Hui Lin and Ming Li

Introduction to Data Science

Contents

List of Tables

List of Figures

Preface

During the first couple years of our career as data scientists, we were bewildered by all kinds of data science hype. There is a lack of definition of many basic terminologies such as "Big Data" and "Data Science." How big is big? If someone ran into you asked what data science was all about, what would you tell them? What is the difference between the sexy role "Data Scientist" and the traditional "Data Analyst"? How suddenly came all kinds of machine algorithms? All those struck us as confusing and vague as real-world data scientists! But we always felt that there was something real there. After applying data science for many years, we explored it more and had a much better idea about data science. And this book is our endeavor to make data science to a more legitimate field.

Goal of the Book

This is an introductory book to data science with a specific focus on the application. Data Science is a cross-disciplinary subject involving hands-on experience and business problem-solving exposures. The majority of existing introduction books on data science are about the modeling techniques and the implementation of models using R or Python. However, they fail to introduce data science in a context of the industrial environment. Moreover, a crucial part, the art of data science in practice, is often missing. This book intends to fill the gap.

Some key features of this book are as follows:

- It is comprehensive. It covers not only technical skills but also soft skills and big data environment in the industry.
- It is hands-on. We provide the data and repeatable R and Python code. You can repeat the analysis in the book using the data and code provided. We also suggest you perform the analyses with your data whenever possible. You can only learn data science by doing it!
- It is based on context. We put methods in the context of industrial data science questions.
- Where appropriate, we point you to more advanced materials on models to dive deeper

Who This Book Is For

Non-mathematical readers will appreciate the emphasis on problem-solving with real data across a wide variety of applications and the reproducibility of the companion R and python code.

Readers should know basic statistical ideas, such as correlation and linear regression analysis. While the text is biased against complex equations, a mathematical background is needed for advanced topics.

What This Book Covers

Based on industry experience, this book outlines the real world scenario and points out pitfalls data science practitioners should avoid. It also covers big data cloud platform and the art of data science such as soft skills. We use R as the main tool and provide code for both R and Python.

Conventions

Acknowledgements

About the Authors

Hui Lin is currently Data Scientist at Netlify where she is building the data science department. Before Netlify, she was a Data Scientist at DowDuPont. She was a leader in the company of applying advanced data science to enhance Marketing and Sales Effectiveness. She was providing data science leadership for a broad range of predictive analytics and market research analysis from 2013 to 2018. She is the co-founder of Central Iowa R User Group, blogger of scientistcafe.com and 2018 Program Chair of ASA Statistics in Marketing Section. She enjoys making analytics accessible to a broad audience and teaches tutorials and workshops for practitioners on data science. She holds MS and Ph.D. in statistics from Iowa State University, BS in mathematical statistics from Beijing Normal University.

Ming Li is currently a Senior Data Scientist at Amazon and an Adjunct Faculty of Department of Marketing and Business Analytics in Texas A&M University - Commerce. He is the Chair of Quality & Productivity Section of ASA for 2017. He was a Data Scientist at Walmart and a Statistical Leader at General Electric Global Research Center. He obtained his Ph.D. in Statistics from Iowa State University at 2010. With deep statistics background and a few years' experience in data science, he has trained and mentored numerous junior data scientist with different background such as statistician, programmer, software developer, database administrator and business analyst. He is also an Instructor of Amazon's internal Machine Learning University and was one of the key founding member of Walmart's Analytics Rotational Program which bridges the skill gaps between new hires and productive data scientists.(

1 Introduction

Interest in data science is at an all-time high and has exploded in popularity in the last couple of years. Data scientists today are from various backgrounds. If someone ran into you asked what data science was all about, what would you tell them? It is not easy to answer. Data science is one of the areas where if you ask ten people you get ten different answers. It is not well-defined as an academic subject but broadly used in the industry. Media has been hyping about "Data Science" "Big Data" and "Artificial Intelligence" over the fast few years. With the data science hype picking up stream, many professionals changed their titles to "Data Scientist" without any of the necessary qualifications. Your first reaction to all of this might be some combination of skepticism and confusion. We want to address this up front that: we had that exact reaction. To make things clear, let's start with the fundamental question.

1.1 What is data science?

David Donoho (**?**) summarizes in "50 Years of Data Science" the main recurring "Memes" about data sciences:

1. The 'Big Data' Meme
2. The 'Skills' Meme
3. The 'Jobs' Meme

Everyone should have heard about big data. Data science trainees now need the skills to cope with such big data sets. What are those skills? You may hear about: Hadoop, a system using Map/Reduce to process large data sets distributed across a cluster of computers or about Spark, a system build atop Hadoop for speeding up the same by loading huge datasets into shared memory(RAM) across clusters. The new skills are for dealing with organizational artifacts of large-scale cluster computing but not for better solving the real problem. A lot of data on its own is worthless. It isn't the size of the data that's important. It's what you do with it. The big data skills that so many are touting today are not skills for better solving the real problem of inference from data.

We are transiting to universal connectivity with a deluge of data filling telecom servers. But these facts don't immediately create a science. The statisticians and computer scientists have been laying the groundwork for data science for at least 50 years. Today's data science is an enlargement and combination of statistics and computer science rather than a brand new discipline.

Data Science doesn't come out of the blue. Its predecessor is data analysis. Back in 1962, John Tukey wrote in "The Future of Data Analysis":

> For a long time I have thought I was a statistician, interested in inferences from the particular to the general. But as I have watched mathematical statistics evolve, I have had cause to wonder and to doubt. ...All in all, I have come to feel that my central interest is in data analysis, which I take to include, among other things: procedures for analyzing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data.

It deeply shocked his academic readers. Aren't you supposed to present something mathematically precise, such as definitions, theorems, and proofs? If we use one sentence to summarize what John said, it is:

---

data analysis is more than mathematics.

In September 2015, the University of Michigan made plans to invest $100 million over the next five years in a new Data Science Initiative (DSI) that will enhance opportunities for student and faculty researchers across the university to tap into the enormous potential of big data. How does DSI define Data science? Their website gives us an idea:

"This coupling of scientific discovery and practice involves the collection, management, processing, analysis, visualization, and interpretation of vast amounts of heterogeneous data associated with a diverse array of scientific, translational, and interdisciplinary applications."

How about data scientist? Here is a list of somewhat whimsical definitions for a "data scientist":

- "A data scientist is a data analyst who lives in California"
- "A data scientist is someone who is better at statistics than any software engineer and better at software engineering than any statistician."
- "A data scientist is a statistician who lives in San Francisco."
- "Data Science is statistics on a Mac."

There is lots of confusion between Data Scientist, Statistician, Business/Financial/Risk(etc.) Analyst and BI professional due to the apparent intersections among skillsets. We see data science as a discipline to make sense of data. The techniques and methodologies of data science stem from the fields of computer science and statistics. One of the most well-cited diagrams describing the area comes from Drew Conway [1] where he suggested data science is the intersection of hacking skills, math and stats knowledge, and substantial expertise. This diagram might be a bit of an oversimplification, but it's a great start.

There are almost as many definitions of data science as there are data scientists. Instead of listing some of these definitions, it might be more informative to let the subject matter define the field.

Let's start from a brief history of data science. If you hit up the Google Trends website which shows search keyword information over time and check the term "data science," you will find the history of data science goes back a little further than 2004. From the way media describes it, you may feel machine learning algorithms were just invented last month, and there was never "big" data before Google. That is not true. There are new and exciting developments of data science, but many of the techniques we are using are based on decades of work by statisticians, computer scientists, mathematicians and scientists of all types.

In the early 19th century when Legendre and Gauss came up the least squares method for linear regression, only physicists would use it to fit linear regression. Now, even non-technical people can fit linear regressions using excel. In 1936 Fisher came up with linear discriminant analysis. In the 1940s, we had another widely used model – logistic regression. In the 1970s, Nelder and Wedderburn formulated "generalized linear model (GLM)" which:

> "generalized linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value." [from Wikipedia]

[1] I suppose you have seen this before. If not, just search "drew conway venn diagram".

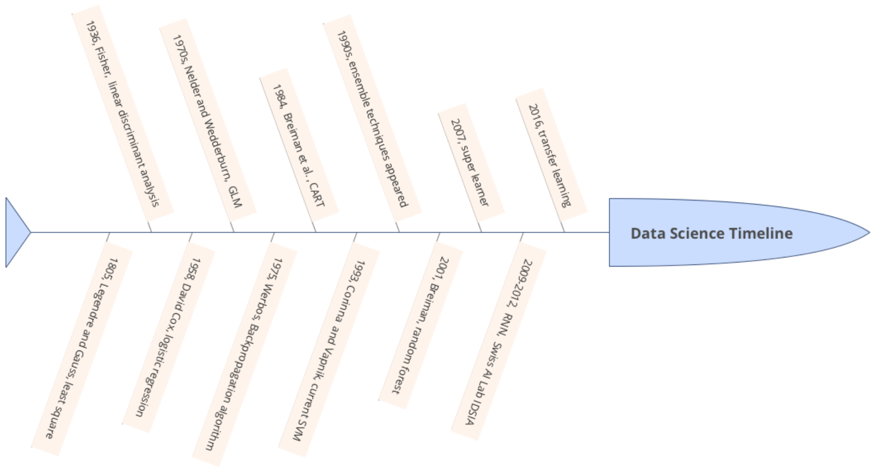

1936, Fisher, linear discriminant analysis
1970s, Nelder and Wedderburn, GLM
1984, Breiman et al., CART
1990s, ensemble techniques appeared
2007, super learner
2016, transfer learning
Data Science Timeline
1805, Legendre and Gauss, least square
1958, David Cox, logistic regression
1975, Werbos, Backpropagation algorithm
1993, Comma and Vapnik, current SVM
2001, Breiman, random forest
2009-2012, RNN, Swiss AI Lab IDSIA

FIGURE 1.1: Data Science Timeline

By the end of the 1970s, there was a range of analytical models and most of them were linear because computers were not powerful enough to fit non-linear model until the 1980s.

In 1984 Breiman et al. introduced classification and regression tree (CART) which is one of the oldest and most utilized classification and regression techniques. After that Ross Quinlan came up with more tree algorithms such as ID3, C4.5, and C5.0. In the 1990s, ensemble techniques (methods that combine many models' predictions) began to appear. Bagging is a general approach that uses bootstrapping in conjunction with any regression or classification model to construct an ensemble. Based on the ensemble idea, Breiman came up with random forest in 2001. Later, Yoav Freund and Robert Schapire came up with the AdaBoost.M1 algorithm. Benefiting from the increasing availability of digitized information, and the possibility to distribute that via the internet, the toolbox has been expanding fast. The applications include business, health, biology, social science, politics, etc.

John Tukey identified four forces driving data analysis (there was no “data science” then):

1. The formal theories of math and statistics
2. Acceleration of developments in computers and display devices
3. The challenge, in many fields, of more and ever larger bodies of data
4. The emphasis on quantification in an ever wider variety of disciplines

Tukey’s 1962 list is surprisingly modern. Let’s inspect those points in today’s context. There is always a time gap between a theory and its application. We had the theories much earlier than application. Fortunately, for the past 50 years, statisticians have been laying the theoretical groundwork for constructing “data science” today. The development of computers enables us to calculate much faster and deliver results in a friendly and intuitive way. The striking transition to the internet of things generates vast amounts of commercial data. Industries have also sensed the value of exploiting that data. Data science seems certain to be a major preoccupation of commercial life in coming decades. All the four forces John identified exist today and have been driving data science.

1.2 What kind of questions can data science solve?

1.2.1 Prerequisites

Data science is not a panacea, and data scientists are not magicians. There are problems data science can’t help. It is best to make a judgment as early in the analytical cycle as possible. Tell your clients honestly and clearly when you think data analytics can’t give the answer they want. What kind of questions can data science solve?

1. Your question needs to be specific enough

Look at two examples:

- Question 1: How can I increase product sales?
- Question 2: Is the new promotional tool introduced at the beginning of this year boosting the annual sales of P1197 in Iowa and Wisconsin? (P1197 is an impressive corn seed product from DuPont Pioneer)

It is easy to see the difference between the two questions. Question 1 is a grammatically correct question, but it is proper for data analysis to answer. Why? It is too general. What is the response variable here? Product sales? Which product? Is it annual sales or monthly sales? What are the candidate predictors? You nearly can't get any useful information from the questions. In contrast, question 2 is much more specific. From the analysis point of view, the response variable is clearly "annual sales of P1197 in Iowa and Wisconsin". Even we don't know all the predictors, but the variable of interest is "the new promotional tool introduced early this year." We want to study the impact of the promotion of the sales. You can start from there and move on to figure out other variables need to include in the model by further communication.

As a data scientist, you may start with something general and unspecific like question 1 and eventually get to question 2. Effective communication and in-depth domain knowledge about the business problem are essential to convert a general business question into a solvable analytical problem. Domain knowledge helps data scientist communicate with the language the other people can understand and obtain the required information.

However, defining the question and variables involved won't guarantee that you can answer it. For example, I encountered this situation with a well-defined supply chain problem. My client asked me to estimate the stock needed for a product in a particular area. Why can't this question be answered? I tried fitting a Multivariate Adaptive Regression Spline (MARS) model and thought I found a reasonable solution. But it turned out later that the data my client gave me was inaccurate. In this case, only estimates rather than actual values of past supply figures were available and there was no way to get accurate data. The lesson lends itself to the next point.

2. You need to have sound and relevant data

One cannot make a silk purse out of a sow's ear. Data scientists need data, sound and relevant data. The supply problem is a case in point. There was relevant data, but not sound. All the later analytics based on that data was a building on sand. Of course, data nearly almost have noise, but it has to be in a certain range. Generally speaking, the accuracy requirement for the independent variables of interest and response variable is higher than others. In question 2, it is data related to the "new promotion" and "sales of P1197".

The data has to be helpful for the question. If you want to predict which product consumers are most likely to buy in the next three months, you need to have historical purchasing data: the last buying time, the amount of invoice, coupons and so on. Information about customers' credit card number, ID number, the email address is not going to help.

Often the quality of the data is more important than the quantity, but the quantity cannot be overlooked. In the premise of guaranteeing quality, usually the more data, the better. If you have a specific and reasonable question, also sound and relevant data, then congratulations, you can start playing data science!

1.2.2 Problem type

Many of the data science books classify the various models from a technical point of view. Such as supervised vs. unsupervised models, linear vs. nonlinear models, parametric models vs. nonparametric models, and so on. Here we will continue on "problem-oriented" track. We first introduce different groups of real problems and then present which models can be used to answer the corresponding category of questions.

1. Description

The basic analytic problem is to summarize and explore a data set with descriptive statistics (mean, standard deviation, and so forth) and visualization methods. It is the simplest problem and yet the most crucial and common one. You will need to describe and explore the dataset before moving on to more complex analysis. In the problem such as customer segmentation, after you cluster the sample, the next step is to figure out the profile of each class by comparing the descriptive statistics of the various variables. Questions of this kind are:

- Is the distribution of annual income normal?
- Are there outliers?
- What are the means of different customer segments?

Data description is often used to check data, find the appropriate data preprocessing method, and demonstrate the model results.

2. Comparison

The first common problem is to compare different groups. Such as: Is A better in some way than B? Or more comparisons: Is there any difference among A, B, and C in a particular aspect? Here are some examples:

- Are males more inclined to buy our products than females?
- Are there any differences in customer satisfaction in different business districts?
- Do soybean carrying a particular gene have higher oil content?

For those problems, it is usually to start exploring from the summary statistics and visualization by groups. After a preliminary visualization, you can test the differences between treatment and control group statistically. The commonly used statistical tests are chi-square test, t-test, and ANOVA. There are also methods using Bayesian methods. In biology industry, such as new drug development, crop breeding, mixed effect models are the dominant technique.

3. Clustering

Clustering is a widespread problem, which is usually related to classification. Clustering answers questions like:

- Which customers have similar product preference?
- Which printer performs similar pattern to the broken ones?
- How many different themes are there in the corpus?

Note that clustering is unsupervised learning. The most common clustering algorithms include K-Means and Hierarchical Clustering.

4. Classification

Usually, a labeled sample set is used as a training set to train the classifier. Then the classifier is used to predict the category of a future sample. Here are some example questions:

- Who is more likely to buy our product?
- Is the borrower going to pay back?
- Is it spam?

There are hundreds of classifiers. In practice, we do not need to try all the models but several models that perform well generally.

5. Regression

In general, regression deals with the problem of "how much is it?" and return a numerical answer. In some cases, it is necessary to coerce the model results to be 0, or round the result to the nearest integer. It is the most common problem.

- What will be the temperature tomorrow?
- What is the projected net income for the next season?
- How much inventory should we have?

1.3 Data Scientist Skill Set

We talked about the bewildering definitions of data scientist. What are the required skills for a data scientist?

- Educational Background

Most of the data scientists today have undergraduate or higher degree from one of the following areas: computer science, electronic engineering, mathematics or statistics. According to a 2017 survey, 25% of US data scientists have a Ph.D. degree, 64% have a Master's degree, and 11% are Bachelors.

- Database Skills

Data scientists in the industry need to use SQL to pull data from the database. So it is necessary to be familiar with how data is structured and how to do basic data manipulation using SQL. Many statistics/mathematics students do not have experience with SQL in school. Don't worry. If you are proficient in one programming language, it is easy to pick up SQL. The main purpose of graduate school should be to develop the ability to learn and analytical thinking rather than the technical skills. Even the technical skills are necessary to enter the professional area. Most of the skills needed at work are not taught in school.

- Programming Skills

Programming skills are critical for data scientists. According to a 2017 survey from Burtch Works[2], 97% of the data scientists today using R or Python. We will provide exemplary code for both in this book. There is not one "have-to-use" tool. The goal is to solve the problem not which tool to choose. However, a good tool needs to be flexible and scalable.

- Modeling Skills

[2] http://www.burtchworks.com/2017/06/19/2017-sas-r-python-flash-survey-results/

Data scientists need to know statistical and machine learning models. There is no clear line separating these two. Many statistical models are also machine learning models, vice versa. Generally speaking, a data scientist is familiar with basic statistical tests such as t-test, chi-square test, and analysis of variance. They can explain the difference between Spearman rank correlation and Pearson correlation, be aware of basic sampling schemes, such as Simple Random Sampling, Stratified Random Sampling, and Multi-Stage Sampling. Know commonly used probability distributions such as Normal distribution, Binomial distribution, Poisson distribution, F distribution, T distribution, and Chi-square distribution. Experimental design plays a significant role in the biological study. Understanding the main tenants of Bayesian methods is necessary (at least be able to write the Bayes theorem on the whiteboard and know what does it mean). Know the difference between supervised and unsupervised learning. Understand commonly used cluster algorithms, classifiers, and regression models. Some powerful tools in predictive analytics are tree models (such as random forest and AdaBoost) and penalized model (such as lasso and SVM). Data scientist working on social science (such as consumer awareness surveys), also needs to know the latent variable model, such as exploratory factor analysis, confirmatory factor analysis, structural equation model.

Is the list getting a little scary? It can get even longer. Don't worry if you don't know all of them now. You will learn as you go. Standard mathematics, statistics or computer science training in graduate school can get you started. But you have to learn lots of new skills after school. Learning is happening increasingly outside of formal educational settings and in unsupervised environments. An excellent data scientist must be a lifetime learner. Fortunately, technological advantages provide new tools and opportunities for lifetime learners, MOOC, online data science workshops and various online tutorials. So above all, being a **life-time learner** is the most critical.

- Soft Skills

In addition to technical knowledge, there are some critical soft skills. These include the ability to translate practical problems into data problems, excellent communication skill, attention to detail, storytelling and so on. We will discuss it in a later chapter in more detail.

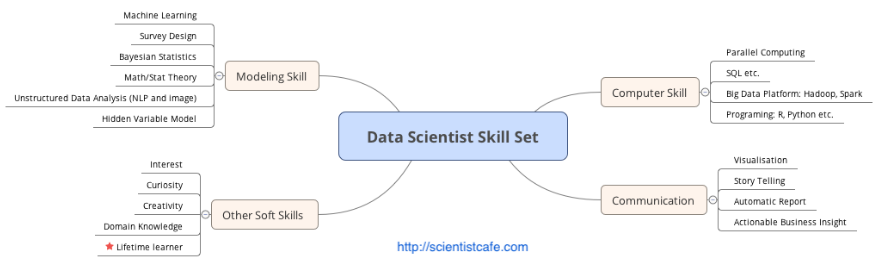


FIGURE 1.2: Data Scientist Skill Set

1.4 Types of Learning

There are three broad groups of styles: supervised learning, unsupervised learning and transfer learning.

In supervised learning, each observation of the predictor measurement(s) corresponds to a response measurement. There are two flavors of supervised learning: regression and classification. In regression, the response is a real number such as the total net sales in 2017, or the yield of corn next year. The goal is to approximate the response measurement as much as possible. In classification, the response is a class label, such as dichotomous response such as yes/no. The response can also have more than two categories, such as four segments of customers. A supervised learning model is a function that maps some input variables with corresponding parameters to a response y. Modeling tuning is to adjust the value of parameters to make the mapping fit the given response. In other words, it is to minimize the discrepancy between given response and the model output. When the response y is a real value, it is intuitive to define discrepancy as the squared difference between model output and given the response. When y is categorical, there are other ways to measure the difference, such as AUC or information gain.

In unsupervised learning, there is no response variable. For a long time, the machine learning community overlooked unsupervised learning except for one called clustering. Moreover, many researchers thought that clustering was the only form of unsupervised learning. One reason is that it is hard to define the goal of unsupervised learning explicitly. Unsupervised learning can be used to do the following:

(1) Identify a good internal representation or pattern of the input that is useful for subsequent supervised or reinforcement learning, such as finding clusters.

(2) It is a dimension reduction tool that is to provide compact, low dimensional representations of the input, such as factor analysis.

(3) Provide a reduced number of uncorrelated learned features from original variables, such as principal component regression.

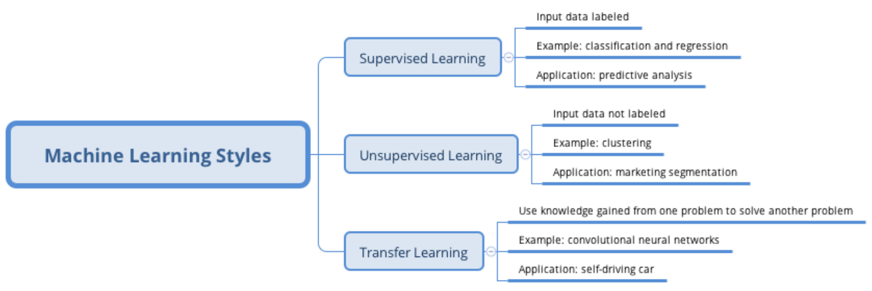

Machine Learning Styles
Supervised Learning
Input data labeled
Example: classification and regression
Application: predictive analysis
Unsupervised Learning
Input data not labeled
Example: clustering
Application: marketing segmentation
Transfer Learning
Use knowledge gained from one problem to solve another problem
Example: convolutional neural networks
Application: self-driving car

FIGURE 1.3: Machine Learning Styles

1.5 Types of Algorithm

The categorization here is based on the structure (such as tree model, Regularization Methods) or type of question to answer (such as regression).[3] It is far less than perfect but will help to show a bigger map of different algorithms. Some can be legitimately classified into multiple categories, such as support vector machine (SVM) can be a classifier, and can also be used for regression. So you may see other ways of grouping. Also, the following summary does not list all the existing algorithms (there are just too many).

1. Regression

Regression can refer to the algorithm or a particular type of problem. It is supervised learning. Regression is one of the oldest and most widely used statistical models. It is often called the statistical machine learning method. Standard regression models are:

- Ordinary Least Squares Regression
- Logistic Regression
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)

The least squares regression and logistic regression are traditional statistical models. Both of them are highly interpretable. MARS is similar to neural networks and partial least squares (PLS) in the respect that they all use surrogate features instead of original predictors.

They differ in how to create the surrogate features. PLS and neural networks use linear combinations of the original predictors as

[3]The summary of various algorithms for data science in this section is based on Jason Brownlee's blog "(A Tour of Machine Learning Algorithms)[http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/]." We added and subtracted some algorithms in each category and gave additional comments.

surrogate features [4]. MARS creates two contrasted versions of a predictor by a truncation point. And LOESS is a non-parametric model, usually only used in visualization.

2. Similarity-based Algorithms

This type of model is based on a similarity measure. There are three main steps: (1) compare the new sample with the existing ones; (2) search for the closest sample; (3) and let the response of the nearest sample be used as the prediction.

- K-Nearest Neighbour [KNN]
- Learning Vector Quantization [LVQ]
- Self-Organizing Map [SOM]

The biggest advantage of this type of model is that they are intuitive. K-Nearest Neighbour is generally the most popular algorithm in this set. The other two are less common. The key to similarity-based algorithms is to find an appropriate distance metric for your data.

3. Feature Selection Algorithms

The primary purpose of feature selection is to exclude non-information or redundant variables and also reduce dimension. Although it is possible that all the independent variables are significant for explaining the response. But more often, the response is only related to a portion of the predictors. We will expand the feature selection in detail later.

- Filter method
- Wrapper method
- Embedded method

Filter method focuses on the relationship between a single feature

[4]To be clear on neural networks, the linear combinations of predictors are put through non-linear activation functions, deeper neural networks have many layers of non-linear transformation

and a target variable. It evaluates each feature (or an independent variable) before modeling and selects “important” variables.

Wrapper method removes the variable according to particular law and finds the feature combination that optimizes the model fitting by evaluating a set of feature combinations. In essence, it is a searching algorithm.

Embedding method is part of the machine learning model. Some model has built-in variable selection function such as lasso, and decision tree.

4. Regularization Method

This method itself is not a complete model, but rather an add-on to other models (such as regression models). It appends a penalty function on the criteria used by the original model to estimate the variables (such as likelihood function or the sum of squared error). In this way, it penalizes model complexity and contracts the model parameters. That is why people call them “shrinkage method.” This approach is advantageous in practice.

- Ridge Regression
- Least Absolute Shrinkage and Selection Operator (LASSO)
- Elastic Net

5. Decision Tree

Decision trees are no doubt one of the most popular machine learning algorithms. Thanks to all kinds of software, implementation is a no-brainer which requires nearly zero understanding of the mechanism. The followings are some of the common trees:

- Classification and Regression Tree (CART)
- Iterative Dichotomiser 3 (ID3)
- C4.5
- Random Forest
- Gradient Boosting Machines (GBM)

6. Bayesian Models

People usually confuse Bayes theorem with Bayesian models. Bayes theorem is an implication of probability theory which gives Bayesian data analysis its name.

$$Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{Pr(y)}$$

The actual Bayesian model is not identical to Bayes theorem. Given a likelihood, parameters to estimate, and a prior for each parameter, a Bayesian model treats the estimates as a purely logical consequence of those assumptions. The resulting estimates are the posterior distribution which is the relative plausibility of different parameter values, conditional on the observations. The Bayesian model here is not strictly in the sense of Bayesian but rather model using Bayes theorem.

- Naïve Bayes
- Averaged One-Dependence Estimators (AODE)
- Bayesian Belief Network (BBN)

7. Kernel Methods

The most common kernel method is the support vector machine (SVM). This type of algorithm maps the input data to a higher order vector space where classification or regression problems are easier to solve.

- Support Vector Machine (SVM)
- Radial Basis Function (RBF)
- Linear Discriminate Analysis (LDA)

8. Clustering Methods

Like regression, when people mention clustering, sometimes they mean a class of problems, sometimes a class of algorithms. The clustering algorithm usually clusters similar samples to categories in a centroidal or hierarchical manner. The two are the most common clustering methods:

- K-Means
- Hierarchical Clustering

9. Association Rule

The basic idea of an association rule is: when events occur together more often than one would expect from their rates of occurrence, such co-occurrence is an interesting pattern. The most used algorithms are:

- Apriori algorithm
- Eclat algorithm

10. Artificial Neural Network

The term neural network has evolved to encompass a repertoire of models and learning methods. There has been lots of hype around the model family making them seem magical and mysterious. A neural network is a two-stage regression or classification model. The basic idea is that it uses linear combinations of the original predictors as surrogate features, and then the new features are put through non-linear activation functions to get hidden units in the 2nd stage. When there are multiple hidden layers, it is called deep learning, another over hyped term. Among varieties of neural network models, the most widely used "vanilla" net is the single hidden layer back-propagation network.

- Perceptron Neural Network
- Back Propagation
- Hopield Network
- Self-Organizing Map (SOM)
- Learning Vector Quantization (LVQ)

11. Deep Learning

The name is a little misleading. As mentioned before, it is multilayer neural network. It is hyped tremendously especially after AlphaGO defeated Li Shishi at the board game Go. We don't have too much experience with the application of deep learning and are not in the right position to talk more about it. Here are some of the common algorithms:

- Restricted Boltzmann Machine (RBN)
- Deep Belief Networks (DBN)
- Convolutional Network
- Stacked Autoencoders
- Long short-term memory (LSTM)

12. Dimensionality Reduction

Its purpose is to construct new features that have significant physical or statistical characteristics, such as capturing as much of the variance as possible.

- Principle Component Analysis (PCA)
- Partial Least Square Regression (PLS)
- Multi-Dimensional Scaling (MDS)
- Exploratory Factor Analysis (EFA)

PCA attempts to find uncorrelated linear combinations of original variables that can explain the variance to the greatest extent possible. EFA also tries to explain as much variance as possible in a lower dimension. MDS maps the observed similarity to a low dimension, such as a two-dimensional plane. Instead of extracting underlying components or latent factors, MDS attempts to find a lower-dimensional map that best preserves all the observed similarities between items. So it needs to define a similarity measure as in clustering methods.

13. Ensemble Methods

Ensemble method made its debut in the 1990s. The idea is to build a prediction model by combining the strengths of a collection of simpler base models. Bagging, originally proposed by Leo Breiman, is one of the earliest ensemble methods. After that, people developed Random Forest (**??**) and Boosting method (**??**). This is a class of powerful and effective algorithms.

- Bootstrapped Aggregation (Bagging)
- Random Forest
- Gradient Boosting Machine (GBM)

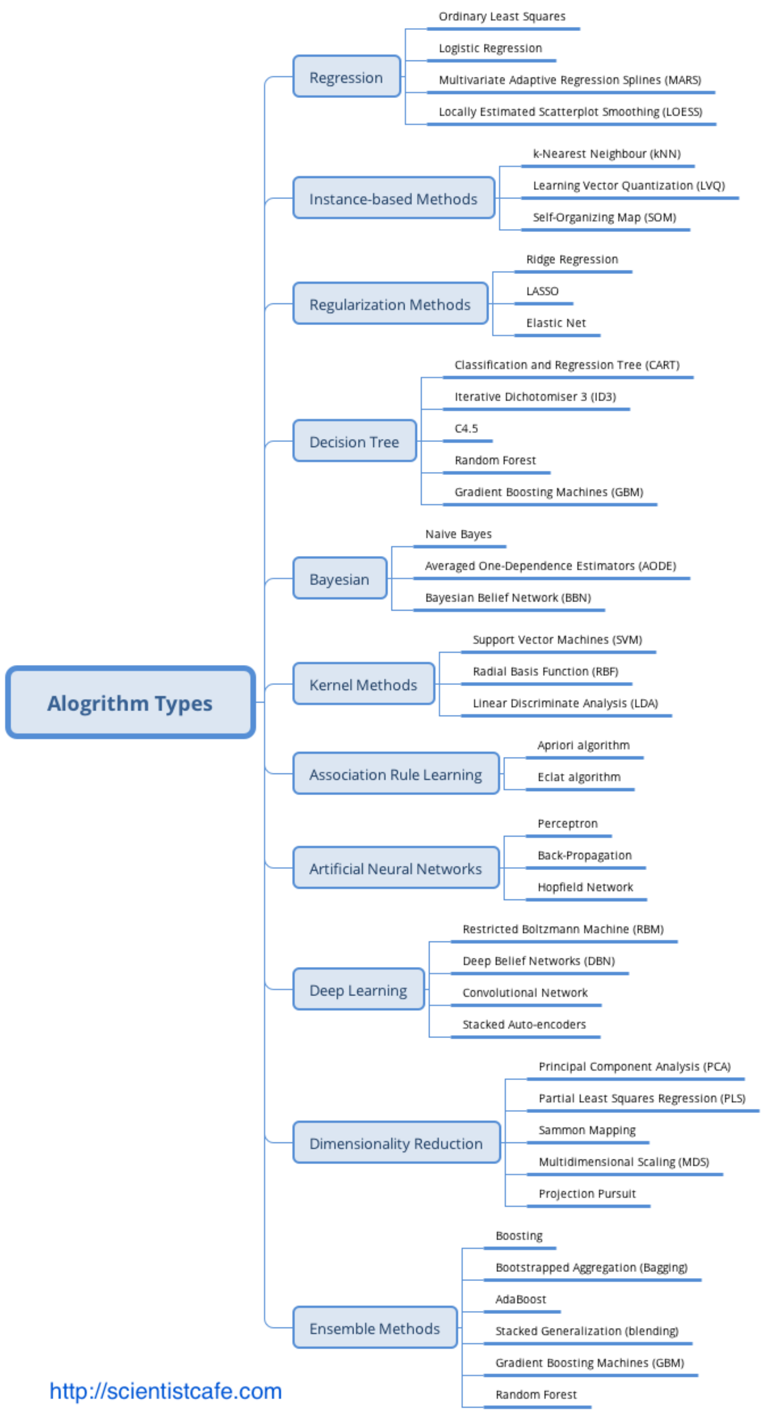

Alogrithm Types
Regression
Ordinary Least Squares
Logistic Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Instance-based Methods
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Regularization Methods
Ridge Regression
LASSO
Elastic Net
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
Random Forest
Gradient Boosting Machines (GBM)
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Kernel Methods
Support Vector Machines (SVM)
Radial Basis Function (RBF)
Linear Discriminate Analysis (LDA)
Association Rule Learning
Apriori algorithm
Eclat algorithm
Artificial Neural Networks
Perceptron
Back-Propagation
Hopfield Network
Deep Learning
Restricted Boltzmann Machine (RBM)
Deep Belief Networks (DBN)
Convolutional Network
Stacked Auto-encoders
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLS)
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Ensemble Methods
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (blending)
Gradient Boosting Machines (GBM)
Random Forest
http://scientistcafe.com

FIGURE 1.4: Machines Learning Algorithms

2

Introduction to the data

Before tackling analytics problem, we start by introducing data to be analyzed in later chapters.

2.1 Customer Data for Clothing Company

Our first data set represents customers of a clothing company who sells products in stores and online. This data is typical of what one might get from a company's marketing data base (the data base will have more data than the one we show here). This data includes 1000 customers for whom we have 3 types of data:

1. Demography
    - `age`: age of the respondent
    - `gender`: male/female
    - `house`: 0/1 variable indicating if the customer owns a house or not
2. Sales in the past year
    - `store_exp`: expense in store
    - `online_exp`: expense online
    - `store_trans`: times of store purchase
    - `online_trans`: times of online purchase
3. Survey on product preference

It is common for companies to survey their customers and draw insights to guide future marketing activities. The survey is as below:

How strongly do you agree or disagree with the following statements:

1. Strong disagree
2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

- Q1. I like to buy clothes from different brands
- Q2. I buy almost all my clothes from some of my favorite brands
- Q3. I like to buy premium brands
- Q4. Quality is the most important factor in my purchasing decision
- Q5. Style is the most important factor in my purchasing decision
- Q6. I prefer to buy clothes in store
- Q7. I prefer to buy clothes online
- Q8. Price is important
- Q9. I like to try different styles
- Q10. I like to make a choice by myself and don't need too much of others' suggestions

There are 4 segments of customers:

1. Price
2. Conspicuous
3. Quality
4. Style

Let's check it:

```r
str(sim.dat,vec.len=3)
```

```
## 'data.frame':    1000 obs. of  19 variables:
##  $ age         : int  57 63 59 60 51 59 57 57 ...
##  $ gender      : Factor w/ 2 levels "Female","Male": 1 1 2 2 2 2 2 2 ...
##  $ income      : num  120963 122008 114202 113616 ...
##  $ house       : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 ...
##  $ store_exp   : num  529 478 491 348 ...
##  $ online_exp  : num  304 110 279 142 ...
##  $ store_trans : int  2 4 7 10 4 4 5 11 ...
##  $ online_trans: int  2 2 2 2 4 5 3 5 ...
##  $ Q1          : int  4 4 5 5 4 4 4 5 ...
##  $ Q2          : int  2 1 2 2 1 2 1 2 ...
##  $ Q3          : int  1 1 1 1 1 1 1 1 ...
##  $ Q4          : int  2 2 2 3 3 2 2 3 ...
##  $ Q5          : int  1 1 1 1 1 1 1 1 ...
##  $ Q6          : int  4 4 4 4 4 4 4 4 ...
##  $ Q7          : int  1 1 1 1 1 1 1 1 ...
##  $ Q8          : int  4 4 4 4 4 4 4 4 ...
##  $ Q9          : int  2 1 1 2 2 1 1 2 ...
##  $ Q10         : int  4 4 4 4 4 4 4 4 ...
##  $ segment     : Factor w/ 4 levels "Conspicuous",..: 2 2 2 2 2 2 2 2 ...
```

2.2 Customer Satisfaction Survey Data from Airline Company

This data set is from a customer satisfaction survey for three airline companies. There are `N=1000` respondents and 15 questions. The market researcher asked respondents to recall the experience with different airline companies and assign a score (1-9) to each airline company for all the 15 questions. The higher the score, the more satisfied the customer to the specific item. The 15 questions are of 4 types (the variable names are in the parentheses):

- How satisfied are you with your_________?

1. Ticketing
    - Ease of making reservation Easy_Reservation
    - Availability of preferred seats Preferred_Seats
    - Variety of flight options Flight_Options
    - Ticket prices Ticket_Prices
2. Aircraft
    - Seat comfort Seat_Comfort

- Roominess of seat area Seat_Roominess
- Availability of Overhead Overhead_Storage
- Cleanliness of aircraft Clean_Aircraft

3. Service
    - Courtesy of flight attendant Courtesy
    - Friendliness Friendliness
    - Helpfulness Helpfulness
    - Food and drinks Service
4. General
    - Overall satisfaction Satisfaction
    - Purchase again Fly_Again
    - Willingness to recommend Recommend

Now check the data frame we have:

```
## Parsed with column specification:
## cols(
##   Easy_Reservation = col_integer(),
##   Preferred_Seats = col_integer(),
##   Flight_Options = col_integer(),
##   Ticket_Prices = col_integer(),
##   Seat_Comfort = col_integer(),
##   Seat_Roominess = col_integer(),
##   Overhead_Storage = col_integer(),
##   Clean_Aircraft = col_integer(),
##   Courtesy = col_integer(),
##   Friendliness = col_integer(),
##   Helpfulness = col_integer(),
##   Service = col_integer(),
##   Satisfaction = col_integer(),
##   Fly_Again = col_integer(),
##   Recommend = col_integer(),
##   ID = col_integer(),
##   Airline = col_character()
## )
```

```
str(rating,vec.len=3)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    3000 obs. of  17 variables:
```

```
##  $ Easy_Reservation: int  6 5 6 5 4 5 6 4 ...
##  $ Preferred_Seats : int  5 7 6 6 5 6 6 6 ...
##  $ Flight_Options  : int  4 7 5 5 3 4 6 3 ...
##  $ Ticket_Prices   : int  5 6 6 5 6 5 5 5 ...
##  $ Seat_Comfort    : int  5 6 7 7 6 6 6 4 ...
##  $ Seat_Roominess  : int  7 8 6 8 7 8 6 5 ...
##  $ Overhead_Storage: int  5 5 7 6 5 4 4 4 ...
##  $ Clean_Aircraft  : int  7 6 7 7 7 7 6 4 ...
##  $ Courtesy        : int  5 6 6 4 2 5 5 4 ...
##  $ Friendliness    : int  4 6 6 6 3 4 5 5 ...
##  $ Helpfulness     : int  6 5 6 4 4 5 5 4 ...
##  $ Service         : int  6 5 6 5 3 5 5 5 ...
##  $ Satisfaction    : int  6 7 7 5 4 6 5 5 ...
##  $ Fly_Again       : int  6 6 6 7 4 5 3 4 ...
##  $ Recommend       : int  3 6 5 5 4 5 6 5 ...
##  $ ID              : int  1 2 3 4 5 6 7 8 ...
##  $ Airline         : chr  "AirlineCo.1" "AirlineCo.1" "AirlineCo.1" ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 17
##   .. ..$ Easy_Reservation: list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Preferred_Seats : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Flight_Options  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Ticket_Prices   : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Seat_Comfort    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Seat_Roominess  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Overhead_Storage: list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Clean_Aircraft  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Courtesy        : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Friendliness    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Helpfulness     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Service         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Satisfaction    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Fly_Again       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Recommend       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ ID              : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ Airline         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
```

3

Data Pre-processing

Many data analysis related books focus on models, algorithms and statistical inferences. However, in practice, raw data is usually not directly used for modeling. Data preprocessing is the process of converting raw data into clean data that is proper for modeling. A model fails for various reasons. One is that the modeler doesn't correctly preprocess data before modeling. Data preprocessing can significantly impact model results, such as imputing missing value and handling with outliers. So data preprocessing is a very critical part.

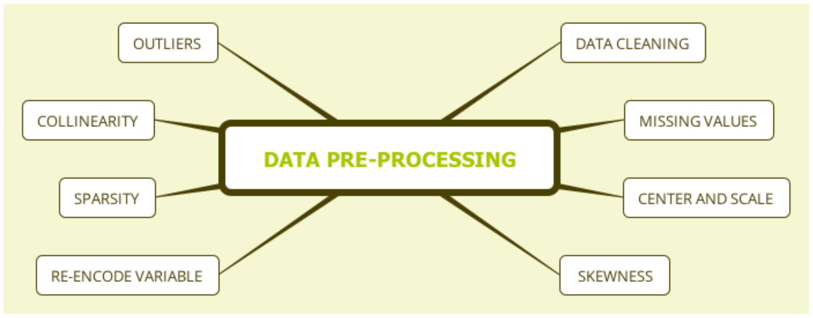


FIGURE 3.1: Data Pre-processing Outline

In real life, depending on the stage of data cleanup, data has the following types:

1. Raw data
2. Technically correct data
3. Data that is proper for the model
4. Summarized data
5. Data with fixed format

The raw data is the first-hand data that analysts pull from the database, market survey responds from your clients, the experimental results collected by the R & D department, and so on. These data may be very rough, and R sometimes can't read them directly. The table title could be multi-line, or the format does not meet the requirements:

- Use 50% to represent the percentage rather than 0.5, so R will read it as a character;
- The missing value of the sales is represented by "-" instead of space so that R will treat the variable as character or factor type;
- The data is in a slideshow document, or the spreadsheet is not ".csv" but ".xlsx"
- ...

Most of the time, you need to clean the data so that R can import them. Some data format requires a specific package. Technically correct data is the data, after preliminary cleaning or format conversion, that R (or another tool you use) can successfully import it.

Assume we have loaded the data into R with reasonable column names, variable format and so on. That does not mean the data is entirely correct. There may be some observations that do not make sense, such as age is negative, the discount percentage is greater than 1, or data is missing. Depending on the situation, there may be a variety of problems with the data. It is necessary to clean the data before modeling. Moreover, different models have different requirements on the data. For example, some model may require the variables are of consistent scale; some may be susceptible to outliers or collinearity, some may not be able to handle categorical variables and so on. The modeler has to preprocess the data to make it proper for the specific model.

Sometimes we need to aggregate the data. For example, add up the daily sales to get annual sales of a product at different locations. In customer segmentation, it is common practice to build a profile for each segment. It requires calculating some statistics such as average age, average income, age standard deviation, etc.

Data aggregation is also necessary for presentation, or for data visualization.

The final table results for clients need to be in a nicer format than what used in the analysis. Usually, data analysts will take the results from data scientists and adjust the format, such as labels, cell color, highlight. It is important for a data scientist to make sure the results look consistent which makes the next step easier for data analysts.

It is highly recommended to store each step of the data and the R code, making the whole process as repeatable as possible. The R markdown reproducible report will be extremely helpful for that. If the data changes, it is easy to rerun the process. In the remainder of this chapter, we will show the most common data preprocessing methods.

Load the R packages first:

```r
source("https://raw.githubusercontent.com/happyrabbit/CE_JSM2017/master/Rcode/00-course-setup.
```

3.1 Data Cleaning

After you load the data, the first thing is to check how many variables are there, the type of variables, the distributions, and data errors. Let's read and check the data:

```r
sim.dat <- read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
summary(sim.dat)
```

```
##       age           gender         income
##  Min.   : 16.0   Female:554   Min.   : 41776
##  1st Qu.: 25.0   Male  :446   1st Qu.: 85832
##  Median : 36.0                Median : 93869
##  Mean   : 38.8                Mean   :113543
##  3rd Qu.: 53.0                3rd Qu.:124572
##  Max.   :300.0                Max.   :319704
##                               NA's   :184
##  house       store_exp       online_exp
##  No :432   Min.   : -500   Min.   :  69
##  Yes:568   1st Qu.:  205   1st Qu.: 420
##            Median :  329   Median :1942
##            Mean   : 1357   Mean   :2120
##            3rd Qu.:  597   3rd Qu.:2441
##            Max.   :50000   Max.   :9479
##
##   store_trans     online_trans        Q1
##  Min.   : 1.00   Min.   : 1.0   Min.   :1.0
##  1st Qu.: 3.00   1st Qu.: 6.0   1st Qu.:2.0
##  Median : 4.00   Median :14.0   Median :3.0
##  Mean   : 5.35   Mean   :13.6   Mean   :3.1
##  3rd Qu.: 7.00   3rd Qu.:20.0   3rd Qu.:4.0
##  Max.   :20.00   Max.   :36.0   Max.   :5.0
##
##        Q2             Q3             Q4
##  Min.   :1.00   Min.   :1.00   Min.   :1.00
##  1st Qu.:1.00   1st Qu.:1.00   1st Qu.:2.00
##  Median :1.00   Median :1.00   Median :3.00
##  Mean   :1.82   Mean   :1.99   Mean   :2.76
##  3rd Qu.:2.00   3rd Qu.:3.00   3rd Qu.:4.00
##  Max.   :5.00   Max.   :5.00   Max.   :5.00
##
##        Q5             Q6             Q7
##  Min.   :1.00   Min.   :1.00   Min.   :1.00
##  1st Qu.:1.75   1st Qu.:1.00   1st Qu.:2.50
##  Median :4.00   Median :2.00   Median :4.00
##  Mean   :2.94   Mean   :2.45   Mean   :3.43
##  3rd Qu.:4.00   3rd Qu.:4.00   3rd Qu.:4.00
##  Max.   :5.00   Max.   :5.00   Max.   :5.00
##
##        Q8            Q9             Q10
##  Min.   :1.0   Min.   :1.00   Min.   :1.00
##  1st Qu.:1.0   1st Qu.:2.00   1st Qu.:1.00
##  Median :2.0   Median :4.00   Median :2.00
##  Mean   :2.4   Mean   :3.08   Mean   :2.32
##  3rd Qu.:3.0   3rd Qu.:4.00   3rd Qu.:3.00
##  Max.   :5.0   Max.   :5.00   Max.   :5.00
##
##        segment
##  Conspicuous:200
##  Price      :250
##  Quality    :200
##  Style      :350
##
##
##
```

Are there any problems? Questionnaire response Q1-Q10 seem reasonable, the minimum is 1 and maximum is 5. Recall that the questionnaire score is 1-5. The number of store transactions (store_trans) and online transactions (store_trans) make sense too. Things need to pay attention are:

- There are some missing values.
- There are outliers for store expenses (`store_exp`). The maximum value is 50000. Who would spend $50000 a year buying clothes? Is it an imputation error?
- There is a negative value (-500) in `store_exp` which is not logical.
- Someone is 300 years old.

How to deal with that? Depending on the real situation, if the sample size is large enough, it will not hurt to delete those problematic samples. Here we have 1000 observations. Since marketing survey is usually expensive, it is better to set these values as missing and impute them instead of deleting the rows.

```
# set problematic values as missings
sim.dat$age[which(sim.dat$age>100)]<-NA
sim.dat$store_exp[which(sim.dat$store_exp<0)]<-NA
# see the results
summary(subset(sim.dat,select=c("age","income")))
```

```
##       age            income
##  Min.   :16.0   Min.   : 41776
##  1st Qu.:25.0   1st Qu.: 85832
##  Median :36.0   Median : 93869
##  Mean   :38.6   Mean   :113543
##  3rd Qu.:53.0   3rd Qu.:124572
##  Max.   :69.0   Max.   :319704
##  NA's   :1      NA's   :184
```

Now we will deal with the missing values in the data.

3.2 Missing Values

You can write a whole book about missing value. This section will only show some of the most commonly used methods without getting too deep into the topic. Chapter 7 of the book by De Waal, Pannekoek and Scholtus (**?**) makes a concise overview of some of the existing imputation methods. The choice of specific method depends on the actual situation. There is no best way.

One question to ask before imputation: Is there any auxiliary information? Being aware of any auxiliary information is critical. For example, if the system set customer who did not purchase as missing, then the real purchasing amount should be 0. Is missing a random occurrence? If so, it may be reasonable to impute with mean or median. If not, is there a potential mechanism for the missing data? For example, older people are more reluctant to disclose their ages in the questionnaire, so that the absence of age is not completely random. In this case, the missing values need to be estimated using the relationship between age and other independent variables. For example, use variables such as whether they have children, income, and other survey questions to build a model to predict age.

Also, the purpose of modeling is important for selecting imputation methods. If the goal is to interpret the parameter estimate or statistical inference, then it is important to study the missing mechanism carefully and to estimate the missing values using non-missing information as much as possible. If the goal is to predict, people usually will not study the absence mechanism rigorously (but sometimes the mechanism is obvious). If the absence mechanism is not clear, treat it as missing at random and use mean, median, or k-nearest neighbor to impute. Since statistical inference is sensitive to missing values, researchers from survey statistics have conducted in-depth studies of various imputation schemes which focus on valid statistical inference. The problem of missing values in the prediction model is different from that in the traditional survey. Therefore, there are not many papers on missing value imputation in the prediction model. Those who want to study further can refer to Saar-Tsechansky and Provost's comparison of different imputation methods (**?**) and De Waal, Pannekoek and Scholtus' book (**?**).

3.2.1 Impute missing values with median/mode

In the case of missing at random, a common method is to impute with the mean (continuous variable) or median (categorical variables). You can use `impute()` function in `imputeMissings` package.

```r
# save the result as another object
demo_imp<-impute(sim.dat,method="median/mode")
# check the first 5 columns, there is no missing values in other columns
summary(demo_imp[,1:5])
```

```
##       age           gender         income      
##  Min.   :16.0   Female:554   Min.   : 41776  
##  1st Qu.:25.0   Male  :446   1st Qu.: 87896  
##  Median :36.0                Median : 93869  
##  Mean   :38.6                Mean   :109923  
##  3rd Qu.:53.0                3rd Qu.:119456  
##  Max.   :69.0                Max.   :319704  
##  house       store_exp    
##  No :432   Min.   :  156  
##  Yes:568   1st Qu.:  205  
##            Median :  330  
##            Mean   : 1358  
##             3rd Qu.:   597  
##             Max.   :50000  
```

After imputation, `demo_imp` has no missing value. This method is straightforward and widely used. The disadvantage is that it does not take into account the relationship between the variables. When there is a significant proportion of missing, it will distort the data. In this case, it is better to consider the relationship between variables and study the missing mechanism. In the example here, the missing variables are numeric. If the missing variable is a categorical/factor variable, the `impute()` function will impute with the mode.

You can also use `preProcess()` function, but it is only for numeric variables, and can not impute categorical variables. Since missing values here are numeric, we can use the `preProcess()` function. The result is the same as the `impute()` function. `PreProcess()` is a powerful function that can link to a variety of data preprocessing methods. We will use the function later for other data preprocessing.

```
imp<-preProcess(sim.dat,method="medianImpute")
demo_imp2<-predict(imp,sim.dat)
summary(demo_imp2[,1:5])
```

```
##       age           gender          income      
##  Min.   :16.0   Female:554   Min.   : 41776  
##  1st Qu.:25.0   Male  :446   1st Qu.: 87896  
##  Median :36.0                Median : 93869  
##  Mean   :38.6                Mean   :109923  
##  3rd Qu.:53.0                3rd Qu.:119456  
##  Max.   :69.0                Max.   :319704  
##  house        store_exp    
##  No :432   Min.   :  156  
##  Yes:568   1st Qu.:  205  
##            Median :  330  
##            Mean   : 1358  
##            3rd Qu.:  597  
##            Max.   :50000  
```

3.2.2 K-nearest neighbors

K-nearest neighbor (KNN) will find the k closest samples (Euclidian distance) in the training set and impute the mean of those "neighbors."

Use `preProcess()` to conduct KNN:

```r
imp<-preProcess(sim.dat,method="knnImpute",k=5)
# need to use predict() to get KNN result
demo_imp<-predict(imp,sim.dat)
```

```
Error in `[.data.frame`(old, , non_missing_cols, drop = FALSE) :
  undefined columns selected
```

Now we get an error saying "undefined columns selected." It is because `sim.dat` has non-numeric variables. The `preProcess()` in the first line will automatically ignore non-numeric columns, so there is no error. However, there is a problem when using `predict()` to get the result. Removing those variable will solve the problem.

```r
# find factor columns
imp<-preProcess(sim.dat,method="knnImpute",k=5)
idx<-which(lapply(sim.dat,class)=="factor")
demo_imp<-predict(imp,sim.dat[,-idx])
summary(demo_imp[,1:3])
```

```
##       age              income          store_exp
##  Min.   :-1.591   Min.   :-1.440   Min.   :-0.433
##  1st Qu.:-0.957   1st Qu.:-0.537   1st Qu.:-0.416
##  Median :-0.182   Median :-0.376   Median :-0.371
##  Mean   : 0.000   Mean   : 0.024   Mean   : 0.000
##  3rd Qu.: 1.016   3rd Qu.: 0.215   3rd Qu.:-0.274
##  Max.   : 2.144   Max.   : 4.136   Max.   :17.527
```

`lapply(data,class)` can return a list of column class. Here the data frame is `sim.dat`, and the following code will give the list of column class:

```r
# only show the first three elements
lapply(sim.dat,class)[1:3]
```

```
## $age
## [1] "integer"
##
## $gender
## [1] "factor"
##
## $income
## [1] "numeric"
```

Comparing the KNN result with the previous median imputation, the two are very different. This is because when you tell the `preProcess()` function to use KNN (the option `method =" knnImpute"`), it will automatically standardize the data. Another way is to use Bagging tree (in the next section). Note that KNN can not impute samples with the entire row missing. The reason is straightforward. Since the algorithm uses the average of its neighbors if none of them has a value, what does it apply to calculate the mean? Let's append a new row with all values missing to the original data frame to get a new object called `temp`. Then apply KNN to `temp` and see what happens:

```
temp<-rbind(sim.dat,rep(NA,ncol(sim.dat)))
imp<-preProcess(sim.dat,method="knnImpute",k=5)
idx<-which(lapply(temp,class)=="factor")
```

```
demo_imp<-predict(imp,temp[,-idx])
```

```
Error in FUN(newX[, i], ...) :
  cannot impute when all predictors are missing in the new data point
```

There is an error saying "cannot impute when all predictors are missing in the new data point". It is easy to fix by finding and removing the problematic row:

```
idx<-apply(temp,1,function(x) sum(is.na(x)) )
as.vector(which(idx==ncol(temp)))
```

```
## [1] 1001
```

It shows that row 1001 is problematic. You can go ahead to delete it.

3.2.3 Bagging Tree

Bagging (Bootstrap aggregating) was originally proposed by Leo Breiman. It is one of the earliest ensemble methods (**?**). When used in missing value imputation, it will use the remaining variables as predictors to train a bagging tree and then use the tree to predict the missing values. Although theoretically, the method is powerful, the computation is much more intense than KNN. In practice, there is a trade-off between computation time and the effect. If a median or mean meet the modeling needs, even bagging tree may improve the accuracy a little, but the upgrade is so marginal that it does not deserve the extra time. The bagging tree itself is a model for regression and classification. Here we use `preProcess()` to impute `sim.dat`:

```
imp<-preProcess(sim.dat,method="bagImpute")
demo_imp<-predict(imp,sim.dat)
summary(demo_imp[,1:5])
```

```
      age           gender         income           house        store_exp
 Min.   :16.00   Female:554   Min.   : 41776   No :432   Min.   :   155.8
 1st Qu.:25.00   Male  :446   1st Qu.: 86762   Yes:568   1st Qu.:   205.1
 Median :36.00                Median : 94739             Median :   329.0
 Mean   :38.58                Mean   :114665             Mean   : 1357.7
 3rd Qu.:53.00                3rd Qu.:123726             3rd Qu.:   597.3
 Max.   :69.00                Max.   :319704             Max.   :50000.0
```

3.3 Centering and Scaling

It is the most straightforward data transformation. It centers and scales a variable to mean 0 and standard deviation 1. It ensures that the criterion for finding linear combinations of the predictors is based on how much variation they explain and therefore improves the numerical stability. Models involving finding linear combinations of the predictors to explain response/predictors variation need data centering and scaling, such as PCA (?), PLS (?) and EFA (?). You can quickly write code yourself to conduct this transformation.

Let's standardize the variable `income` from `sim.dat`:

```r
income<-sim.dat$income
# calculate the mean of income
mux<-mean(income,na.rm=T)
# calculate the standard deviation of income
sdx<-sd(income,na.rm=T)
# centering
tr1<-income-mux
# scaling
tr2<-tr1/sdx
```

Or the function `preProcess()` in package `caret` can apply this transformation to a set of predictors.

```r
sdat<-subset(sim.dat,select=c("age","income"))
# set the "method" option
trans<-preProcess(sdat,method=c("center","scale"))
# use predict() function to get the final result
transformed<-predict(trans,sdat)
```

Now the two variables are in the same scale:

```r
summary(transformed)
```

```
##       age              income     
##  Min.   :-1.591   Min.   :-1.44  
##  1st Qu.:-0.957   1st Qu.:-0.56  
##  Median :-0.182   Median :-0.39  
##  Mean   : 0.000   Mean   : 0.00  
##  3rd Qu.: 1.016   3rd Qu.: 0.22  
##  Max.   : 2.144   Max.   : 4.14  
##  NA's   :1        NA's   :184    
```

Sometimes you only need to scale the variable. For example, if the model adds a penalty to the parameter estimates (such as L_2 penalty is ridge regression and L_1 penalty in LASSO), the variables need to have a similar scale to ensure a fair variable selection. I am a heavy user of this kind of penalty-based model in my work, and I used the following quantile transformation:

$$x_{ij}^{*} = \frac{x_{ij} - quantile(x_{.j}, 0.01)}{quantile(x_{.j} - 0.99) - quantile(x_{-j}, 0.01)}$$

The reason to use 99% and 1% quantile instead of maximum and minimum values is to resist the impact of outliers.

It is easy to write a function to do it:

```r
qscale<-function(dat){
  for (i in 1:ncol(dat)){
    up<-quantile(dat[,i],0.99)
    low<-quantile(dat[,i],0.01)
    diff<-up-low
    dat[,i]<-(dat[,i]-low)/diff
  }
  return(dat)
}
```

In order to illustrate, let's apply it to some variables from ‘demo_imp2:

```r
demo_imp3<-qscale(subset(demo_imp2,select=c("income","store_exp","online_exp")))
summary(demo_imp3)
```

```
##      income          store_exp        online_exp
##  Min.   :-0.0578   Min.   :-0.003   Min.   :-0.0060
##  1st Qu.: 0.1574   1st Qu.: 0.004   1st Qu.: 0.0427
##  Median : 0.1852   Median : 0.023   Median : 0.2537
##  Mean   : 0.2601   Mean   : 0.177   Mean   : 0.2784
##  3rd Qu.: 0.3046   3rd Qu.: 0.063   3rd Qu.: 0.3229
##  Max.   : 1.2386   Max.   : 7.477   Max.   : 1.2988
```

After transformation, most of the variables are between 0-1.

3.4 Resolve Skewness

Skewness[1] is defined to be the third standardized central moment. The formula for the sample skewness statistics is:

$$skewness = \frac{\sum(x_i - \bar{x})^3}{(n-1)v^{3/2}}$$

$$v = \frac{\sum(x_i - \bar{x})^2}{(n-1)}$$

Skewness=0 means that the destribution is symmetric, i.e. the probability of falling on either side of the distribution's mean is equal.

```r
# need skewness() function from e1071 package
set.seed(1000)
par(mfrow=c(1,2),oma=c(2,2,2,2))
# random sample 1000 chi-square distribution with df=2
# right skew
x1<-rchisq(1000,2, ncp = 0)
# get left skew variable x2 from x1
x2<-max(x1)-x1
plot(density(x2),main=paste("left skew, skewnwss =",round(skewness(x2),2)), xlab="X2")
plot(density(x1),main=paste("right skew, skewness =",round(skewness(x1),2)), xlab="X1")
```

You can easily tell if a distribution is skewed by simple visualization(Figure**??**). There are different ways may help to remove skewness such as log, square root or inverse. However, it is often difficult to determine from plots which transformation is most appropriate for correcting skewness. The Box-Cox procedure automatically identified a transformation from the family of power transformations that are indexed by a parameter λ(**?**).

$$x^* = \begin{cases} \frac{x^\lambda - 1}{\lambda} & if\ \lambda \neq 0 \\ log(x) & if\ \lambda = 0 \end{cases}$$

[1] https://en.wikipedia.org/wiki/Skewness

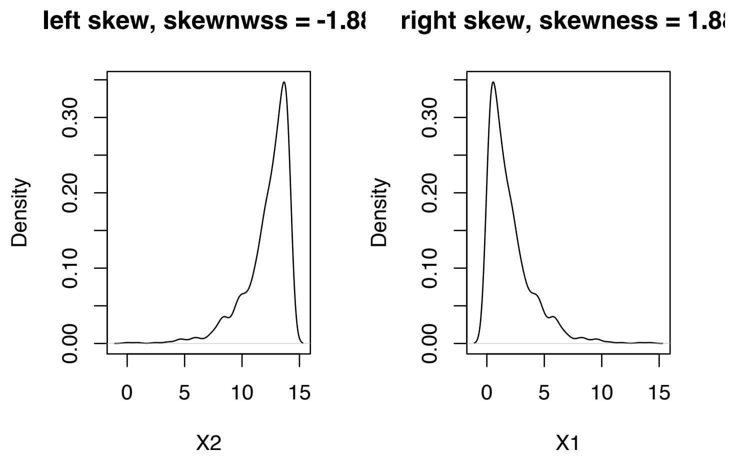


FIGURE 3.2: Shewed Distribution

It is easy to see that this family includes log transformation ($\lambda = 0$), square transformation ($\lambda = 2$), square root ($\lambda = 0.5$), inverse ($\lambda = -1$) and others in-between. We can still use function `preProcess()` in package `caret` to apply this transformation by chaning the `method` argument.

```
describe(sim.dat)
```

```
##              vars    n      mean       sd  median
## age             1  999     38.58    14.19    36.0
## gender*         2 1000      1.45     0.50     1.0
## income          3  816 113543.07 49842.29 93868.7
## house*          4 1000      1.57     0.50     2.0
## store_exp       5  999   1358.71  2775.17   329.8
## online_exp      6 1000   2120.18  1731.22  1941.9
## store_trans     7 1000      5.35     3.70     4.0
## online_trans    8 1000     13.55     7.96    14.0
## Q1              9 1000      3.10     1.45     3.0
## Q2             10 1000      1.82     1.17     1.0
## Q3             11 1000      1.99     1.40     1.0
## Q4             12 1000      2.76     1.16     3.0
## Q5             13 1000      2.94     1.28     4.0
## Q6             14 1000      2.45     1.44     2.0
## Q7             15 1000      3.43     1.46     4.0
## Q8             16 1000      2.40     1.15     2.0
## Q9             17 1000      3.08     1.12     4.0
## Q10            18 1000      2.32     1.14     2.0
## segment*       19 1000      2.70     1.15     3.0
##                trimmed      mad      min    max  range
## age              37.67    16.31    16.00     69     53
## gender*           1.43     0.00     1.00      2      1
## income       104841.94 28989.47 41775.64 319704 277929
## house*            1.58     0.00     1.00      2      1
## store_exp       845.14   197.47   155.81  50000  49844
## online_exp     1874.51  1015.21    68.82   9479   9411
## store_trans       4.89     2.97     1.00     20     19
## online_trans     13.42    10.38     1.00     36     35
## Q1                3.13     1.48     1.00      5      4
## Q2                1.65     0.00     1.00      5      4
## Q3                1.75     0.00     1.00      5      4
## Q4                2.83     1.48     1.00      5      4
## Q5                3.05     0.00     1.00      5      4
## Q6                2.43     1.48     1.00      5      4
## Q7                3.54     0.00     1.00      5      4
## Q8                2.36     1.48     1.00      5      4
## Q9                3.23     0.00     1.00      5      4
## Q10               2.27     1.48     1.00      5      4
## segment*          2.75     1.48     1.00      4      3
##               skew kurtosis      se
## age           0.47    -1.18    0.45
## gender*       0.22    -1.95    0.02
## income        1.69     2.57 1744.83
## house*       -0.27    -1.93    0.02
## store_exp     8.08   115.04   87.80
## online_exp    1.18     1.31   54.75
## store_trans   1.11     0.69    0.12
## online_trans  0.03    -0.98    0.25
## Q1           -0.12    -1.36    0.05
## Q2            1.13    -0.32    0.04
## Q3            1.06    -0.40    0.04
## Q4           -0.18    -1.46    0.04
## Q5            -0.60    -1.40    0.04
## Q6             0.11    -1.89    0.05
## Q7            -0.90    -0.79    0.05
## Q8             0.21    -1.33    0.04
## Q9            -0.68    -1.10    0.04
## Q10            0.39    -1.23    0.04
## segment*      -0.20    -1.41    0.04
```

It is easy to see the skewed variables. If `mean` and `trimmed` differ a lot, there is very likely outliers. By default, `trimmed` reports mean by dropping the top and bottom 10%. It can be adjusted by setting argument `trim=`. It is clear that `store_exp` has outliers.

As an example, we will apply Box-Cox transformation on `store_trans` and `online_trans`:

```
# select the two columns and save them as dat_bc
dat_bc<-subset(sim.dat,select=c("store_trans","online_trans"))
(trans<-preProcess(dat_bc,method=c("BoxCox")))
```

```
## Created from 1000 samples and 2 variables
##
## Pre-processing:
##   - Box-Cox transformation (2)
##   - ignored (0)
##
## Lambda estimates for Box-Cox transformation:
## 0.1, 0.7
```

The last line of the output shows the estimates of λ for each variable. As before, use `predict()` to get the transformed result:

```
transformed<-predict(trans,dat_bc)
par(mfrow=c(1,2),oma=c(2,2,2,2))
hist(dat_bc$store_trans,main="Before Transformation",xlab="store_trans")
hist(transformed$store_trans,main="After Transformation",xlab="store_trans")
```

Before the transformation, the `stroe_trans` is skewed right. The situation is significantly improved after (figure??). `BoxCoxTrans ()` can also conduct Box-Cox transform. But note that `BoxCoxTrans ()`

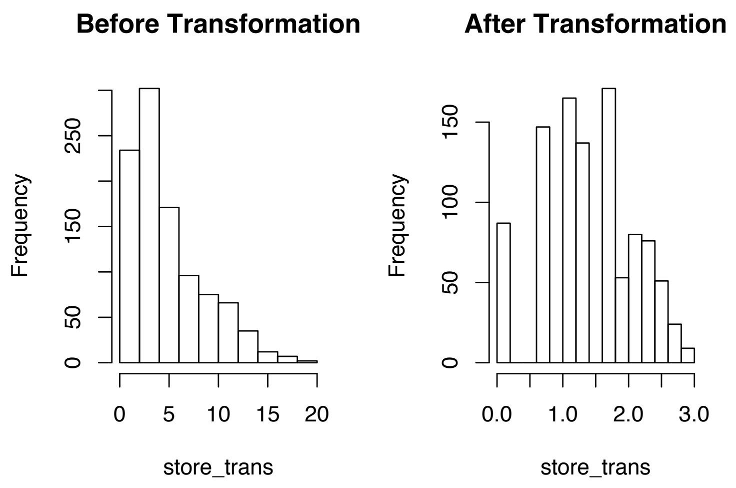


FIGURE 3.3: Box-Cox Transformation

can only be applied to a single variable, and it is not possible to transform difference columns in a data frame at the same time.

```
(trans<-BoxCoxTrans(dat_bc$store_trans))
```

```
## Box-Cox Transformation
##
## 1000 data points used to estimate Lambda
##
## Input data summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.00    4.00    5.35    7.00   20.00
##
## Largest/Smallest: 20
## Sample Skewness: 1.11
##
## Estimated Lambda: 0.1
## With fudge factor, Lambda = 0 will be used for transformations
```

```
transformed<-predict(trans,dat_bc$store_trans)
skewness(transformed)
```

```
## [1] -0.2155
```

The estimate of λ is the same as before (0.1). The skewness of the original observation is 1.1, and -0.2 after transformation. Although it is not strictly 0, it is greatly improved.

3.5 Resolve Outliers

Even under certain assumptions we can statistically define outliers, it can be hard to define in some situations. Box plot, histogram and some other basic visualizations can be used to initially check whether there are outliers. For example, we can visualize numerical non-survey variables in `sim.dat`:

```
# select numerical non-survey data
sdat<-subset(sim.dat,select=c("age","income","store_exp","online_exp","store_trans","online_tr
# use scatterplotMatrix() function from car package
par(oma=c(2,2,1,2))
scatterplotMatrix(sdat,diagonal="boxplot",smoother=FALSE)
```

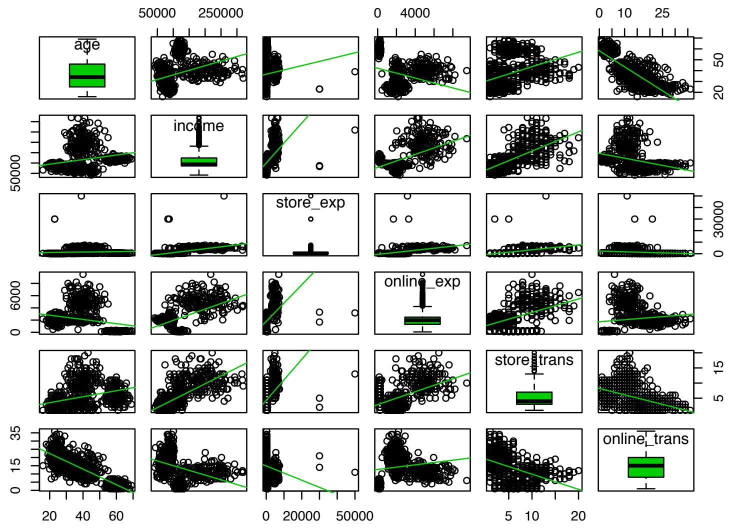


FIGURE 3.4: Use basic visualization to check outliers

As figure **??** shows, `store_exp` has outliers. It is also easy to observe the pair relationship from the plot. `age` is negatively correlated with `online_trans` but positively correlated with `store_trans`. It seems that older people tend to purchase from the local store. The amount of expense is positively correlated with income. Scatterplot matrix like this can reveal lots of information before modeling.

In addition to visualization, there are some statistical methods to define outliers, such as the commonly used Z-score. The Z-score for variable $\mathbf{Y}$ is defined as:

$$Z_i = \frac{Y_i - \bar{Y}}{s}$$

where $\bar{Y}$ and s are mean and standard deviation for Y. Z-score is a measurement of the distance between each observation and the mean. This method may be misleading, especially when the sample size is small. Iglewicz and Hoaglin proposed to use the modified Z-score to determine the outlier(**?**)

$$M_i = \frac{0.6745(Y_i - \bar{Y})}{MAD}$$

Where MAD is the median of a series of $|Y_i - \bar{Y}|$, called the median of the absolute dispersion. Iglewicz and Hoaglin suggest that the points with the Z-score greater than 3.5 corrected above are possible outliers. Let's apply it to `income`:

```r
# calculate median of the absolute dispersion for income
ymad<-mad(na.omit(sdat$income))
# calculate z-score
zs<-(sdat$income-mean(na.omit(sdat$income)))/ymad
# count the number of outliers
sum(na.omit(zs>3.5))
```

```
## [1] 59
```

According to modified Z-score, variable income has 59 outliers. Refer to (**?**) for other ways of detecting outliers.

The impact of outliers depends on the model. Some models are sensitive to outliers, such as linear regression, logistic regression. Some are pretty robust to outliers, such as tree models, support vector machine. Also, the outlier is not wrong data. It is real observation so cannot be deleted at will. If a model is sensitive to outliers, we can use *spatial sign transformation* (**?**) to minimize the problem. It projects the original sample points to the surface of a sphere by:

$$x_{ij}^{*}=\frac{x_{ij}}{\sqrt{\sum_{j=1}^{p}x_{ij}^{2}}}$$

where x_{ij} represents the i^{th} observation and j^{th} variable. As shown in the equation, every observation for sample i is divided by its square mode. The denominator is the Euclidean distance to the center of the p-dimensional predictor space. Three things to pay attention here:

1. It is important to center and scale the predictor data before using this transformation
2. Unlike centering or scaling, this manipulation of the predictors transforms them as a group
3. If there are some variables to remove (for example, highly correlated variables), do it before the transformation

Function `spatialSign()` `caret` package can conduct the transformation. Take `income` and `age` as an example:

```r
# KNN imputation
sdat<-sim.dat[,c("income","age")]
imp<-preProcess(sdat,method=c("knnImpute"),k=5)
sdat<-predict(imp,sdat)
transformed <- spatialSign(sdat)
transformed <- as.data.frame(transformed)
par(mfrow=c(1,2),oma=c(2,2,2,2))
plot(income ~ age,data = sdat,col="blue",main="Before")
plot(income ~ age,data = transformed,col="blue",main="After")
```

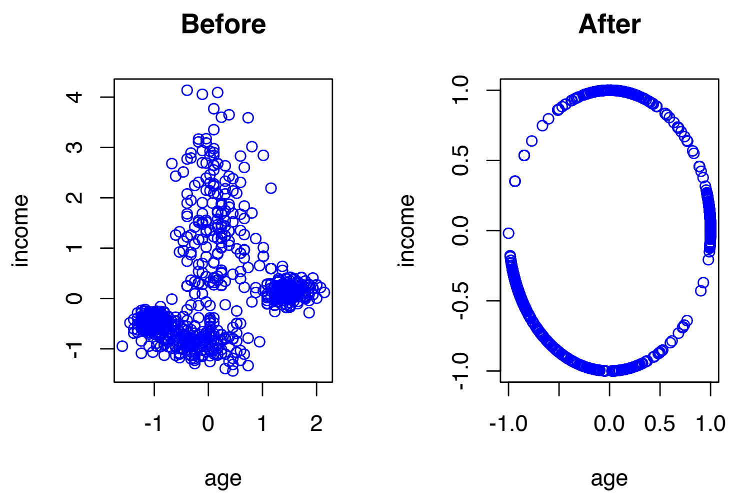


FIGURE 3.5: Spatial sign transformation

Some readers may have found that the above code does not seem to standardize the data before transformation. Recall the introduction of KNN, `preProcess()` with `method="knnImpute"` by default will standardize data.

3.6 Collinearity

It is probably the technical term known by the most un-technical people. When two predictors are very strongly correlated, including both in a model may lead to confusion or problem with a singular matrix. There is an excellent function in `corrplot` package with the same name `corrplot()` that can visualize correlation structure of a set of predictors. The function has the option to reorder the variables in a way that reveals clusters of highly correlated ones.

```
# select non-survey numerical variables
sdat<-subset(sim.dat,select=c("age","income","store_exp","online_exp","store_trans","online_tr
# use bagging imputation here
```

```r
imp<-preProcess(sdat,method="bagImpute")
sdat<-predict(imp,sdat)
# get the correlation matrix
correlation<-cor(sdat)
# plot
par(oma=c(2,2,2,2))
corrplot.mixed(correlation,order="hclust",tl.pos="lt",upper="ellipse")
```

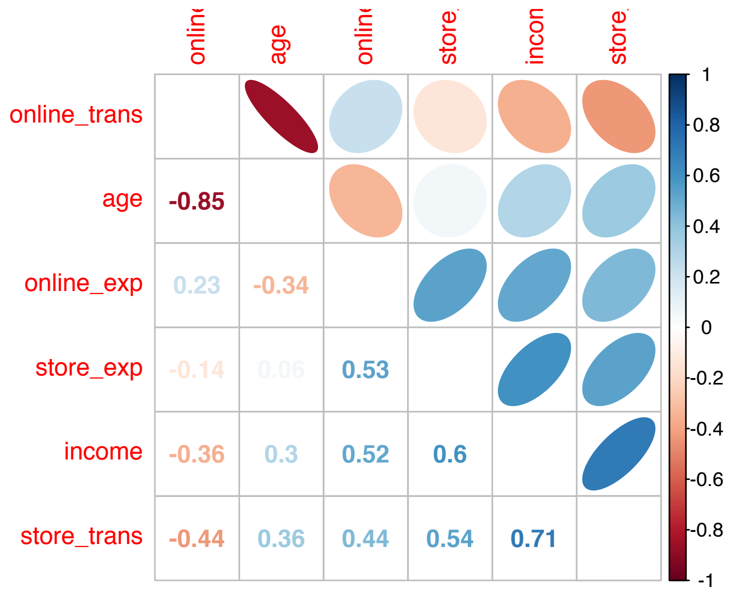


FIGURE 3.6: Correlation Matrix

Here use `corrplot.mixed()` function to visualize the correlation matrix (figure **??**). The closer the correlation is to 0, the lighter the color is and the closer the shape is to a circle. The elliptical means the correlation is not equal to 0 (because we set the `upper = "ellipse"`), the greater the correlation, the narrower the ellipse. Blue represents a positive correlation; red represents a negative correlation. The direction of the ellipse also changes with the correlation. The correlation coefficient is shown in the lower triangle of the matrix. The variables relationship from previous scatter matrix (figure @ref(fig: scm)) are clear here: the negative correlation between age and online shopping, the positive correlation between income and amount of purchasing. Some correlation is very strong (such as the correlation between `online_trans` and`age` is -0.85) which means the two variables contain duplicate information.

Section 3.5 of "Applied Predictive Modeling" (**?**) presents a heuristic algorithm to remove a minimum number of predictors to ensure all pairwise correlations are below a certain threshold:

---

(1) Calculate the correlation matrix of the predictors.
(2) Determine the two predictors associated with the largest absolute pairwise correlation (call them predictors A and B).
(3) Determine the average correlation between A and the other variables. Do the same for predictor B.
(4) If A has a larger average correlation, remove it; otherwise, remove predictor B.
(5) Repeat Step 2-4 until no absolute correlations are above the threshold.

The `findCorrelation()` function in package `caret` will apply the above algorithm.

```
(highCorr<-findCorrelation(cor(sdat),cutoff=.75))
```

```
## [1] 1
```

It returns the index of columns need to be deleted. It tells us that we need to remove the first column to make sure the correlations are all below 0.75.

```
# delete highly correlated columns
sdat<-sdat[-highCorr]
# check the new correlation matrix
cor(sdat)
```

```
##                income store_exp online_exp store_trans
## income         1.0000    0.6004     0.5199      0.7070
## store_exp     0.6004    1.0000     0.5350      0.5399
## online_exp    0.5199    0.5350     1.0000      0.4421
## store_trans   0.7070    0.5399     0.4421      1.0000
## online_trans -0.3573   -0.1367     0.2256     -0.4368
##              online_trans
## income            -0.3573
## store_exp         -0.1367
## online_exp         0.2256
## store_trans       -0.4368
## online_trans       1.0000
```

The absolute value of the elements in the correlation matrix after removal are all below 0.75. How strong does a correlation have to get, before you should start worrying about multicollinearity? There is no easy answer to that question. You can treat the threshold as a tuning parameter and pick one that gives you best prediction accuracy.

3.7 Sparse Variables

Other than the highly related predictors, predictors with degenerate distributions can cause the problem too. Removing those variables can significantly improve some models' performance and stability (such as linear regression and logistic regression but the tree based model is impervious to this type of predictors). One extreme example is a variable with a single value which is called zero-variance variable. Variables with very low frequency of unique values are near-zero variance predictors. In general, detecting those variables follows two rules:

- The fraction of unique values over the sample size
- The ratio of the frequency of the most prevalent value to the frequency of the second most prevalent value.

`nearZeroVar()` function in the `caret` package can filter near-zero variance predictors according to the above rules. In order to show the useage of the function, let's arbitaryly add some problematic variables to the origional data `sim.dat`:

```
# make a copy
zero_demo<-sim.dat
# add two sparse variable
# zero1 only has one unique value
# zero2 is a vector with the first element 1 and the rest are 0s
zero_demo$zero1<-rep(1,nrow(zero_demo))
zero_demo$zero2<-c(1,rep(0,nrow(zero_demo)-1))
```

The function will return a vector of integers indicating which columns to remove:

```
nearZeroVar(zero_demo,freqCut = 95/5, uniqueCut = 10)
```

As expected, it returns the two columns we generated. You can go ahead to remove them. Note the two arguments in the function `freqCut =` and `uniqueCut =` are corresponding to the previous two rules.

- `freqCut`: the cutoff for the ratio of the most common value to the second most common value
- `uniqueCut`: the cutoff for the percentage of distinct values out of the number of total samples

3.8 Re-encode Dummy Variables

A dummy variable is a binary variable (0/1) to represent subgroups of the sample. Sometimes we need to recode categories to smaller bits of information named "dummy variables." For example, some questionnaires have five options for each question, A, B, C, D, and E. After you get the data, you will usually convert the corresponding categorical variables for each question into five nominal variables, and then use one of the options as the baseline.

Let's encode `gender` and `house` from `sim.dat` to dummy variables.

There are two ways to implement this. The first is to use `class.ind()` from `nnet` package. However, it only works on one variable at a time.

```
dumVar<-nnet::class.ind(sim.dat$gender)
head(dumVar)
```

```
##      Female Male
## [1,]      1    0
## [2,]      1    0
## [3,]      0    1
## [4,]      0    1
## [5,]      0    1
## [6,]      0    1
```

Since it is redundant to keep both, we need to remove one of them when modeling. Another more powerful function is `dummyVars()` from `caret`:

```
dumMod<-dummyVars(~gender+house+income,
                  data=sim.dat,
                  # use "origional variable name + level" as new name
                  levelsOnly=F)
head(predict(dumMod,sim.dat))
```

```
##   gender.Female gender.Male house.No house.Yes income
## 1             1           0        0         1 120963
## 2             1           0        0         1 122008
## 3             0           1        0         1 114202
## 4             0           1        0         1 113616
## 5             0           1        0         1 124253
## 6             0           1        0         1 107661
```

`dummyVars()` can also use formula format. The variable on the right-hand side can be both categorical and numeric. For a numerical variable, the function will keep the variable unchanged. The advantage is that you can apply the function to a data frame without removing numerical variables. Other than that, the function can create interaction term:

```r
dumMod<-dummyVars(~gender+house+income+income:gender,
                  data=sim.dat,
                  levelsOnly=F)
head(predict(dumMod,sim.dat))
```

```
##   gender.Female gender.Male house.No house.Yes income
## 1             1           0        0         1 120963
## 2             1           0        0         1 122008
## 3             0           1        0         1 114202
## 4             0           1        0         1 113616
## 5             0           1        0         1 124253
## 6             0           1        0         1 107661
##   gender.Female:income gender.Male:income
## 1               120963                  0
## 2               122008                  0
## 3                    0             114202
## 4                    0             113616
## 5                    0             124253
## 6                    0             107661
```

If you think the impact income levels on purchasing behavior is different for male and female, then you may add the interaction term between `income` and `gender`. You can do this by adding `income: gender` in the formula.

3.9 Python Computing

Environmental Setup

```python
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
import scipy as sp
import pandas as pd
import math

from sklearn.preprocessing import Imputer
from sklearn.preprocessing import StandardScaler

from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
```

3.9.1 Data Cleaning

4

Data Wrangling

This chapter focuses on some of the most frequently used data manipulations and shows how to implement them in R and Python. It is critical to explore the data with descriptive statistics (mean, standard deviation, etc.) and data visualization before analysis. Transform data so that the data structure is in line with the requirements of the model. You also need to summarize the results after analysis.

4.1 Data Wrangling Using R

4.1.1 Read and write data

4.1.1.1 `readr`

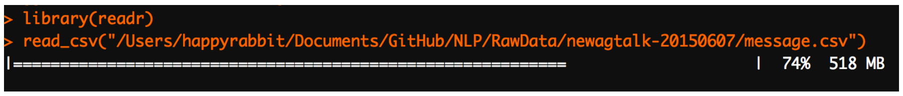
You must be familiar with `read.csv()`, `read.table()` and `write.csv()` in base R. Here we will introduce a more efficient package from RStudio in 2015 for reading and writing data: `readr` package. The corresponding functions are `read_csv()`, `read_table()` and `write_csv()`. The commands look quite similar, but `readr` is different in the following respects:

1. It is 10x faster. The trick is that `readr` uses C++ to process the data quickly.
2. It doesn't change the column names. The names can start with a number and "`.`" will not be substituted to "`_`". For example:

```r
library(readr)
read_csv("2015,2016,2017
1,2,3
4,5,6")
```

```
## # A tibble: 2 x 3
##   `2015` `2016` `2017`
##    <int>  <int>  <int>
## 1      1      2      3
## 2      4      5      6
```

3. `readr` functions do not convert strings to factors by default, are able to parse dates and times and can automatically determine the data types in each column.
4. The killing character, in my opinion, is that `readr` provides **progress bar**. What makes you feel worse than waiting is not knowing how long you have to wait.



```
> library(readr)
> read_csv("/Users/happyrabbit/Documents/GitHub/NLP/RawData/newagtalk-20150607/message.csv")
|===========================================================                  |  74%  518 MB
```

The major functions of readr is to turn flat files into data frames:

- `read_csv()`: reads comma delimited files
- `read_csv2()`: reads semicolon separated files (common in countries where `,` is used as the decimal place)
- `read_tsv()`: reads tab delimited files
- `read_delim()`: reads in files with any delimiter
- `read_fwf()`: reads fixed width files. You can specify fields either by their widths with `fwf_widths()` or their position with `fwf_positions()`

- `read_table()`: reads a common variation of fixed width files where columns are separated by white space
- `read_log()`: reads Apache style log files

The good thing is that those functions have similar syntax. Once you learn one, the others become easy. Here we will focus on `read_csv()`.

The most important information for `read_csv()` is the path to your data:

```
library(readr)
sim.dat <- read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
head(sim.dat)
```

```
## # A tibble: 6 x 19
##     age gender  income house store_exp online_exp
##   <int> <chr>    <dbl> <chr>     <dbl>      <dbl>
## 1    57 Female 120963. Yes        529.       304.
## 2    63 Female 122008. Yes        478.       110.
## 3    59 Male   114202. Yes        491.       279.
## 4    60 Male   113616. Yes        348.       142.
## 5    51 Male   124253. Yes        380.       112.
## 6    59 Male   107661. Yes        338.       196.
## # ... with 13 more variables: store_trans <int>,
## #   online_trans <int>, Q1 <int>, Q2 <int>, Q3 <int>,
## #   Q4 <int>, Q5 <int>, Q6 <int>, Q7 <int>, Q8 <int>,
## #   Q9 <int>, Q10 <int>, segment <chr>
```

The function reads the file to R as a `tibble`. You can consider `tibble` as next iteration of the data frame. They are different with data frame for the following aspects:

- It never changes an input's type (i.e., no more `stringsAsFactors = FALSE`!)
- It never adjusts the names of variables
- It has a refined print method that shows only the first 10 rows and all the columns that fit on the screen. You can also control the default print behavior by setting options.

Refer to `http://r4ds.had.co.nz/tibbles.html` for more information about 'tibble'.

When you run `read_csv()` it prints out a column specification that gives the name and type of each column. To better understanding how `readr` works, it is helpful to type in some baby data set and check the results:

```
dat=read_csv("2015,2016,2017
100,200,300
canola,soybean,corn")
print(dat)
```

```
## # A tibble: 2 x 3
##   `2015` `2016`  `2017`
##   <chr>  <chr>   <chr>
## 1 100    200     300
## 2 canola soybean corn
```

You can also add comments on the top and tell R to skip those lines:

```
dat=read_csv("# I will never let you know that
          # my favorite food is carrot
          Date,Food,Mood
          Monday,carrot,happy
          Tuesday,carrot,happy
          Wednesday,carrot,happy
          Thursday,carrot,happy
          Friday,carrot,happy
          Saturday,carrot,extremely happy
          Sunday,carrot,extremely happy", skip = 2)
print(dat)
```

```
## # A tibble: 7 x 3
##   Date      Food   Mood
##   <chr>     <chr>  <chr>
## 1 Monday    carrot happy
## 2 Tuesday   carrot happy
## 3 Wednesday carrot happy
## 4 Thursday  carrot happy
## 5 Friday    carrot happy
## 6 Saturday  carrot extremely happy
## 7 Sunday    carrot extremely happy
```

If you don't have column names, set `col_names = FALSE` then R will assign names "`X1`","`X2`"... to the columns:

```
dat=read_csv("Saturday,carrot,extremely happy
          Sunday,carrot,extremely happy", col_names=FALSE)
print(dat)
```

```
## # A tibble: 2 x 3
##   X1       X2     X3
##   <chr>    <chr>  <chr>
## 1 Saturday carrot extremely happy
## 2 Sunday   carrot extremely happy
```

You can also pass `col_names` a character vector which will be used as the column names. Try to replace `col_names=FALSE` with `col_names=c("Date","Food","Mood")` and see what happen.

As mentioned before, you can use `read_csv2()` to read semicolon separated files:

```
dat=read_csv2("Saturday; carrot; extremely happy \n Sunday; carrot; extremely happy", col_name
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

```
print(dat)
```

```
## # A tibble: 2 x 3
##   X1       X2     X3
##   <chr>    <chr>  <chr>
## 1 Saturday carrot extremely happy
## 2 Sunday   carrot extremely happy
```

Here "`\n`" is a convenient shortcut for adding a new line.

You can use `read_tsv()` to read tab delimited files

```
dat=read_tsv("every\tman\tis\ta\tpoet\twhen\the\tis\tin\tlove\n", col_names = FALSE)
print(dat)
```

```
## # A tibble: 1 x 10
##   X1    X2    X3    X4    X5    X6    X7    X8
##   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 every man   is    a     poet  when  he    is
## # ... with 2 more variables: X9 <chr>, X10 <chr>
```

Or more generally, you can use `read_delim()` and assign separating character

```r
dat=read_delim("THE|UNBEARABLE|RANDOMNESS|OF|LIFE\n", delim = "|", col_names = FALSE)
print(dat)
```

```
## # A tibble: 1 x 5
##   X1    X2         X3         X4    X5   
##   <chr> <chr>      <chr>      <chr> <chr>
## 1 THE   UNBEARABLE RANDOMNESS OF    LIFE 
```

Another situation you will often run into is the missing value. In marketing survey, people like to use "99" to represent missing. You can tell R to set all observation with value "99" as missing when you read the data:

```r
dat=read_csv("Q1,Q2,Q3
               5, 4,99",na="99")
print(dat)
```

```
## # A tibble: 1 x 3
##      Q1    Q2 Q3   
##   <int> <int> <chr>
## 1     5     4 <NA> 
```

For writing data back to disk, you can use `write_csv()` and `write_tsv()`. The following two characters of the two functions increase the chances of the output file being read back in correctly:

- Encode strings in UTF-8
- Save dates and date-times in ISO8601 format so they are easily parsed elsewhere

For example:

```r
write_csv(sim.dat, "sim_dat.csv")
```

For other data types, you can use the following packages:

- `Haven`: SPSS, Stata and SAS data
- `Readxl` and `xlsx`: excel data(.xls and .xlsx)

- `DBI`: given data base, such as RMySQL, RSQLite and RPostgreSQL, read data directly from the database using SQL

Some other useful materials:

- For getting data from the internet, you can refer to the book "XML and Web Technologies for Data Sciences with R".

- R data import/export manual[1]
- `rio` package `https://github.com/leeper/rio`

4.1.1.2 `data.table`— enhanced `data.frame`

What is `data.table`? It is an R package that provides an enhanced version of `data.frame`. The most used object in R is `data frame`. Before we move on, let's briefly review some basic characters and manipulations of data.frame:

- It is a set of rows and columns.
- Each row is of the same length and data type
- Every column is of the same length but can be of differing data types
- It has characteristics of both a matrix and a list
- It uses `[]` to subset data

We will use the clothes customer data to illustrate. There are two dimensions in `[]`. The first one indicates the row and second one indicates column. It uses a comma to separate them.

```
# read data
sim.dat<-readr::read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/
```

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   gender = col_character(),
##   income = col_double(),
##   house = col_character(),
```

[1]`https://cran.r-project.org/doc/manuals/r-release/R-data.html#Acknowledgements`

```
##   store_exp = col_double(),
##   online_exp = col_double(),
##   store_trans = col_integer(),
##   online_trans = col_integer(),
##   Q1 = col_integer(),
##   Q2 = col_integer(),
##   Q3 = col_integer(),
##   Q4 = col_integer(),
##   Q5 = col_integer(),
##   Q6 = col_integer(),
##   Q7 = col_integer(),
##   Q8 = col_integer(),
##   Q9 = col_integer(),
##   Q10 = col_integer(),
##   segment = col_character()
## )
```

```
# subset the first two rows
sim.dat[1:2,]
```

```
## # A tibble: 2 x 19
##     age gender  income house store_exp online_exp
##   <int> <chr>    <dbl> <chr>     <dbl>      <dbl>
## 1    57 Female 120963. Yes        529.       304.
## 2    63 Female 122008. Yes        478.       110.
## # ... with 13 more variables: store_trans <int>,
## #   online_trans <int>, Q1 <int>, Q2 <int>, Q3 <int>,
## #   Q4 <int>, Q5 <int>, Q6 <int>, Q7 <int>, Q8 <int>,
## #   Q9 <int>, Q10 <int>, segment <chr>
```

```
# subset the first two rows and column 3 and 5
sim.dat[1:2,c(3,5)]
```

```
## # A tibble: 2 x 2
##    income store_exp
##     <dbl>     <dbl>
## 1 120963.      529.
## 2 122008.      478.
```

```r
# get all rows with age>70
sim.dat[sim.dat$age>70,]
```

```
## # A tibble: 1 x 19
##     age gender  income house store_exp online_exp
##   <int> <chr>    <dbl> <chr>     <dbl>      <dbl>
## 1   300 Male   208017. Yes       5077.      6053.
## # ... with 13 more variables: store_trans <int>,
## #   online_trans <int>, Q1 <int>, Q2 <int>, Q3 <int>,
## #   Q4 <int>, Q5 <int>, Q6 <int>, Q7 <int>, Q8 <int>,
## #   Q9 <int>, Q10 <int>, segment <chr>
```

```r
# get rows with age> 60 and gender is Male
# select column 3 and 4
sim.dat[sim.dat$age>68 & sim.dat$gender == "Male", 3:4]
```

```
## # A tibble: 2 x 2
##    income house
##     <dbl> <chr>
## 1 119552. No
## 2 208017. Yes
```

Remember that there are usually different ways to conduct the same manipulation. For example, the following code presents three ways to calculate an average number of online transactions for male and female:

```r
tapply(sim.dat$online_trans, sim.dat$gender, mean )
```

```
## Female   Male
##  15.38  11.26
```

```r
aggregate(online_trans ~ gender, data = sim.dat, mean)
```

```
##   gender online_trans
## 1 Female        15.38
## 2   Male        11.26
```

```r
library(dplyr)
sim.dat%>%
  group_by(gender)%>%
  summarise(Avg_online_trans=mean(online_trans))
```

```
## # A tibble: 2 x 2
##   gender Avg_online_trans
##   <chr>             <dbl>
## 1 Female             15.4
## 2 Male               11.3
```

There is no gold standard to choose a specific function to manipulate data. The goal is to solve the real problem, not the tool itself. So just use whatever tool that is convenient for you.

The way to use `[]` is straightforward. But the manipulations are limited. If you need more complicated data reshaping or aggregation, there are other packages to use such as `dplyr`, `reshape2`, `tidyr` etc. But the usage of those packages are not as straightforward as `[]`. You often need to change functions. Keeping related operations together, such as subset, group, update, join etc, will allow for:

- concise, consistent and readable syntax irrespective of the set of operations you would like to perform to achieve your end goal
- performing data manipulation fluidly without the cognitive burden of having to change among different functions
- by knowing precisely the data required for each operation, you can automatically optimize operations effectively

`data.table` is the package for that. If you are not familiar with other data manipulating packages and are interested in reducing programming time tremendously, then this package is for you.

Other than extending the function of `[]`, `data.table` has the following advantages:

Offers fast import, subset, grouping, update, and joins for large data files It is easy to turn data frame to data table Can behave just like a data frame

You need to install and load the package:

```r
# If you haven't install it, use the code to instal
# install.packages("data.table")
# load packagw
library(data.table)
```

Use `data.table()` to covert the existing data frame `sim.dat` to data table:

```r
dt <- data.table(sim.dat)
class(dt)
```

```
## [1] "data.table" "data.frame"
```

Calculate mean for counts of online transactions:

```r
dt[, mean(online_trans)]
```

```
## [1] 13.55
```

You can't do the same thing using data frame:

```r
sim.dat[,mean(online_trans)]
```

```
Error in mean(online_trans) : object 'online_trans' not found
```

If you want to calculate mean by group as before, set "`by =`" argument:

```r
dt[ , mean(online_trans), by = gender]
```

```
##    gender    V1
## 1: Female 15.38
## 2:   Male 11.26
```

You can group by more than one variables. For example, group by "`gender`" and "`house`":

```r
dt[ , mean(online_trans), by = .(gender, house)]
```

```
##    gender house     V1
## 1: Female   Yes 11.312
## 2:   Male   Yes  8.772
## 3: Female    No 19.146
## 4:   Male    No 16.486
```

Assign column names for aggregated variables:

```
dt[ , .(avg = mean(online_trans)), by = .(gender, house)]
```

```
##    gender house    avg
## 1: Female   Yes 11.312
## 2:   Male   Yes  8.772
## 3: Female    No 19.146
## 4:   Male    No 16.486
```

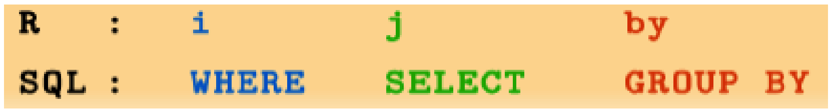
`data.table` can accomplish all operations that `aggregate()` and `tapply()`can do for data frame.

- General setting of `data.table`

Different from data frame, there are three arguments for data table:

DT[i, j, by]

It is analogous to SQL. You don't have to know SQL to learn data table. But experience with SQL will help you understand data table. In SQL, you select column `j` (use command `SELECT`) for row `i` (using command `WHERE`). `GROUP BY` in SQL will assign the variable to group the observations.



```
R   :  i      j       by
SQL :  WHERE  SELECT  GROUP BY
```

Let's review our previous code:

```
dt[ , mean(online_trans), by = gender]
```

```
##    gender    V1
## 1: Female 15.38
## 2:   Male 11.26
```

The code above is equal to the following SQL

```
SELECT  gender, avg(online_trans) FROM sim.dat GROUP BY gender
```

R code:

```
dt[ , .(avg = mean(online_trans)), by = .(gender, house)]
```

```
##    gender house    avg
## 1: Female   Yes 11.312
## 2:   Male   Yes  8.772
## 3: Female    No 19.146
## 4:   Male    No 16.486
```

is equal to SQL

```
SELECT gender, house, avg(online_trans) AS avg FROM sim.dat GROUP BY gender, house
```

R code

```
dt[ age < 40, .(avg = mean(online_trans)), by = .(gender, house)]
```

```
##    gender house   avg
## 1:   Male   Yes 14.46
## 2: Female   Yes 18.14
## 3:   Male    No 18.24
## 4: Female    No 20.10
```

is equal to SQL

```
SELECT gender, house, avg(online_trans) AS avg FROM sim.dat WHERE age < 40 GROUP BY gender, ho
```

You can see the analogy between `data.table` and `SQL`. Now let's focus on operations in data table.

- select row

```
# select rows with age<20 and income > 80000
dt[age < 20 & income > 80000]
```

```
##    age gender income house store_exp online_exp
## 1:  19 Female  83535    No     227.7       1491
## 2:  18 Female  89416   Yes     209.5       1926
## 3:  19 Female  92813    No     186.7       1042
##    store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
## 1:           1           22  2  1  1  2  4  1  4  2  4
## 2:           3           28  2  1  1  1  4  1  4  2  4
## 3:           2           18  3  1  1  2  4  1  4  3  4
##    Q10 segment
## 1:   1   Style
## 2:   1   Style
## 3:   1   Style
```

```
# select the first two rows
dt[1:2]
```

```
##    age gender income house store_exp online_exp
## 1:  57 Female 120963   Yes     529.1      303.5
## 2:  63 Female 122008   Yes     478.0      109.5
##    store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
## 1:           2            2  4  2  1  2  1  4  1  4  2
## 2:           4            2  4  1  1  2  1  4  1  4  1
##    Q10 segment
## 1:   4   Price
## 2:   4   Price
```

- select column

Selecting columns in `data.table` don't need `$`:

```
# select column “age” but return it as a vector
# the argument for row is empty so the result will return all observations
ans <- dt[, age]
head(ans)
```

```
## [1] 57 63 59 60 51 59
```

To return `data.table` object, put column names in `list()`:

```
# Select age and online_exp columns and return as a data.table instead
ans <- dt[, list(age, online_exp)]
head(ans)
```

```
##    age online_exp
## 1:  57      303.5
## 2:  63      109.5
## 3:  59      279.2
## 4:  60      141.7
## 5:  51      112.2
## 6:  59      195.7
```

Or you can also put column names in `.()`:

```
ans <- dt[, .(age, online_exp)]
# head(ans)
```

To select all columns from "`age`" to "`income`":

```
ans <- dt[, age:income, with = FALSE]
head(ans,2)
```

```
##    age gender income
## 1:  57 Female 120963
## 2:  63 Female 122008
```

Delete columns using `-` or `!`:

```
# delete columns from  age to online_exp
ans <- dt[, -(age:online_exp), with = FALSE]
ans <- dt[, !(age:online_exp), with = FALSE]
```

- tabulation

In data table. `.N` means to count

```
# row count
dt[, .N]
```

```
## [1] 1000
```

If you assign the group variable, then it will count by groups:

```
# counts by gender
dt[, .N, by= gender]
```

```
##    gender   N
## 1: Female 554
## 2:   Male 446
```

```
# for those younger than 30, count by gender
 dt[age < 30, .(count=.N), by= gender]
```

```
##    gender count
## 1: Female   292
## 2:   Male    86
```

Order table:

```
# get records with the highest 5 online expense:
head(dt[order(-online_exp)],5)
```

```
##    age gender income house store_exp online_exp
## 1:  40 Female 217600    No      7024       9479
## 2:  41 Female     NA   Yes      3787       8638
## 3:  36   Male 228550   Yes      3280       8221
## 4:  31 Female 159508   Yes      5177       8006
## 5:  43 Female 190407   Yes      4695       7876
##    store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
## 1:          10            6  1  4  5  4  3  4  4  1  4
## 2:          14           10  1  4  4  4  4  4  4  1  4
## 3:           8           12  1  4  5  4  4  4  4  1  4
## 4:          11           13  1  4  4  4  4  4  4  1  4
## 5:           6           11  1  4  5  4  4  4  4  1  4
##    Q10     segment
## 1:   2 Conspicuous
## 2:   2 Conspicuous
## 3:   1 Conspicuous
## 4:   2 Conspicuous
## 5:   2 Conspicuous
```

Since data table keep some characters of data frame, they share some operations:

```
dt[order(-online_exp)][1:5]
```

```
##    age gender income house store_exp online_exp
## 1:  40 Female 217600    No      7024       9479
## 2:  41 Female     NA   Yes      3787       8638
## 3:  36   Male 228550   Yes      3280       8221
## 4:  31 Female 159508   Yes      5177       8006
## 5:  43 Female 190407   Yes      4695       7876
##    store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
## 1:          10            6  1  4  5  4  3  4  4  1  4
## 2:          14           10  1  4  4  4  4  4  4  1  4
## 3:           8           12  1  4  5  4  4  4  4  1  4
## 4:          11           13  1  4  4  4  4  4  4  1  4
## 5:           6           11  1  4  5  4  4  4  4  1  4
##    Q10     segment
## 1:   2 Conspicuous
## 2:   2 Conspicuous
## 3:   1 Conspicuous
## 4:   2 Conspicuous
## 5:   2 Conspicuous
```

You can also order the table by more than one variable. The following code will order the table by `gender`, then order within `gender` by `online_exp`:

```
dt[order(gender, -online_exp)][1:5]
```

```
##    age gender income house store_exp online_exp
## 1:  40 Female 217600    No      7024       9479
## 2:  41 Female     NA   Yes      3787       8638
## 3:  31 Female 159508   Yes      5177       8006
## 4:  43 Female 190407   Yes      4695       7876
## 5:  50 Female 263858   Yes      5814       7449
##    store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9
## 1:          10            6  1  4  5  4  3  4  4  1  4
## 2:          14           10  1  4  4  4  4  4  4  1  4
## 3:          11           13  1  4  4  4  4  4  4  1  4
## 4:           6           11  1  4  5  4  4  4  4  1  4
## 5:          11           11  1  4  5  4  4  4  4  1  4
##    Q10     segment
## 1:   2 Conspicuous
## 2:   2 Conspicuous
## 3:   2 Conspicuous
## 4:   2 Conspicuous
## 5:   1 Conspicuous
```

- Use `fread()` to import dat

Other than `read.csv` in base R, we have introduced 'read_csv' in 'readr'. `read_csv` is much faster and will provide progress bar which makes user feel much better (at least make me feel better). `fread()` in `data.table` further increase the efficiency of reading data. The following are three examples of reading the same data file `topic.csv`. The file includes text data scraped from an agriculture forum with 209670 rows and 6 columns:

```
system.time(topic<-read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/mast
```

```
  user  system elapsed
 4.313   0.027   4.340
```

```
system.time(topic<-readr::read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientis
```

```
   user  system elapsed
  0.267   0.008   0.274
```

```
system.time(topic<-data.table::fread("https://raw.githubusercontent.com/happyrabbit/DataScient
```

```
   user  system elapsed
  0.217   0.005   0.221
```

It is clear that `read_csv()` is much faster than `read.csv()`. `fread()` is a little faster than `read_csv()`. As the size increasing, the difference will become for significant. Note that `fread()` will read file as `data.table` by default.

4.1.2 Summarize data

4.1.2.1 `apply()`, `lapply()` and `sapply()` in base R

There are some powerful functions to summarize data in base R, such as `apply()`, `lapply()` and `sapply()`. They do the same basic things and are all from "apply" family: apply functions over parts of data. They differ in two important respects:

1. the type of object they apply to
2. the type of result they will return

When do we use `apply()`? When we want to apply a function to margins of an array or matrix. That means our data need to be structured. The operations can be very flexible. It returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

For example you can compute row and column sums for a matrix:

```
## simulate a matrix
x <- cbind(x1 =1:8, x2 = c(4:1, 2:5))
dimnames(x)[[1]] <- letters[1:8]
apply(x, 2, mean)
```

```
##  x1  x2
## 4.5 3.0
```

```
col.sums <- apply(x, 2, sum)
row.sums <- apply(x, 1, sum)
```

You can also apply other functions:

```
ma <- matrix(c(1:4, 1, 6:8), nrow = 2)
ma
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    1    7
## [2,]    2    4    6    8
```

```
apply(ma, 1, table)  #--> a list of length 2
```

```
## [[1]]
##
## 1 3 7
## 2 1 1
##
## [[2]]
##
## 2 4 6 8
## 1 1 1 1
```

```
apply(ma, 1, stats::quantile) # 5 x n matrix with rownames
```

```
##      [,1] [,2]
## 0%      1  2.0
## 25%     1  3.5
## 50%     2  5.0
## 75%     4  6.5
## 100%    7  8.0
```

Results can have different lengths for each call. This is a trickier example. What will you get?

```
## Example with different lengths for each call
z <- array(1:24, dim = 2:4)
zseq <- apply(z, 1:2, function(x) seq_len(max(x)))
zseq         ## a 2 x 3 matrix
typeof(zseq) ## list
dim(zseq) ## 2 3
zseq[1,]
apply(z, 3, function(x) seq_len(max(x)))
```

- `lapply()` applies a function over a list, data.frame or vector and returns a list of the same length.
- `sapply()` is a user-friendly version and wrapper of `lapply()`. By default it returns a vector, matrix or if `simplify = "array"`, an array if appropriate. `apply(x, f, simplify = FALSE, USE.NAMES = FALSE)` is the same as `lapply(x, f)`. If `simplify=TRUE`, then it will return a `data.frame` instead of `list`.

Let's use some data with context to help you better understand the functions.

- Get the mean and standard deviation of all numerical variables in the dataset.

```
# Read data
sim.dat<-read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/Se
# Get numerical variables
sdat<-sim.dat[,!lapply(sim.dat,class)=="factor"]
## Try the following code with apply() function
## apply(sim.dat,2,class)
## What is the problem?
```

The data frame `sdat` only includes numeric columns. Now we can go head and use `apply()` to get mean and standard deviation for each column:

```r
apply(sdat, MARGIN=2,function(x) mean(na.omit(x)))
```

```
##         age      income   store_exp  online_exp 
##   3.884e+01   1.135e+05   1.357e+03   2.120e+03 
##  store_trans online_trans          Q1          Q2 
##   5.350e+00   1.355e+01   3.101e+00   1.823e+00 
##          Q3          Q4          Q5          Q6 
##   1.992e+00   2.763e+00   2.945e+00   2.448e+00 
##          Q7          Q8          Q9         Q10 
##   3.434e+00   2.396e+00   3.085e+00   2.320e+00
```

Here we defined a function using `function(x) mean(na.omit(x))`. It is a very simple function. It tells R to ignore the missing value when calculating the mean. `MARGIN=2` tells R to apply the function to each column. It is not hard to guess what `MARGIN=1` mean. The result show that the average online expense is much higher than store expense. You can also compare the average scores across different questions. The command to calculate standard deviation is very similar. The only difference is to change `mean()` to `sd()`:

```r
apply(sdat, MARGIN=2,function(x) sd(na.omit(x)))
```

```
##         age      income   store_exp  online_exp 
##      16.417   49842.287    2774.400    1731.224 
##  store_trans online_trans          Q1          Q2 
##       3.696       7.957       1.450       1.168 
##          Q3          Q4          Q5          Q6 
##       1.402       1.155       1.284       1.439 
##          Q7          Q8          Q9         Q10 
##       1.456       1.154       1.118       1.136
```

Even the average online expense is higher than store expense, the standard deviation for store expense is much higher than online expense which indicates there is very likely some big/small purchase in store. We can check it quickly:

```r
summary(sdat$store_exp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    -500     205     329    1357     597   50000
```

```r
summary(sdat$online_exp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      69     420    1942    2120    2441    9479
```

There are some odd values in store expense. The minimum value is -500 which is a wrong imputation which indicates that you should preprocess data before analyzing it. Checking those simple statistics will help you better understand your data. It then gives you some idea how to preprocess and analyze them. How about using `lapply()` and `sapply()`?

Run the following code and compare the results:

```r
lapply(sdat, function(x) sd(na.omit(x)))
sapply(sdat, function(x) sd(na.omit(x)))
sapply(sdat, function(x) sd(na.omit(x)), simplify = FALSE)
```

4.1.3 `dplyr` package

`dplyr` provides a flexible grammar of data manipulation focusing on tools for working with data frames (hence the `d` in the name). It is faster and more friendly:

- It identifies the most important data manipulations and make they easy to use from R
- It performs faster for in-memory data by writing key pieces in C++ using `Rcpp`
- The interface is the same for data frame, data table or database.

We will illustrate the following functions in order:

1. Display

2. Subset
3. Summarize
4. Create new variable
5. Merge

Display

- `tbl_df()`: Convert the data to `tibble` which offers better checking and printing capabilities than traditional data frames. It will adjust output width according to fit the current window.

```r
library(dplyr)
tbl_df(sim.dat)
```

- `glimpse()`: This is like a transposed version of `tbl_df()`

```r
glimpse(sim.dat)
```

Subset

Get rows with `income` more than 300000:

```r
library(magrittr)
filter(sim.dat, income >300000) %>%
  tbl_df()
```

```
## # A tibble: 4 x 19
##     age gender  income house store_exp online_exp
##   <int> <fct>    <dbl> <fct>     <dbl>      <dbl>
## 1    40 Male   301398. Yes       4840.      3618.
## 2    33 Male   319704. Yes       5998.      4396.
## 3    41 Male   317476. Yes       3030.      4180.
## 4    37 Female 315697. Yes       6549.      4284.
## # ... with 13 more variables: store_trans <int>,
## #   online_trans <int>, Q1 <int>, Q2 <int>, Q3 <int>,
## #   Q4 <int>, Q5 <int>, Q6 <int>, Q7 <int>, Q8 <int>,
## #   Q9 <int>, Q10 <int>, segment <fct>
```

Here we meet a new operator `%>%`. It is called "Pipe operator" which pipes a value forward into an expression or function call. What you get in the left operation will be the first argument or the only argument in the right operation.

```
x %>% f(y) = f(x, y)
y %>% f(x, ., z) = f(x, y, z )
```

It is an operator from `magrittr` which can be really beneficial. Look at the following code. Can you tell me what it does?

```
ave_exp <- filter(
  summarise(
    group_by(
      filter(
        sim.dat,
        !is.na(income)
      ),
      segment
    ),
    ave_online_exp = mean(online_exp),
    n = n()
  ),
  n > 200
)
```

Now look at the identical code using "`%>%`":

```
ave_exp <- sim.dat %>%
 filter(!is.na(income)) %>%
 group_by(segment) %>%
 summarise(
   ave_online_exp = mean(online_exp),
   n = n() ) %>%
  filter(n > 200)
```

Isn't it much more straightforward now? Let's read it:

1. Delete observations from `sim.dat` with missing income values
2. Group the data from step 1 by variable `segment`
3. Calculate mean of online expense for each segment and save the result as a new variable named `ave_online_exp`

4. Calculate the size of each segment and saved it as a new variable named `n`
5. Get segments with size larger than 200

You can use `distinct()` to delete duplicated rows.

```r
dplyr::distinct(sim.dat)
```

`sample_frac()` will randomly select some rows with a specified percentage. `sample_n()` can randomly select rows with a specified number.

```r
dplyr::sample_frac(sim.dat, 0.5, replace = TRUE)
dplyr::sample_n(sim.dat, 10, replace = TRUE)
```

`slice()` will select rows by position:

```r
dplyr::slice(sim.dat, 10:15)
```

It is equivalent to `sim.dat[10:15,]`.

`top_n()` will select the order top n entries:

```r
dplyr::top_n(sim.dat,2,income)
```

If you want to select columns instead of rows, you can use `select()`. The following are some sample codes:

```r
# select by column name
dplyr::select(sim.dat,income,age,store_exp)

# select columns whose name contains a character string
dplyr::select(sim.dat, contains("_"))

# select columns whose name ends with a character string
# similar there is "starts_with"
dplyr::select(sim.dat, ends_with("e"))

# select columns Q1,Q2,Q3,Q4 and Q5
select(sim.dat, num_range("Q", 1:5))

# select columns whose names are in a group of names
```

```r
dplyr::select(sim.dat, one_of(c("age", "income")))

# select columns between age and online_exp
dplyr::select(sim.dat, age:online_exp)

# select all columns except for age
dplyr::select(sim.dat, -age)
```

Summarize

A standard marketing problem is customer segmentation. It usually starts with designing survey and collecting data. Then run a cluster analysis on the data to get customer segments. Once we have different segments, the next is to understand how each group of customer look like by summarizing some key metrics. For example, we can do the following data aggregation for different segments of clothes customers.

```r
sim.dat%>%
  group_by(segment)%>%
  summarise(Age=round(mean(na.omit(age)),0),
      FemalePct=round(mean(gender=="Female"),2),
      HouseYes=round(mean(house=="Yes"),2),
      store_exp=round(mean(na.omit(store_exp),trim=0.1),0),
      online_exp=round(mean(online_exp),0),
      store_trans=round(mean(store_trans),1),
      online_trans=round(mean(online_trans),1))
```

```
## # A tibble: 4 x 8
##   segment       Age FemalePct HouseYes store_exp
##   <fct>       <dbl>     <dbl>    <dbl>     <dbl>
## 1 Conspicuous   42.     0.320    0.860     4990.
## 2 Price         60.     0.450    0.940      501.
## 3 Quality       35.     0.470    0.340      301.
## 4 Style         24.     0.810    0.270      200.
## # ... with 3 more variables: online_exp <dbl>,
## #   store_trans <dbl>, online_trans <dbl>
```

Now, let's peel the onion in order.

The first line `sim.dat` is easy. It is the data you want to work on. The second line `group_by(segment)` tells R that in the following steps you want to summarise by variable `segment`. Here we only summarize data by one categorical variable, but you can group by multiple variables, such as `group_by(segment, house)`. The third argument `summarise` tells R the manipulation(s) to do. Then list the exact actions inside `summarise()` . For example, `Age=round(mean(na.omit(age)),0)` tell R the following things:

1. Calculate the mean of column `age` ignoring missing value for each customer segment
2. Round the result to the specified number of decimal places
3. Store the result in a new variable named `Age`

The rest of the command above is similar. In the end, we calculate the following for each segment:

1. `Age`: average age for each segment
2. `FemalePct`: percentage for each segment
3. `HouseYes`: percentage of people who own a house
4. `stroe_exp`: average expense in store
5. `online_exp`: average expense online
6. `store_trans`: average times of transactions in the store
7. `online_trans`: average times of online transactions

There is a lot of information you can extract from those simple averages.

- Conspicuous: average age is about 40. It is a group of middle-age wealthy people. 1/3 of them are female, and 2/3 are male. They buy regardless the price. Almost all of them own house (0.86). It makes us wonder what is wrong with the rest 14%?

- Price: They are older people with average age 60. Nearly all of them own a house(0.94). They are less likely to purchase online (store_trans=6 while online_trans=3). It is the only group that is less likely to buy online.

- Quality: The average age is 35. They are not way different with

Conspicuous regarding age. But they spend much less. The percentages of male and female are similar. They prefer online shopping. More than half of them don't own a house (0.66).

- Style: They are young people with average age 24. The majority of them are female (0.81). Most of them don't own a house (0.73). They are very likely to be digital natives and prefer online shopping.

You may notice that Style group purchase more frequently online (`online_trans`) but the expense (`online_exp`) is not higher. It makes us wonder what is the average expense each time, so you have a better idea about the price range of the group.

The analytical process is aggregated instead of independent steps. The current step will shed new light on what to do next. Sometimes you need to go back to fix something in the previous steps. Let's check average one-time online and instore purchase amounts:

```
sim.dat%>%
group_by(segment)%>%
summarise(avg_online=round(sum(online_exp)/sum(online_trans),2),
    avg_store=round(sum(store_exp)/sum(store_trans),2))
```

```
## # A tibble: 4 x 3
##   segment     avg_online avg_store
##   <fct>            <dbl>     <dbl>
## 1 Conspicuous      442.      479.
## 2 Price             69.3      81.3
## 3 Quality          126.      105.
## 4 Style             92.8     121.
```

Price group has the lowest averaged one-time purchase. The Conspicuous group will pay the highest price. When we build customer profile in real life, we will also need to look at the survey summarization. You may be surprised how much information simple data manipulations can provide.

Another comman task is to check which column has missing values. It requires the program to look at each column in the data. In this case you can use `summarise_all`:

```
# apply function anyNA() to each column
# you can also assign a function vector such as: c("anyNA","is.factor")
dplyr::summarise_all(sim.dat, funs_(c("anyNA")))
```

```
##     age gender income house store_exp online_exp
## 1 FALSE  FALSE   TRUE FALSE     FALSE      FALSE
##   store_trans online_trans    Q1    Q2    Q3    Q4
## 1       FALSE        FALSE FALSE FALSE FALSE FALSE
##      Q5    Q6    Q7    Q8    Q9   Q10 segment
## 1 FALSE FALSE FALSE FALSE FALSE FALSE   FALSE
```

The above code returns a vector indicating if there is any value missing in each column.

Create new variable

There are often situations where you need to create new variables. For example, adding online and store expense to get total expense. In this case, you will apply **window function** to the columns and return a column with the same length. `mutate()` can do it for you and append one or more new columns:

```
dplyr::mutate(sim.dat, total_exp = store_exp + online_exp)
```

```
##      age gender income house store_exp online_exp
## 1     57 Female 120963   Yes     529.1     303.51
## 2     63 Female 122008   Yes     478.0     109.53
## 3     59   Male 114202   Yes     490.8     279.25
## 4     60   Male 113616   Yes     347.8     141.67
## 5     51   Male 124253   Yes     379.6     112.24
## 6     59   Male 107661   Yes     338.3     195.69
## 7     57   Male 120483   Yes     482.5     284.54
## 8     57   Male 110542   Yes     340.7     135.26
## 9     61 Female 132061   Yes     608.2     142.55
## 10    60   Male 105049   Yes     470.3     163.47
## 11    58   Male 107197   Yes     366.6     170.13
## 12    59   Male     NA   Yes     674.9     310.27
## 13    64   Male 119020   Yes     613.9     160.85
## 14    57 Female     NA   Yes     737.0     224.53
## 15    64   Male 114539   Yes     402.5     241.83
## 16    61 Female     NA   Yes     615.1     238.10
## 17    57 Female 133078   Yes     429.4     262.66
## 18    63   Male 115709   Yes     552.6     187.52
## 19    57 Female 113211    No     540.3     254.58
## 20    57 Female 129774    No     384.3     311.03
## 21    58   Male     NA   Yes     372.4     296.83
## 22    58   Male 124357   Yes     535.5     205.54
## 23    59 Female 123117   Yes     481.3     157.29
## 24    61 Female     NA   Yes     546.2     161.23
## 25    60 Female     NA   Yes     411.1     123.78
## 26    56   Male     NA   Yes     492.7     128.02
## 27    58   Male 127887   Yes     519.3     142.63
## 28    64   Male 115925   Yes     627.7     240.98
## 29    63 Female     NA   Yes     511.1     203.49
## 30    56 Female 112621   Yes     699.3     223.44
## 31    57   Male     NA   Yes     530.6     217.92
## 32    59 Female 121773   Yes     532.1     176.88
## 33    63   Male 126903   Yes     601.6     257.84
## 34    55   Male 128254   Yes     595.3     248.06
## 35    60 Female     NA   Yes     403.6     236.96
## 36    64 Female     NA   Yes     611.7     102.24
## 37    57 Female 118170   Yes     482.7     183.36
## 38    57 Female 119148   Yes     412.9     246.87
## 39    58   Male 125844   Yes     473.8     261.67
## 40    55   Male 128194   Yes     595.7     156.93
## 41    59 Female     NA   Yes     548.3     186.50
## 42    61   Male     NA   Yes     597.2     209.68
## 43    59 Female 122338   Yes     455.3     205.16
## 44    58 Female 114519   Yes     493.5     271.67
## 45    62   Male 123459   Yes     561.7     209.23
## 46    59   Male 125626   Yes     518.2     261.31
## 47    56   Male 100583   Yes     633.0     195.95
## 48    66   Male     NA   Yes     510.3     198.46
## 49    62 Female 128606   Yes     499.7     244.05
## 50    54   Male 114337   Yes     518.0     217.25
##      store_trans online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
## 1              2            2  4  2  1  2  1  4  1  4
## 2              4            2  4  1  1  2  1  4  1  4
## 3              7            2  5  2  1  2  1  4  1  4
## 4             10            2  5  2  1  3  1  4  1  4
## 5              4            4  4  1  1  3  1  4  1  4
## 6              4            5  4  2  1  2  1  4  1  4
## 7              5            3  4  1  1  2  1  4  1  4
## 8             11            5  5  2  1  3  1  4  1  4
## 9              6            1  4  1  1  2  1  4  1  4
## 10            12            1  4  2  1  3  1  4  1  4
## 11             5            4  4  1  1  3  1  4  1  4
## 12             6            2  4  1  1  3  1  4  1  4
## 13             7            4  5  1  1  3  1  4  1  4
## 14             7            3  4  2  1  3  1  4  1  4
## 15             5            5  4  2  1  2  1  4  1  4
## 16             5            1  5  2  1  2  1  4  1  4
## 17             5            3  5  2  1  2  1  4  1  4
## 18             5            2  5  2  1  3  1  4  1  4
## 19             7            2  4  2  1  3  1  4  1  4
## 20             4            2  5  2  1  2  1  4  1  5
## 21            11            2  5  1  1  3  1  4  1  4
## 22             7            3  4  2  1  2  1  4  1  4
## 23             6            1  5  2  1  2  1  4  1  4
## 24             8            3  5  1  1  2  1  4  1  4
## 25            11            3  4  1  1  3  1  4  1  4
## 26             7            1  5  1  1  3  1  4  1  4
## 27             6            2  4  1  1  3  1  4  1  4
## 28             4            4  5  1  1  2  1  4  1  4
## 29             4            3  5  1  1  2  1  4  1  4
## 30             9            2  5  2  1  2  1  4  1  4
## 31             4            2  4  2  1  3  1  4  1  4
## 32             2            3  5  2  1  2  1  4  1  4
## 33             6            4  4  1  1  3  1  4  1  4
## 34             8            3  4  1  1  2  1  4  1  4
## 35             9            4  4  2  1  3  1  4  1  4
## 36             6            4  5  2  1  2  1  4  1  4
## 37             6            2  4  2  1  3  1  4  1  4
## 38             6            5  5  1  1  3  1  4  1  4
## 39             5            4  4  2  1  2  1  4  1  4
## 40             6            2  4  1  1  2  1  4  1  4
## 41             2            3  4  1  1  3  1  4  1  4
## 42             5            1  5  1  1  3  1  4  1  4
## 43             9            4  5  1  1  2  1  4  1  3
## 44             4            5  4  1  1  3  1  4  1  4
## 45             9            1  4  1  1  3  1  4  1  4
## 46             8            2  5  1  1  3  1  4  1  4
## 47            11            3  5  1  1  3  1  4  1  4
## 48             9            4  5  2  1  3  1  4  1  4
## 49             5            2  4  1  1  2  1  4  1  4
## 50             6            2  4  1  1  2  1  4  1  4
##      Q9 Q10     segment total_exp
## 1     2   4       Price     832.6
## 2     1   4       Price     587.5
## 3     1   4       Price     770.1
## 4     2   4       Price     489.5
## 5     2   4       Price     491.9
## 6     1   4       Price     534.0
## 7     1   4       Price     767.1
## 8     2   4       Price     476.0
## 9     1   4       Price     750.8
## 10    1   4       Price     633.8
## 11    2   4       Price     536.8
## 12    1   4       Price     985.1
## 13    2   4       Price     774.8
## 14    1   4       Price     961.6
## 15    1   4       Price     644.3
## 16    1   4       Price     853.2
## 17    1   4       Price     692.1
## 18    2   4       Price     740.1
## 19    2   4       Price     794.9
## 20    2   4       Price     695.4
## 21    1   4       Price     669.2
## 22    2   4       Price     741.0
## 23    1   4       Price     638.5
## 24    2   4       Price     707.4
## 25    1   4       Price     534.9
## 26    2   4       Price     620.7
## 27    2   4       Price     661.9
## 28    2   4       Price     868.7
## 29    1   4       Price     714.5
## 30    2   4       Price     922.8
## 31    1   4       Price     748.5
## 32    1   4       Price     709.0
## 33    2   4       Price     859.5
## 34    2   4       Price     843.3
## 35    1   4       Price     640.6
## 36    1   4       Price     713.9
## 37    1   4       Price     666.0
## 38    1   4       Price     659.8
## 39    2   4       Price     735.4
## 40    2   4       Price     752.6
## 41    1   4       Price     734.8
## 42    2   4       Price     806.9
## 43    2   4       Price     660.5
## 44    1   4       Price     765.2
## 45    1   4       Price     771.0
## 46    1   4       Price     779.5
## 47    2   4       Price     829.0
## 48    1   4       Price     708.7
## 49    2   4       Price     743.7
## 50    2   4       Price     735.3
##  [ reached getOption("max.print") -- omitted 950 rows ]
```

The above code sums up two columns and appends the result (`total_exp`) to `sim.dat`. Another similar function is `transmute()`. The difference is that `transmute()` will delete the original columns and only keep the new ones.

```
dplyr::transmute(sim.dat, total_exp = store_exp + online_exp)
```

Merge

Similar to SQL, there are different joins in `dplyr`. We create two baby data sets to show how the functions work.

```
(x<-data.frame(cbind(ID=c("A","B","C"),x1=c(1,2,3))))
```

```
##   ID x1
## 1  A  1
## 2  B  2
## 3  C  3
```

```
(y<-data.frame(cbind(ID=c("B","C","D"),y1=c(T,T,F))))
```

```
##   ID    y1
## 1  B  TRUE
## 2  C  TRUE
## 3  D FALSE
```

```
# join to the left
# keep all rows in x
left_join(x,y,by="ID")
```

```
##   ID x1   y1
## 1  A  1 <NA>
## 2  B  2 TRUE
## 3  C  3 TRUE
```

```
# get rows matched in both data sets
inner_join(x,y,by="ID")
```

```
##   ID x1   y1
## 1  B  2 TRUE
## 2  C  3 TRUE
```

```
# get rows in either data set
full_join(x,y,by="ID")
```

```
##   ID   x1    y1
## 1  A    1  <NA>
## 2  B    2  TRUE
## 3  C    3  TRUE
## 4  D <NA> FALSE
```

```
# filter out rows in x that can be matched in y
```

```r
# it doesn't bring in any values from y
semi_join(x,y,by="ID")
```

```
##   ID x1
## 1  B  2
## 2  C  3
```

```r
# the opposite of  semi_join()
# it gets rows in x that cannot be matched in y
# it doesn't bring in any values from y
anti_join(x,y,by="ID")
```

```
##   ID x1
## 1  A  1
```

There are other functions(`intersect()`, `union()` and `setdiff()`). Also the data frame version of `rbind` and `cbind` which are `bind_rows()` and `bind_col()`. We are not going to go through them all. You can try them yourself. If you understand the functions we introduced so far. It should be easy for you to figure out the rest.

4.2 Tidy and Reshape Data

"Tidy data" represent the information from a dataset as data frames where each row is an observation, and each column contains the values of a variable (i.e., an attribute of what we are observing). Depending on the situation, the requirements on what to present as rows and columns may change. To make data easy to work with for the problem at hand, in practice, we often need to convert data between the "wide" and the "long" format. The process feels like kneading the dough.

There are two commonly used packages for this kind of manipulations: `tidyr` and `reshape2`. We will show how to tidy and reshape data using the two packages. By comparing the functions to show how they overlap and where they differ.

4.2.1 `reshape2` package

It is a reboot of the previous package `reshape`. Take a baby subset of our exemplary clothes consumers data to illustrate:

```
(sdat<-sim.dat[1:5,1:6])
```

```
##   age gender income house store_exp online_exp
## 1  57 Female 120963   Yes     529.1      303.5
## 2  63 Female 122008   Yes     478.0      109.5
## 3  59   Male 114202   Yes     490.8      279.2
## 4  60   Male 113616   Yes     347.8      141.7
## 5  51   Male 124253   Yes     379.6      112.2
```

For the above data `sdat`, what if we want to have a variable indicating the purchasing channel (i.e. online or in-store) and another column with the corresponding expense amount? Assume we want to keep the rest of the columns the same. It is a task to change data from "wide" to "long". There are two general ways to shape data:

- Use `melt()` to convert an object into a molten data frame, i.e., from wide to long
- Use `dcast()` to cast a molten data frame into the shape you want, i.e., from long to wide

```
library(reshape2)
(mdat <- melt(sdat, measure.vars=c("store_exp","online_exp"),
              variable.name = "Channel",
              value.name = "Expense"))
```

```
##    age gender income house    Channel Expense
## 1   57 Female 120963   Yes  store_exp   529.1
## 2   63 Female 122008   Yes  store_exp   478.0
## 3   59   Male 114202   Yes  store_exp   490.8
## 4   60   Male 113616   Yes  store_exp   347.8
## 5   51   Male 124253   Yes  store_exp   379.6
## 6   57 Female 120963   Yes online_exp   303.5
## 7   63 Female 122008   Yes online_exp   109.5
## 8   59   Male 114202   Yes online_exp   279.2
## 9   60   Male 113616   Yes online_exp   141.7
## 10  51   Male 124253   Yes online_exp   112.2
```

You melted the data frame `sdat` by two variables: `store_exp` and `online_exp` (`measure.vars=c("store_exp","online_exp")`). The new variable name is `Channel` set by command `variable.name = "Channel"`. The value name is `Expense` set by command `value.name = "Expense"`.

You can run a regression to study the effect of purchasing channel as follows:

```
# Here we use all observations from sim.dat
# Don't show result here
mdat<-melt(sim.dat[,1:6], measure.vars=c("store_exp","online_exp"),
            variable.name = "Channel",
              value.name = "Expense")
fit<-lm(Expense~gender+house+income+Channel+age,data=mdat)
summary(fit)
```

You can `melt()` list, matrix, table too. The syntax is similar, and we won't go through every situation. Sometimes we want to convert the data from "long" to "wide". For example, **you want to compare the online and in-store expense between male and female based on the house ownership.**

```
dcast(mdat, house + gender ~ Channel, sum)
```

```
## Using 'Expense' as value column. Use 'value.var' to override
```

```
##   house gender store_exp online_exp
## 1   Yes Female      1007      413.0
## 2   Yes   Male      1218      533.2
```

In the above code, what is the left side of `~` are variables that you want to group by. The right side is the variable you want to spread as columns. It will use the column indicating value from `melt()` before. Here is "`Expense`" .

4.2.2 `tidyr` package

The other package that will do similar manipulations is `tidyr`. Let's get a subset to illustrate the usage.

```
library(dplyr)
# practice functions we learnt before
sdat<-sim.dat[1:5,]%>%
  dplyr::select(age,gender,store_exp,store_trans)
sdat %>% tbl_df()
```

```
## # A tibble: 5 x 4
##     age gender store_exp store_trans
## * <int> <fct>      <dbl>       <int>
## 1    57 Female      529.           2
## 2    63 Female      478.           4
## 3    59 Male        491.           7
## 4    60 Male        348.          10
## 5    51 Male        380.           4
```

`gather()` function in `tidyr` is analogous to `melt()` in `reshape2`. The following code will do the same thing as we did before using `melt()`:

```
library(tidyr)
msdat<-tidyr::gather(sdat,"variable","value",store_exp,store_trans)
msdat %>% tbl_df()
```

```
## # A tibble: 10 x 4
##      age gender variable     value
##    <int> <fct>  <chr>        <dbl>
##  1    57 Female store_exp   529.
##  2    63 Female store_exp   478.
##  3    59 Male   store_exp   491.
##  4    60 Male   store_exp   348.
##  5    51 Male   store_exp   380.
##  6    57 Female store_trans   2.00
##  7    63 Female store_trans   4.00
##  8    59 Male   store_trans   7.00
##  9    60 Male   store_trans  10.0
## 10    51 Male   store_trans   4.00
```

Or if we use the pipe operation, we can write the above code as:

```
sdat%>%gather("variable","value",store_exp,store_trans)
```

It is identical with the following code using `melt()`:

```
melt(sdat, measure.vars=c("store_exp","store_trans"),
            variable.name = "variable",
              value.name = "value")
```

The opposite operation to `gather()` is `spread()`. The previous one stacks columns and the latter one spread the columns.

```
msdat %>% spread(variable,value)
```

```
##   age gender store_exp store_trans
## 1  51   Male     379.6           4
## 2  57 Female     529.1           2
## 3  59   Male     490.8           7
## 4  60   Male     347.8          10
## 5  63 Female     478.0           4
```

Another pair of functions that do opposite manipulations are `separate()` and `unite()`.

```
sepdat<- msdat %>%
  separate(variable,c("Source","Type"))
sepdat %>% tbl_df()
```

```
## # A tibble: 10 x 5
##      age gender Source Type   value
##    <int> <fct>  <chr>  <chr>  <dbl>
##  1    57 Female store  exp   529.
##  2    63 Female store  exp   478.
##  3    59 Male   store  exp   491.
##  4    60 Male   store  exp   348.
##  5    51 Male   store  exp   380.
##  6    57 Female store  trans   2.00
##  7    63 Female store  trans   4.00
##  8    59 Male   store  trans   7.00
##  9    60 Male   store  trans  10.0
## 10    51 Male   store  trans   4.00
```

You can see that the function separates the original column "variable" to two new columns "Source" and "Type". You can use sep= to set the string or regular expression to separate the column. By default, it is "_".

The unite() function will do the opposite: combining two columns. It is the generalization of paste() to a data frame.

```
sepdat %>%
  unite("variable",Source,Type,sep="_")
```

```
##     age gender    variable value
## 1    57 Female   store_exp 529.1
## 2    63 Female   store_exp 478.0
## 3    59   Male   store_exp 490.8
## 4    60   Male   store_exp 347.8
## 5    51   Male   store_exp 379.6
## 6    57 Female store_trans   2.0
## 7    63 Female store_trans   4.0
## 8    59   Male store_trans   7.0
## 9    60   Male store_trans  10.0
## 10   51   Male store_trans   4.0
```

The reshaping manipulations may be the trickiest part. You have to practice a lot to get familiar with those functions. Unfortunately, there is no shortcut.

5

Model Tuning Strategy

When training a machine learning model, there are many decisions to make. For example, when training a random forest, you need to decide the number of trees and the number of variables at each node. For lasso method, you need to determine the penalty parameter. There may be standard settings for some of the parameters, but it's unlikely to guess the right values for all of these correctly. Other than that, making good choices on how you split the data into training and testing sets can make a huge difference in helping you find a high-performance model efficiently.

This chapter will illustrate the practical aspects of model tuning. We will talk about different types of model error, sources of model error, hyperparameter tuning, how to set up your data and how to make sure your model implementation is correct. In practice applying machine learning is a highly iterative process.

5.1 Systematic Error and Random Error

Assume $\mathbf{X}$ is $n \times p$ observation matrix and $\mathbf{y}$ is response variable, we have:

$$\mathbf{y} = f(\mathbf{X}) + \epsilon$$

where ϵ is the random error with a mean of zero. The function $f(\cdot)$ is our modeling target, which represents the information in the response variable that predictors can explain. The main goal of estimating $f(\cdot)$ is inference or prediction, or sometimes both. In general, there is a trade-off between flexibility and interpretability of the model. So data scientists need to comprehend the delicate balance between these two.

Depending on the modeling purposes, the requirement for interpretability varies. If the prediction is the only goal, then as long as the prediction is accurate enough, the interpretability is not under consideration. In this case, people can use "black box" model, such as random forest, boosting tree, neural network and so on. These models are very flexible but nearly impossible to explain. Their accuracy is usually higher on the training set, but not necessary when it predicts. It is not surprising since those models have a huge number of parameters and high flexibility that they can "memorize" the entire training data. A paper by Chiyuan Zhang et al. in 2017 pointed out that "Deep neural networks (even just two-layer net) easily fit random labels" (**?**). The traditional forms of regularization, such as weight decay, dropout, and data augmentation, fail to control generalization error. It poses a conceptual challenge to statistical theory and also calls our attention when we use such black-box models.

There are two kinds of application problems: complete information problem and incomplete information problem. The complete information problem has all the information you need to know the correct response. Take the famous cat recognition, for example, all the information you need to identify a cat is in the picture. In this situation, the algorithm that penetrates the data the most wins. There are some other similar problems such as the self-driving car, chess game, facial recognition and speech recognition. But in most of the data science applications, the information is incomplete. If you want to know whether a customer is going to purchase again or not, it is unlikely to have 360-degree of the customer's information. You may have their historical purchasing record, discounts and service received. But you don't know if the customer sees your advertisement, or has a friend recommends competitor's product, or encounters some unhappy purchasing experience somewhere. There could be a myriad of factors that will influence the customer's purchase decision while what you have as data is only a small part. To make things worse, in many cases, you don't even know what you don't know. Deep learning doesn't have any advantage in solving those problems. Instead, some parametric models often work better in this situation. You will comprehend this more after learning the different types of model error. Assume we have $\hat{f}$ which is an estimator of f. Then we can further get $\hat{\mathbf{y}} = \hat{f}(\mathbf{X})$. The predicted error is divided into two parts, systematic error, and random error:

$$E(\mathbf{y} - \hat{\mathbf{y}})^2 = E[f(\mathbf{X}) + \epsilon - \hat{f}(\mathbf{X})]^2 = \underbrace{E[f(\mathbf{X}) - \hat{f}(\mathbf{X})]^2}_{(1)} + \underbrace{Var(\epsilon)}_{(2)}$$

It is also called Mean Square Error (MSE) where (1) is the systematic error. It exists because $\hat{f}$ usually does not entirely describe the "systematic relation" between X and y which refers to the stable relationship that exists across different samples or time. Model improvement can help reduce this kind of error; (2) is the random error which represents the part of y that cannot be explained by X. A more complex model does not reduce the error. There are three reasons for random error:

1. the current sample is not representative, so the pattern in one sample set does not generalize to a broader scale.
2. The information is incomplete. In other words, you don't have all variables needed to explain the response.
3. Measurement error in the variables.

Deep learning has significant success solving problems with complete information and usually low measurement error. As mentioned before, in a task like image recognition, all you need are the pixels in the pictures. So in deep learning applications, increasing the sample size can improve the model performance significantly. But it may not perform well in problems with incomplete information. The biggest problem with the black-box model is that it fits random error, i.e., over-fitting. The notable feature of random error is that it varies over different samples. So one way to determine whether overfitting happens is to reserve a part of the data as the test set and then check the performance of the trained model on the test data. Note that overfitting is a general problem from which any model could suffer. However, since black-box models usually have a large number of parameters, it is much more suspectable to over-fitting.

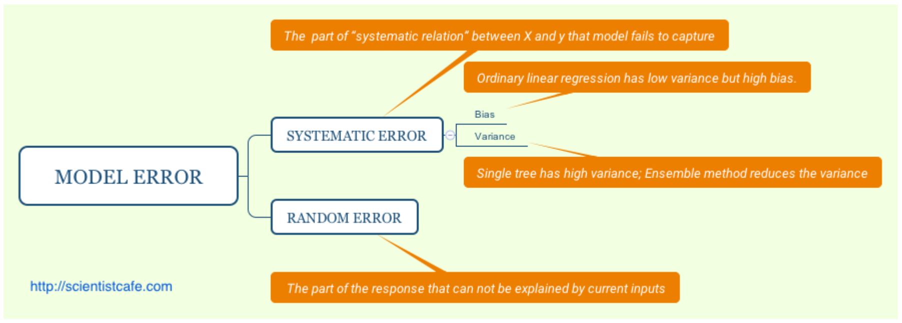

MODEL ERROR
SYSTEMATIC ERROR
RANDOM ERROR
Bias
Variance
The part of "systematic relation" between X and y that model fails to capture
Ordinary linear regression has low variance but high bias.
Single tree has high variance; Ensemble method reduces the variance
The part of the response that can not be explained by current inputs
http://scientistcafe.com

FIGURE 5.1: Types of Model Error

The systematic error can be further decomposed as:

$$\begin{aligned}
E[f(\mathbf{X})-\hat{f}(\mathbf{X})]^2 &= E\left(f(\mathbf{X})-E[\hat{f}(\mathbf{X})]+E[\hat{f}(\mathbf{X})]-\hat{f}(\mathbf{X})\right)^2\\
&= E\left(E[\hat{f}(\mathbf{X})]-f(\mathbf{X})\right)^2+E\left(\hat{f}(\mathbf{X})-E[\hat{f}(\mathbf{X})]\right)^2\\
&= [Bias(\hat{f}(\mathbf{X}))]^2+Var(\hat{f}(\mathbf{X}))
\end{aligned}$$

The systematic error consists of two parts, $Bias(\hat{f}(\mathbf{X}))$ and $Var(\hat{f}(\mathbf{X}))$. To minimize the systematic error, we need to minimize both. The bias represents the error caused by the model's approximation of the reality, i.e., systematic relation, which may be very complex. For example, linear regression assumes a linear relationship between the predictors and the response, but rarely is there a perfect linear relationship in real life. So linear regression is more likely to have a high bias.

To explore bias and variance, let's begin with a simple simulation. We will simulate a data with a non-linear relationship and fit different models on it. An intuitive way to show these is to compare the plots of various models.

The code below simulates one predictor (`x`) and one response variable (`fx`). The relationship between `x` and `fx` is non-linear.

```r
source(ids_url('R/multiplot.r'))
# randomly simulate some non-linear samples
x = seq(1, 10, 0.01) * pi
e = rnorm(length(x), mean = 0, sd = 0.2)
fx <- sin(x) + e + sqrt(x)
dat = data.frame(x, fx)
```

Then fit a simple linear regression on these data:

```r
# plot fitting result
library(ggplot2)
ggplot(dat, aes(x, fx)) + geom_point() + geom_smooth(method = "lm", se = FALSE)
```

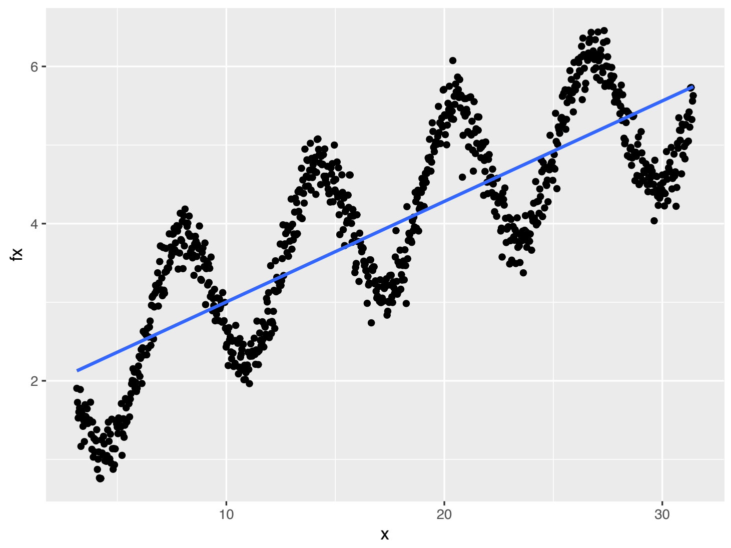


FIGURE 5.2: High bias model

Despite a large sample size, trained linear regression cannot describe the relationship very well. In other words, in this case, the model has a high bias (Fig. **??**). People also call it underfitting.

Since the estimated parameters will be somewhat different for the different samples, there is the variance of estimates. Intuitively, it gives you some sense that if we fit the same model with different samples (presumably, they are from the same population), how much will the estimates change. Ideally, the change is trivial. For high variance models, small changes in the training data result in very different estimates. In general, a model with high flexibility also has high variance., such as the CART tree, and the initial boosting method. To overcome that problem, the Random Forest and Gradient Boosting Model aim to reduce the variance by summarizing the results obtained from different samples.

Let's fit the above data using a smoothing method which is highly flexible and can fit the current data tightly:

```r
ggplot(dat, aes(x, fx)) + geom_smooth(span = 0.03)
```

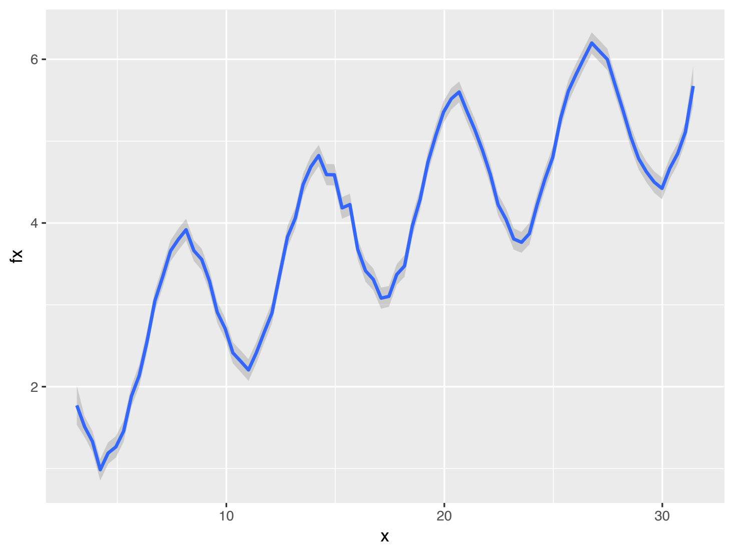
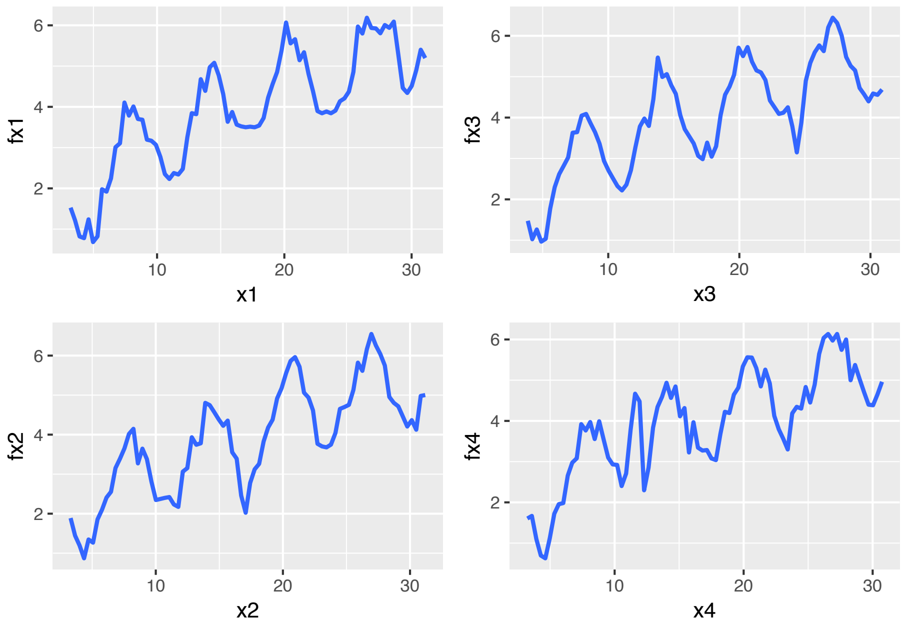


FIGURE 5.3: High variance model

The resulting plot (Fig. **??**) indicates the smoothing method fit the data much better so it has a much smaller bias. However, this method has a high variance. If we simulate different subsets of the sample, the result curve will change significantly:

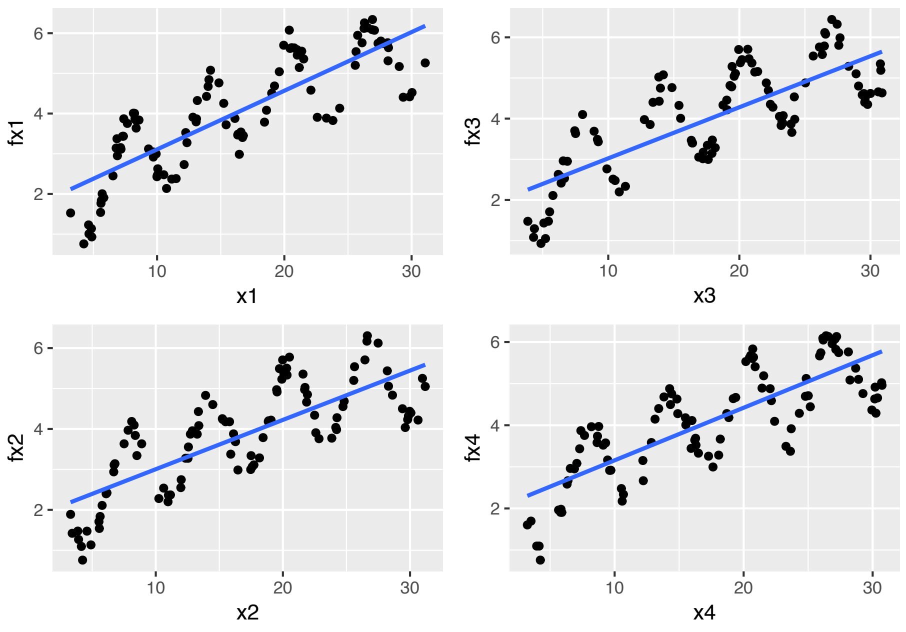
```r
# set random seed
set.seed(2016)
# sample part of the data to fit model sample 1
idx1 = sample(1:length(x), 100)
dat1 = data.frame(x1 = x[idx1], fx1 = fx[idx1])
p1 = ggplot(dat1, aes(x1, fx1)) + geom_smooth(span = 0.03)
```

```r
# sample 2
idx2 = sample(1:length(x), 100)
dat2 = data.frame(x2 = x[idx2], fx2 = fx[idx2])
p2 = ggplot(dat2, aes(x2, fx2)) + geom_smooth(span = 0.03)
# sample 3
idx3 = sample(1:length(x), 100)
dat3 = data.frame(x3 = x[idx3], fx3 = fx[idx3])
p3 = ggplot(dat3, aes(x3, fx3)) + geom_smooth(span = 0.03)
# sample 4
idx4 = sample(1:length(x), 100)
dat4 = data.frame(x4 = x[idx4], fx4 = fx[idx4])
p4 = ggplot(dat4, aes(x4, fx4)) + geom_smooth(span = 0.03)
multiplot(p1, p2, p3, p4, cols = 2)
```



The fitted lines (blue) change over different samples which means it has high variance. People also call it overfitting. Fitting the linear model using the same four subsets, the result barely changes:

```r
p1 = ggplot(dat1, aes(x1, fx1)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
p2 = ggplot(dat2, aes(x2, fx2)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
p3 = ggplot(dat3, aes(x3, fx3)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
p4 = ggplot(dat4, aes(x4, fx4)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE)
multiplot(p1, p2, p3, p4, cols = 2)
```



In general, the variance $(Var(\hat{f}(\mathbf{X})))$ **increases** and the bias $(Bias(\hat{f}(\mathbf{X})))$ **decreases** as the model flexibility increases. Variance and bias together determine the systematic error. As we increase the flexibility of the model, at first the rate at which $Bias(\hat{f}(\mathbf{X}))$ decreases is faster than $Var(\hat{f}(\mathbf{X}))$, so the MSE decreases. However, to some degree, higher flexibility has little effect on $Bias(\hat{f}(\mathbf{X}))$ but $Var(\hat{f}(\mathbf{X}))$ increases significantly, so the MSE increases.

5.1.1 Measurement Error in the Response

The measurement error in the response contributes to the random error (ϵ). This part of the error is irreducible if you change the data collection mechanism, and so it makes the root mean square error (RMSE) and R^2 have the corresponding upper and lower limits. RMSE and R^2 are commonly used performance measures for the regression model which we will talk in more detail later. Therefore, the random error term not only represents the part of fluctuations the model cannot explain but also contains measurement error in the response variables. Section 20.2 of Applied Predictive Modeling (**?**) has an example that shows the effect of the measurement error in the response variable on the model performance (RMSE and R^2).

The authors increased the error in the response proportional to a base level error which was gotten using the original data without introducing extra noise. Then fit a set of models repeatedly using the "contaminated" data sets to study the change of $RMSE$ and R^2 as the level of noise. Here we use clothing consumer data for a similar illustration. Suppose many people do not want to disclose their income and so we need to use other variables to establish a model to predict income. We set up the following model:

```
# load data
sim.dat <- read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
ymad <- mad(na.omit(sim.dat$income))
# calculate z-score
zs <- (sim.dat$income - mean(na.omit(sim.dat$income)))/ymad
# which(na.omit(zs>3.5)): identify outliers which(is.na(zs)):
# identify missing values
idex <- c(which(na.omit(zs > 3.5)), which(is.na(zs)))
# delete rows with outliers and missing values
sim.dat <- sim.dat[-idex, ]
fit <- lm(income ~ store_exp + online_exp + store_trans + online_trans,
    data = sim.dat)
```

The output shows that without additional noise, the root mean square error (RMSE) of the model is 29567, R^2 is 0.6.

Let's add various degrees of noise (0 to 3 times the RMSE) to the variable `income`:

$$RMSE \times (0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0)$$

```r
noise <- matrix(rep(NA, 7 * nrow(sim.dat)), nrow = nrow(sim.dat),
    ncol = 7)
for (i in 1:nrow(sim.dat)) {
    noise[i, ] <- rnorm(7, rep(0, 7), summary(fit)$sigma * seq(0,
        3, by = 0.5))
}
```

We then examine the effect of noise intensity on R^2 for models with different complexity. The models with complexity from low to high are: ordinary linear regression, partial least square regression(PLS), multivariate adaptive regression spline (MARS), support vector machine (SVM, the kernel function is radial basis function), and random forest.

```r
# fit ordinary linear regression
rsq_linear <- rep(0, ncol(noise))
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- lm(withnoise ~ store_exp + online_exp + store_trans +
        online_trans, data = sim.dat)
    rsq_linear[i] <- summary(fit0)$adj.r.squared
}
```

PLS is a method of linearizing nonlinear relationships through hidden layers. It is similar to the principal component regression (PCR), except that PCR does not take into account the information of the dependent variable when selecting the components, and its purpose is to find the linear combinations (i.e., unsupervised) that capture the most variance of the independent variables. When the independent variables and response variables are related, PCR can well identify the systematic relationship between them. However, when there exist independent variables not associated with response variable, it will undermine PCR's performance. And PLS maximizes the linear combination of dependencies with the response variable. In the current case, the more complicated PLS does not perform better than simple linear regression.

```r
# pls: conduct PLS and PCR
library(pls)
rsq_pls <- rep(0, ncol(noise))
# fit PLS
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- plsr(withnoise ~ store_exp + online_exp + store_trans +
        online_trans, data = sim.dat)
    rsq_pls[i] <- max(drop(R2(fit0, estimate = "train", intercept = FALSE)$val))
}
```

```r
# earth: fit mars
library(earth)
rsq_mars <- rep(0, ncol(noise))
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- earth(withnoise ~ store_exp + online_exp + store_trans +
        online_trans, data = sim.dat)
    rsq_mars[i] <- fit0$rsq
}
```

```r
# caret: awesome package for tuning predictive model
library(caret)
rsq_svm <- rep(0, ncol(noise))
# Need some time to run
for (i in 1:7) {
    idex <- which(is.na(sim.dat$income))
    withnoise <- sim.dat$income + noise[, i]
    trainX <- sim.dat[, c("store_exp", "online_exp", "store_trans",
        "online_trans")]
    trainY <- withnoise
    fit0 <- train(trainX, trainY, method = "svmRadial", tuneLength = 15,
        trControl = trainControl(method = "cv"))
    rsq_svm[i] <- max(fit0$results$Rsquared)
}
```

```r
# randomForest: random forest model
library(randomForest)
```

```r
rsq_rf <- rep(0, ncol(noise))
# ntree=500 number of trees na.action = na.omit ignore
# missing value
for (i in 1:7) {
    withnoise <- sim.dat$income + noise[, i]
    fit0 <- randomForest(withnoise ~ store_exp + online_exp +
        store_trans + online_trans, data = sim.dat, ntree = 500,
        na.action = na.omit)
    rsq_rf[i] <- tail(fit0$rsq, 1)
}
library(reshape2)
rsq <- data.frame(cbind(Noise = c(0, 0.5, 1, 1.5, 2, 2.5, 3),
    rsq_linear, rsq_pls, rsq_mars, rsq_svm, rsq_rf))
rsq <- melt(rsq, id.vars = "Noise", measure.vars = c("rsq_linear",
    "rsq_pls", "rsq_mars", "rsq_svm", "rsq_rf"))
```

```r
library(ggplot2)
ggplot(data = rsq, aes(x = Noise, y = value, group = variable,
    colour = variable)) + geom_line() + geom_point() + ylab("R2")
```

Fig. **??** shows that:

All model performance decreases sharply with increasing noise intensity. To better anticipate model performance, it helps to understand the way variable is measured. It is something need to make clear at the beginning of an analytical project. A data scientist should be aware of the quality of the data in the database. For data from the clients, it is an important to understand the quality of the data by communication.

More complex model is not necessarily better. The best model in this situation is MARS, not random forests or SVM. Simple linear regression and PLS perform the worst when noise is low. MARS is more complicated than the linear regression and PLS, but it is simpler and easier to explain than random forest and SVM.

When noise increases to a certain extent, the potential structure becomes vaguer, and complex random forest model starts to fail. When the systematic measurement error is significant, a more

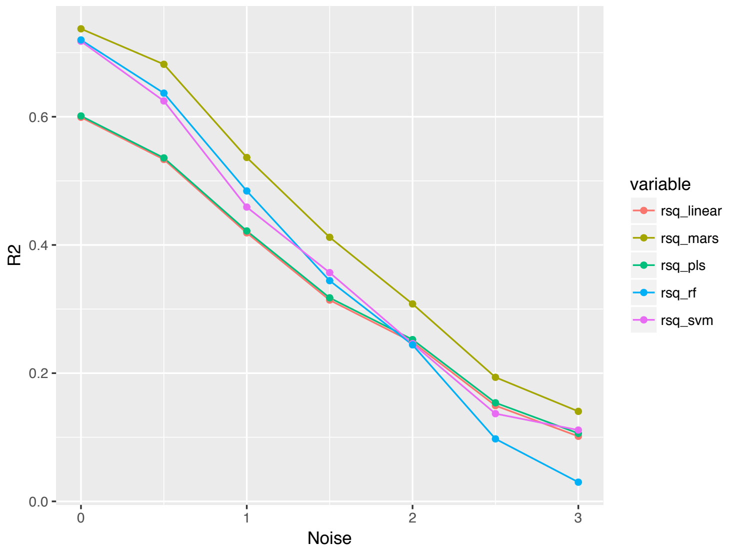


FIGURE 5.4: Test set R^2 profiles for income models when measurement system noise increases. `rsq_linear`: linear regression, `rsq_pls`: Partial Least Square, `rsq_mars`: Multiple Adaptive Regression Spline Regression, `rsq_svm`: Support Vector Machine `rsq_rf`: Random Forest

straightforward but not naive model may be a better choice. It is always a good practice to try different models, and select the simplest model in the case of similar performance. Model evaluation and selection represent the career "maturity" of a data scientist.

5.1.2 Measurement Error in the Independent Variables

The traditional statistical model usually assumes that the measurement of the independent variable has no error which is not possible in practice. Considering the error in the independent variables is necessary. The impact of the error depends on the following factors: (1) the magnitude of the randomness; (2) the importance of the corresponding variable in the model, and (3) the type of model used. Use variable `online_exp` as an example. The approach is similar to the previous section. Add varying degrees of noise and see its impact on the model performance. We add the following different levels of noise (0 to 3 times the standard deviation) to`online_exp`:

$$\sigma_0 \times (0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0)$$

where σ_0 is the standard error of `online_exp`.

```
noise<-matrix(rep(NA,7*nrow(sim.dat)),nrow=nrow(sim.dat),ncol=7)
for (i in 1:nrow(sim.dat)){
noise[i,]<-rnorm(7,rep(0,7),sd(sim.dat$online_exp)*seq(0,3,by=0.5))
}
```

Likewise, we examine the effect of noise intensity on different models (R^2). The models with complexity from low to high are: ordinary linear regression, partial least square regression(PLS), multivariate adaptive regression spline (MARS), support vector machine (SVM, the Kernel function is radial basis function), and random forest. The code is similar as before so not shown here.

Comparing Fig. **??** and Fig. **??**, the influence of the two types of error is very different. The error in response cannot be overcome for any model, but it is not the case for the independent variables. Imagine an extreme case, if `online_exp` is completely random, that is, no information in it, the impact on the performance of random forest and support vector machine is marginal. Linear regression and PLS still perform similarly. With the increase of noise, the performance starts to decline faster. To a certain extent, it becomes steady. In general, if an independent variable contains error, other variables associated with it can compensate to some extent.

5.2 Data Splitting and Resampling

Those highly adaptable models can model complex relationships. However, they tend to overfit which leads to the poor prediction by learning too much from the data. It means that the model is susceptible to the specific sample used to fit it. When future data is not

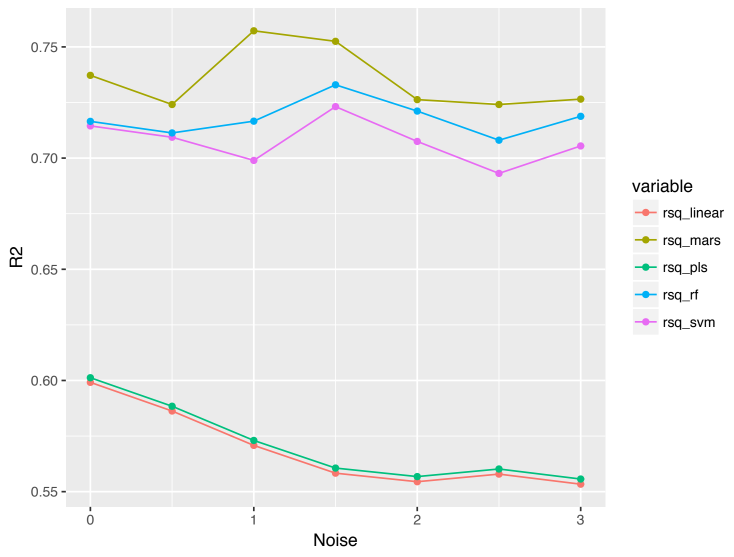


FIGURE 5.5: Test set R^2 profiles for income models when noise in `online_exp` increases. `rsq_linear` : linear regression, `rsq_pls` : Partial Least Square, `rsq_mars`: Multiple Adaptive Regression Spline Regression, `rsq_svm`: Support Vector Machine `rsq_rf`: Random Forest

exactly like the past data, the model prediction may have big mistakes. A simple model like ordinary linear regression tends instead to underfit which leads to a bad prediction by learning too little from the data. It systematically over-predicts or under-predicts the data regardless of how well future data resemble past data. Without evaluating models, the modeler will not know about the problem before the future samples. Data splitting and resampling are fundamental techniques to build sound models for prediction.

5.2.1 Data Splitting

Data splitting is to put part of the data aside as testing set (or Hold-outs, out of bag samples) and use the rest for model training. Training samples are also called in-sample. Model performance metrics evaluated using in-sample are retrodictive, not predictive.

The traditional business intelligence usually handles data description. Answer simple questions by querying and summarizing the data, such as:

- What is the monthly sales of a product in 2015?
- What is the number of visits to our site in the past month?
- What is the sales difference in 2015 for two different product designs?

There is no need to go through the tedious process of splitting the data, tuning and testing model to answer questions of this kind. Instead, people usually use as complete data as possible and then sum or average the parts of interest.

Many models have parameters which cannot be directly estimated from the data, such as λ in the lasso (penalty parameter), the number of trees in the random forest. This type of model parameter is called tuning parameter, and there is no analytical formula available to calculate the optimized value. Tuning parameters often control the complexity of the model. A poor choice can result in over-fitting or under-fitting. A standard approach to estimate tuning parameters is through cross-validation which is a data resampling approach.

To get a reasonable precision of the performance based on a single test set, the size of the test set may need to be large. So a conventional approach is to use a subset of samples to fit the model and use the rest to evaluate model performance. This process will repeat multiple times to get a performance profile. In that sense, resampling is based on splitting. The general steps are:

- Define a set of candidate values for tuning parameter(s)
    - For each candidate value in the set
        - Resample data
        - Fit model
        - Predict hold-out
        - Calculate performance
- Aggregate the results
- Determine the final tuning parameter

- Refit the model with the entire data set

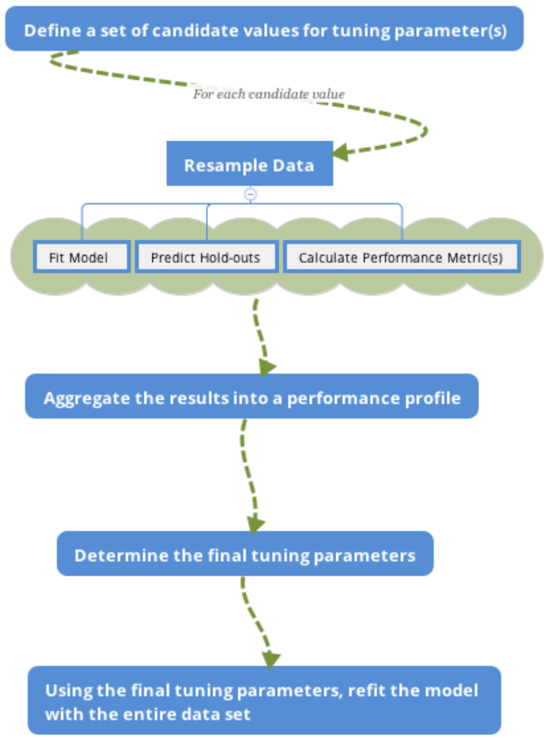

Define a set of candidate values for tuning parameter(s)
For each candidate value
Resample Data
Fit Model
Predict Hold-outs
Calculate Performance Metric(s)
Aggregate the results into a performance profile
Determine the final tuning parameters
Using the final tuning parameters, refit the model with the entire data set

FIGURE 5.6: Parameter Tuning Process

The above is an outline of the general procedure to tune parameters. Now let's focus on the critical part of the process: data splitting. Ideally, we should evaluate model using samples that were not used to build or fine-tune the model. So it provides an unbiased sense of model effectiveness. When the sample size is large, it is a good practice to set aside part of the samples to evaluate the final model. People use "training" data to indicate samples used to fit or fine-tune the model and "test" or "validation" data set is used to validate performance.

The first decision to make for data splitting is to decide the proportion of data in the test set. There are two factors to consider here: (1) sample size; (2) computation intensity. If the sample size is large enough which is the most common situation according to my experience, you can try to use 20%, 30% and 40% of the data as the test set, and see which one works the best. If the model is computationally intense, then you may consider starting from a smaller sample of data to train the model hence will have a higher portion of data in the test set. Depending on how it performs, you may need to increase the training set. If the sample size is small, you can use cross-validation or bootstrap which is the topic in the next section.

The next decision is to decide which samples are in the test set. There is a desire to make the training and test sets as similar as possible. A simple way is to split data by random sampling which, however, does not control for any of the data attributes, such as the percentage of the retained customer in the data. So it is possible that the distribution of outcomes is substantially different between the training and test sets. There are three main ways to split the data that account for the similarity of resulted data sets. We will describe the three approaches using the clothing company customer data as examples.

(1) Split data according to the outcome variable

Assume the outcome variable is customer segment (column `segment`) and we decide to use 80% as training and 20% test. The goal is to make the proportions of the categories in the two sets as similar as possible. The `createDataPartition()` function in `caret` will return a balanced splitting based on assigned variable.

```r
# load data
sim.dat <- read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master/Data/
library(caret)
# set random seed to make sure reproducibility
set.seed(3456)
trainIndex <- createDataPartition(sim.dat$segment, p = 0.8, list = FALSE,
    times = 1)
head(trainIndex)
```

```
##      Resample1
## [1,]         1
## [2,]         2
## [3,]         3
## [4,]         4
## [5,]         6
## [6,]         7
```

The `list = FALSE` in the call to `createDataPartition` is to return a data frame. The `times = 1` tells R how many times you want to split the data. Here we only do it once, but you can repeat the splitting multiple times. In that case, the function will return multiple vectors indicating the rows to training/test. You can set `times2` and rerun the above code to see the result. Then we can use the returned indicator vector `trainIndex` to get training and test sets:

```
# get training set
datTrain <- sim.dat[trainIndex, ]
# get test set
datTest <- sim.dat[-trainIndex, ]
```

According to the setting, there are 800 samples in the training set and 200 in test set. Let's check the distribution of the two sets:

```
library(plyr)
ddply(datTrain, "segment", summarise, count = length(segment),
    percentage = round(length(segment)/nrow(datTrain), 2))
```

```
##       segment count percentage
## 1 Conspicuous   160       0.20
## 2       Price   200       0.25
## 3     Quality   160       0.20
## 4       Style   280       0.35
```

```
ddply(datTest, "segment", summarise, count = length(segment),
    percentage = round(length(segment)/nrow(datTest), 2))
```

```
##       segment count percentage
## 1 Conspicuous    40       0.20
## 2       Price    50       0.25
## 3     Quality    40       0.20
## 4       Style    70       0.35
```

The percentages are the same for these two sets. In practice, it is possible that the distributions are not exactly identical but should be close.

(2) Divide data according to predictors

An alternative way is to split data based on the predictors. The goal is to get a diverse subset from a dataset so that the sample is representative. In other words, we need an algorithm to identify the n most diverse samples from a dataset with size N. However, the task is generally infeasible for non-trivial values of n and N (**?**). And hence practicable approaches to dissimilarity-based selection involve approximate methods that are sub-optimal. A major class of algorithms split the data on *maximum dissimilarity sampling*. The process starts from:

- Initialize a single sample as starting test set
- Calculate the dissimilarity between this initial sample and each remaining samples in the dataset
- Add the most dissimilar unallocated sample to the test set

To move forward, we need to define the dissimilarity between groups. Each definition results in a different version of the algorithm and hence a different subset. It is the same problem as in hierarchical clustering where you need to define a way to measure the distance between clusters. The possible approaches are to use minimum, maximum, sum of all distances, the average of all distances, etc. Unfortunately, there is not a single best choice, and you may have to try multiple methods and check the resulted sample sets. R users can implement the algorithm using `maxDissim()` function from `caret` package. The `obj` argument is to set the definition of dissimilarity. Refer to the help documentation for more details (`?maxDissim`).

Let's use two variables (`age` and `income`) from the customer data as an example to illustrate how it works in R and compare maximum dissimilarity sampling with random sampling.

```r
library(lattice)
# select variables
testing <- subset(sim.dat, select = c("age", "income"))
```

Random select 5 samples as initial subset (`start`) , the rest will be in `samplePool`:

```r
set.seed(5)
# select 5 random samples
startSet <- sample(1:dim(testing)[1], 5)
start <- testing[startSet, ]
# save the rest in data frame 'samplePool'
samplePool <- testing[-startSet, ]
```

Use `maxDissim()` to select another 5 samples from `samplePool` that are as different as possible with the initical set `start`:

```r
selectId <- maxDissim(start, samplePool, obj = minDiss, n = 5)
minDissSet <- samplePool[selectId, ]
```

The `obj = minDiss` in the above code tells R to use minimum dissimilarity to define the distance between groups. Next, random select 5 samples from `samplePool` in data frame `RandomSet`:

```r
selectId <- sample(1:dim(samplePool)[1], 5)
RandomSet <- samplePool[selectId, ]
```

Plot the resulted set to compare different sampling methods:

```r
start$group <- rep("Initial Set", nrow(start))
minDissSet$group <- rep("Maximum Dissimilarity Sampling", nrow(minDissSet))
RandomSet$group <- rep("Random Sampling", nrow(RandomSet))
xyplot(age ~ income, data = rbind(start, minDissSet, RandomSet), grid = TRUE,
    group = group, auto.key = TRUE)
```

The points from maximum dissimilarity sampling are far away from the initial samples (Fig. **??**, while the random samples are much closer to the initial ones. Why do we need a diverse subset? Because we hope the test set to be representative. If all test set samples are from respondents younger than 30, model performance

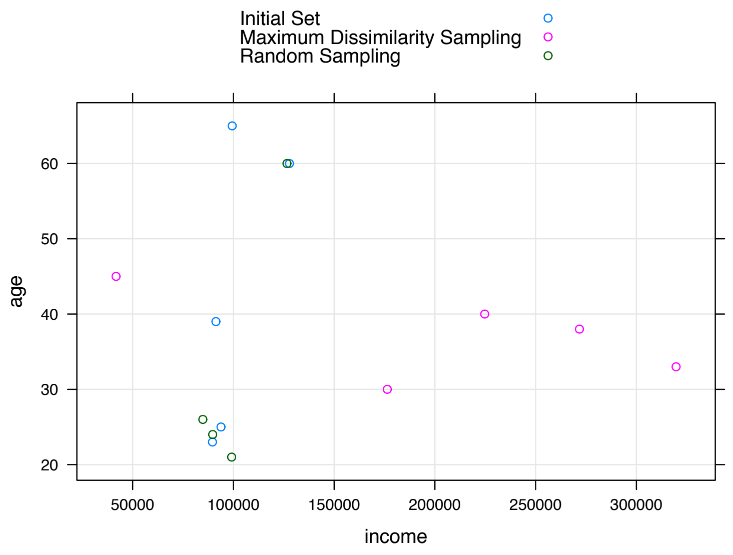


FIGURE 5.7: Compare Maximum Dissimilarity Sampling with Random Sampling

on the test set has a high risk to fail to tell you how the model will perform on more general population.

- Divide data according to time

For time series data, random sampling is usually not the best way. There is an approach to divide data according to time-series. Since time series is beyond the scope of this book, there is not much discussion here. For more detail of this method, see (**?**). We will use a simulated first-order autoregressive model [AR (1)] time-series data with 100 observations to show how to implement using the function `createTimeSlices ()` in the `caret` package.

```r
# simulte AR(1) time series samples
timedata = arima.sim(list(order=c(1,0,0), ar=-.9), n=100)
# plot time series
plot(timedata, main=(expression(AR(1)~~~phi==-.9)))
```

Fig. **??** shows 100 simulated time series observation. The goal is to make sure both training and test set to cover the whole period.

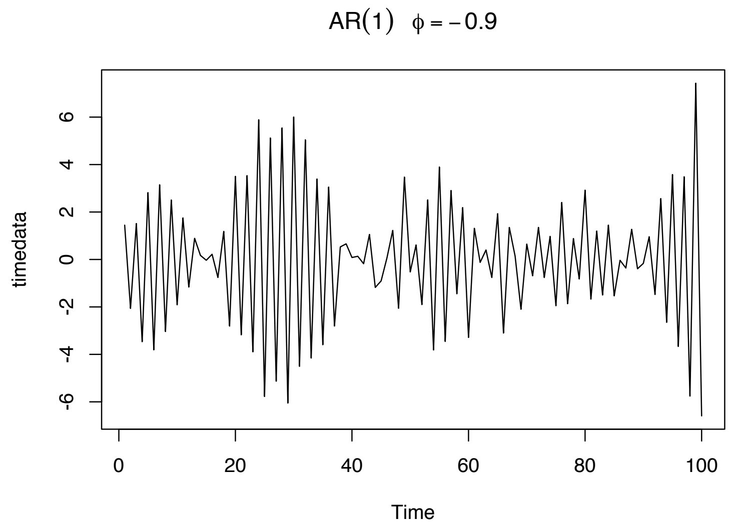


FIGURE 5.8: Divide data according to time

```
timeSlices <- createTimeSlices(1:length(timedata),
                   initialWindow = 36, horizon = 12, fixedWindow = T)
str(timeSlices,max.level = 1)
```

```
## List of 2
##  $ train:List of 53
##  $ test :List of 53
```

There are three arguments in the above `createTimeSlices()`.

- `initialWindow`: The initial number of consecutive values in each training set sample
- `horizon`: the number of consecutive values in test set sample
- `fixedWindow`: if FALSE, all training samples start at 1

The function returns two lists, one for the training set, the other for the test set. Let's look at the first training sample:

```
# get result for the 1st training set
trainSlices <- timeSlices[[1]]
# get result for the 1st test set
testSlices <- timeSlices[[2]]
```

```
# check the index for the 1st training and test set
trainSlices[[1]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [18] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
## [35] 35 36
```

```
testSlices[[1]]
```

```
##  [1] 37 38 39 40 41 42 43 44 45 46 47 48
```

The first training set is consist of sample 1-36 in the dataset (`initialWindow = 36`). Then sample 37-48 are in the first test set (`horizon = 12`). Type `head(trainSlices)` or `head(testSlices)` to check the later samples. If you are not clear about the argument `fixedWindow`, try to change the setting to be `F` and check the change in `trainSlices` and `testSlices`.

Understand and implement data splitting is not difficult. But there are two things to note:

1. The randomness in the splitting process will lead to uncertainty in performance measurement.
2. When the dataset is small, it can be too expensive to leave out test set. In this situation, if collecting more data is just not possible, the best shot is to use leave-one-out cross-validation which is in the next section.

5.2.2 Resampling

You can consider resampling as repeated splitting. The basic idea is: use part of the data to fit model and then use the rest of data to calculate model performance. Repeat the process multiple times and aggregate the results. The differences in resampling techniques usually center around the ways to choose subsamples. There are two main reasons that we may need resampling:

1. Estimate tuning parameters through resampling. Some examples of models with such parameters are Support

Vector Machine (SVM), models including the penalty (LASSO) and random forest.

2. For models without tuning parameter, such as ordinary linear regression and partial least square regression, the model fitting doesn't require resampling. But you can study the model stability through resampling.

We will introduce three most common resampling techniques: k-fold cross-validation, repeated training/test splitting, and bootstrap.

5.2.2.1 k-fold cross-validation

k-fold cross-validation is to partition the original sample into k equal size subsamples (folds). Use one of the k folds to validate the model and the rest $k-1$ to train model. Then repeat the process k times with each of the k folds as the test set. Aggregate the results into a performance profile.

Denote by $\hat{f}^{-\kappa}(X)$ the fitted function, computed with the κ^{th} fold removed and x_i^κ the predictors for samples in left-out fold. The process of k-fold cross-validation is as follows:

---

1. Partition the original sample into k equal size folds
2. for $\kappa = 1k$

- Use data other than fold κ to train the model $\hat{f}^{-\kappa}(X)$
- Apply $\hat{f}^{-\kappa}(X)$ to predict fold κ to get $\hat{f}^{-\kappa}(x_i^\kappa)$

3. Aggregate the results

$$\hat{Error} = \frac{1}{N}\Sigma_{\kappa=1}^{k}\Sigma_{x_i^\kappa}L(y_i^\kappa, \hat{f}^{-\kappa}(x_i^\kappa))$$

It is a standard way to find the value of tuning parameter that gives you the best performance. It is also a way to study the variability of model performance.

The following figure represents a 5-fold cross-validation example.

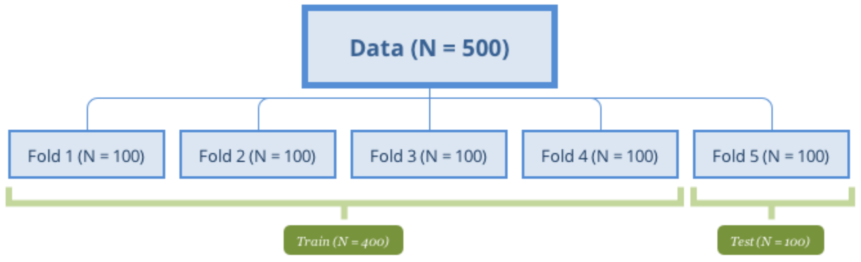


FIGURE 5.9: 5-fold cross-validation

A special case of k-fold cross-validation is Leave One Out Cross Validation (LOOCV) where $k = 1$. When sample size is small, it is desired to use as many data to train the model. Most of the functions have default setting $k = 10$. The choice is usually 5-10 in practice, but there is no standard rule. The more folds to use, the more samples are used to fit model, and then the performance estimate is closer to the theoretical performance. Meanwhile, the variance of the performance is larger since the samples to fit model in different iterations are more similar. However, LOOCV has high computational cost since the number of interactions is the same as the sample size and each model fit uses a subset that is nearly the same size of the training set. On the other hand, when k is small (such as 2 or 3), the computation is more efficient, but the bias will increase. When the sample size is large, the impact of k becomes marginal.

Chapter 7 of (**?**) presents a more in-depth and more detailed discussion about the bias-variance trade-off in k-fold cross-validation.

You can implement k-fold cross-validation using `createFolds()` in `caret`:

```r
library(caret)
class<-sim.dat$segment
```

```
# creat k-folds
set.seed(1)
cv<-createFolds(class,k=10,returnTrain=T)
str(cv)
```

```
## List of 10
##  $ Fold01: int [1:900] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold02: int [1:900] 1 2 3 4 5 6 7 9 10 11 ...
##  $ Fold03: int [1:900] 1 2 3 4 5 6 7 8 10 11 ...
##  $ Fold04: int [1:900] 1 2 3 4 5 6 7 8 9 11 ...
##  $ Fold05: int [1:900] 1 3 4 6 7 8 9 10 11 12 ...
##  $ Fold06: int [1:900] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Fold07: int [1:900] 2 3 4 5 6 7 8 9 10 11 ...
##  $ Fold08: int [1:900] 1 2 3 4 5 8 9 10 11 12 ...
##  $ Fold09: int [1:900] 1 2 4 5 6 7 8 9 10 11 ...
##  $ Fold10: int [1:900] 1 2 3 5 6 7 8 9 10 11 ...
```

The above code creates ten folds (`k=10`) according to the customer segments (we set `class` to be the categorical variable `segment`). The function returns a list of 10 with the index of rows in training set.

5.2.2.2 Repeated Training/Test Splits

In fact, this method is nothing but repeating the training/test set division on the original data. Fit the model with the training set, and evaluate the model with the test set. Unlike k-fold cross-validation, the test set generated by this procedure may have duplicate samples. A sample usually shows up in more than one test sets. There is no standard rule for split ratio and number of repetitions. The most common choice in practice is to use 75% to 80% of the total sample for training. The remaining samples are for validation. The more sample in the training set, the less biased the model performance estimate is. Increasing the repetitions can reduce the uncertainty in the performance estimates. Of course, it is at the cost of computational time when the model is complex. The number of repetitions is also related to the sample size of the test set. If the size is small, the performance estimate is more volatile. In this case, the number of repetitions needs to be higher to deal with the uncertainty of the evaluation results.

We can use the same function (`createDataPartition ()`) as before. If you look back, you will see `times = 1`. The only thing to change is to set it to the number of repetitions.

```
trainIndex <- createDataPartition(sim.dat$segment, p = .8, list = FALSE, times = 5)
dplyr::glimpse(trainIndex)
```

```
##  int [1:800, 1:5] 1 3 4 5 6 7 8 9 10 11 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "Resample1" "Resample2" "Resample3" "Resample4" ...
```

Once know how to split the data, the repetition comes naturally.

5.2.2.3 Bootstrap Methods

Bootstrap is a powerful statistical tool (a little magic too). It can be used to analyze the uncertainty of parameter estimates (**?**) quantitatively. For example, estimate the standard deviation of linear regression coefficients. The power of this method is that the concept is so simple that it can be easily applied to any model as long as the computation allows. However, you can hardly obtain the standard deviation for some models by using the traditional statistical inference.

Since it is with replacement, a sample can be selected multiple times, and the bootstrap sample size is the same as the original data. So for every bootstrap set, there are some left-out samples, which is also called "out-of-bag samples." The out-of-bag sample is used to evaluate the model. Efron points out that under normal circumstances (**?**), bootstrap estimates the error rate of the model with more certainty.The probability of an observation i in bootstrap sample B is:

$$\begin{aligned} Pri \in B &= 1 - \left(1 - \frac{1}{N}\right)^N \\ &\approx 1 - e^{-1} \\ &= 0.632 \end{aligned}$$

On average, 63.2% of the observations appeared at least once in a bootstrap sample, so the estimation bias is similar to 2-fold cross-validation. As mentioned earlier, the smaller the number of folds, the larger the bias. Increasing the sample size will ease the problem. In general, bootstrap has larger bias and smaller uncertainty than cross-validation. Efron came up the following ".632 estimator" to alleviate this bias:

$$(0.632\text{»}\mathit{original\ bootstrap\ estimate})+(0.368\text{»}\mathit{apparent\ error\ rate})$$

The apparent error rate is the error rate when the data is used twice, both to fit the model and to check its accuracy and it is apparently over-optimistic. The modified bootstrap estimate reduces the bias but can be unstable with small samples size. This estimate can also be unduly optimistic when the model severely over-fits since the apparent error rate will be close to zero. Efron and Tibshirani (**?**) discuss another technique, called the "632+ method," for adjusting the bootstrap estimates.

6

Measuring Performance