3, figsize=(1
Auto.plot.scatter('horsepower', 'mpg', a
```

Note also that the columns of a data frame can be accessed as attributes: try typing in Auto.horsepower.

In [106]:

In [107]: In [108]: In [109]:

We now consider the cylinders variable. Typing in Auto.cylinders.dtype reveals that it is being treated as a quantitative variable. However, since there is only a small number of possible values for this variable, we may wish to treat it as qualitative. Below, we replace the cylinders column with a categorical version of Auto.cylinders. The function pd.Series() owes its name to the fact that pandas is often used in time series applications.

pd.Series()

```
Auto.cylinders = pd.Series(Auto.cylinders, dtype='category')
Auto.cylinders.dtype
```

Now that cylinders is qualitative, we can display it using the boxplot() method.

.boxplot()

```
fig, ax = subplots(figsize=(8, 8))
Auto.boxplot('mpg', by='cylinders', ax=ax);
```

The hist() method can be used to plot a *histogram*.

.hist()

```
fig, ax = subplots(figsize=(8, 8))
Auto.hist('mpg', ax=ax);
```

The color of the bars and the number of bins can be changed:

```
fig, ax = subplots(figsize=(8, 8))
Auto.hist('mpg', color='red', bins=12, ax=ax);
```

See Auto.hist? for more plotting options. We can use the pd.plotting.scatter_matrix() function to create a *scatterplot matrix* to visualize all of the pairwise relationships between the columns in a data frame.

pd.plotting.

In [110]: In [111]:

```
pd.plotting.scatter_matrix
```

We can also produce scatterplots for a subset of the variables.

scatter_ matrix()

```
pd.plotting.scatter_matrix(Auto[['mpg',
```

In [112]: In [113]:

The describe() method produces a numerical summary of each column in .describe() a data frame.

```
Auto[['mpg', 'weight']].describe()
```

We can also produce a summary of just a single column.

```
Auto['cylinders'].describe()
Auto['mpg'].describe()
```

To exit Jupyter, select File / Close and Halt.

2.4 Exercises

Conceptual

1. For each of parts (a) through (d), indicate whether we would generally expect the performance of a flexible statistical learning method to be better or worse than an inflexible method. Justify your answer.

   (a) The sample size *n* is extremely large, and the number of predic tors *p* is small.

   (b) The number of predictors *p* is extremely large, and the number of observations *n* is small.

   (c) The relationship between the predictors and response is highly non-linear.

   (d) The variance of the error terms, i.e. σ^2 = Var(″), is extremely high.

2. Explain whether each scenario is a classification or regression problem, and indicate whether we are most interested in inference or pre diction. Finally, provide *n* and *p*.

   (a) We collect a set of data on the top 500 firms in the US. For each firm we record profit, number of employees, industry and the CEO salary. We are interested in understanding which factors affect CEO salary.

   (b) We are considering launching a new product and wish to know whether it will be a *success* or a *failure*. We collect data on 20 similar products that were previously launched. For each product we have recorded whether it was a success or failure, price charged for the product, marketing budget, competition price, and ten other variables.

   (c) We are interested in predicting the % change in the USD/Euro exchange rate in relation to the weekly changes in the world stock markets. Hence we collect weekly data for all of 2012.

For each week we record the % change in the USD/Euro, the % change in the US market, the % change in the British market, and the % change in the German market.

3. We now revisit the bias-variance decomposition.

   (a) Provide a sketch of typical (squared) bias, variance, training error, test error, and Bayes (or irreducible) error curves, on a single plot, as we go from less fexible statistical learning methods towards more fexible approaches. The *x*-axis should represent the amount of fexibility in the method, and the *y*-axis should represent the values for each curve. There should be fve curves. Make sure to label each one.

   (b) Explain why each of the fve curves has the shape displayed in part (a).

4. You will now think of some real-life applications for statistical learning.

   (a) Describe three real-life applications in which *classifcation* might be useful. Describe the response, as well as the predictors. Is the goal of each application inference or prediction? Explain your answer.

   (b) Describe three real-life applications in which *regression* might be useful. Describe the response, as well as the predictors. Is the goal of each application inference or prediction? Explain your answer.

   (c) Describe three real-life applications in which *cluster analysis* might be useful.

5. What are the advantages and disadvantages of a very fexible (versus a less fexible) approach for regression or classifcation? Under what circumstances might a more fexible approach be preferred to a less fexible approach? When might a less fexible approach be preferred?

6. Describe the diferences between a parametric and a non-parametric statistical learning approach. What are the advantages of a para metric approach to regression or classifcation (as opposed to a non parametric approach)? What are its disadvantages?

7. The table below provides a training data set containing six observations, three predictors, and one qualitative response variable.

Obs.	X_1	X_2	X_3	Y
1	0	3	0	Red
2	2	0	0	Red
3	0	1	3	Red
4	0	1	2	Green
5	−1	0	1	Green

Suppose we wish to use this data set to make a prediction for Y when $X_1 = X_2 = X_3 = 0$ using K-nearest neighbors.

(a) Compute the Euclidean distance between each observation and the test point, $X_1 = X_2 = X_3 = 0$.

(b) What is our prediction with $K = 1$? Why?

(c) What is our prediction with $K = 3$? Why?

(d) If the Bayes decision boundary in this problem is highly non linear, then would we expect the *best* value for K to be large or small? Why?

Applied

8. This exercise relates to the College data set, which can be found in the fle College.csv on the book website. It contains a number of variables for 777 diferent universities and colleges in the US. The variables are

- Private : Public/private indicator
- Apps : Number of applications received
- Accept : Number of applicants accepted
- Enroll : Number of new students enrolled
- Top10perc : New students from top 10 % of high school class
- Top25perc : New students from top 25 % of high school class
- F.Undergrad : Number of full-time undergraduates
- P.Undergrad : Number of part-time undergraduates
- Outstate : Out-of-state tuition
- Room.Board : Room and board costs
- Books : Estimated book costs
- Personal : Estimated personal spending
- PhD : Percent of faculty with Ph.D.s
- Terminal : Percent of faculty with terminal degree
- S.F.Ratio : Student/faculty ratio
- perc.alumni : Percent of alumni who donate
- Expend : Instructional expenditure per student
- Grad.Rate : Graduation rate

Before reading the data into Python, it can be viewed in Excel or a text editor.

(a) Use the pd.read_csv() function to read the data into Python. Call the loaded data college. Make sure that you have the directory set to the correct location for the data.

(b) Look at the data used in the notebook by creating and running a new cell with just the code college in it. You should notice that the frst column is just the name of each university in a column named something like Unnamed: 0. We don't really want pandas to treat this as data. However, it may be handy to have these names for later. Try the following commands and similarly look at the resulting data frames:

```
  college2 = pd.read_csv('College.csv', index_col=0) college3 =
       college.rename({'Unnamed: 0': 'College'}, axis=1)
college3 = college3.set_index('College')
```

This has used the frst column in the fle as an index for the data frame. This means that pandas has given each row a name corresponding to the appropriate university. Now you should see that the frst data column is Private. Note that the names of the colleges appear on the left of the table. We also introduced a new python object above: a *dictionary*, which is specifed by dictionary (key, value) pairs. Keep your modifed version of the data with the following:

```
college = college3
```

(c) Use the describe() method of to produce a numerical summary of the variables in the data set.

(d) Use the pd.plotting.scatter_matrix() function to produce a scatterplot matrix of the frst columns [Top10perc, Apps, Enroll]. Recall that you can reference a list C of columns of a data frame A using A[C].

(e) Use the boxplot() method of college to produce side-by-side boxplots of Outstate versus Private.

(f) Create a new qualitative variable, called Elite, by *binning* the Top10perc variable into two groups based on whether or not the proportion of students coming from the top 10% of their high school classes exceeds 50%.

```
    college['Elite'] = pd.cut(college['Top10perc'], [0,0.5,1],
                                                  labels=['No', 'Yes'])
```

Use the value_counts() method of college['Elite'] to see how many elite universities there are. Finally, use the boxplot() method again to produce side-by-side boxplots of Outstate versus Elite.

(g) Use the plot.hist() method of college to produce some his tograms with difering numbers of bins for a few of the quanti tative variables. The command plt.subplots(2, 2) may be use ful: it will divide the plot window into four regions so that four plots can be made simultaneously. By changing the arguments you can divide the screen up in other combinations.

(h) Continue exploring the data, and provide a brief summary of what you discover.

9. This exercise involves the Auto data set studied in the lab. Make sure that the missing values have been removed from the data.

(a) Which of the predictors are quantitative, and which are quali tative?

(b) What is the *range* of each quantitative predictor? You can an swer this using the min() and max() methods in numpy. .min() (c) What is the mean and standard deviation of each quantitative .max() predictor?

(d) Now remove the 10th through 85th observations. What is the range, mean, and standard deviation of each predictor in the subset of the data that remains?

(e) Using the full data set, investigate the predictors graphically, using scatterplots or other tools of your choice. Create some plots highlighting the relationships among the predictors. Comment on your fndings.

(f) Suppose that we wish to predict gas mileage (mpg) on the basis of the other variables. Do your plots suggest that any of the other variables might be useful in predicting mpg? Justify your answer.

10. This exercise involves the Boston housing data set.

(a) To begin, load in the Boston data set, which is part of the ISLP library.

(b) How many rows are in this data set? How many columns? What do the rows and columns represent?

(c) Make some pairwise scatterplots of the predictors (columns) in this data set. Describe your fndings.

(d) Are any of the predictors associated with per capita crime rate? If so, explain the relationship.

(e) Do any of the suburbs of Boston appear to have particularly high crime rates? Tax rates? Pupil-teacher ratios? Comment on the range of each predictor.

(f) How many of the suburbs in this data set bound the Charles river?

(g) What is the median pupil-teacher ratio among the towns in this data set?

(h) Which suburb of Boston has lowest median value of owner occupied homes? What are the values of the other predictors for that suburb, and how do those values compare to the overall ranges for those predictors? Comment on your fndings.

(i) In this data set, how many of the suburbs average more than seven rooms per dwelling? More than eight rooms per dwelling? Comment on the suburbs that average more than eight rooms per dwelling.

3

Linear Regression

This chapter is about *linear regression*, a very simple approach for super vised learning. In particular, linear regression is a useful tool for predicting a quantitative response. It has been around for a long time and is the topic of innumerable textbooks. Though it may seem somewhat dull compared to some of the more modern statistical learning approaches described in later chapters of this book, linear regression is still a useful and widely used sta tistical learning method. Moreover, it serves as a good jumping-off point for newer approaches: as we will see in later chapters, many fancy statistical learning approaches can be seen as generalizations or extensions of linear regression. Consequently, the importance of having a good understanding of linear regression before studying more complex learning methods cannot be overstated. In this chapter, we review some of the key ideas underlying the linear regression model, as well as the least squares approach that is most commonly used to fit this model.

Recall the Advertising data from Chapter 2. Figure 2.1 displays sales (in thousands of units) for a particular product as a function of advertis ing budgets (in thousands of dollars) for TV, radio, and newspaper media. Suppose that in our role as statistical consultants we are asked to suggest, on the basis of this data, a marketing plan for next year that will result in high product sales. What information would be useful in order to provide such a recommendation? Here are a few important questions that we might seek to address:

1. *Is there a relationship between advertising budget and sales?* Our first goal should be to determine whether the data provide evi dence of an association between advertising expenditure and sales. If the

evidence is weak, then one might argue that no money should be spent on advertising!

G. James et al., *An Introduction to Statistical Learning*, Springer Texts in Statistics,
https://doi.org/10.1007/978-3-031-38747-0_3

2. *How strong is the relationship between advertising budget and sales?* Assuming that there is a relationship between advertising and sales, we would like to know the strength of this relationship. Does knowl edge of the advertising budget provide a lot of information about product sales?

3. *Which media are associated with sales?*
   Are all three media—TV, radio, and newspaper—associated with sales, or are just one or two of the media associated? To answer this question, we must find a way to separate out the individual contribu tion of each medium to sales when we have spent money on all three media.

4. *How large is the association between each medium and sales?*
   For every dollar spent on advertising in a particular medium, by what amount will sales increase? How accurately can we predict this amount of increase?

5. *How accurately can we predict future sales?*
   For any given level of television, radio, or newspaper advertising, what is our prediction for sales, and what is the accuracy of this prediction?

6. *Is the relationship linear?*
   If there is approximately a straight-line relationship between advertis ing expenditure in the various media and sales, then linear regression is an appropriate tool. If not, then it may still be possible to trans form the predictor or the response so that linear regression can be used.

7. *Is there synergy among the advertising media?*
   Perhaps spending $50,000 on television advertising and $50,000 on ra dio advertising is associated with higher sales than allocating $100,000 to either television or radio individually. In marketing, this is known as a *synergy* effect, while in statistics it is called an *interaction* effect. synergy

interaction It turns out that linear regression can be used to answer each of these questions. We will first discuss all of these questions in a general context, and then return to them in this specific context in Section 3.4.

3.1 Simple Linear Regression

Simple linear regression lives up to its name: it is a very straightforward simple linear approach for predicting a quantitative response Y on the basis of a sin gle regression predictor variable X. It assumes that there is approximately a linear relationship between X and Y . Mathematically, we can write this linear relationship as

$$Y \approx \beta_0 + \beta_1 X. \quad (3.1)$$

You might read "≈" as *"is approximately modeled as"*. We will sometimes describe (3.1) by saying that we are *regressing Y on X* (or *Y onto X*).

For example, X may represent TV advertising and Y may represent sales. Then we can regress sales onto TV by ftting the model

$$\text{sales} \approx \beta_0 + \beta_1 \times \text{TV}.$$

In Equation 3.1, β_0 and β_1 are two unknown constants that represent the *intercept* and *slope* terms in the linear model. Together, β_0 and β_1 are intercept slope known as the model *coefcients* or *parameters*. Once we have used our

training data to produce estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ for the model coefcients, we can predict future sales on the basis of a particular value of TV advertising

coefcient parameter

by computing

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x, \quad (3.2)$$

where $\hat{y}$ indicates a prediction of Y on the basis of $X = x$. Here we use a *hat* symbol, ˆ , to denote the estimated value for an unknown parameter or coefcient, or to denote the predicted value of the response.

3.1.1 Estimating the Coefcients

In practice, β_0 and β_1 are unknown. So before we can use (3.1) to make predictions, we must use data to estimate the coefcients. Let

$$(x_1, y_1), (x_2, y_2),\ldots, (x_n, y_n)$$

represent n observation pairs, each of which consists of a measurement of X and a measurement of Y . In the Advertising example, this data set con sists of the TV advertising budget and product sales in n = 200 diferent markets. (Recall that the data are displayed in Figure 2.1.) Our goal is to obtain coefcient estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ such that the linear model (3.1) fts the available data well—that is, so that $y_i \approx \hat{\beta}_0 + \hat{\beta}_1 x_i$ for $i = 1,\ldots,n$. In other words, we want to fnd an intercept $\hat{\beta}_0$ and a slope $\hat{\beta}_1$ such that the resulting line is as close as possible to the n = 200 data points. There are a number of ways of measuring *closeness*. However, by far the most com mon approach involves minimizing the *least squares* criterion, and we take least squares that approach in this chapter. Alternative approaches will be considered in Chapter 6.

Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the prediction for Y based on the ith value of X. Then $e_i = y_i - \hat{y}_i$ represents the ith *residual*—this is the diference between residual the ith observed response value and the ith response value that is predicted by our linear model. We defne the *residual sum of squares* (RSS) as residual sum of squares

$$\text{RSS} = e^2_1 + e^2_2 + \cdots + e^2_n,$$

or equivalently as

$$\text{RSS} = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \cdots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2. \quad (3.3)$$

The least squares approach chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimize the RSS. Using some calculus, one can show that the minimizers are

$)_n$

$\hat{\beta}_1 =$ $\quad \bar{y})\ )_n$

$_{i=1}(x_i - \bar{x})(y_i -$ $\quad _{i=1}(x_i - \bar{x})^2,$ $\quad$ (3.4)

$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 x, \bar{}$