JSM 2017 CE: Preparing Statistician/Statistics Graduates to Be Data Scientist

After taking the CE course, participants will:

- Understand data science is the process of defining the problem with business knowledge, gathering needed data from various sources, developing models, extracting insight and recommend actions.
- Get familiar with the cloud-based big data platform (Hadoop/ Hive/ Spark/ GPU etc.) that are widely used in the development and production setting for industry and know how to quickly transit from academia environment to enterprise environment.
- Get familiar with data extraction, transformation and load from various database systems such that participates can learn how to become self-sufficient to get needed data in enterprise environment.
- Understand how to leverage interactive dashboard to present insight and results and communicate efficient with business partners and customers.
- Understand what data scientists "in the wild" are doing to better prepared to be successfully data scientist in the future.
- Learn how to encode a real problem to a data science problem, search for the right data, preprocess data and deploy of analytical results through a case study.
- Get familiar with how to achieve high-performance computing for standard statistical procedures with big data infrastructure.

Table of Contents

The Science and Art of Data

Hui Lin and Ming Li

2017-07-04

Copyright Statement

About the Authors

Hui Lin is currently Data Scientist at DuPont Pioneer. She is a leader within DuPont at applying advanced data science to enhance Marketing and Sales effectiveness. She has been providing statistical leadership for a broad range of predictive analytics and market research analysis since 2013. She is the co-founder of Central Iowa R User Group, blogger of scientistcafe.com and program Chair of Statistics in Marketing Section of ASA for 2018. 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 Sr. Data Scientist at Amazon and an Adjunct Faculty of Department and Marketing and Business Analytics in TAMU – Commerce. He is also the Chair of Quality & Productivity Section of ASA for 2017. He was a Data Scientist at Wal-Mart, a Statistical Leader at General Electric Global Research Center and Research Statistician at SAS Institute. 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 diverse backgrounds such as statistics, operation research, economics, computer science, and business analytics.

Acknowledgements

We want to give special thanks to Alex Shum and David Body for their editing and comments on the sections of this material.

1. THE ART OF DATA SCIENCE

Data science and data scientist have become buzz words. Allow me to reiterate what you may have already heard a million times in the media: **data scientists are in demand and demand continues to grow**. A study by the McKinsey Global Institute concludes,

"a shortage of the analytical and managerial talent necessary to make the most of Big Data is a significant and pressing challenge (for the U.S.)."

You may expect that statisticians and graduate students from traditional statistics departments are great data scientist candidates. But the situation is that the majority of current data scientists do not have a statistical background. As David Donoho pointed out:

"statistics is being marginalized here; the implicit message is that statistics is a part of what goes on in data science but not a very big part." (from "50 years of Data Science").

What is wrong? The activities that preoccupied statistics over centuries are now in the limelight, but those activities are claimed to belong to a new discipline and are practiced by professionals from various backgrounds. Various professional statistics organizations are reacting to this confusing situation. (Page 5-7, "50 Years of Data Science") From those discussions, Donoho summarizes the main recurring "Memes" about data sciences:

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

The first two are linked together which leads to statisticians' current position on data science. I assume everyone has heard the 3V (volume, variety and velocity) definition of big data. The media hasn't taken a minute break from touting "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, system using Map/Reduce to process large data sets distributed across a cluster of computers. 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.

Some media think they sense the trends in hiring and government funding. 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 have been laying the groundwork of data science for at least 50 years. Today's data science is an enlargement of traditional academic statistics rather than a brand new discipline.

Our goal is to help you enlarge your background to become a data scientist in US enterprise environments. We will use case studies to cover how to leverage big data distributed platforms (Hadoop / Hive / Spark), data wrangling, modeling, dynamic reporting (R

markdown) and interactive dashboards (R-Shiny) to tackle real-world data science problems. One typical skill gap for statisticians is data ETL (extraction, transformation and load) in production environments, and we will cover this topic as well. Data science is a combination of science and art with data as the foundation. We will also cover the "art" part to guide participants to learn soft skills to define data science problems and to effectively communicate with business stakeholders. The prerequisite knowledge is MS level education in statistics and entry level knowledge of R-Studio.

1.1.What is Data Science?

This question is not new. When you tell people "I am a data scientist". "Ah, data scientist!" Yes, who doesn't know that data scientist is the sexist job in 21th century? If they ask further what is data science and what exactly do data scientists do, it may effectively kill the conversation.

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 make plans to invest $100 million over the next five years in a new Data Science Initiative that will enhance opportunities for student and faculty researchers across the university to tap into the enormous potential of big data. UM Provost Martha Pollack said:

"Data science has become a fourth approach to scientific discovery, in addition to experimentation, modeling and computation,…"

How does the Data Science Initiative 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."

With the data science hype picking up stream, many professionals changed their titles to Data Scientist without any of the necessary qualifications. But at that time, the data scientist title was not well defined which lead to confusion in the market, obfuscation in resumes, and exaggeration of skills. 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 obvious intersections among skillsets. We see data science as a discipline to make sense of data. In order to make sense of data, statistics is indispensable. But a data scientist also needs many other skills.

In the obscenity case of Jacobellis v. Ohio (1964), Potter Stewart wrote in his short concurrence that "hard-core pornography" was hard to define, but that "I know it when I see it." This applies to many things including data science. It is hard to define but you know it when you see it.

So instead of scratching my head to figure out a one sentence definition, We are going to sketch the history of data science, what kind of questions data science can answer, and describe the skills required for being a data scientist. We hope this can give you a better depiction of data science.

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]

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 tool box has been expanding fast. The applications include business, health, biology, social science, politics etc.

John Tukey identified 4 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.Is it Science? Totally?

Let's take one step back. What is science? Here is what John Tukey said:

There are diverse views as to what makes a science, but three constituents will be judged essential by most, viz.:

(a1) intellectual content,
(a2) organization in an understandable form,
(a3) reliance upon the test of experience as the ultimate standard of validity

The first one (a1) doesn't provide useful information. And (a2) can't distinguish science from art very well. The last one is a key character of science. The influential philosopher of science Karl Popper argued that science advances by falsifying hypotheses. If science needs to be falsifiable, then data science is not 100% science. It is true that there are some analytical results that can be validated (falsified) through cross validation or comparing prediction with future outcomes. But certainly not all of them. Even in the problem of prediction, we can't validate predictions in the 2nd order chaotic systems.

We can't scientifically validate many unsupervised learning or descriptive analysis, especially in the context of marketing. In that sense, data science is a combination of science and art.

There is another definition of science from the famous computer scientist Donald Knuth. He said in his legendary 1974 essay Computer Programming as an Art:

"Science is knowledge which we understand so well that we can teach it to a computer."

Computers are indispensable for data science. But can we teach computers to do all the work data scientists do today? No. So it is not totally science. Computers can't communicate with stakeholders to transform a real life problem to be data problem. Computers don't know which questions can be answered through analytics. Computers don't know how to explain the results to different audiences using different ways according to their backgrounds. Computers are powerful in many ways but certainly not all. Would a computer enter a 'runaway reaction' of self-improvement cycles so that it could surpass human in every way in the future? Well, that is not a question we are trying to answer here. If you are interested in the future of technology, there are some books you can refer to. Ray Kurzweil (The Singularity Is Near), Yuval Noah Harari (Homo Deus: A Brief History of Tomorrow) and Kevin

Kelly (The Inevitable). At the risk of being short-sighted, we will assume it won't happen in foreseeable future.

To be simple I will still use data science in the rest of the book. But it is important to realize that data science includes art.

1.3.What Kind of Questions Can Data Science Solve?

1.3.1.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 figure out data analytics can't give the answer they want. What kind of questions can data science solve? What are the requirements for our question?

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 not 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 on 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 don't guarantee that you can answer it. I have encountered a well-defined supply chain problem. My client asked about the stock needed for a product in a particular area. Why can not this question be answered? I did fit a Multivariate Adaptive Regression Spline (MARS) model and thought I found a reasonable solution. But it turned out later that the data they gave me was inaccurate. In some areas, only estimates of past supply figures were available. 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 can not 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!

1.3.2.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. non-parametric 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. 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, C in a certain aspect? Here are some examples:

- Are the purchasing amounts different between consumers receiving coupons and those without coupons?
- Are males more inclined to buy our products than females?
- Are there any differences in customer satisfaction in different business districts?
- Do the mice receiving a drug have a faster weight gain than the control group?
- Do soybeans carrying a particular gene contain more oil than the control group?

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.

2. Description

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 income of the family's annual observations unbiased?
- What is the mean/variance of the monthly sales volume of a product in different regions?
- What is the difference in the magnitude of the variable? (Decide whether the data needs to be standardized)
- What is the prediction variable in the model?
- What is the age distribution of the respondents? Data description is often used to check data, find the appropriate data preprocessing method, and demonstrate the model results.

3. Clustering

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

- Which consumers have similar product preferences? (Marketing)
- Which printer performs similar pattern to the broken ones? (Quality Control)
- How many different kinds of employees are there in the company? (Human Resources)
- How many different themes are there in the corpus? (Natural Language Processing)

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

- Is this customer going to buy our product? (yes/no)
- Is there a risk that a lender does not repay?
- Who is the author of this book?
- Is this spam email?

There are hundreds of classifiers. In practice, we do not have to try all the models as long as we fit in several of the best models in most cases.

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 will be the company's sales in the fourth quarter of this year?
- How long will the engine work?
- How much beer should we prepare for this event?

1.4.What Are the Required Skills for Data Scientist?

We talked about the bewildering definitions of data scientist. With the data science hype picking up, some professionals have begun changing their titles to Data Scientist without any of the necessary qualifications (see "Data Scientists…or Data Wannabes"). What are the required skills for 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 PhD 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, 97% of the data scientists today using R or Python. We will focus on R in this book, but both are great tools for data science. 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

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 and online retail. 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, **self-learning ability** is the most critical skill.

- Soft Skills

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

1.5.Types of Learning

There are three broad groups of styles: supervised learning, reinforcement learning, and unsupervised 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 confusion matrix, accuracy, AUC or information gain.

In reinforcement learning, the correct input/output pairs are not present. The model will learn from a sequence of actions and select the action maximizing the expected sum of the future rewards. There is a discount factor that makes future rewards less valuable than current rewards. Reinforcement learning is difficult for the following reasons:

(1) The rewards are not instant. If the action sequence is long, it is hard to know which action was wrong.

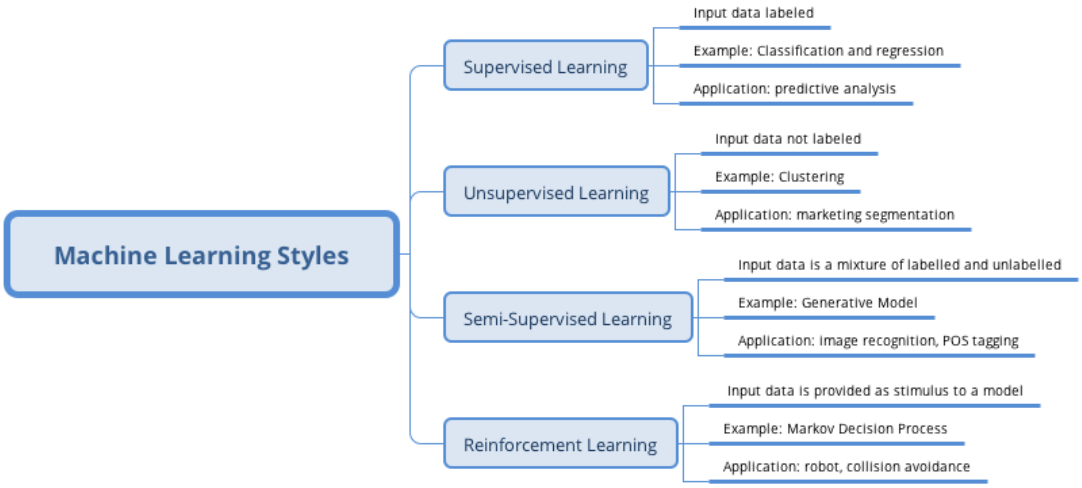
(2) The rewards are occasional. Each reward does not supply much information, so its impact of parameter change is limited. Typically, it is not likely to learn a large number of parameters using reinforcement learning. However, it is possible for supervised and unsupervised learning. The number of parameters in reinforcement learning usually range from dozens to maybe 1,000, but not millions.

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 principle component regression.



1.6.Types of Algorithm

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. The categorization here is based on the structure (such as tree model, Regularization Methods) or type of question to answer (such as regression). 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 surrogate features.[1] 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]

---

[1] 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

- 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 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 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

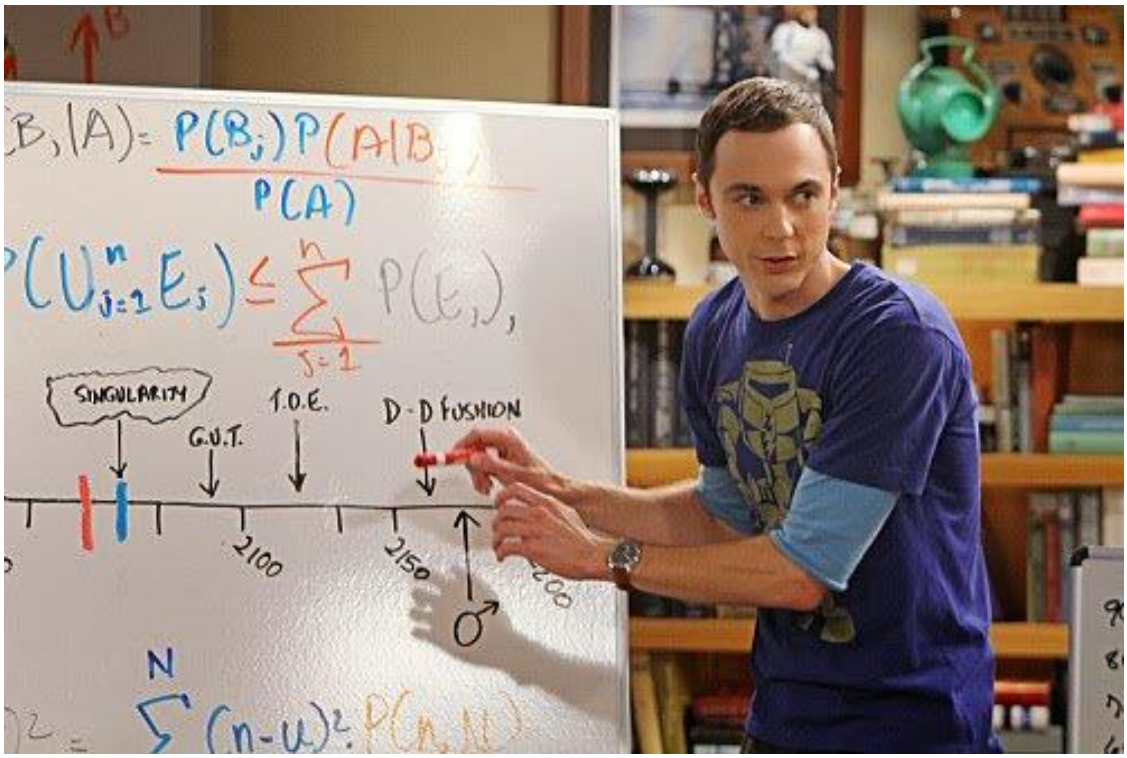
Decision trees are no doubt one of the most popular machine learning algorithms. Thanks to all kinds of software, implementation is a no brainer which requires nearly zero understanding of the mechanism. The followings are some of the common trees:

- 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 individual 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 human champion Ke Jie. 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 [1, 2] and Boosting method [3, 4]. This is a class of powerful and effective algorithms.

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

2. BIG DATA CLOUD PLATFORM

2.1.How Data becomes Science?

Data has been a friend of statistician for hundreds of years. Tabulated data are the most familiar format that we use daily. People used to store data on papers, tapes, diskettes, or hard drives. Only recently, with the development of the computer hardware, software, and algorithms, the volume, variety, and velocity of the data suddenly beyond the capacity of a traditional statistician. And data becomes a special science with the very first focus on a fundamental question: with a huge amount of data, how can we store the data and quick access and process the data. In the past a few years, by utilizing commodity hardware and open source software, a big data analytics ecosystem was created for data storage, data retrieval, and parallel computation. Hadoop and Spark have become a popular platform that enables data scientist, statistician, and business analyst to access the data and to build models. Programming skills in the big data platform have been the largest gap for a statistician to become a successful data scientist. However, with the recent wave of cloud computing environment, this gap is significantly reduced. Many of the technical details have been pushed to the background, and the user interface becomes much easier to learn. Cloud environment also enables quick implementation to the production system. Now data science is emphasis more on the data itself as well as models and algorithms on top of the data instead of platform and infrastructure.

2.2.Power of Cluster of Computers

We are all familiar with our laptop/desktop computers which contain mainly three components to finish computation with data: (1) Hard disk, (2) Memory, and (3) CPU as shown in Figure 2-1 below (left). The data and codes are stored in the hard disk which has certain features such as relatively slow for reading and writes and relatively large capacity of around a few TB in today's market. Memory is relatively fast for reading and writing but relatively small in capacity in the order of a few dozens of GB in today's market. CPU is where all the computation is done.

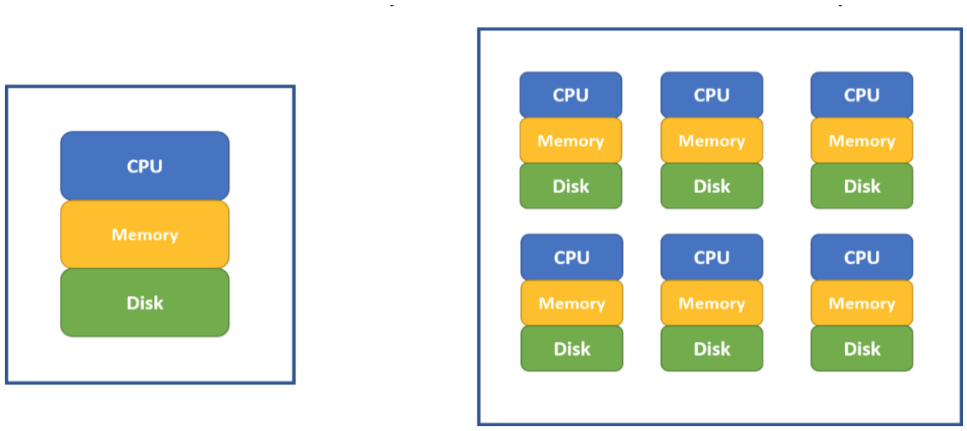


Figure 21 Single Computer vs Cluster of Computers

For statistical software such as R, the amount of data that it can process is limited by the computer's memory. For a typical computer before the year 2000, the memory is less than 1 GB. The memory capacity grows far slower than the availability of the data to analyze. Now it is quite often that we need to analyze data far beyond the capacity of a single computer's memory, especially in an enterprise environment. Meanwhile, the computation time is growing faster than linear to solve the same problem (such as regressions) as the data size increases. Using a cluster of computers become a common way to solve big data problem. In Figure 2-1 above (right), a cluster of computers can be viewed as one powerful machine with total memory, hard disk and CPU equivale to the sum of individual computers. It is common to have thousands of nodes for a cluster.

In the past, to use a cluster of computers, users must write code (such as MPI) to take care of how data is distributed and how the computation is done in a parallel fashion. Luckily with the recent new development, the cloud environment for big data analysis is more user-friendly. As data is typically beyond the size of one hard disk, the dataset itself is stored across different nodes' hard disk (i.e. the Hadoop system mentioned below). When we perform analysis, we can assume the needed data is already distributed across many node's memories in the cluster and algorithms are parallel in nature to leverage corresponding nodes' CPUs to compute (i.e. the Spark system mentioned below).

2.3.Evolution of Clustering Computing

Using computer clusters to solve general purpose data and analytics problems needs a lot of efforts if we have to specifically control every element and steps such as data storage, memory allocation, and parallel computation. Fortunately, high tech IT companies and open source communities have developed the entire ecosystem based on Hadoop and Spark. Users need only to know high-level scripting language such as Python and R to leverage computer clusters' storage, memory and computation power.

2.3.1.Hadoop

The very first problem internet companies face is that a lot of data has been collected and how to better store these data for future analysis. Google developed its own file system to provide efficient, reliable access to data using large clusters of commodity hardware. The open source version is known as Hadoop Distributed File System (HDFS). Both systems use Map-Reduce to allocate computation across computation nodes on top of the file system. Hadoop is written in Java and writing map-reduce job using Java is a direct way to interact with Hadoop which is not familiar to many in the data and analytics community. To help better use Hadoop system, an SQL-like data warehouse system called Hive, and a scripting language for analytics interface called Pig were introduced for people with analytics background to interact with Hadoop system. Within Hive, we can create user defined function through R or Python to leverage the distributed and parallel computing infrastructure. Map-reduce on top of HDFS is the main concept of the Hadoop ecosystem. Each map-reduce operation requires retrieving data from hard disk, then computing and processing, and then storing the result onto disk again. So, jobs on top of Hadoop require a lot of disk operation which may slow down the computation process.

2.3.2.Spark

Spark works on top of distributed file system including HDFS with better data and analytics efficiency by leveraging in-memory operations and is more tailored for data processing and analytics. The Spark system includes an SQL-like framework called Spark SQL and a parallel machine learning library called MLib. Fortunately for many in the analytics community, Spark also supports R and Python. We can interact with data stored in distributed file system using parallel computing across nodes easily with R and Python through the Spark API and do not need to worry about lower level details of distributed computing. We will introduce how to use R notebook to drive Spark computations.

2.4.Introduction of Cloud Environment

There are many cloud computing environments such as Amazon's AWS which provides a complete list of functions for heavy-duty enterprise applications. For example, Netflix runs its business entirely on AWS and Netflix does not own any data centers. For beginners, Databricks provides an easy to use cloud system for learning purpose. Databricks is a company founded by the creators of Apache Spark and it provides a user-friendly web-based notebook environment that can create Hadoop/Spark/GPU cluster on the fly to run R/Python/Scala/SQL. We will use Databricks' community edition to run demos in this book. Please note the content of this section is adopted from the following web pages:

- https://docs.databricks.com/user-guide/faq/sparklyr.html
- http://spark.rstudio.com/index.html

We can follow these a few steps below to get familiar with the Databricks environment:

1. Open Account and Create a Cluster

Anyone can apply for a community edition for free through https://databricks.com/try-databricks and a short YouTube video illustrates the application process can be found through https://youtu.be/vx-3-htFvrg. Another short YouTube video shows how to create a computing cluster in a cloud computing environment and then create an R notebook to run R codes which can be found at https://youtu.be/0HFujX3t6TU. In fact, you can run Python/R/Scala/SQL cells, as well as markdown cells, in the same notebook by including a keyword at the beginning of each cell that we will discuss later.

2. R Notebook

In last section of the video, we created an R notebook. For an R notebook, it contains multiple cells and by default, the content within each cell are R scripts. Usually, each cell is a well-managed segment of a few lines of codes that accomplish a specific task. For example, the Figure below shows the default cell for an R notebook for cell 1. We can type in R scripts and comments same as we are using R console. By default, only the result from the last line will be shown following the cell. However, you can use `print()` function to output results for any lines. If we move the mouse to the middle of the lower edge of the cell below the results, a "+" symbol will show up and click on the symbol will insert a new cell below. When you click

any area within a cell, it will make it editable and you will see a few icons on the top right corn of the cell where you can run the cell, as well as add a cell below or above, copy the cell, cut the cell etc. One quick way to run the cell is Shift+Enter when the cell is chosen. You will become familiar with the notebook environment quickly.

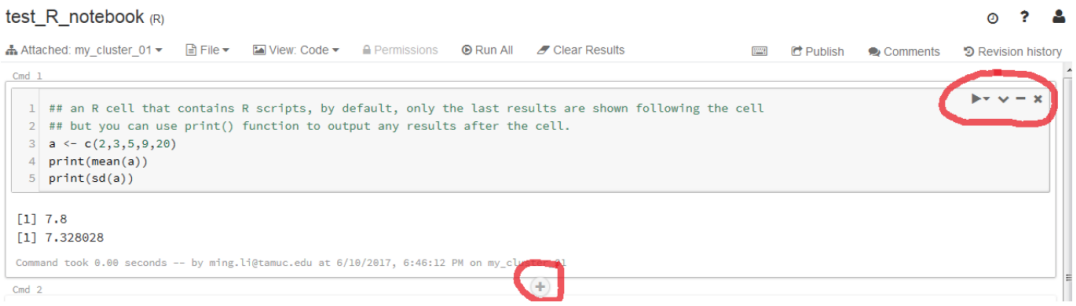


Figure 22: Example of R Notebook

3. Markdown Cells

For an R notebook, every cell by default will contain R scripts. But if we put %md, %sql or %python at the first line of a cell, that cell becomes Markdown cell, SQL script cell, and Python script cell accordingly. For example, Figure below shows a markdown cell with scripts and the actual appearance when exits editing mode. Markdown cell provides a straightforward way to descript what each cell is doing as well as what the entire notebook is about. It is a better way than simple comment within in the code.

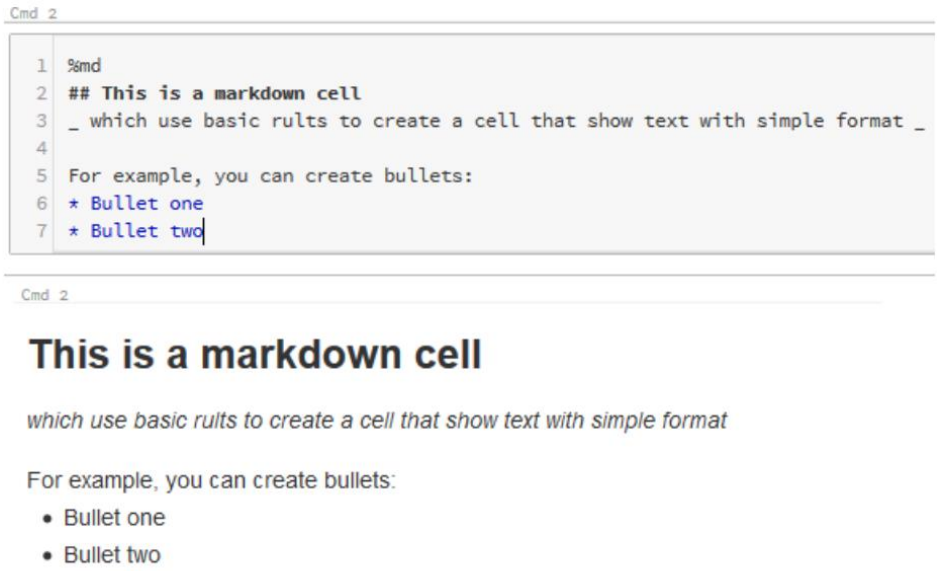


Figure 23 Example of Markdown Cell

2.5.Leverage Hadoop and Spark Using R Notebook

R is a powerful tool for data analysis given the data can be fit into memory. Because of the memory bounded dataset limit, R itself cannot be used directly for big data analysis where the data is likely stored in Hadoop and Spark system. By leverage `sparklyr` package created by RStudio, we can use Databricks' R notebook to analyze data stored in Spark system where the data are stored across different nodes and computation are parallel in nature to use the collection of memory units across all nodes. The fundamental data element in Spark system is called Spark DataFrames. And the process is relatively simple through Spark DataFrames. In this section, we will illustrate how to use Databricks' R notebook for big data analysis on top of Spark environment through `sparklyr` package.

2.5.1.Library Installation

First, we need to install `sparklyr` package which enables the connection between master or local node to Spark cluster environments. As it will install more than 10 dependencies, it may take more than 10 minutes to finish. Be patient while it is installing! Once the installation finishes, load the `sparklyr` package as illustrated by the following code:

```
# Installing sparklyr takes a few minutes,
# because it installs +10 dependencies.

if (!require("sparklyr")) {
  install.packages("sparklyr")
}

# Load sparklyr package.
library(sparklyr)
```

2.5.2.Create Connection

Once the library is loaded, we need to create a Spark Connection to link the computing node (i.e. local node) running the R notebook to the Spark environment. Here we use the "databricks" option for parameter ***method*** which is specific for databricks' system. In the enterprise environment, please consult your administrator for details. The created Spark Connection (i.e. ***sc***) will be the pipe that connects R notebook in local node to the Spark Cluster. We can think of the R notebook is running on a local node which has its memory and CPU; the Spark system has a cluster of connected computation nodes; and the Spark created a pipe to connect both systems. The Spark Connection can be established with:

```
# create a sparklyr connection
sc <- spark_connect(method = "databricks")
```

2.5.3.Sample Dataset

To simplify the learning process, let us use a very familiar dataset: the iris dataset. It is part of the `dplyr` library and let's load that library to use the ***iris*** data frame. Here the ***iris*** dataset

is still on the local node where the R notebook is running on. And we can see that the first a few lines of the ***iris*** dataset below the code after running:

```
library(dplyr)
head(iris)
```

2.5.4.IMPORTANT - Copy Data to Spark Environment

In real applications, your data maybe massive and cannot fit onto a single hard disk and most likely such data are already stored in Spark system. If the data is already in Hadoop/Spark ecosystem, you can leverage Spark DataFrames to analyze it with Spark Connections and R notebook. Here, we illustrate how to copy a local dataset to Spark environment and then work on that dataset in the Spark system. As we have already created the Spark Connection ***sc***, it is easy to copy data to spark system using sdf_copy_to() function as below:

```
iris_tbl <- sdf_copy_to(sc = sc, x = iris, overwrite = T)
```

The above one line code copies iris dataset from the local node to Spark cluster environment. "**sc**" is the Spark Connection we just created; "**x**" is the data frame that we want to copy; "**overwrite**" is the option whether we want to overwrite the target object if the same name Spark DataFrame exists in the Spark environment. Finally, sdf_copy_to() function will return an R object wrapping the copied Spark DataFrame (i.e. creating a "pointer" to the Spark DataFrame such that we can refer ***iris_tbl*** in the R notebook to operate ***iris*** Spark DataFrame). So ***irir_tbl*** can be used to refer to the ***iris*** Spark DataFrame.

To check whether the ***iris*** data was copied to Spark environment successfully or not, we can use `src_tbls( )` function to the Spark Connection (***sc***):

```
src_tbls(sc) ## code to return all the data frames associated with sc
```

2.5.5.Analyzing the Data

Now we have successfully copied the iris dataset to the Spark environment as a Spark DataFrame. This means that ***iris_tbl*** is an R object wrapping the ***iris*** Spark DataFrame and we can use ***iris_tbl*** to refer the ***iris*** dataset in the Spark system (i.e. the ***iris*** Spark DataFrame). With the `sparklyr` packages, we can use many functions in `dplyr` to Spark DataFrame directly through ***iris_tbl***, same as we are applying `dplyr` functions to a local R data frame in our laptop. For example, we can use the `%>%` operator to pass ***iris_tbl*** to the `count( )` function:

```
iris_tbl %>% count
```

or using the `head( )` function to return the first few rows in ***iris_tbl***:

```
head(iris_tbl)
```

or applying more advanced data manipulation directly to ***iris_tbl***:

```
iris_tbl %>%
  mutate(Sepal_Width = ROUND(Sepal_Width * 2) / 2) %>%
  group_by(Species, Sepal_Width) %>%
```

```
  summarize(count = n(), Sepal_Length = mean(Sepal_Length), stdev =
sd(Sepal_Length))
```

2.5.6.Collect Results Back to Master Node

Even though we can run many of the `dplyr` functions on Spark DataFrame, we cannot apply functions from other packages to Spark DataFrame direction (such as `ggplot()`). For functions that can only work on local R data frames, we must copy the Spark DataFrame back to the local node. To copy Spark DataFrame back to the local node, we use the `collect()` function where the argument to it is the name of the Spark DataFrame. The following code `collect()` the results of a few operations and assign the collected data to iris_summary variable:

```
iris_summary <-
  iris_tbl %>%
  mutate(Sepal_Width = ROUND(Sepal_Width * 2) / 2) %>%
  group_by(Species, Sepal_Width) %>%
  summarize(count = n(), Sepal_Length = mean(Sepal_Length), stdev =
sd(Sepal_Length)) %>%
  collect
```

Now, ***iris_summary*** is a local R object to the R notebook and we can use apply R packages and functions to it. In the following code, we will apply `ggplot()` to it, exactly the same as a stand along R console:

```
library(ggplot2)
ggplot(iris_summary, aes(Sepal_Width, Sepal_Length, color = Species)) +
  geom_line(size = 1.2) +
  geom_errorbar(aes(ymin = Sepal_Length - stdev, ymax = Sepal_Length +
stdev), width = 0.05) +
  geom_text(aes(label = count), vjust = -0.2, hjust = 1.2, color = "black") +
  theme(legend.position="top")
```

2.5.7.Fit Regression to SparkDataFrame

One of the largest advantages is that, within Spark system, there are already many statistical and machine learning algorithms developed to run parallelly across many CPUs with data across many memory units through Spark DataFrames. In this example, we have already uploaded the data to Spark system, and the data in the Spark system can be referred through ***iris_tbl***. The linear regression implemented in Spark system can be called through `ml_linear_regression()` function. The syntax to call the function is to define the Spark DataFrame (i.e. ***iris*** through ***iris_tbl***), response variable (i.e. y-variable in linear regression in the Spark DataFrame ***iris*** through ***iris_tbl***) and features (i.e. the x-variables in linear regression in the Spark DataFrame ***iris*** through ***iris_tbl***). So, we can easily fit a linear regression for large dataset far beyond the memory limit of one single computer, and it is truly scalable and only constrained by the resource of the Spark cluster. Below is an illustration of how to fit a linear regression to Spark DataFrame using R notebook:

```
fit1 <-  ml_linear_regression(x = iris_tbl, response = "Sepal_Length",
                              features = c("Sepal_Width", "Petal_Length",
"Petal_Width"))
summary(fit1)
```

In the above code, **x** is the R object pointing to the Spark DataFrame; **response** is y-variable, **features** are the collection of explanatory variables. For this function, both the data and computation are in the Spark cluster which leverages multiple CPUs and distributed memories.

2.5.8.Fit a K-means Cluster

Through the `sparklyr` package, we can use an R notebook to access many Spark Machine Learning Library (MLlib) algorithms such as linear regression, logistic regression, Survival Regression, Generalized Linear Regression, Decision Trees, Random Forests, Gradient-Boosted Trees, Principal Components Analysis, Naive-Bayes, K-Means Clustering and a few other methods. Below codes fit a k-means cluster algorithm:

```
## Now fit a k-means clustering using iris_tbl data
## with only two out of four features in iris_tbl
fit2 <- ml_kmeans(x = iris_tbl, centers = 3,
                  features = c("Petal_Length", "Petal_Width"))

# print our model fit
print(fit2)
```

After the k-means model is fit, we can apply the model to predict other datasets through `sdf_predict()` function. Below code apply the model to ***iris_tbl*** again to predict and then the results are collected back to a local R object (i.e. ***prediction*)** through `collect()` function:

```
prediction = collect(sdf_predict(fit2, iris_tbl))
```

As ***prediction*** is a local R object, we can apply any R functions from any libraries to it. For example:

```
prediction  %>%
  ggplot(aes(Petal_Length, Petal_Width)) +
  geom_point(aes(Petal_Width, Petal_Length, col = factor(prediction + 1)),
             size = 2, alpha = 0.5) +
  geom_point(data = fit2$centers, aes(Petal_Width, Petal_Length),
             col = scales::muted(c("red", "green", "blue")),
             pch = 'x', size = 12) +
  scale_color_discrete(name = "Predicted Cluster",
                       labels = paste("Cluster", 1:3)) +
  labs(
    x = "Petal Length",
    y = "Petal Width",
    title = "K-Means Clustering",
    subtitle = "Use Spark.ML to predict cluster membership with the iris
```

```
dataset."
  )
```

2.5.9.Summary

In the above a few sub-sections, we illustrated

(1) the relationship between local node (i.e. where R notebook is running) and Spark Clusters (i..e where data are stored and computation are done);

(2) how to copy a local data frame to a Spark DataFrames (please note if your data is already in Spark environment, there is no need to copy. This is likely to be the case for enterprise environment);

(3) how to manipulate Spark DataFrames for data cleaning and preprocessing through `dplyr` functions with the installation of `sparklyr` package;

(4) how to fit statistical and machine learning models to Spark DataFrame in a truly parallel manner;

(5) how to collect information from Spark DataFrames back to a local R object (i.e. local R data frame) for future analysis.

These procedures are pretty much covered the basics of big data analysis that a data scientist needs to know. The above steps are published as an R notebook: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2961012104553482/3725396058299890/1806228006848429/latest.html

2.9.Databases and SQL

Databases have been around for many years to efficiently store, organize, retrieve, and update data systematically. In the past, statisticians usually dealt with small datasets stored in Excel files and often did not interact with databases. Students from traditional statistics departments usually lack the needed database knowledge which is essential and required in an enterprise environment where data are stored in some form of database. Databases often contain a collection of tables and the relationship among these tables (i.e. schema). The table is the fundamental structure for databases which contains rows and columns similar to data frames in R or Python pandas. Database management systems (DBMS) ensure data integration and security in real time operations. There are many different DBMS such as Oracle, SQL Server, MySQL, Teradata, Hive, Redshift and Hana. The majority of database operations are very similar among different DBMS, and Structured Query Language (SQL) is the standard language to use these systems.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. The most recent version is published in December 2016. For typical users, the fundamental knowledge is nearly the same. In addition to the standard features, each DBMS providers include their own functions and features. So, for the same query, it may be slightly different implementations (i.e. SQL

script) for different systems. In this section, we use the Databricks' SQL implementation (i.e. all the SQL scripts can run in Databricks SQL notebook).

More recently data is stored in distributed system such as Hive or in-memory such as Hana. Most relational databases are row-based (i.e. data for each row are stored closely), whereas analytics workflows often favor column-based systems (i.e. data for each column are stored closely). Fortunately, as a database user, we only need to learn how to write SQL scripts to retrieve and manipulate data. Even though there are different implantations of SQL across different DBMS, SQL is nearly universal across relational database including Hive and Spark, which means once we know SQL, our knowledge can be transferred among different database systems. SQL is easy to learn even if you do not have previous experience. In this session, we will go over the key concepts in database and SQL. A more detailed description of database basic can be found through a list of YouTube videos using a specific textbook: https://www.youtube.com/playlist?list=PLtqstN-ayEb0H5AAo6_V5qEzWs0D-igpw

2.9.1.Database, Table and View

A database is a collection of tables that are related to each other. A database has its own database name and each table has its name as well. We can think database is a "folder" where tables within a database are "files" within the folder. A table has rows and columns exactly as an R or pandas data frame. Each row (also called record) represents a unique instance of the subject and each column (also called field or attribute) represents a characteristic of the subject on the table. For each table, there is a special column called primary key which uniquely identifies each of its record.

Tables within a specific database contains related information and the schema of a database illustrates all fields in every table as well as how these tables and fields relate to each other. Tables can be joined and aggregated to return specific information. View is a virtual table composed of fields from one or more base tables. View does not store data, and only store table structure. View is also referred as a saved query. View is typically used to protect the data stored in the table and users can only query information from a view and cannot change or update its contents.

2.9.2.Sample Tables

We will use two simple tables to illustrate basic SQL operations. These two tables are from an R dataset which contains the 50 states' population and income. The first table is called "**divisions**" which has two columns: state and division and the first few rows are shown in the following table:

state	division
Alabama	East South Central
Alaska	Pacific
Arizona	Mountain
Arkansas	West South Central
California	Pacific

The second table is called **metrics** which contains three columns: state, population and income and first few rows of the table is shown below:

state	population	income
Alabama	3615	3624
Alaska	365	6315
Arizona	2212	4530
Arkansas	2110	3378
California	21198	5114

To illustrate missing information, three more rows are added at the end of the original division table with state Alberta, Ontario, and Quebec with their corresponding division NULL.

Please watch the following YouTube video on how to upload a .csv file to a Databricks table: https://youtu.be/H5LxjaJgpSk

2.9.3.Basic SQL Statement

After logging into Databricks and creating two tables, you can now create a notebook and choose the type of the notebook to be SQL. There are a few very easy SQL statements to help us understand the database and table structure:

- `show database`: show current databases in the system
- `create database db_name`: create a new database with name db_name

- `drop database db_name`: delete database db_name (be careful when using it!!)
- `use db_name`: set up the current database to be used
- `show tables`: show all the tables within the currently used database
- `describe tbl_name`: show the structure of table with name tbl_name (i.e. list of column name and data type)
- `drop tbl_name`: delete a table with name tbl_name (be careful when using it!!)
- `select * from metrics limit 10`: show the first 10 rows of a table

If you are familiar with a procedural programming language such as C and FORTRN or scripting languages such as R and Python, you may find SQL code a little bit strange. We should view SQL code by each specific chunk where it finishes a specific task. SQL codes descript a specific task and DBMS will run and finish the task. SQL does not follow typical procedure program rules and we can think SQL is "descriptive" (i.e. we describe what we want using SQL and DBMS figures out how to do it).

2.9.4.Simple SELECT Statement

SELECT is the most used statements in SQL, especially for database users and business analyst. It is used to extract specific information (i.e. column or columns) FROM one or multiple tables. It can be used to combine multiple tables. WHERE can be used in SELECT statement to selected rows with specific conditions. ORDER BY can be used in SELECT statement to order the results in descending or ascending order. We can use * after SELECT to represent all columns in the table. Below is the basic structure of a SELECT statement:

```
SELECT Col_Name1, Col_Name2
FROM Table_Name
WHERE Specific_Condition
ORDER BY Col_Name1, Col_Name2;
```

Here specific_condition is the typical logical conditions and only columns with TRUE for this condition will be chosen. For example, if we want to choose states and its total income where the population larger than 10000 and income less than 5000 with the result order by state name, we can use the following query

```
select state, income*population as total_income
from metrics
where population > 10000 and income < 5000
order by state
```

The SELECT statement is used to slicing and dicing the dataset as well as create new columns of interest using basic computation functions.

2.9.5.Aggregation Functions and GROUP BY

We can also use aggregation functions in SELECT statement to summarize the data. For example, count(col_name) function will return the total number of not NULL rows for a specific column. Other aggregation function on numerical values include min(col_name),

max(col_name), avg(col_name). Let's use **metrics** table again to illustrate aggregation functions. For aggregation function, it takes all the rows that match WHERE condition (if any) and return one number. The following statement will calculate the maximum, minimum, and average population for all states starts with letter A to E.

```
select sum(population) as sum_pop, max(population) as
    max_pop, min(population) as min_pop, avg(population)
    as avg_pop, count(population) as count_pop
from metrics
where substring(state, 1, 1) in ('A', 'B', 'C', 'D', 'E')
```

The results from the above query only return one row as expected. Sometimes we want to find the aggregated value based on groups that can be defined by one or more columns. Instead of writing multiple SQL to calculate the aggregated value for each group, we can easily use the GROUP BY to calculate the aggregated value for each group in SELECT statement. For example, if we want of find how many states in each division, we can use the following:

```
select division, count(state) as number_of_states
from divisions
group by division
```

Another special aggregation function is to return distinct values for one column or a combination of multiple columns. Simple use SELECT DISTINCT col_name1, col_name2 in the first line of the SELECT statement.

2.9.6.Join Multiple Tables

The database system is usually designed such that each table contains a piece of specific information and oftentimes we need to JOIN multiple tables to achieve a specific task. There are few types typically JOINs: inner join (keep only rows that match the join condition from both tables), left outer join (rows from inner join + unmatched rows from the first table), right outer join (rows from inner join + unmatched rows from the second table) and full outer join (rows from inner join + unmatched rows from both tables). The typical JOIN statement is illustrated below:

```
SELECT a.col_name1 as var1, b.col_name2 as var2
FROM tbl_one as a
INNER/LEFT JOIN tabl_two ad b
ON a.col_to_match = b.col_to_match
```

For example, lets join the division table and metrics table to find what is the average population and income for each division, and the results order by division names:

```
select a.division, avg(b.population) as avg_pop,
     avg(b.income) as avg_inc
from divisions as a
inner join metrics as b
on a.state = b.state
```

```
group by division
order by division
```

2.9.7.Add More Content into a Table

We can use INSERT statement to add additional rows into a particular table, for example, we can add one more row to the metrics table by using the following query:

```
insert into metrics
values ('Alberta', 4146, 7370)
```

2.9.8.Advanced Topics in Database

Database management is a specific research area and there are many advanced topics such as how to efficiently query data using index; how to take care of data integrity when multiple users are using the same table; algorithm behind data storage (i.e. column-wise or row-wise data storage); how to design the database schema. Users can learn these advanced topics gradually. We hope the basic knowledge covered in this section will kick off the initial momentum to learn SQL. As you can see, it is fairly easy to write SQL statement to retrieve, join, slice, dice and aggregate data. All the SQL scripts can be found in this notebook: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2961012104553482/4213233595765185/1806228006848429/latest.html

2.10. Other Useful Topics

For data scientist, in addition to the above-mentioned areas, the following topics are also very important to get some exposure.

2.10.1. Linux Operation System

Many of the cloud environment, servers, and production systems are usually running on top of Linux operation system and some basic understanding of Linux is essential to solving various problems in a data science project. Linux system is a multiple-user system that runs robustly without interrupt for months. For a typical user, we can access to some of the functions through a commend-line type terminal. Here is a list of commonly used command:

- `ls` : show files in current directory
- `pwd` : display current directory and path
- `mkdir dir_name` : create a new directory
- `cd dir_path`: change directory through its path
- `cd ..` : go one directory level up
- `cp file1 file2` : copy file1 to file2
- `mv file1 file2`: rename file1 to file2
- `head file` : show the first a few rows of file

- `tail file` : show last a few rows of file
- `top` : show current running job
- `who` : list all users that log in the system

There are many Linux training material available online. Once you have a need to learn Linux, you can always learn by yourself through these online training materials. The Linux system will be configured by your system administrator, please always ask your colleague and system administrator for suggestions. Some of the commands may knock the entire system down or permanently delete useful information, please be very careful and never try any commands that you do not know exactly the consequence. There are horror stories about deleting files or taking down systems caused by misusing Linux commands or SQL scripts.

2.10.2. Visualization

R and Python both provide nice visualization capability. And you can combine `shiny` and `flexdashboard` packages to build a dynamic and interactive dashboard using RStudio. However, in a business environment, Tableau is still the most used dashboard visualization system. More recently, HTML5 and D3 have become popular for data visualization. For a successful data scientist, we need to have our own recommendation of what types of visualization are useful. We may not need to implement these dashboard systems on our own, but we need to at least guide the team that manages these systems to implement.

2.10.3. GPU

Many of the machine learning methods are based on linear algebra, especially deep learning algorithms. The CPU is not designed to handle large-size matrix linear algebra and using a GPU is an efficient alternative for matrix-based linear algebra computation. In Databricks Community Edition, we can also create a GPU machine to use. If you are interested in deep learning using Spark and GPU through Databrick, please watch this video for more detail: https://databricks.com/blog/2016/10/27/gpu-acceleration-in-databricks.html

3. INTRODUCTION TO THE DATA

Before tackling analytics problem, we start by creating data to be analyzed in later chapters. Why do we simulate data instead of using real data set?

- Going through the simulation code helps you practice R skills.
- It makes the book less dependent on downloading online data set
- It allows you to manipulate the synthetic data, run analysis and examine how the results change
- The authors do not need to worry about intellectual property rights.

3.1.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

The simulation is not very straightforward and I will break it into three parts:

1. Define data structure: variable names, variable distribution, customer segment names, segment size
2. Variable distribution parameters: mean and variance
3. Iterate across segments and variables. Simulate data according to specific parameters assigned

By organizing code this way, it makes easy for us to change specific parts of the simulation. For example, if we want to change the distribution of one variable, we can just change the corresponding part of the code.

Here is code to define data structure:

```
# set a random number seed to make the process repeatable
set.seed(12345)
# define the number of observations
ncust<-1000
# create a data frmae for simulated data
seg_dat<-data.frame(id=as.factor(c(1:ncust)))
# assign the variable names
vars<-
c("age","gender","income","house","store_exp","online_exp","store_trans","onl
ine_trans")
# assign distribution for each variable
vartype<-c("norm","binom","norm","binom","norm","norm","pois","pois")
# names of 4 segments
group_name<-c("Price","Conspicuous","Quality","Style")
# size of each segments
group_size<-c(250,200,200,350)
```

The next step is to define variable distribution parameters. There are 4 segments of customers and 8 parameters. Different segments correspond to different parameters. Let's store the parameters in a 4×8 matrix:

```
# matrix for mean
mus <- matrix( c(
  # Price
  60, 0.5, 120000,0.9, 500,200,5,2,
  # Conspicuous
  40, 0.7, 200000,0.9, 5000,5000,10,10,
  # Quality
  36, 0.5, 70000, 0.4, 300, 2000,2,15,
  # Style
  25, 0.2, 90000, 0.2, 200, 2000,2,20), ncol=length(vars), byrow=TRUE)

# matrix for variance
sds<- matrix( c(
  # Price
  3,NA,8000,NA,100,50,NA,NA,
  # Conspicuous
  5,NA,50000,NA,1000,1500,NA,NA,
  # Quality
  7,NA,10000,NA,50,200,NA,NA,
  # Style
  2,NA,5000,NA,10,500,NA,NA), ncol=length(vars), byrow=TRUE)
```

Now we are ready to simulate data using the parameters defined above:

```
# simulate non-survey data
sim.dat<-NULL
set.seed(2016)
# loop on customer segment (i)
 for (i in seq_along(group_name)){

   # add this line in order to moniter the process
   cat (i, group_name[i],"\n")

  # create an empty matrix to store relevent data
  seg<-data.frame(matrix(NA,nrow=group_size[i], ncol=length(vars)))

  # Simulate data within segment i
  for (j in seq_along(vars)){

    # loop on every variable (j)
    if (vartype[j]=="norm"){
      # simulate normal distribution
      seg[,j]<-rnorm(group_size[i], mean=mus[i,j], sd=sds[i,j])
    } else if (vartype[j]=="pois") {
      # simulate poisson distribution
      seg[,j]<-rpois(group_size[i], lambda=mus[i,j])
```

```
    } else if (vartype[j]=="binom"){
      # simulate binomial distribution
      seg[,j]<-rbinom(group_size[i],size=1,prob=mus[i,j])
    } else{
      # if the distribution name is not one of the above, stop and return a
message
      stop ("Don't have type:",vartype[j])
    }
  }
  sim.dat<-rbind(sim.dat,seg)
 }
```

Now let's edit the data we just simulated a little by adding tags to 0/1 binomial variables:

```
# assign variable names
names(sim.dat)<-vars
# assign factor levels to segment variable
sim.dat$segment<-factor(rep(group_name,times=group_size))
# recode gender and house variable
sim.dat$gender<-factor(sim.dat$gender, labels=c("Female","Male"))
sim.dat$house<-factor(sim.dat$house, labels=c("No","Yes"))
# store_trans and online_trans are at least 1
sim.dat$store_trans<-sim.dat$store_trans+1
sim.dat$online_trans<-sim.dat$online_trans+1
# age is integer
sim.dat$age<-floor(sim.dat$age)
```

In the real world, the data always includes some noise such as missing, wrong imputation. So we will add some noise to the data:

```
# add missing values
idxm <- as.logical(rbinom(ncust, size=1, prob=sim.dat$age/200))
sim.dat$income[idxm]<-NA
# add wrong imputations and outliers
set.seed(123)
idx<-sample(1:ncust,5)
sim.dat$age[idx[1]]<-300
sim.dat$store_exp[idx[2]]<- -500
sim.dat$store_exp[idx[3:5]]<-c(50000,30000,30000)
```

So far we have created part of the data. You can check it using `summary(sim.dat).' Next, we will move on to simulate survey data.

```
# number of survey questions
nq<-10
# mean matrix for different segments
mus2 <- matrix( c(
  # Price
 5,2,1,3,1,4,1,4,2,4,
  # Conspicuous
 1,4,5,4,4,4,4,1,4,2,
```

```
  # Quality
 5,2,3,4,3,2,4,2,3,3,
  # Style
 3,1,1,2,4,1,5,3,4,2), ncol=nq, byrow=TRUE)

# assume the variance is 0.2 for all
sd2<-0.2
sim.dat2<-NULL
set.seed(1000)
# loop for customer segment (i)
for (i in seq_along(group_name)){
  # the following line is used for checking the progress
  # cat (i, group_name[i],"\n")
  # create an empty data frame to store data
  seg<-data.frame(matrix(NA,nrow=group_size[i], ncol=nq))
  # simulate data within segment
  for (j in 1:nq){
    # simulate normal distribution
    res<-rnorm(group_size[i], mean=mus2[i,j], sd=sd2)
    # set upper and lower limit
    res[res>5]<-5
    res[res<1]<-1
    # convert continuous values to discrete integers
    seg[,j]<-floor(res)
  }
  sim.dat2<-rbind(sim.dat2,seg)
}

names(sim.dat2)<-paste("Q",1:10,sep="")
sim.dat<-cbind(sim.dat,sim.dat2)
sim.dat$segment<-factor(rep(group_name,times=group_size))
```

So far we have gotten all the data. Let's check it:

```
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 ...
```

3.2.Customer Satisfaction Survey Data from Airline Company

We will simulate 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）

```
# Create a matrix of factor loadings
# This pattern is called bifactor because it has a general factor for
separate components.
# For example, "Ease of making reservation" has general factor loading 0.33,
```

```
specific factor loading 0.58
# The outcome variables are formed as combinations of these general and
specific factors

loadings <- matrix(c (
  # Ticketing
  .33, .58, .00, .00,  # Ease of making reservation
  .35, .55, .00, .00,  # Availability of preferred seats
  .30, .52, .00, .00,  # Variety of flight options
  .40, .50, .00, .00,  # Ticket prices
  # Aircraft
  .50, .00, .55, .00,  # Seat comfort
  .41, .00, .51, .00,  # Roominess of seat area
  .45, .00, .57, .00,  # Availability of Overhead
  .32, .00, .54, .00,  # Cleanliness of aircraft
  # Service
  .35, .00, .00, .50,  # Courtesy of flight attendant
  .38, .00, .00, .57,  # Friendliness
  .60, .00, .00, .50,  # Helpfulness
  .52, .00, .00, .58,  # Food and drinks
  # General
  .43, .10, .30, .30,  # Overall satisfaction
  .35, .50, .40, .20,  # Purchase again
  .25, .50, .50, .20), # Willingness to recommend
  nrow=15,ncol=4, byrow=TRUE)

# Matrix multiplication produces the correlation matrix except for the
diagonal
cor_matrix<-loadings %*% t(loadings)
# Diagonal set to ones
diag(cor_matrix)<-1

# use the mvtnorm package to randomly generate a data set with a given
correlation pattern

library(mvtnorm)
# mean vectors of the 3 airline companies
mu1=c(5,6,5,6, 7,8,6,7, 5,5,5,5, 6,6,6)
mu2=c(3,3,2,3, 5,4,5,6, 8,8,8,8, 3,3,3)
mu3=c(2,2,2,2, 8,8,8,8, 8,8,8,8, 8,8,8)

# set random seed
set.seed(123456)
# respondent ID
resp.id <- 1:1000

library(MASS)
rating1 <- mvrnorm(length(resp.id),
                   mu=mu1,
```

```
                      Sigma=cor_matrix)
rating2 <- mvrnorm(length(resp.id),
                   mu=mu2,
                   Sigma=cor_matrix)
rating3 <- mvrnorm(length(resp.id),
                   mu=mu3,
                   Sigma=cor_matrix)


# truncates scale to be between 1 and 9
rating1[rating1>9]<-9
rating1[rating1<1]<-1
rating2[rating2>9]<-9
rating2[rating2<1]<-1
rating3[rating3>9]<-9
rating3[rating3<1]<-1

# Round to single digit
rating1<-data.frame(round(rating1,0))
rating2<-data.frame(round(rating2,0))
rating3<-data.frame(round(rating3,0))
rating1$ID<-resp.id
rating2$ID<-resp.id
rating3$ID<-resp.id
rating1$Airline<-rep("AirlineCo.1",length(resp.id))
rating2$Airline<-rep("AirlineCo.2",length(resp.id))
rating3$Airline<-rep("AirlineCo.3",length(resp.id))
rating<-rbind(rating1,rating2,rating3)

# assign names to the variables in the data frame
names(rating)<-c(
  "Easy_Reservation",
  "Preferred_Seats",
  "Flight_Options",
  "Ticket_Prices",
  "Seat_Comfort",
  "Seat_Roominess",
  "Overhead_Storage",
  "Clean_Aircraft",
  "Courtesy",
  "Friendliness",
  "Helpfulness",
  "Service",
  "Satisfaction",
  "Fly_Again",
  "Recommend",
  "ID",
  "Airline")
```

Now check the data frame we have:

```r
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" ...
```

3.3.Swine Disease Breakout Data

In this section, we are going to simulate a data set about swine disease. We simulate 800 farms (i.e. n=800) and 120 survey questions (i.e. G=120) in each data set. There are three possible answers for each question. The outbreak status for the i^{th} farm is generated from a $Bernoulli(1, p_i)$ distribution with p_i being a function of the question answers:

$$ln(\frac{p_i}{1-p_i}) = \beta_0 + \Sigma_{g=1}^{G} \mathbf{x_{i,g}^T \beta_g}$$

where β_0 is the intercept, $\mathbf{x_{i,g}}$ is a three-dimensional indication vector for question answer and β_g is the parameter vector corresponding to the g^{th} predictor. Three types of questions are considered regarding their effects on the outcome. The first forty survey questions are important questions such that the coefficients of the three answers to these questions are all different:

$$\boldsymbol{\beta_g} = (1,0,-1)\times\gamma, g = 1, \dots, 40$$

The second forty survey questions are also important questions but only one answer has a coefficient that is different from the other two answers:

$$\boldsymbol{\beta_g} = (1,0,0)\times\gamma, g = 41, \dots, 80$$

The last forty survey questions are also unimportant questions such that all three answers have the same coefficients:

$$\boldsymbol{\beta_g} = (0,0,0)\times\gamma, g = 81, \dots, 120$$

The baseline coefficient β_0 is set to be $-\frac{40}{3}\gamma$ so that on average a farm has 50 of chance to have an outbreak. The parameter γ in the above simulation is set to control the strength of the questions' effect on the outcome. In this simulation study, we consider the situations where $\gamma = 0.1, 0.25, 0.5, 1, 2$. So the parameter settings are:

$$\boldsymbol{\beta}^{\mathbf{T}} = \left(\frac{40}{3}, \underbrace{1,0,-1,\ldots,}_{question1} \underbrace{1,0,-1}_{question40}, \underbrace{1,0,0}_{question41}, \ldots, \underbrace{1,0,0}_{question80}, \underbrace{0,0,0}_{question81}, \ldots, \underbrace{0,0,0}_{question120} \right) * \gamma$$

```
# sim1_da1.csv  the 1st simulated data
# similar sim1_da2 and sim1_da3
# sim1.csv  simulated data, the first simulation
# dummy.sim1.csv dummy variables for the first simulated data with all the
baseline in
#code for simulation

# setwd(dirname(file.choose()))
# library(grplasso)

nf<-800
for (j in 1:20){
set.seed(19870+j)
x<-c('A','B','C')
sim.da1<-NULL
for (i in 1:nf){
# sample(x, 120, replace=TRUE)->sam
sim.da1<-rbind(sim.da1,sample(x, 120, replace=TRUE))
}

data.frame(sim.da1)->sim.da1
paste("Q", 1:120, sep = "")->col
paste("Farm", 1:nf, sep = "")->row
colnames(sim.da1)<-col
rownames(sim.da1)<-row

# use class.ind() function in nnet package to encode dummy variables
library(nnet)
dummy.sim1<-NULL
for (k in 1:ncol(sim.da1)) {
tmp=class.ind(sim.da1[,k])
colnames(tmp)=paste(col[k],colnames(tmp))
dummy.sim1=cbind(dummy.sim1,tmp)
}
data.frame(dummy.sim1)->dummy.sim1

# set "C" as the baseline
# delete baseline dummy variable
```

```
base.idx<-3*c(1:120)
dummy1<-dummy.sim1[,-base.idx]

# simulate independent variable for different values of r
# simulate based on one value of r each time
# r=0.1, get the link function
c(rep(c(1/10,0,-1/10),40),rep(c(1/10,0,0),40),rep(c(0,0,0),40))->s1
as.matrix(dummy.sim1)%*%s1-40/3/10->link1

# r=0.25
# c(rep(c(1/4,0,-1/4),40),rep(c(1/4,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3/4->link1

# r=0.5
# c(rep(c(1/2,0,-1/2),40),rep(c(1/2,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3/2->link1

# r=1
# c(rep(c(1,0,-1),40),rep(c(1,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3->link1

# r=2
# c(rep(c(2,0,-2),40),rep(c(2,0,0),40),rep(c(0,0,0),40))->s1
# as.matrix(dummy.sim1)%*%s1-40/3/0.5->link1


# calculate the outbreak probability
exp(link1)/(exp(link1)+1)->hp1

# based on the probability hp1, simulate response variable: res
res<-rep(9,nf)
for (i in 1:nf){
sample( c(1,0),1,prob=c(hp1[i],1-hp1[i]))->res[i]
}

# da1 with response variable, without group indicator
# da2 without response variable, with group indicator
# da3 without response variable, without group indicator

dummy1$y<-res
da1<-dummy1
y<-da1$y
ind<-NULL
for (i in 1:120){
c(ind,rep(i,2))->ind
}

da2<-rbind(da1[,1:240],ind)
da3<-da1[,1:240]
```

```
# save simulated data
write.csv(da1,paste('sim',j,'_da',1,'.csv',sep=''),row.names=F)
write.csv(da2,paste('sim',j,'_da',2,'.csv',sep=''),row.names=F)
write.csv(da3,paste('sim',j,'_da',3,'.csv',sep=''),row.names=F)
write.csv(sim.da1,paste('sim',j,'.csv',sep=''),row.names=F)
write.csv(dummy.sim1,paste('dummy.sim',j,'.csv',sep=''),row.names=F)
}
```

For each value of γ, 20 data sets are simulated. The bigger γ is, the larger the corresponding parameter. We provided the data sets with $\gamma = 2$. Let's check the data:

```
disease_dat<-
read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master
/Data/sim1_da1.csv")
#       only       show       the       last       7       columns       here
head(subset(disease_dat,select=c(  "Q118.A","Q118.B","Q119.A","Q119.B","Q120.A
","Q120.B","y")))
```

```
##   Q118.A Q118.B Q119.A Q119.B Q120.A Q120.B y
## 1      1      0      0      0      0      1 1
## 2      0      1      0      1      0      0 1
## 3      1      0      0      0      1      0 1
## 4      1      0      0      0      0      1 1
## 5      1      0      0      0      1      0 0
## 6      1      0      0      1      1      0 1
```

Here `y` indicates the outbreak situation of the farms. `y=1` means there is an outbreak in 5 years after the survey. The rest columns indicate survey responses. For example `Q120.A = 1` means the respondent chose `A` in Q120. We consider `C` as the baseline.

4. DATA WRANGLING

This chapter focuses on some of the most frequently used data manipulations and shows how to implement them in R. It is important to explore the data set 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.

Here we assume the readers are already familiar with some of the traditional R data operations, such as subsetting data frame, deleting variables, read and write functions (`read.csv ()`, `write.csv ()`, etc.) in base R. We will also skip some basic descriptive functions in R. For example, for discrete variables, we often use the frequency table to look at the frequency (`table ()`) of the variable at various levels as needed, or a crosstab of two variables. You can also draw a bar chart for discrete variables (`bar()`). For continuous variables, we need to look at the mean (`mean ()`), standard deviation (`sd()`), quantile (`quantile()`) of a variable from time to time. There are also functions like `summary()`, `str()` and `describe()` (a functions in the 'psych' package) that give a summary of a data frame.

The focus here is to introduce some of the more efficient data wrangling methods in R.

4.1.Read and write data

4.1.1. 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:

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

##   2015 2016 2017
## 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. Without

"progress bar" might be the No.1 reason that people break up with the one they have been dating.

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/SegData.csv ")
head(sim.dat)
```

```
##   age gender   income house store_exp online_exp store_trans online_trans
## 1  57 Female 120963.4   Yes  529.1344   303.5125           2            2
## 2  63 Female 122008.1   Yes  478.0058   109.5297           4            2
## 3  59   Male 114202.3   Yes  490.8107   279.2496           7            2
## 4  60   Male 113616.3   Yes  347.8090   141.6698          10            2
## 5  51   Male 124252.6   Yes  379.6259   112.2372           4            4
## 6  59   Male 107661.5   Yes  338.3154   195.6870           4            5
##   Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 segment
## 1  4  2  1  2  1  4  1  4  2   4   Price
## 2  4  1  1  2  1  4  1  4  1   4   Price
## 3  5  2  1  2  1  4  1  4  1   4   Price
## 4  5  2  1  3  1  4  1  4  2   4   Price
## 5  4  1  1  3  1  4  1  4  2   4   Price
## 6  4  2  1  2  1  4  1  4  1   4   Price
```

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)
```

```
##     2015    2016 2017
## 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)
```

```
##        Date   Food            Mood
## 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)
```

```
##         X1     X2              X3
## 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_names=FALSE)
print(dat)
```

```
##         X1     X2              X3
## 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)
```

```
##      X1  X2 X3 X4   X5   X6 X7 X8 X9  X10
## 1 every man is  a poet when he is in love
```

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

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

```
##    X1         X2         X3 X4   X5
## 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:

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

```
##   Q1 Q2   Q3
## 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:

```
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
- `rio` package：https://github.com/leeper/rio

4.1.2. `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

I 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/Data/SegData.csv")
```

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

```
##   age gender   income house store_exp online_exp store_trans online_trans
## 1  57 Female 120963.4   Yes  529.1344   303.5125           2            2
## 2  63 Female 122008.1   Yes  478.0058   109.5297           4            2
##   Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 segment
## 1  4  2  1  2  1  4  1  4  2   4   Price
## 2  4  1  1  2  1  4  1  4  1   4   Price
```

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

```
##     income store_exp
## 1 120963.4  529.1344
## 2 122008.1  478.0058
```

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

```
##     age gender   income house store_exp online_exp store_trans
## 288 300   Male 208017.5   Yes  5076.801   6053.485          12
##     online_trans Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10     segment
## 288           11  1  4  5  4  4  4  4  1  4   2 Conspicuous
```

```
# 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]
```

```
##       income house
## 204 119552.0    No
## 288 208017.5   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:

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

```
##   Female     Male
## 15.38448 11.26233
```

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

```
##   gender online_trans
## 1 Female     15.38448
## 2   Male     11.26233
```

```
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         15.38448
## 2   Male         11.26233
```

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:

```
# 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:

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

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

Calculate mean for counts of online transactions:

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

```
## [1] 13.546
```

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

```
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:

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

```
##    gender       V1
## 1: Female 15.38448
## 2:   Male 11.26233
```

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

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

```
##    gender house        V1
## 1: Female   Yes 11.312030
## 2:   Male   Yes  8.771523
## 3: Female    No 19.145833
## 4:   Male    No 16.486111
```

Assign column names for aggregated variables:

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

```
##    gender house       avg
## 1: Female   Yes 11.312030
## 2:   Male   Yes  8.771523
## 3: Female    No 19.145833
## 4:   Male    No 16.486111
```

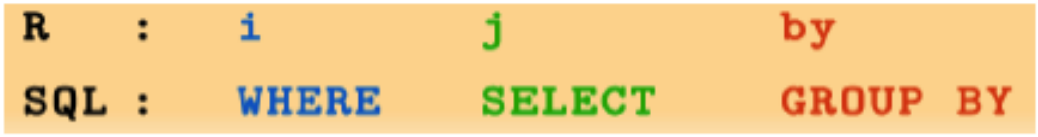
`data.table` can accomplish all operations that `aggregate()` and `tapply()`can do for data frame.

- General setting of `data.table`

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

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.



Let's review our previous code:

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

```
##    gender       V1
## 1: Female 15.38448
## 2:   Male 11.26233
```

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 11.312030
## 2:   Male   Yes  8.771523
## 3: Female    No 19.145833
## 4:   Male    No 16.486111
```

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 14.45977
## 2: Female   Yes 18.14062
## 3:   Male    No 18.24299
## 4: Female    No 20.10196
```

is equal to SQL:

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

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 store_trans online_trans
## 1:  19 Female 83534.70    No  227.6686   1490.719           1           22
## 2:  18 Female 89415.97   Yes  209.5487   1926.470           3           28
## 3:  19 Female 92812.81    No  186.7475   1041.539           2           18
##    Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 segment
## 1:  2  1  1  2  4  1  4  2  4   1   Style
## 2:  2  1  1  1  4  1  4  2  4   1   Style
## 3:  3  1  1  2  4  1  4  3  4   1   Style
```

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

```
##    age gender   income house store_exp online_exp store_trans online_trans
## 1:  57 Female 120963.4   Yes  529.1344   303.5125           2            2
## 2:  63 Female 122008.1   Yes  478.0058   109.5297           4            2
##    Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 segment
## 1:  4  2  1  2  1  4  1  4  2   4   Price
## 2:  4  1  1  2  1  4  1  4  1   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   303.5125
## 2:  63   109.5297
## 3:  59   279.2496
## 4:  60   141.6698
## 5:  51   112.2372
## 6:  59   195.6870
```

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 120963.4
## 2:  63 Female 122008.1
```

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 store_trans online_trans
## 1:  40 Female 217599.7    No  7023.684   9479.442          10            6
## 2:  41 Female       NA   Yes  3786.740   8638.239          14           10
## 3:  36   Male 228550.1   Yes  3279.621   8220.555           8           12
## 4:  31 Female 159508.1   Yes  5177.081   8005.932          11           13
## 5:  43 Female 190407.4   Yes  4694.922   7875.562           6           11
##    Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10     segment
## 1:  1  4  5  4  3  4  4  1  4   2 Conspicuous
## 2:  1  4  4  4  4  4  4  1  4   2 Conspicuous
```

```
## 3:  1  4  5  4  4  4  4  1  4   1 Conspicuous
## 4:  1  4  4  4  4  4  4  1  4   2 Conspicuous
## 5:  1  4  5  4  4  4  4  1  4   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 store_trans online_trans
## 1:  40 Female 217599.7    No  7023.684   9479.442          10            6
## 2:  41 Female       NA   Yes  3786.740   8638.239          14           10
## 3:  36   Male 228550.1   Yes  3279.621   8220.555           8           12
## 4:  31 Female 159508.1   Yes  5177.081   8005.932          11           13
## 5:  43 Female 190407.4   Yes  4694.922   7875.562           6           11
##    Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10     segment
## 1:  1  4  5  4  3  4  4  1  4   2 Conspicuous
## 2:  1  4  4  4  4  4  4  1  4   2 Conspicuous
## 3:  1  4  5  4  4  4  4  1  4   1 Conspicuous
## 4:  1  4  4  4  4  4  4  1  4   2 Conspicuous
## 5:  1  4  5  4  4  4  4  1  4   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 store_trans online_trans
## 1:  40 Female 217599.7    No  7023.684   9479.442          10            6
## 2:  41 Female       NA   Yes  3786.740   8638.239          14           10
## 3:  31 Female 159508.1   Yes  5177.081   8005.932          11           13
## 4:  43 Female 190407.4   Yes  4694.922   7875.562           6           11
## 5:  50 Female 263858.0   Yes  5813.802   7448.729          11           11
##    Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10     segment
## 1:  1  4  5  4  3  4  4  1  4   2 Conspicuous
## 2:  1  4  4  4  4  4  4  1  4   2 Conspicuous
## 3:  1  4  4  4  4  4  4  1  4   2 Conspicuous
## 4:  1  4  5  4  4  4  4  1  4   2 Conspicuous
## 5:  1  4  5  4  4  4  4  1  4   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/master
/Data/topic.csv"))
```

```
  user  system elapsed
  4.313   0.027   4.340

system.time(topic<-
readr::read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR
/master/Data/topic.csv"))

   user  system elapsed
  0.267   0.008   0.274

system.time(topic<-
data.table::fread("https://raw.githubusercontent.com/happyrabbit/DataScientis
tR/master/Data/topic.csv"))

   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.

4.2.Summarize data

4.2.1. `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 data set.

```
# Read data
sim.dat<-
read.csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR/master
/Data/SegData.csv")
# 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:

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

```
##          age       income    store_exp   online_exp  store_trans
##       38.840   113543.065     1356.851     2120.181        5.350
## online_trans           Q1           Q2           Q3           Q4
##       13.546        3.101        1.823        1.992        2.763
##           Q5           Q6           Q7           Q8           Q9
##        2.945        2.448        3.434        2.396        3.085
##          Q10
##        2.320
```

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 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()`:

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

```
##          age       income    store_exp   online_exp  store_trans
##    16.416818 49842.287197  2774.399785  1731.224308     3.695559
## online_trans           Q1           Q2           Q3           Q4
##     7.956959     1.450139     1.168348     1.402106     1.155061
##           Q5           Q6           Q7           Q8           Q9
##     1.284377     1.438529     1.455941     1.154347     1.118493
##          Q10
##     1.136174
```

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 are very likely some big/small purchase in store. We can check it quickly:

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -500.0   205.0   329.0  1357.0   597.3 50000.0
```

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   68.82  420.30 1942.00 2120.00 2441.00 9479.00
```

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:

```
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)
```

4.2.2. `ddply()` in `plyr` package

`dplyr` is a set of clean and consistent tools that implement the split-apply-combine pattern in R. This is an extremely common pattern in data analysis: you solve a complex problem by breaking it down into small pieces, doing something to each piece and then combining the results back together again. [From package description]

You may find the description sounds familiar. The package is sort of a wrapper of apply family. We will only introduce the main function `ddply()`. Because the package has next iteration which is `dplyr` package. We will introduce `dplyr` in more details. The reason we still want to spend some time on the older version is that they have the similar idea and knowing the lineage will deeper your understanding of the whole family.

We will use the same data frame `sim.dat` to illustrate. Run the following command:

```
library(plyr)
```

```
## ------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first,
then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
ddply(sim.dat,"segment",summarize, 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))
```

```
##       segment Age FemalePct HouseYes store_exp online_exp store_trans
## 1 Conspicuous  42      0.32     0.86      4990       4898        10.9
## 2       Price  60      0.45     0.94       501        205         6.1
## 3     Quality  35      0.47     0.34       301       2013         2.9
## 4       Style  24      0.81     0.27       200       1962         3.0
##   online_trans
## 1         11.1
## 2          3.0
## 3         16.0
## 4         21.1
```

Now, let's peel the onion in order.

The first argument `sim.dat` is easy. It is the data you want to work on.

The second argument `"segment"` is the column you want to group by. It is a very standard marketing segmentation problem. The final segment is the result you want to get by designing survey, collecting and analyzing data. Here we assume those segments are known, and we want to understand how each group of customer look like. It is a common task in segmentation: figuring out a profile. Here we only summarize data by one categorical variable, but you can group by multiple variables using `ddply(sim.dat, c("segment","house"), .)`. So the second argument tells the function we want to divide data by customer segment.

The third argument `summarize` tells R the kind of manipulation you want to do which is to summarize data. There are other choices for this argument such as `transform` (transform data within each group) and `subset`(subset data within each group).

Then the rest commands tell R the exact action. For example, `Age=round(mean(na.omit(age)),0)` tell R the following things:

1. Calculate the mean of column `age` ignoring missing value
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 store
7. `online_trans`: average times of online transactions

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

- Conspicuous: average age is about 40. Target for middle-age wealthy people. 1/3 of them are female and 2/3 are male. They may be a good target for candy dad. They buy regardless the price. Almost all of them own house (0.86). It makes us wonder what is wrong with the rest 14%? They may live in Manhattan

- Price: They are older people, 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). This 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 definitely prefer online shopping.

You may notice that Style group purchase more frequently online (online_trans=21) but the expense (online_exp=1962) is not higher. This makes us wondering what is the average expense each time so you have a better idea about the price range the group fall in.

The analytical process is aggregated instead of independent steps. What you learn before will help you decide what to do next. Sometimes you also need to go backward to fix something in the previous steps. For example, you may need to check those negative expense value.

We continue to use `ddply()` to calculate the two statistics:

```
ddply(sim.dat,"segment",summarize,avg_online=round(sum(online_exp)/sum(online
_trans),2),
      avg_store=round(sum(store_exp)/sum(store_trans),2))
```

```
##       segment avg_online avg_store
## 1 Conspicuous     442.27    479.25
## 2       Price      69.28     81.30
## 3     Quality     126.05    105.12
## 4       Style      92.83    121.07
```

Price group has the lowest averaged one-time purchasing price. The Conspicuous group will pay the highest price. When we build the profile in real life, we will need to look at the survey results too. Those simple data manipulations can provide you lots of information already. As mentioned before, other than "`summarize`" there are other functions such as "`transform`" and "`subset`".

For simplicity, I draw 11 random samples and 3 variables (`age`, `store_exp` and `segment`) from the original data according to the different segments. We will explain stratified sampling later. Here we just do it without explanation.

```
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

set.seed(2016)
trainIndex<-createDataPartition(sim.dat$segment,p=0.01,list=F,times=1)
examp<-sim.dat[trainIndex,c("age","store_exp","segment")]
```

Now data frame `examp` only has 11 rows and 3 columns. Let's look at the function of `transform`:

```
ddply(examp,"segment",transform,store_pct=round(store_exp/sum(store_exp),2))

##    age store_exp     segment store_pct
## 1   42 6319.0718 Conspicuous      0.55
## 2   42 5106.4816 Conspicuous      0.45
## 3   55  595.2520       Price      0.42
## 4   64  399.3550       Price      0.28
## 5   64  426.6653       Price      0.30
## 6   39  362.4795     Quality      0.58
## 7   35  260.5065     Quality      0.42
## 8   23  205.6099       Style      0.25
## 9   24  212.3040       Style      0.26
## 10  24  202.1017       Style      0.25
## 11  28  200.1906       Style      0.24
```

What "`transform`" does is to transform data within the specified group (segment) and append the result as a new column.

Next let's look at the function of "`subset`":

```
ddply(examp,"segment",subset,store_exp>median(store_exp))

##   age store_exp     segment
## 1  42 6319.0718 Conspicuous
## 2  55  595.2520       Price
## 3  39  362.4795     Quality
## 4  23  205.6099       Style
## 5  24  212.3040       Style
```

You get all rows with `store_exp` greater than its group median.

4.2.3. `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

I 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.

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

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

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

Subset

Get rows with `income` more than 300000:

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

```
## # A tibble: 4 x 19
##     age gender   income  house store_exp online_exp store_trans
##   <int> <fctr>    <dbl> <fctr>     <dbl>      <dbl>       <int>
## 1    40   Male 301398.0    Yes  4840.461   3618.212          10
## 2    33   Male 319704.3    Yes  5998.305   4395.923           9
## 3    41   Male 317476.2    Yes  3029.844   4179.671          11
## 4    37 Female 315697.2    Yes  6548.970   4284.065          13
## # ... with 12 more variables: online_trans <int>, Q1 <int>, Q2 <int>,
## #   Q3 <int>, Q4 <int>, Q5 <int>, Q6 <int>, Q7 <int>, Q8 <int>, Q9 <int>,
## #   Q10 <int>, segment <fctr>
```

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 straight forward 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.

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

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

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

`slice()` will select rows by position:

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

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

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

```
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:

```
# 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
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

The operations here are similar what we did before with `apply()` and `ddply()`.

```
dplyr::summarise(sim.dat, avg_online = mean(online_trans))
```

```
##   avg_online
## 1     13.546
```

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

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over all variables, use `summarise_all()`
```

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

You can use `group_by()` to indicate the variables you want to group by as before:

```
sim.dat %>% group_by(segment) %>% summarise_each(funs_(c("anyNA")))
```

```
## `summarise_each()` is deprecated.
## Use `summarise_all()`, `summarise_at()` or `summarise_if()` instead.
## To map `funs` over all variables, use `summarise_all()`

## # A tibble: 4 x 19
##       segment   age gender income house store_exp online_exp store_trans
##        <fctr> <lgl>  <lgl>  <lgl> <lgl>     <lgl>      <lgl>       <lgl>
## 1 Conspicuous FALSE  FALSE   TRUE FALSE     FALSE      FALSE       FALSE
## 2       Price FALSE  FALSE   TRUE FALSE     FALSE      FALSE       FALSE
## 3     Quality FALSE  FALSE   TRUE FALSE     FALSE      FALSE       FALSE
## 4       Style FALSE  FALSE   TRUE FALSE     FALSE      FALSE       FALSE
## # ... with 11 more variables: online_trans <lgl>, Q1 <lgl>, Q2 <lgl>,
## #   Q3 <lgl>, Q4 <lgl>, Q5 <lgl>, Q6 <lgl>, Q7 <lgl>, Q8 <lgl>, Q9 <lgl>,
## #   Q10 <lgl>
```

Create new variable

`mutate()` will compute and append one or more new columns:

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

It will apply **window function** to the columns and return a column with the same length. It is a different type of function as before.

```
# min_rank=rank(ties.method = "min")
# mutate_each() means apply function to each column
dplyr::mutate_each(sim.dat, funs(min_rank))
```

The other 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

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

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

```
# 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.

4.3.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. In order 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 playing with a 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.

4.3.1. `reshape2` package

It is a reboot of the previous package `reshape`. Why? Here is what I got from Stack Overflow:

"`reshape2` let Hadley make a rebooted reshape that was way, way faster, while avoiding busting up people's dependencies and habits."

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 120963.4   Yes  529.1344   303.5125
## 2  63 Female 122008.1   Yes  478.0058   109.5297
## 3  59   Male 114202.3   Yes  490.8107   279.2496
## 4  60   Male 113616.3   Yes  347.8090   141.6698
## 5  51   Male 124252.6   Yes  379.6259   112.2372
```

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)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
```

```
(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 120963.4   Yes  store_exp 529.1344
## 2   63 Female 122008.1   Yes  store_exp 478.0058
## 3   59   Male 114202.3   Yes  store_exp 490.8107
## 4   60   Male 113616.3   Yes  store_exp 347.8090
## 5   51   Male 124252.6   Yes  store_exp 379.6259
## 6   57 Female 120963.4   Yes online_exp 303.5125
## 7   63 Female 122008.1   Yes online_exp 109.5297
```

```
## 8   59   Male 114202.3   Yes online_exp 279.2496
## 9   60   Male 113616.3   Yes online_exp 141.6698
## 10  51   Male 124252.6   Yes online_exp 112.2372
```

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:

```
# Here we use all observations from sim.dat
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)

##
## Call:
## lm(formula = Expense ~ gender + house + income + Channel + age,
##     data = mdat)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -4208   -821   -275    533  44353
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -9.132e+02  1.560e+02  -5.855 5.76e-09 ***
## genderMale         3.572e+02  1.028e+02   3.475 0.000524 ***
## houseYes          -5.687e+01  1.138e+02  -0.500 0.617275
## income             2.834e-02  1.079e-03  26.268  < 2e-16 ***
## Channelonline_exp  8.296e+02  9.772e+01   8.489  < 2e-16 ***
## age               -2.793e+01  3.356e+00  -8.321  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1974 on 1626 degrees of freedom
##   (368 observations deleted due to missingness)
## Multiple R-squared:  0.348,  Adjusted R-squared:  0.346
## F-statistic: 173.5 on 5 and 1626 DF,  p-value: < 2.2e-16
```

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    No Female  171102.2   583492.4
## 2    No   Male  133130.8   332499.9
## 3   Yes Female  355320.2   500856.9
## 4   Yes   Male  697297.3   703332.0
```

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`" .

4.3.2. `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> <fctr>     <dbl>       <int>
## 1    57 Female  529.1344           2
## 2    63 Female  478.0058           4
## 3    59   Male  490.8107           7
## 4    60   Male  347.8090          10
## 5    51   Male  379.6259           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> <fctr>       <chr>    <dbl>
##  1    57 Female   store_exp 529.1344
##  2    63 Female   store_exp 478.0058
##  3    59   Male   store_exp 490.8107
##  4    60   Male   store_exp 347.8090
##  5    51   Male   store_exp 379.6259
##  6    57 Female store_trans   2.0000
##  7    63 Female store_trans   4.0000
##  8    59   Male store_trans   7.0000
##  9    60   Male store_trans  10.0000
## 10    51   Male store_trans   4.0000
```

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  379.6259           4
## 2  57 Female  529.1344           2
## 3  59   Male  490.8107           7
## 4  60   Male  347.8090          10
## 5  63 Female  478.0058           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> <fctr>  <chr> <chr>    <dbl>
##  1    57 Female  store   exp 529.1344
##  2    63 Female  store   exp 478.0058
##  3    59   Male  store   exp 490.8107
##  4    60   Male  store   exp 347.8090
##  5    51   Male  store   exp 379.6259
##  6    57 Female  store trans   2.0000
##  7    63 Female  store trans   4.0000
##  8    59   Male  store trans   7.0000
##  9    60   Male  store trans  10.0000
## 10    51   Male  store trans   4.0000
```

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 529.1344
## 2   63 Female   store_exp 478.0058
## 3   59   Male   store_exp 490.8107
## 4   60   Male   store_exp 347.8090
## 5   51   Male   store_exp 379.6259
## 6   57 Female store_trans   2.0000
## 7   63 Female store_trans   4.0000
## 8   59   Male store_trans   7.0000
## 9   60   Male store_trans  10.0000
## 10  51   Male store_trans   4.0000
```

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. 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.

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 analyst 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:

```
source("https://raw.githubusercontent.com/happyrabbit/CE_JSM2017/master/Rcode
/00Require.R")

##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine

##
## Attaching package: 'imputeMissings'

## The following object is masked from 'package:e1071':
##
##     impute

## The following object is masked from 'package:dplyr':
##
##     compute

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode

##
## Attaching package: 'psych'

## The following object is masked from 'package:car':
##
##     logit

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

5.1.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:

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

##       age            gender        income       house
##  Min.   : 16.00   Female:554   Min.   : 41776   No :432
##  1st Qu.: 25.00   Male  :446   1st Qu.: 85832   Yes:568
##  Median : 36.00                Median : 93869
##  Mean   : 38.84                Mean   :113543
##  3rd Qu.: 53.00                3rd Qu.:124572
##  Max.   :300.00                Max.   :319704
##                                NA's   :184
##    store_exp         online_exp       store_trans     online_trans
##  Min.   : -500.0   Min.   :  68.82   Min.   : 1.00   Min.   : 1.00
```

```
##  1st Qu.:  205.0   1st Qu.: 420.34   1st Qu.: 3.00   1st Qu.: 6.00
##  Median :  329.0   Median :1941.86   Median : 4.00   Median :14.00
##  Mean   : 1356.8   Mean   :2120.18   Mean   : 5.35   Mean   :13.55
##  3rd Qu.:  597.3   3rd Qu.:2440.78   3rd Qu.: 7.00   3rd Qu.:20.00
##  Max.   :50000.0   Max.   :9479.44   Max.   :20.00   Max.   :36.00
##
##        Q1              Q2              Q3              Q4
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000
##  Median :3.000   Median :1.000   Median :1.000   Median :3.000
##  Mean   :3.101   Mean   :1.823   Mean   :1.992   Mean   :2.763
##  3rd Qu.:4.000   3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##
##        Q5              Q6              Q7              Q8
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.750   1st Qu.:1.000   1st Qu.:2.500   1st Qu.:1.000
##  Median :4.000   Median :2.000   Median :4.000   Median :2.000
##  Mean   :2.945   Mean   :2.448   Mean   :3.434   Mean   :2.396
##  3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:3.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##
##        Q9             Q10             segment
##  Min.   :1.000   Min.   :1.00   Conspicuous:200
##  1st Qu.:2.000   1st Qu.:1.00   Price      :250
##  Median :4.000   Median :2.00   Quality    :200
##  Mean   :3.085   Mean   :2.32   Style      :350
##  3rd Qu.:4.000   3rd Qu.:3.00
##  Max.   :5.000   Max.   :5.00
##
```

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 imput them instead of deleting the rows.

```r
# 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.   :16.00   Min.   : 41776
##  1st Qu.:25.00   1st Qu.: 85832
##  Median :36.00   Median : 93869
##  Mean   :38.58   Mean   :113543
##  3rd Qu.:53.00   3rd Qu.:124572
##  Max.   :69.00   Max.   :319704
##  NA's   :1       NA's   :184
```

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

5.2.Missing Values

Missing value imputation can be the topic for a book. This section will show some of the commonly used methods without getting too deep into the topic. Chapter 7 of the book by De Waal, Pannekoek and Scholtus [5] makes a concise overview of some of the existing imputation methods. The choice of specific method depends on the actual situation. There is no method always better than the others.

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 imput with mean or median. If not, what is the potential mechanism for the missing? 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 imput. 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 [6]and De Waal, Pannekoek and Scholtus' book [5].

5.2.1.Impute missing values with median/mode

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

```
# 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       house       store_exp
##  Min.   :16.00   Female:554   Min.   : 41776   No :432   Min.   :  155.8
##  1st Qu.:25.00   Male  :446   1st Qu.: 87896   Yes:568   1st Qu.:  205.1
##  Median :36.00                Median : 93869             Median :  329.8
##  Mean   :38.58                Mean   :109923             Mean   : 1357.7
##  3rd Qu.:53.00                3rd Qu.:119456             3rd Qu.:  597.3
##  Max.   :69.00                Max.   :319704             Max.   :50000.0
```

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 imput with the mode.

You can also use `preProcess ()` function, but it is only for numeric variables, and can not imput 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       house       store_exp
##  Min.   :16.00   Female:554   Min.   : 41776   No :432   Min.   :  155.8
##  1st Qu.:25.00   Male  :446   1st Qu.: 87896   Yes:568   1st Qu.:  205.1
##  Median :36.00                Median : 93869             Median :  329.8
##  Mean   :38.58                Mean   :109923             Mean   : 1357.7
##  3rd Qu.:53.00                3rd Qu.:119456             3rd Qu.:  597.3
##  Max.   :69.00                Max.   :319704             Max.   :50000.0
```

5.2.2.K-nearest neighbors

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

Use `preProcess()` to conduct KNN:

```
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.

```
# 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.   :-1.5910972   Min.   :-1.43989   Min.   :-0.43345
##  1st Qu.:-0.9568733   1st Qu.:-0.53732   1st Qu.:-0.41574
##  Median :-0.1817107   Median :-0.37606   Median :-0.37105
##  Mean   : 0.0000156   Mean   : 0.02389   Mean   :-0.00042
##  3rd Qu.: 1.0162678   3rd Qu.: 0.21540   3rd Qu.:-0.27437
##  Max.   : 2.1437770   Max.   : 4.13627   Max.   :17.52734
```

`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:

```
# only show the fist 3 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 imput sample 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 use 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.

5.2.3.Bagging Tree

Bagging (Bootstrap aggregation abbreviation) was originally proposed by Leo Breiman. It is one of the earliest ensemble methods [7]. 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 imput `sim.dat`:

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

5.3.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 [8], PLS [9] and EFA [10]. You can easily writing code yourself to conduct this transformation.

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

```
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.

```
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:

```
summary(transformed)
```

```
##       age              income        
##  Min.   :-1.5911   Min.   :-1.4399  
##  1st Qu.:-0.9569   1st Qu.:-0.5560  
##  Median :-0.1817   Median :-0.3947  
##  Mean   : 0.0000   Mean   : 0.0000  
##  3rd Qu.: 1.0163   3rd Qu.: 0.2213  
##  Max.   : 2.1438   Max.   : 4.1363  
##  NA's   :1         NA's   :184
```

Sometimes you only need to scale the variable. For example, if the model adds penalty to the parameter estimates (such as L_2 penalty is ridge regression and L_1 penalty in LASSO), the variables need to have similar scale to ensure a fair variable selection. I am 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:

```
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:

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

```
##      income            store_exp           online_exp
##  Min.   :-0.05776   Min.   :-0.003407   Min.   :-0.006023
##  1st Qu.: 0.15736   1st Qu.: 0.003984   1st Qu.: 0.042719
##  Median : 0.18521   Median : 0.022704   Median : 0.253691
##  Mean   : 0.26009   Mean   : 0.176965   Mean   : 0.278417
##  3rd Qu.: 0.30456   3rd Qu.: 0.062849   3rd Qu.: 0.322871
##  Max.   : 1.23857   Max.   : 7.476996   Max.   : 1.298845
```

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

5.4.Resolve Skewness

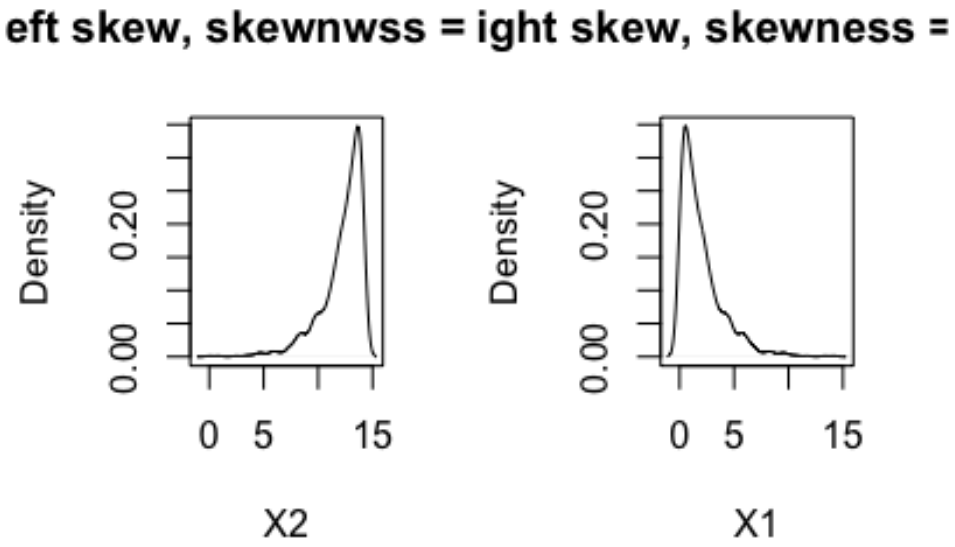
Skewness is defined to be the third standardized central moment. The formula for the sample skewness statistics is:

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

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

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

```
# 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")
```



Shewed Distribution

You can easily tell if a distribution is skewed by simple visualization. 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[11].

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

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   trimmed      mad
## age             1  999     38.58    14.19    36.00     37.67    16.31
## gender*         2 1000      1.45     0.50     1.00      1.43     0.00
## income          3  816 113543.07 49842.29 93868.68 104841.94 28989.47
## house*          4 1000      1.57     0.50     2.00      1.58     0.00
## store_exp       5  999   1358.71  2775.17   329.80    845.14   197.47
```

```
## online_exp      6 1000   2120.18  1731.22  1941.86   1874.51  1015.21
## store_trans     7 1000      5.35     3.70     4.00      4.89     2.97
## online_trans    8 1000     13.55     7.96    14.00     13.42    10.38
## Q1              9 1000      3.10     1.45     3.00      3.13     1.48
## Q2             10 1000      1.82     1.17     1.00      1.65     0.00
## Q3             11 1000      1.99     1.40     1.00      1.75     0.00
## Q4             12 1000      2.76     1.16     3.00      2.83     1.48
## Q5             13 1000      2.94     1.28     4.00      3.05     0.00
## Q6             14 1000      2.45     1.44     2.00      2.43     1.48
## Q7             15 1000      3.43     1.46     4.00      3.54     0.00
## Q8             16 1000      2.40     1.15     2.00      2.36     1.48
## Q9             17 1000      3.08     1.12     4.00      3.23     0.00
## Q10            18 1000      2.32     1.14     2.00      2.27     1.48
## segment*       19 1000      2.70     1.15     3.00      2.75     1.48
##                   min       max     range  skew kurtosis      se
## age             16.00     69.00     53.00  0.47    -1.18    0.45
## gender*          1.00      2.00      1.00  0.22    -1.95    0.02
## income       41775.64 319704.34 277928.70  1.69     2.57 1744.83
## house*           1.00      2.00      1.00 -0.27    -1.93    0.02
## store_exp      155.81  50000.00  49844.19  8.08   115.04   87.80
## online_exp      68.82   9479.44   9410.63  1.18     1.31   54.75
## store_trans      1.00     20.00     19.00  1.11     0.69    0.12
## online_trans     1.00     36.00     35.00  0.03    -0.98    0.25
## Q1               1.00      5.00      4.00 -0.12    -1.36    0.05
## Q2               1.00      5.00      4.00  1.13    -0.32    0.04
## Q3               1.00      5.00      4.00  1.06    -0.40    0.04
## Q4               1.00      5.00      4.00 -0.18    -1.46    0.04
## Q5               1.00      5.00      4.00 -0.60    -1.40    0.04
## Q6               1.00      5.00      4.00  0.11    -1.89    0.05
## Q7               1.00      5.00      4.00 -0.90    -0.79    0.05
## Q8               1.00      5.00      4.00  0.21    -1.33    0.04
## Q9               1.00      5.00      4.00 -0.68    -1.10    0.04
## Q10              1.00      5.00      4.00  0.39    -1.23    0.04
## segment*         1.00      4.00      3.00 -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`:

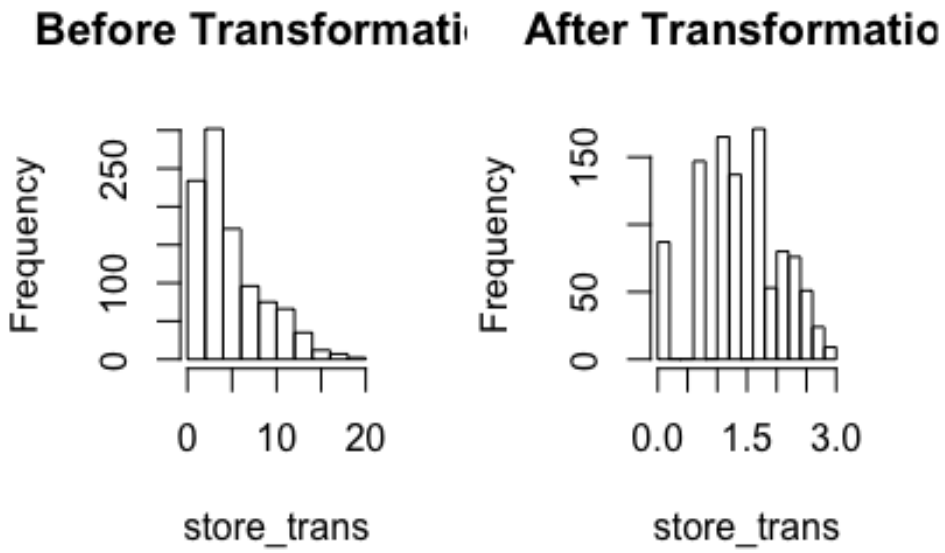
```
# select the two columns and save them as dat_bc
dat_bc<-subset(sim.dat,select=c("store_trans","online_trans"))
(trans<-preProcess(dat_bc,method=c("BoxCox")))

## 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")
```



Box-Cox Transformation

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

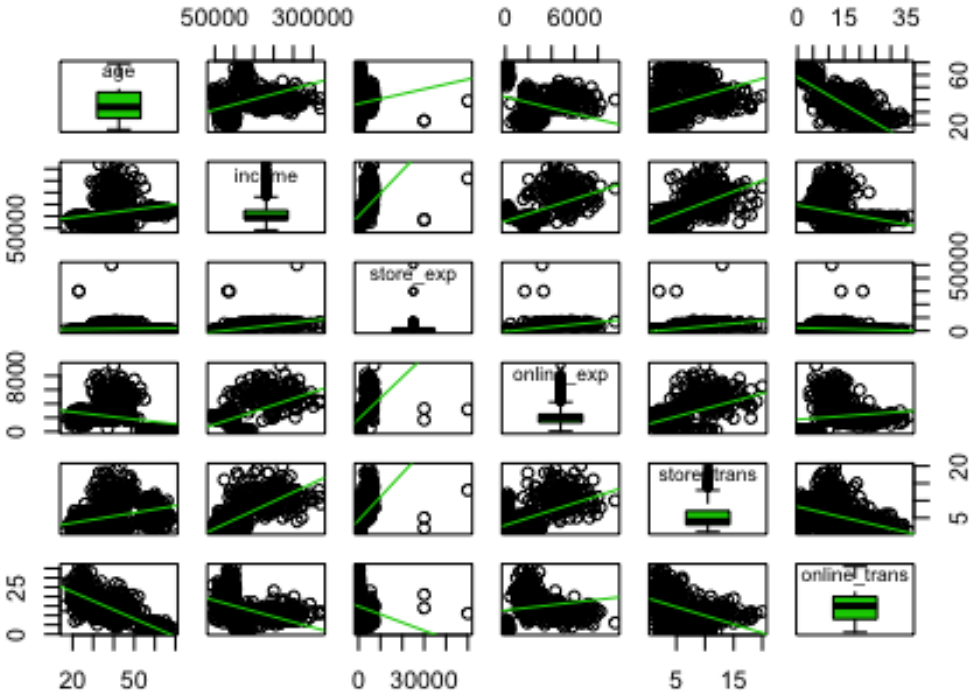
```
## [1] -0.2154708
```

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.

5.5.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_trans" ))
# use scatterplotMatrix() function from car package
par(oma=c(2,2,1,2))
scatterplotMatrix(sdat,diagonal="boxplot",smoother=FALSE)
```



As above 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 **Y** is defined as:

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

where $\overline{Y}$ 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[12]：

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

Where MAD is the median of a series of $| Y_ {i} - {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`:

```
# 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 [12] 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 can not be deleted at will. If a model is sensitive to outliers, we can use *spatial sign transformation* [13] to minimize the problem. It projects the original sample points to the surface of a sphere by:

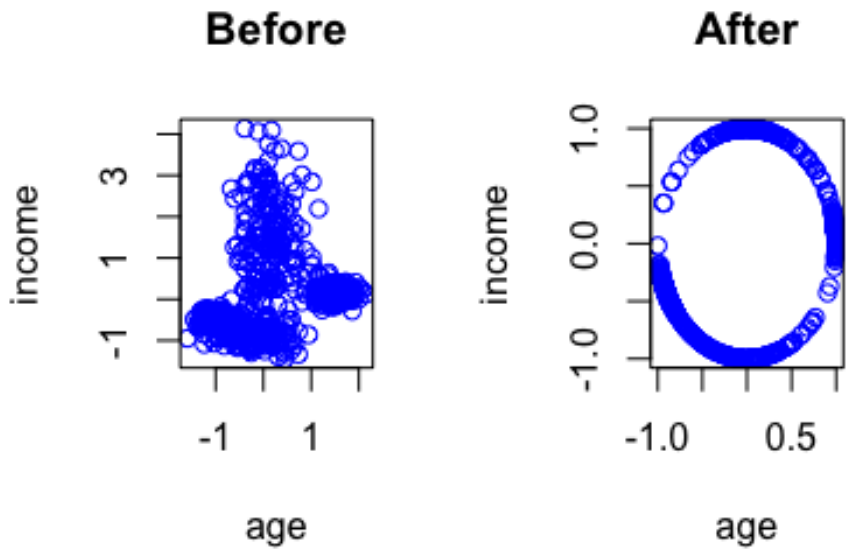
$$x_{ij}^{*} = \frac{x_{ij}}{\sqrt{\sum_{j=1}^{p} x_{ij}^{2}}}$$

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:

```
# 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")
```



spatial sign transformation

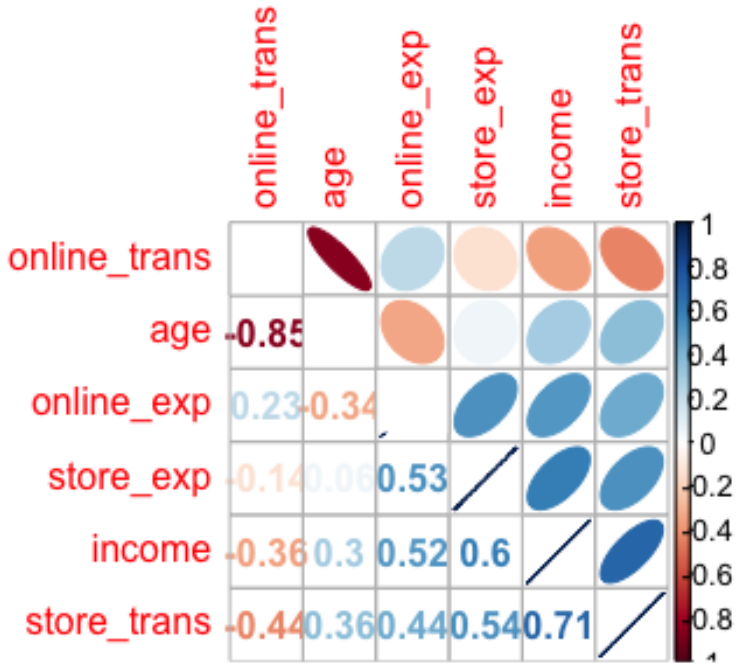
Some readers may have found that the above code does not seem to standardize the data before transformation. Recall the introduction of KNN, `preProcess ()` with `method="knnImpute"` by default will standardize data.

5.6.Collinearity

It is probably a technical term that many un-technical people also know. 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_trans" ))
# use bagging imputation here
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")
```



Here use `corrplot.mixed()` function to visualize the correlation matrix. 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 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" [14] 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 online_trans
## income        1.0000000  0.6004006  0.5198623   0.7069595   -0.3572884
## store_exp     0.6004006  1.0000000  0.5349527   0.5399121   -0.1367411
## online_exp    0.5198623  0.5349527  1.0000000   0.4420638    0.2256370
## store_trans   0.7069595  0.5399121  0.4420638   1.0000000   -0.4367544
## online_trans -0.3572884 -0.1367411  0.2256370  -0.4367544    1.0000000
```

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.

5.7.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

5.8.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<-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 120963.4
```

```
## 2             1           0        0         1 122008.1
## 3             0           1        0         1 114202.3
## 4             0           1        0         1 113616.3
## 5             0           1        0         1 124252.6
## 6             0           1        0         1 107661.5
```

`dummyVars()` can also use formula format. The variable on the right-hand side can be both categorical and numeric. For 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:

```
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 120963.4
## 2             1           0        0         1 122008.1
## 3             0           1        0         1 114202.3
## 4             0           1        0         1 113616.3
## 5             0           1        0         1 124252.6
## 6             0           1        0         1 107661.5
##   gender.Female:income gender.Male:income
## 1             120963.4                0.0
## 2             122008.1                0.0
## 3                  0.0           114202.3
## 4                  0.0           113616.3
## 5                  0.0           124252.6
## 6                  0.0           107661.5
```

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.

6. DYNAMIC/REPRODUCIBLE REPORT

6.1.What is R Markdown?

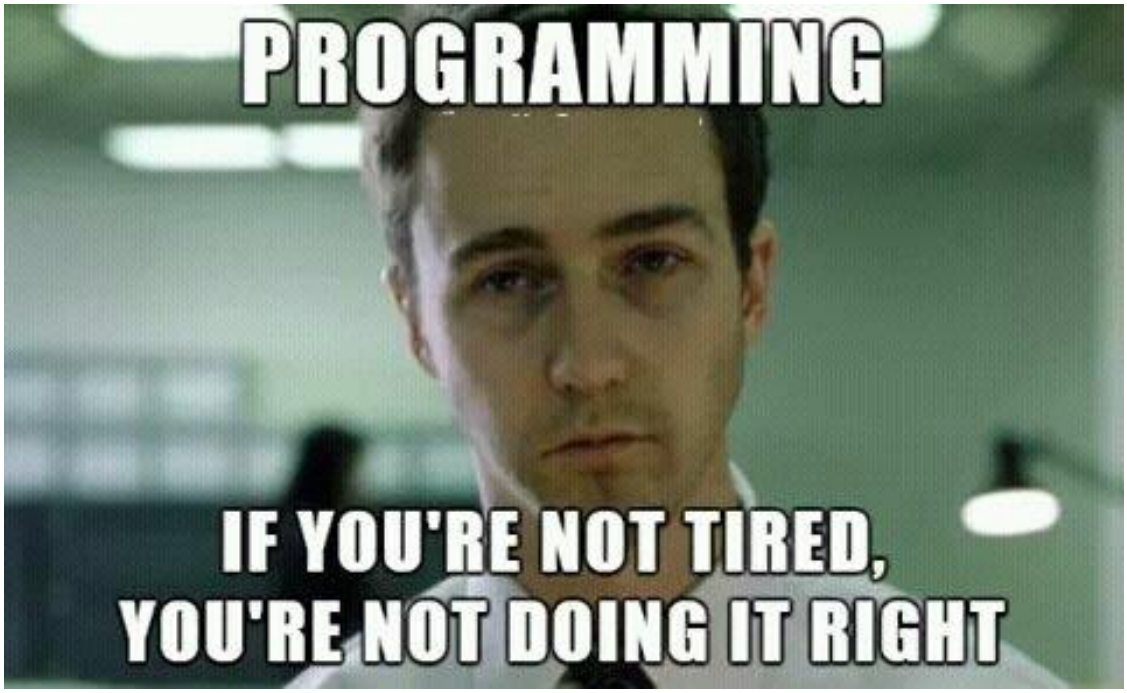
Let's start from markdown. Markdown is a lightweight markup language designed to make authoring content easy for everyone. Here is a definition of markup language:

Markup languages are designed for the processing, definition, and presentation of text. The language specifies code for formatting, both the layout and style, within a text file. The code used to specify the format are called tags.

HTML is an example of a widely known and used markup language.

Rather than writing complex markup code (e.g. LyX, XML, HTML or LaTeX), Markdown enables the use of a syntax much more like plain-text email. It is young compared to the other markup languages. What makes markdown distinct is that it is both machine-readable and human-readable.

R Markdown combines the core syntax of Markdown and embedded R code chunks that are run so their output can be included in the final document. Consider how people typically create an analytical report. The author makes the graph/table, saves it as a file, and then copy and pastes it into the final report. This process relies on manual labor. The author may take a deep breath when the report is finally well-shaped. If the data changes, the author must repeat the entire process to update the graph.



R Markdown comes to rescue! It provides an authoring framework for data science. You can use a single R Markdown file to do both:

- save and execute code

- generate high-quality reports that can be shared with an audience

R Markdown documents are fully reproducible and the most important and it is simple!

6.2.How to Start?

6.2.1.How It Works?

When you run `render`, R Markdown feeds the `.rmd` file to `knitr`.`knitr` is an R package that will execute all of the code chunks and creates a new markdown (.md) document which includes the code and its output. The markdown file is then processed by `pandoc` which is responsible for creating the finished format. `pandoc` is a swiss-knife to convert files from one markup format into another.

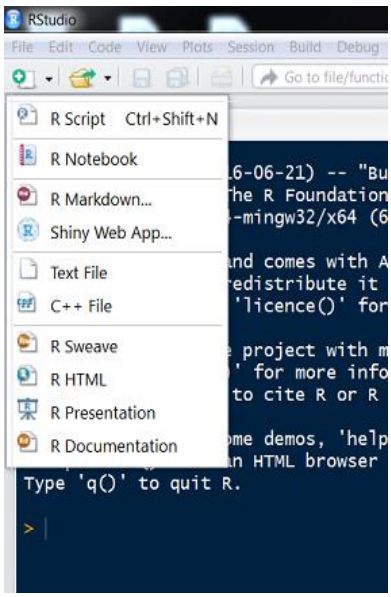
R Markdown encapsulates all of the above processing into a single render function.

6.2.2.Get Started

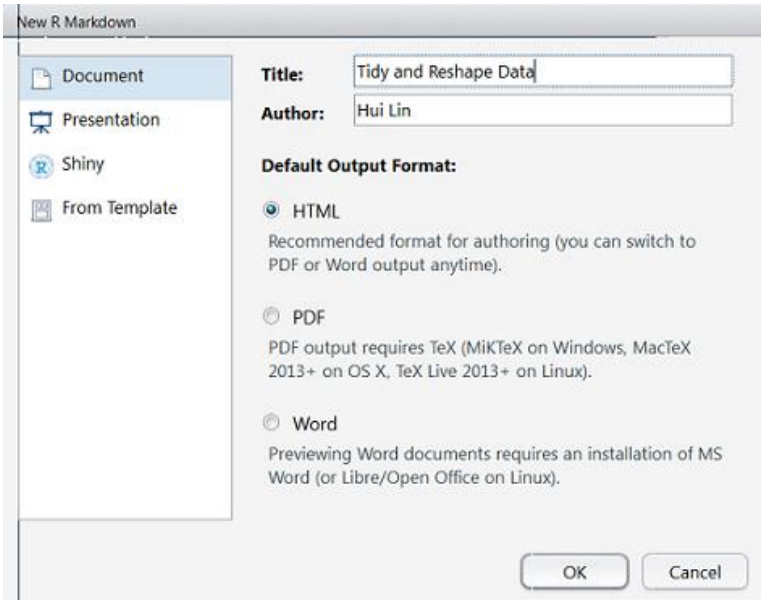
Install R and RStudio. If you have RStudio installed ready, I suggest you make sure it is in the latest version. You can install the `rmarkdown` package from CRAN with:

```
install.packages("rmarkdown")
```

R Markdown file is a plain text file that has the extension `.Rmd`. You can create a sample `.Rmd` file in R Studio:



Input your document title and author name and click "OK":



The file contains three types of content:

- An (optional) YAML header surrounded by `---`
- R code chunks surrounded by ```` ```{r} ```` and ```` ``` ````
- text mixed with simple text formatting

R Markdown generates a new file that contains selected text, code, and results from the `.Rmd` file. The new file can be in the following formats:

- HTML
- PDF
- MS Word document
- slide show
- book
- dashboard
- package vignette
- Others

6.2.3.Markdown Basic

Don't worry if you are new to Markdown. You can quickly pick up only by looking at a few examples of it in action. We will show some examples in a before/after style. You will see example syntax and the HTML output in R Studio. The webpage provides complete, detailed documentation for every markdown feature.

6.2.3.1.Paragraphs, Headers

A paragraph is simply one or more consecutive lines of text, separated by one or more blank lines. Standard paragraphs **should not** be indented with spaces or tabs.

You can put 1-6 hash marks (#) at the beginning of the line - the number of hashes equals the resulting HTML header level.

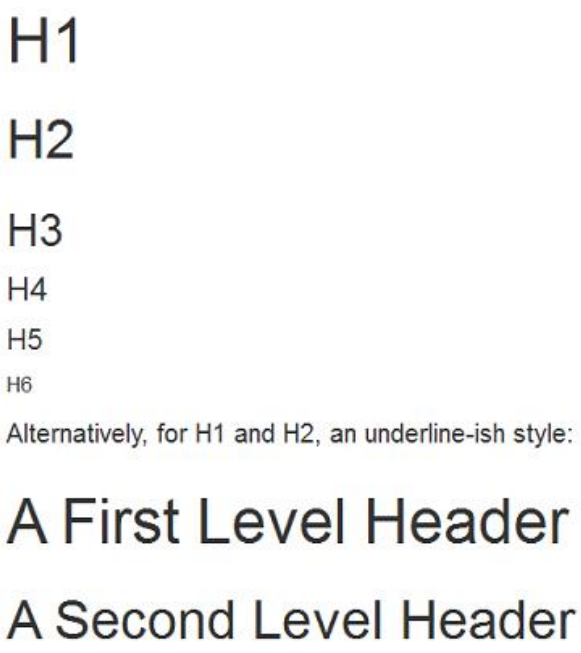
```
# H1
## H2
### H3
#### H4
##### H5
###### H6

Alternatively, for H1 and H2, an underline-ish style:

A First Level Header
====================

A Second Level Header
---------------------
```

Output:



6.2.3.2.Blockquotes

Blockquotes are indicated using email-style '>' angle brackets.

```
A statistician gave birth to twins, but only had one of them baptised. She
kept the other as a control.

News bulletin: A local Physicist declared that he has figured out the
ingredients in McDonald’s secret sauce: protons, nuetrons, and electrons.

> All you need in this life is ignorance and confidence, and then success is
sure. [Mark Twain]
>
> A bartender says, “We don’t serve faster than light particles in here.” A
tachyon walks into a bar. [A joke from Prof. Bill Rand]
```

Output:

A statistician gave birth to twins, but only had one of them baptised. She kept the other as a control.

News bulletin: A local Physicist declared that he has figured out the ingredients in McDonald’s secret sauce: protons, nuetrons, and electrons.

> All you need in this life is ignorance and confidence, and then success is sure. [Mark Twain]
>
> A bartender says, “We don’t serve faster than light particles in here.” A tachyon walks into a bar. [A joke from Prof. Bill Rand]

6.2.3.3.Phrase Emphasis

Markdown uses asterisks and underscores to indicate spans of emphasis.

```
Some of these words *are italic*.
Some of these words _are italic also_.

Use two asterisks for **bold**.
Or, if you prefer, __use two underscores instead__.

Strikethrough uses two tildes. ~~Scratch this.~~
```

Output:

Some of these words *are italic*.

Some of these words *are italic also*.

Use two asterisks for **bold**.

Or, if you prefer, **use two underscores instead**.

Strikethrough uses two tildes. ~~Scratch this.~~

6.2.3.3.1.1.Lists

Unordered (bulleted) lists use asterisks, pluses, and hyphens (*, +, and -) as list markers. These three markers are interchangeable; this:

```
* If it’s green and wiggles, it’s biology.
* If it stinks, it’s chemistry.
* If it doesn’t work, it’s Physics.
```

Output:

- If it’s green and wiggles, it’s biology.
- If it stinks, it’s chemistry.
- If it doesn’t work, it’s Physics.

this:

```
+ Engineers think that equations approximate the real world.
+ Scientists think that the real world approximates equations.
+ Mathematicians don’t care.
```

Output:

- Engineers think that equations approximate the real world.
- Scientists think that the real world approximates equations.
- Mathematicians don’t care.

and this:

```
An engineer, a physicist, and a mathematician were on a train heading north,
and had just crossed the border into Scotland.

- The engineer looked out of the window and said “Look! Scottish sheep are
black!”
- The physicist said, “No, no. Some Scottish sheep are black.”
- The mathematician looked irritated. “There is at least one field,
containing at least one sheep, of - which at least one side is black.”
- The statistician : “It’s not significant. We only know there’s one black
sheep”
- The computer scientist : “Oh, no! A special case!”
```

Output:

An engineer, a physicist, and a mathematician were on a train heading north, and had just crossed the border into Scotland.

- The engineer looked out of the window and said “Look! Scottish sheep are black!”
- The physicist said, “No, no. Some Scottish sheep are black.”

- The mathematician looked irritated. “There is at least one field, containing at least one sheep, of - which at least one side is black.”
- The statistician : “It’s not significant. We only know there’s one black sheep”
- The computer scientist : “Oh, no! A special case!”

Next, we will show how to build HTML report and dashboard in more detail.

6.3.HTML

6.3.1.Create an HTML document

To create an HTML document from R Markdown you specify the html_document output format in the front-matter of your document:

```
---
title: "Tidy and Reshape Data"
author: Hui Lin
date: May 11, 2017
output: html_document
---
```

You can add a table of contents using the toc option and specify the depth of headers that it applies to using the toc_depth option. For example:

```
---
title: "Tidy and Reshape Data"
author: Hui Lin
date: May 11, 2017
output:
  html_document:
    toc: true
    toc_depth: 3
---
```

6.3.2.Floating TOC

You can specify the toc_float option to float the table of contents to the left of the main document content. The floating table of contents will always be visible even when the document is scrolled. For example:

```
---
title: "Tidy and Reshape Data"
author: Hui Lin
date: May 11, 2017
output:
  html_document:
    toc: true
    toc_depth: 3
```

```
    toc_float: true
---
```

There are some options for `toc_float` parameter:

- `collapsed` (defaults to `TRUE`) controls whether the table of contents appears with only the top-level (e.g. H2) headers. When collapsed the table of contents is automatically expanded in line when necessary.

- `smooth_scroll` (defaults to `TRUE`) controls whether page scrolls are animated when the table of contents items are navigated to via mouse clicks.

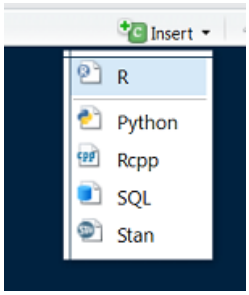
For example:

```
---
title: "Tidy and Reshape Data"
author: Hui Lin
date: May 11, 2017
output:
  html_document:
    toc: true
    toc_depth: 3
    toc_float:
      collapsed: false
      smooth_scroll: false
---
```

6.3.3.Code Chunks

Every code chunk will start with ```` ```{r} ```` and end with ```` ``` ````. You can type the chunk delimiters. Or there are two quick ways to insert chunks to you file:

(1) the keyboard shortcut Ctrl + Alt + I (OS X: Cmd + Option + I)
(2) the Add Chunk command in the editor toolbar



When you render your `.Rmd` file, R Markdown will run each code chunk and embed the results beneath the code chunk in your final report.

- Customize Chunks

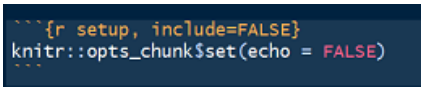
  Chunk output can be customized with options which are arguments in the {} of a chunk header. Here are some of the most common arguments:

- `include = FALSE` prevents code and results from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.
- `echo = FALSE` prevents code, but not the results from appearing in the finished file. This is a useful way to embed figures.
- `message = FALSE` prevents messages that are generated by code from appearing in the finished file.
- `warning = FALSE` prevents warnings that are generated by code from appearing in the finished.
- `fig.height`, `fig.width` The width and height to use in R for plots created by the chunk (in inches).

See the R Markdown Reference Guide for a complete list of knitr chunk options.

- Global Options

  To set global options that apply to every chunk in your file, call `knitr::opts_chunk$set` in a code chunk. Knitr will treat each option that you pass to `knitr::opts_chunk$set` as a global default that can be overwritten in individual chunk headers. For example, you can put the following after front-matter of your document:



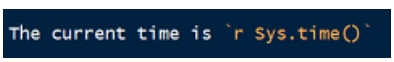
  If you set global option as above, r markdown will prevent code for all chunks unless you overwrite in individual chunk header.

- Caching

  If the computations are long and document rendering becomes time consuming, you can use knitr caching to improve performance. You can use the chunk option `cache=TRUE` to enable cache, and `cache.path` to set the cache directory.

- Inline Code

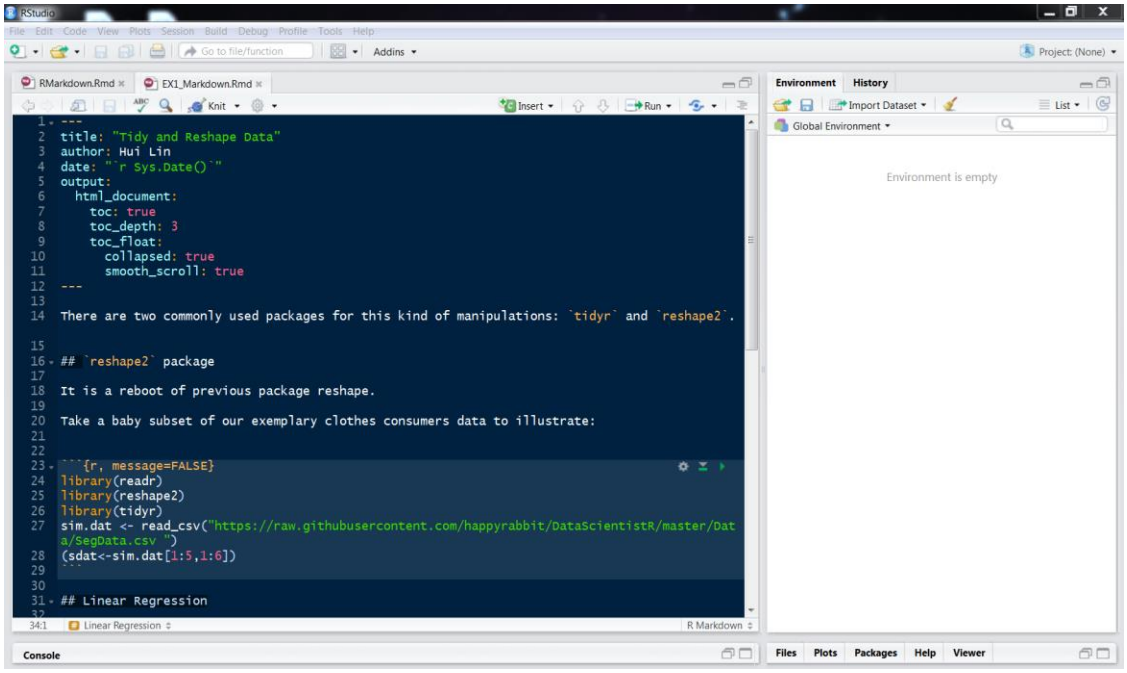
  Code results can be inserted directly into the text of a `.Rmd` file by enclosing the code with `r`. In this way, R Markdown will display the results of inline code, but not the code. For example:



  Output:

As a result, an inline output is indistinguishable from the surrounding text. Inline expressions do not take knitr options.

This is an R Markdown file. You can download a copy: EX1_Markdown.Rmd(output).



6.4.HTML5 Slides

R Markdown supports several HTML presentation (slide show) formats.

- `ioslides_presentation` - HTML presentations with ioslides
- `slidy_presentation` - HTML presentations with slidy
- `revealjs::revealjs_presentation` - HTML presentations with reveal.js

6.4.1. `ioslides` presentation

To create an ioslides presentation from R Markdown you specify the `ioslides_presentation` output format in the front-matter of your document. You can use # and ## to create a new slide. You can also use a horizontal rule (----) to create slide without a header. For example here's a simple slide show. You can download a copy: Ex_ioslide.Rmd(output).

```
---
title: "Dynamic/Reproducible report"
author: "Hui Lin"
date: "`r Sys.Date()`"
output: ioslides_presentation
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
```

Slide One with Bullets

- Bullet 1
- Bullet 2
- Bullet 3

Slide Two with R Output

```{r cars, echo = TRUE}
summary(cars)
```

- Another way to build slide without
```

You can add a subtitle to a slide or section by including text after the pipe (|) character. For example:

```
## Slide One with Bullets | I will never let you know my favourite food is carrot

- Bullet 1
- Bullet 2
- Bullet 3
```

There are different display modes. The following are keyboard shortcuts for each:

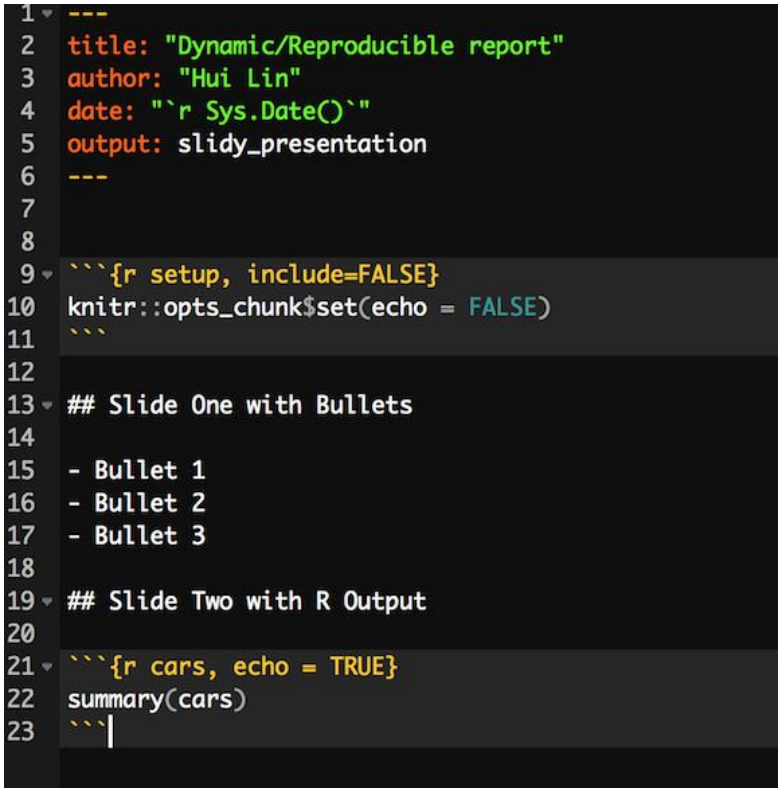
- 'f': fullscreen mode
- 'w': toggle widescreen mode
- 'o': overview mode
- 'h': code highlight mode
- 'p': show presenter notes

Press `Esc` to exit any mode. The code highlight mode enables to select subsets of code for additional emphasis by adding a special "highlight" comment around the code. For example:

When you press `h` key, the highlighted region will be displayed with a bold font and the rest of the code will fade away. So you can help the audience focus exclusively on the highlighted region.

6.4.2. `slidy` presentation

Creating `slidy` presentation is very similar to that of `ioslides` presentation. You specify the `slidy_presentation` output format in the front-matter of your document instead of `ioslides_presentation`. The way you break up slides is the same with `ioslides`. For example here's a simple slide show. You can download a copy: Ex_slidy.Rmd(output).



Like before, there are different display modes. The following are keyboard shortcuts for each:

- `C` Show table of contents
- `F` Toggles the display of the footer

- `A` Toggles display of current vs. all slides (useful for printing handouts)
- `S` Make fonts smaller
- `B` Make fonts larger

For more information about other adjustments, such as appearance text style, CSS, footer elements, etc. please refer to "Presentations with Slidy"

6.5.Dashboards

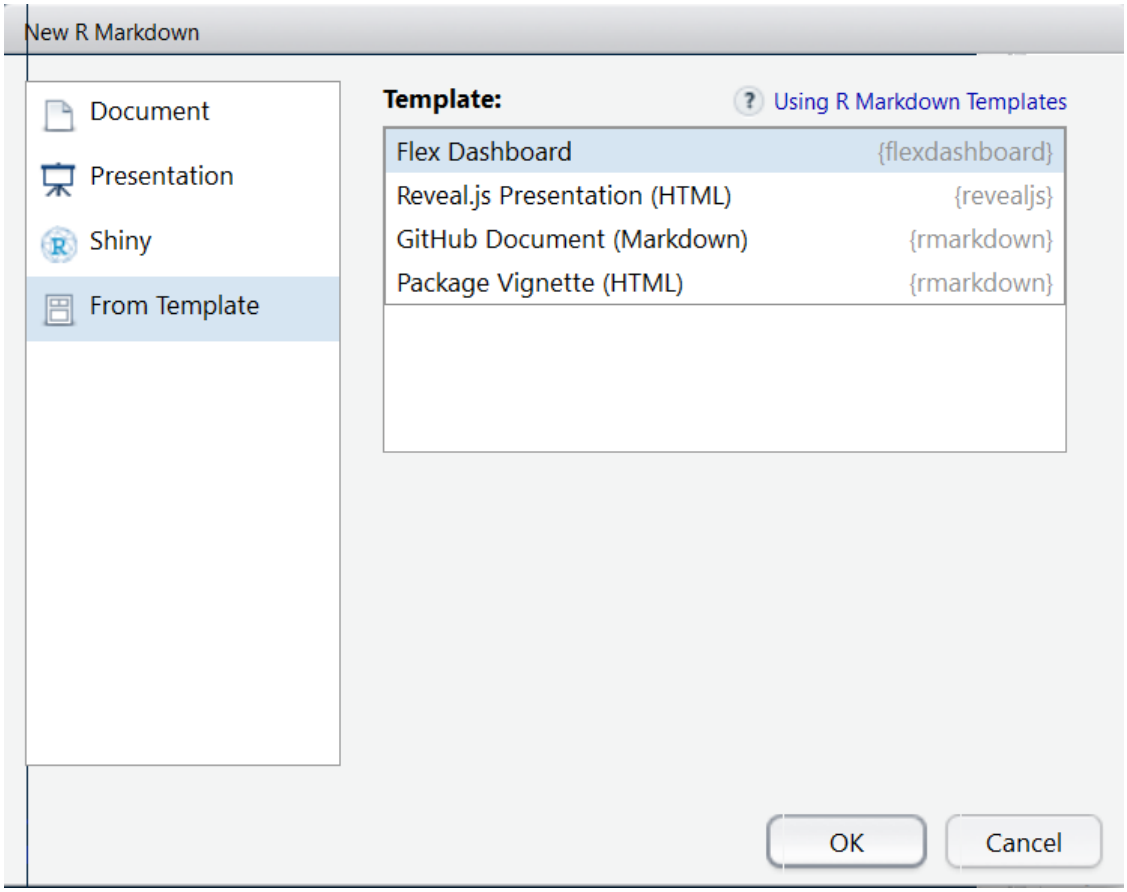
Use R Markdown and `felxdashboard` package to build flexible, attractive, interactive dashboards. Some features of `flexdashboard` + `R Markdown` are:

- Reproducible and highly flexible to specify the row and column-based layouts.
- Nice display: components are intelligently re-sized to fill the browser and adapted for display on mobile devices.
- Support for a wide variety of components including htmlwidgets; base, lattice, and grid graphics; tabular data; gauges and value boxes; and text annotations.
- Extensive support for text annotations to include assumptions, contextual narrative, and analysis within dashboards.
- Storyboard layouts for presenting sequences of visualizations and related commentary.
- By default, dashboards are standard HTML documents that can be deployed on any web server or even attached to an email message. You can optionally add Shiny components for additional interactivity and then deploy on your server or Shiny Server

Install `flexdashboard` package using:

```
install.packages("flexdashboard")
```

Then you can create an R Markdown document with the `flexdashboard::flex_dashboard` output format within RStudio using the New R Markdown dialog:



6.5.1.Layouts

6.5.1.1.Layout by Column

There is no better way to illustrate the syntax of latout than using example. Here is an example of two-column dashboard:

```
---
title: "Layout by Column"
output: flexdashboard::flex_dashboard
---

Column 1
-----------------------------------------------------------------------

### Chart A

```{r}

```

Column 2

Chart B

```{r}

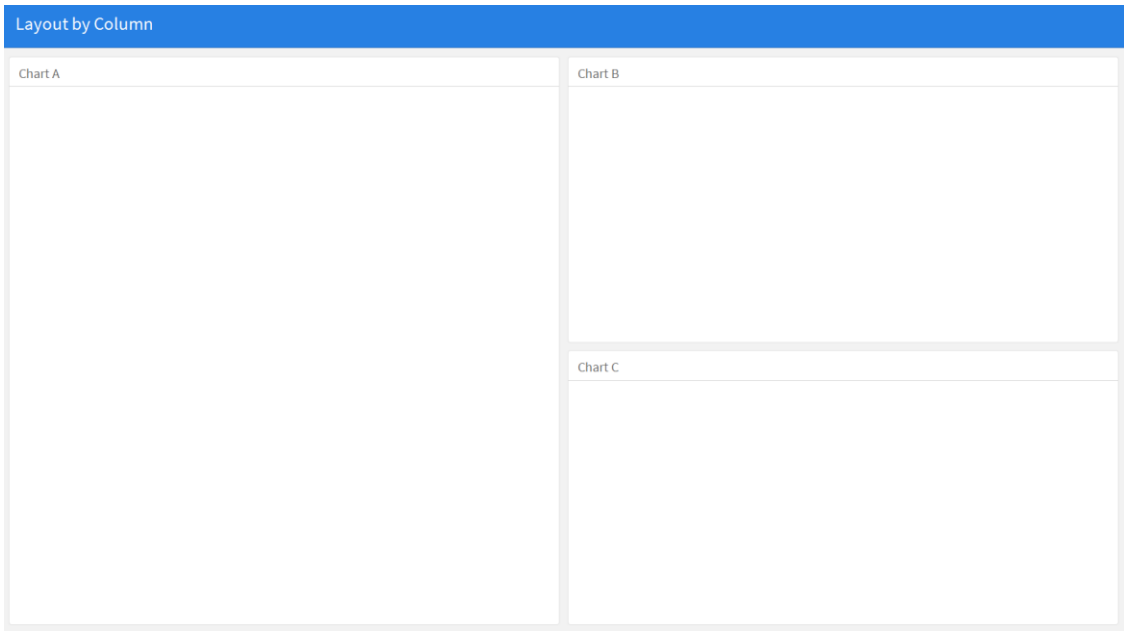
```

Chart C

```{r}

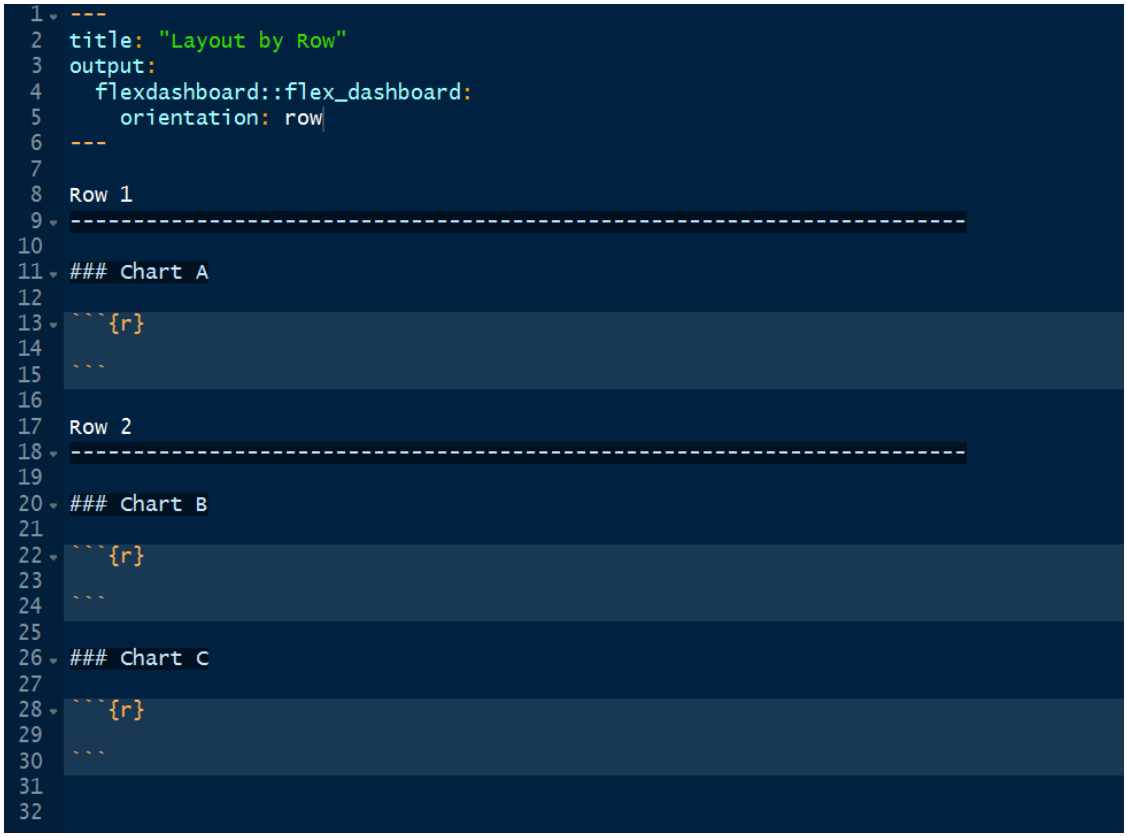
```
```

The ---------- defines columns with individual charts stacked vertically within each column. The above document defines a two-column dashboard with one chart on the left and two charts on the right. The output layout is:



6.5.1.2.Layout by Row

You can similarly define row orientation by setting `orientation: rows`. Here is an example of two-row dashboard:



```
---
title: "Layout by Row"
output:
  flexdashboard::flex_dashboard:
    orientation: row
---

Row 1
-----------------------------------------------------------------------

### Chart A

```{r}

```

Row 2

Chart B

```{r}

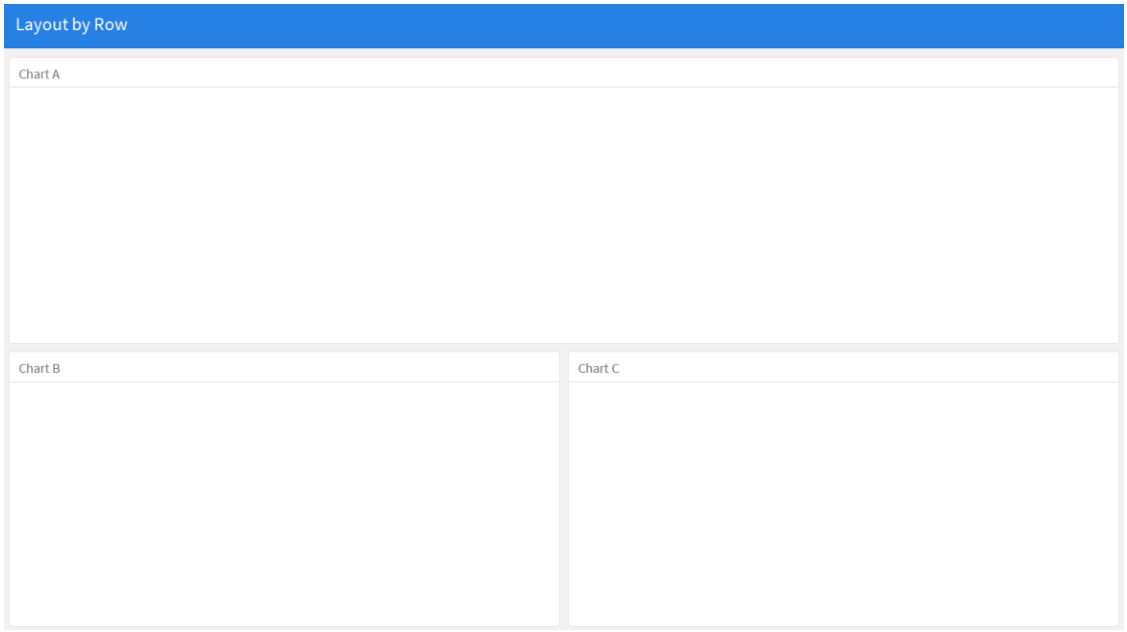
```

Chart C

```{r}

```
```

The `----------` here defines rows. The dashboard has two rows, the first of which has one chart and the second of which has two charts:



6.5.1.3.Scrolling Layout

You may want to scroll rather than fit all the charts onto the page when there are lots of charts. You can set the scrolling function using the `vertical_layout` option. The default setting for `vertical_layout` is `fill`. You can use scrolling layout to demonstrate more charts. However, we recommend you consider using multiple pages instead which we will introduce later.

````
---
title: "Layout by Column (Scrolling)"
output:
  flexdashboard::flex_dashboard:
    vertical_layout: scroll
---

Col 1
-----------------------------------------------------------------------

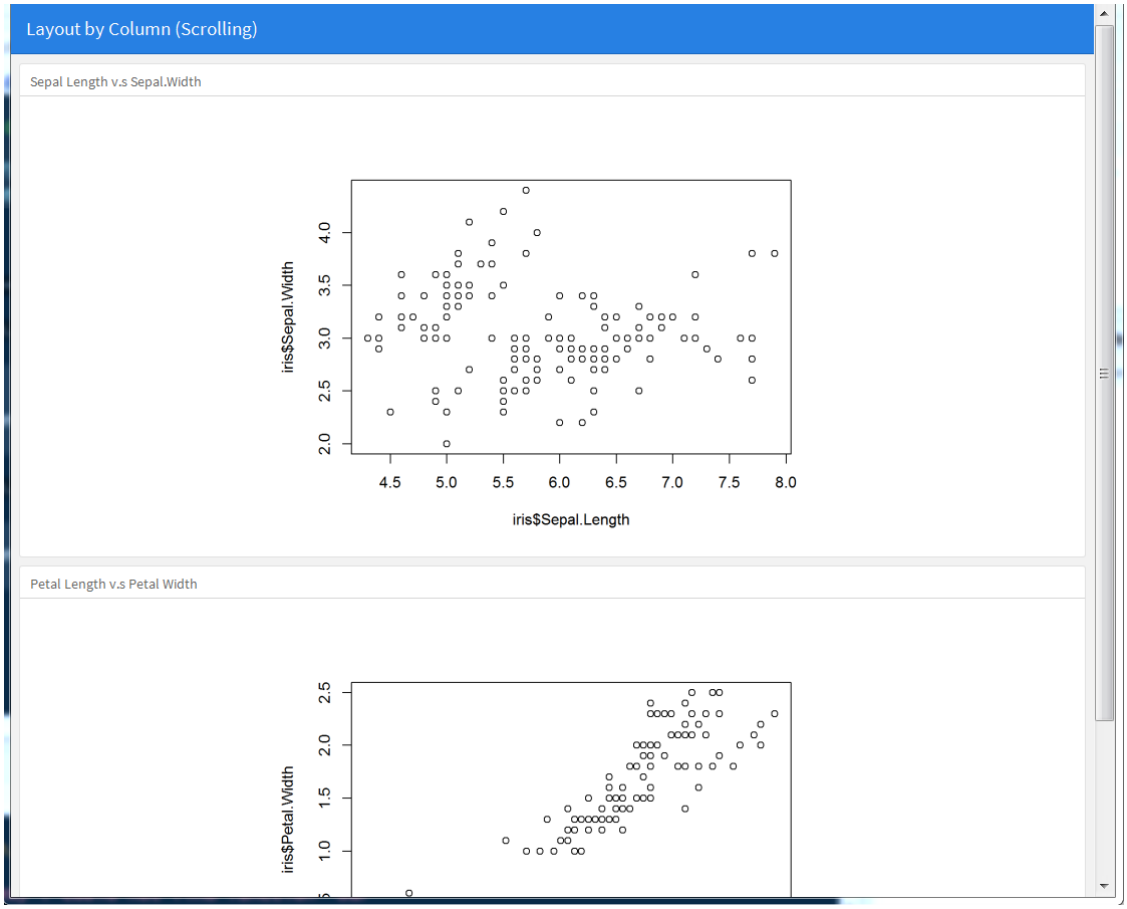
### Sepal Length v.s Sepal.Width

```{r}
data("iris")
plot(iris$Sepal.Length,iris$Sepal.Width)
```

### Petal Length v.s Petal Width

```{r}
plot(iris$Petal.Length,iris$Petal.Width)
```
````

The dashboard has one column with two charts:



6.5.1.4.Focal Chart

This layout fills the page completely and gives prominence to a single chart at the top or on the left. For example:

```
---
title: "Focal Chart (Top)"
output:
  flexdashboard::flex_dashboard:
    orientation: rows
---

Row 1 {data-height=500}
-----------------------------------------------------------------------

### Summary Table by Segment

```{r}
library(DataScienceR)
library(dplyr)
library(ggplot2)
library(DT)
data("SegData")
SegData%>%
  group_by(segment)%>%
  summarise( 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))%>%
  datatable()
```

Row 2 {data-height=500}

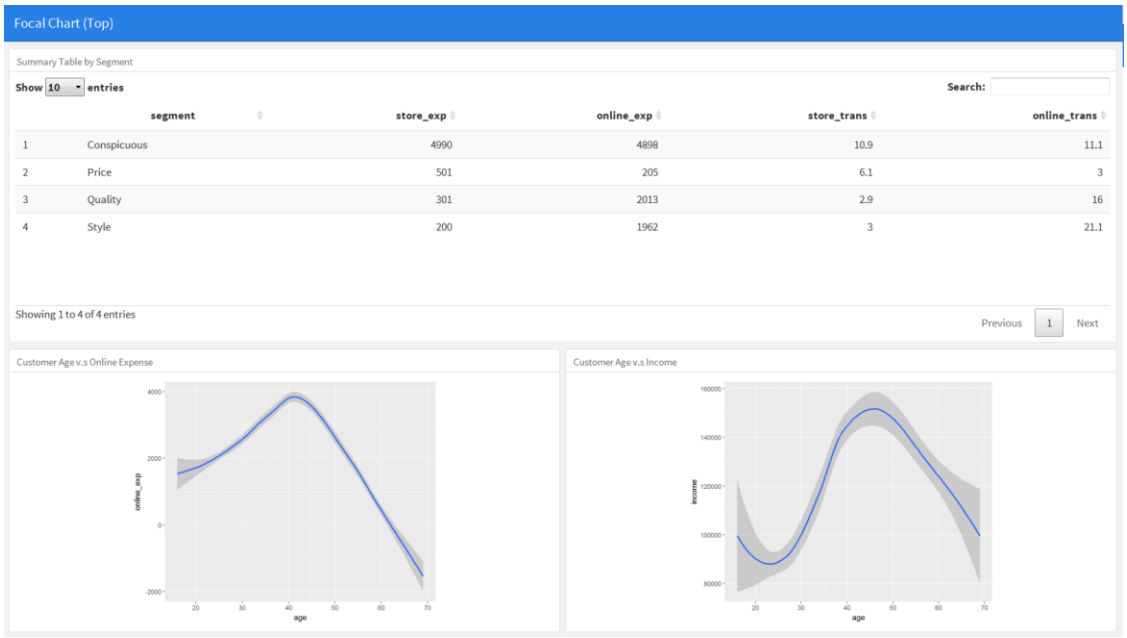
Customer Age v.s Online Expense

```{r}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,online_exp))+geom_smooth()
```

Customer Age v.s Income

```{r}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,income))+geom_smooth()
```
```

You can download the source code here. The resulted dashboard includes 3 charts. You can specify `data-height` attributes on each row to establish their relative sizes.



You can also give prominence to a single chart on the left such as:

```
---
title: "Focal Chart (Left)"
output: flexdashboard::flex_dashboard
---

Column 1 {data-width=600}
-------------------------------------

### Chart 1

```{r}
```

Column 2 {data-width=400}

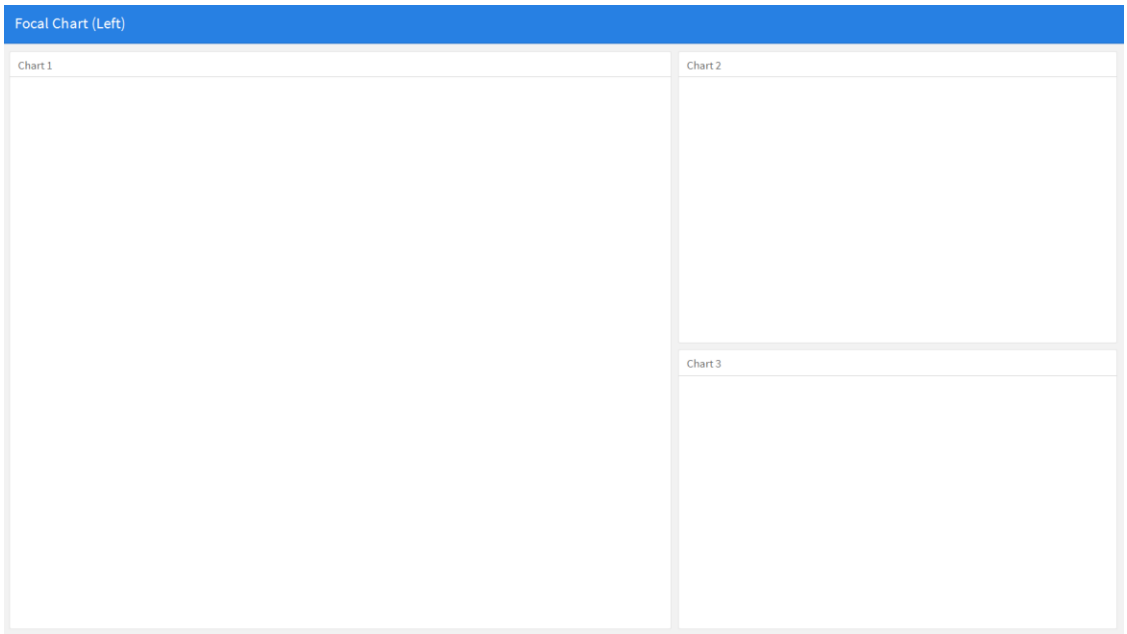
Chart 2

```{r}
```

Chart 3

```{r}
```
```

The resulted dashboard includes 3 charts:



6.5.1.5.Tabset

This layout displays column or row as a set of tabs. It is an alternative to scrolling layout when you have more charts. For example:

```
---
title: "Tabset Column"
output: flexdashboard::flex_dashboard
---

Column 1
-----------------------------------------------------------------------

### Summary Table by Segment

```{r}
library(DataScienceR)
library(dplyr)
library(ggplot2)
library(DT)
data("SegData")
SegData%>%
  group_by(segment)%>%
  summarise( 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))%>%
  datatable()
```

Column 2 {.tabset}

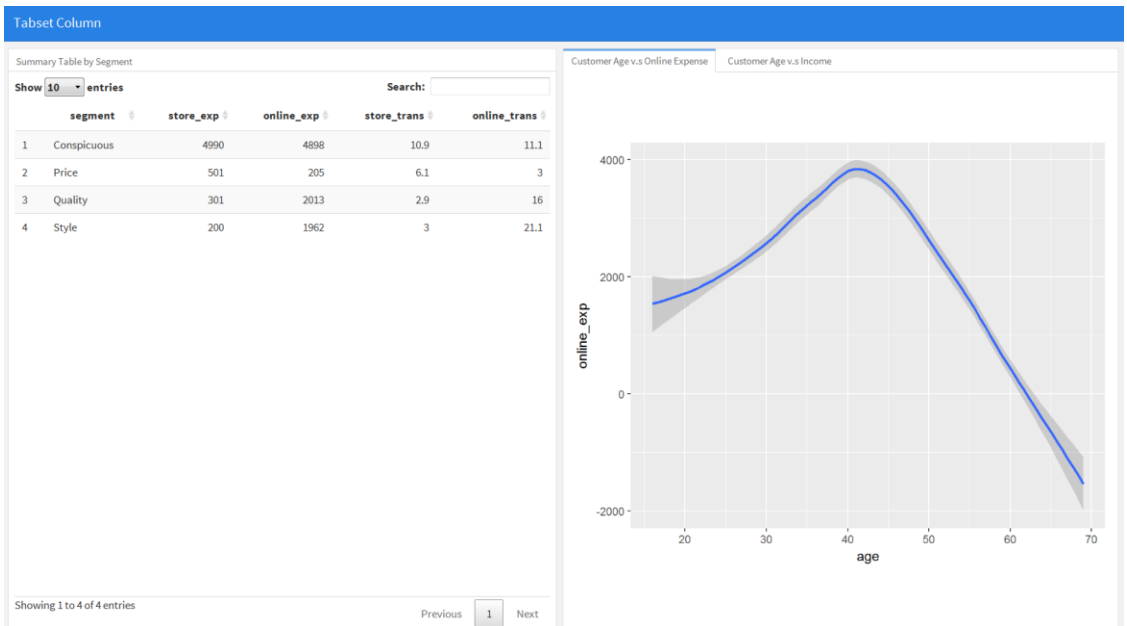
Customer Age v.s Online Expense

```{r}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,online_exp))+geom_smooth()
```

Customer Age v.s Income

```{r}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,income))+geom_smooth()
```
```

You can download the source code here. The dashboard displays the right column as a set of two tabs:



You can also add tabs to row:

```
---
title: "Tabset Row"
output:
  flexdashboard::flex_dashboard:
    orientation: rows
---

Row 1
-----------------------------------------------------------------------

### Summary Table by Segment

```{r}
library(DataScienceR)
library(dplyr)
library(ggplot2)
library(DT)
data("SegData")
SegData%>%
  group_by(segment)%>%
  summarise( 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))%>%
  datatable()
```

Row 2 {.tabset .tabset-fade}

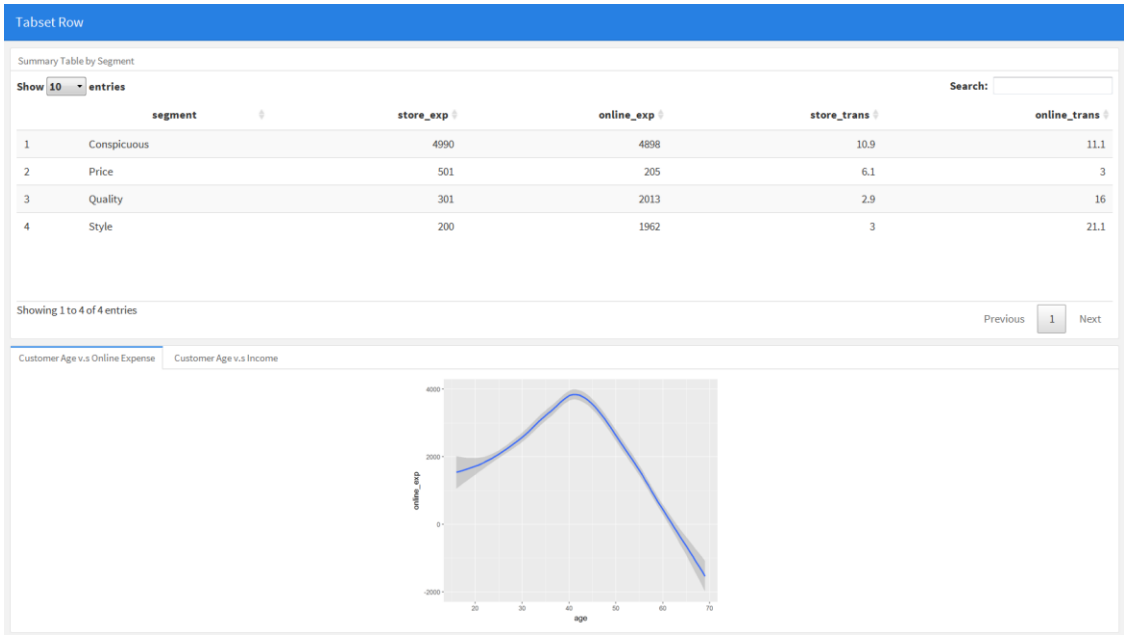
Customer Age v.s Online Expense

```{r}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,online_exp))+geom_smooth()
```

Customer Age v.s Income

```{r}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,income))+geom_smooth()
```
```

You can download the source code here. The dashboard displays the bottom row as a set of two tabs. Here the `{.tabset-fade}` attribute is used to enable a fade in/out effect when switching tabs:



6.5.1.6.Multiple pages

This layout defines multiple pages using (==========). Each page can have its own top-level navigation tab and orientation. You can set the orientation via the `data-orientation` attribute:

```
Page 1
====================================

Column 1 {data-width=600}
------------------------------------

### Chart 1


Column 2 {data-width=400}
------------------------------------

### Chart 2


Page 2 {data-orientation=rows}
====================================

Row 1 {data-height=600}
------------------------------------
```

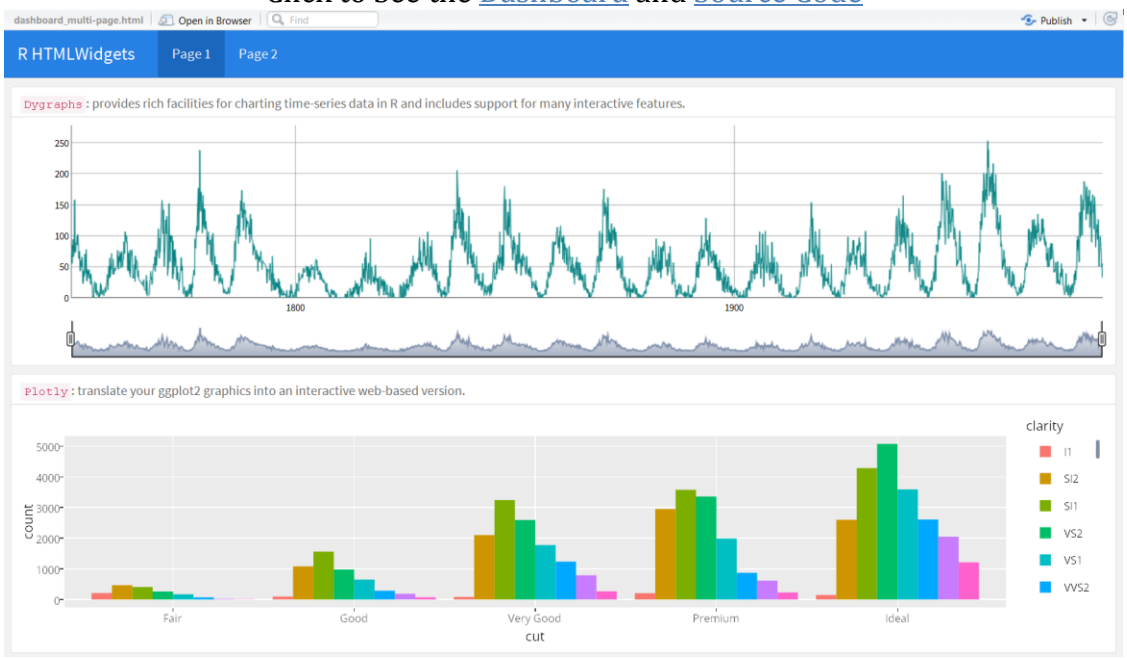
```
### Chart 1

Row 1 {data-height=600}
-------------------------------------

### Chart 2
```

You can easily build the following dashboard:

Click to See the Dashboard and Source Code



6.5.1.7.Storyboard

If you want to present a sequence of charts and related commentary, stroyboard will be a great choice.

```
---
title: "Storyboard Layout"
output:
  flexdashboard::flex_dashboard:
    storyboard: true
    social: menu
    source: embed
---

### Frame 1

```{r}

```

***

Some commentary about Frame 1.

Frame 2

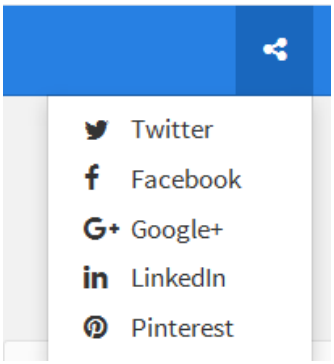
```{r}

```

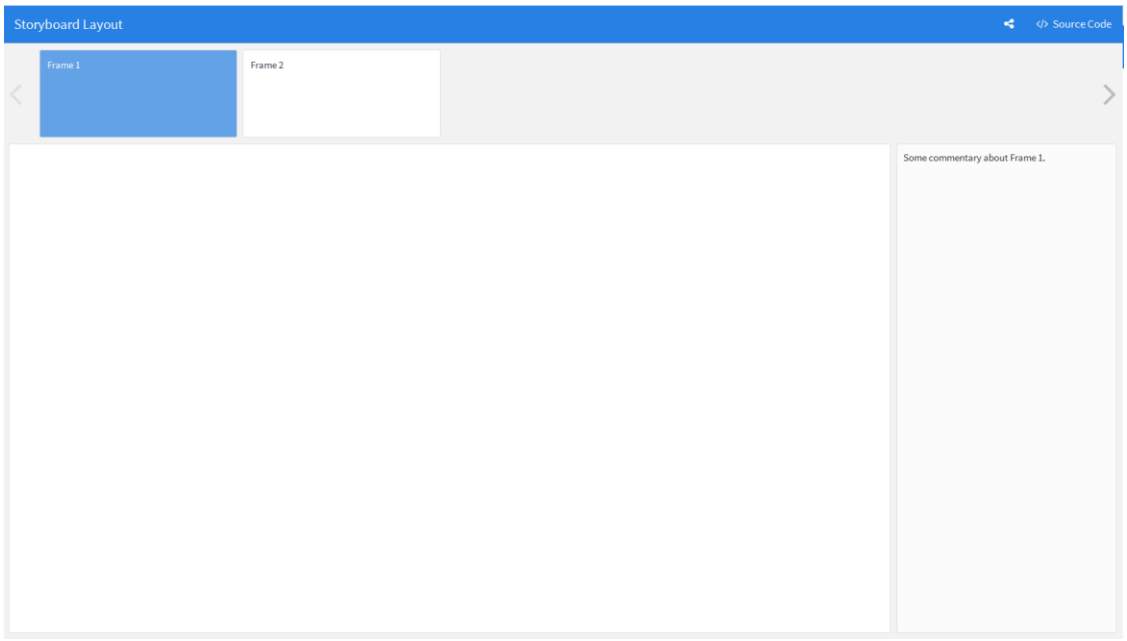
***

Some commentary about Frame 2.
```

You need to specify `storyboard: true`and additional commentary will show up alongside the storyboard frames (the content after the *** separator in each section). `social: menu` will enable an icon to share the storyboard to your social network:

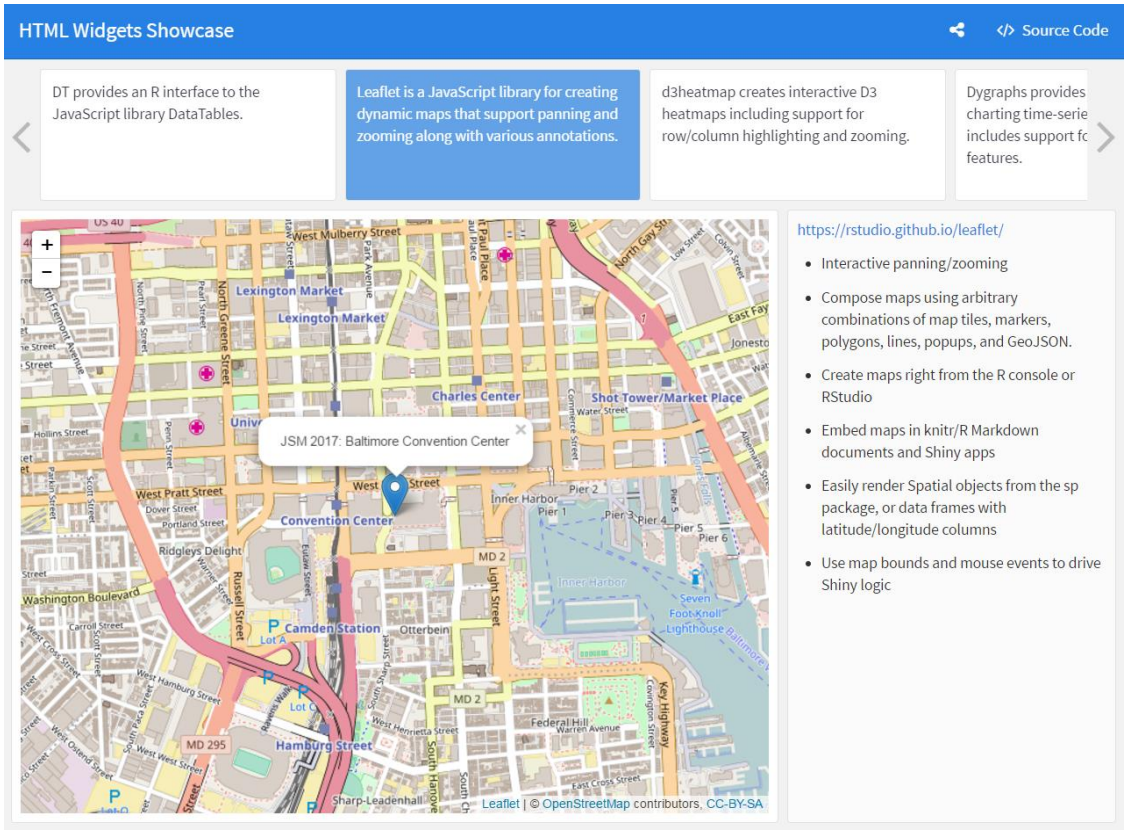


`source: embed` allows you to embed the source code. The layout is:



Here is an example of HTML Widgets Showcase storyboard. You can look at the source code by clicking "Source Code" tab at the upright corner.

See the storyboard here.



6.5.2.Components

6.5.2.1.HTML Widgets

The `htmlwidgets` framework brings JavaScript data visualization to R. The biggest advantage is the interactive character. As of writing this book, there are over 40 packages on CRAN which provide htmlwidgets. Charts based on `htmlwidgets` can dynamically re-size themselves so will fit within the bounds of their flexdashboard containers.

Some htmlwidgets:

- `DT`: provides an R interface to the JavaScript library DataTables
- `leaflet`: a JavaScript library for creating dynamic maps that support panning and zooming along with various annotations.
- `rbokeh`: an interface to Bokeh, a powerful declarative Bokeh framework for creating web-based plots.
- `d3heatmap`: creates interactive D3 heatmaps including support for row/column highlighting and zooming.
- `networkD3`: provides tools for creating D3 JavaScript network graphs from R

- `dygraphs`: provides rich facilities for charting time-series data in R and includes support for many interactive features.
- `plotly`: provides bindings to the plotly.js library and allows you to easily translate your ggplot2 graphics into an interactive web-based version.
- `metricsgraphics`: enables easy creation of D3 scatterplots, line charts, and histograms.
- `threejs`: provides interactive 3D scatterplots and globe plot

One disadvantage of htmlwidgets is that there may be a performance problem for larger datasets. Because they embed their data directly in their host web page. You can use standard R graphics in the case of a large dataset.

6.5.2.2.Standard R graphics

A static dashboard is also a great tool for story-telling. Standard R graphics are also scaled in static dashboard with the same aspect ratios. However, it is possible for the PNG images fill the bounds of their container seamlessly. To solve that problem, you can scale figure by defining knitr `fig.width` and `fig.height` values to approximate the actual size on the page. For example:

````
Row 1
-----------------------------------------------------------------------

### Online Expense v.s Store Expense

```{r, fig.width=10, fig.height=7}
library(DataScienceR)
library(dplyr)
library(ggplot2)
library(DT)
data("SegData")
SegData%>%
  ggplot(aes(online_exp,store_exp))%>%
  +geom_smooth()

```

Row 2
-----------------------------------------------------------------------

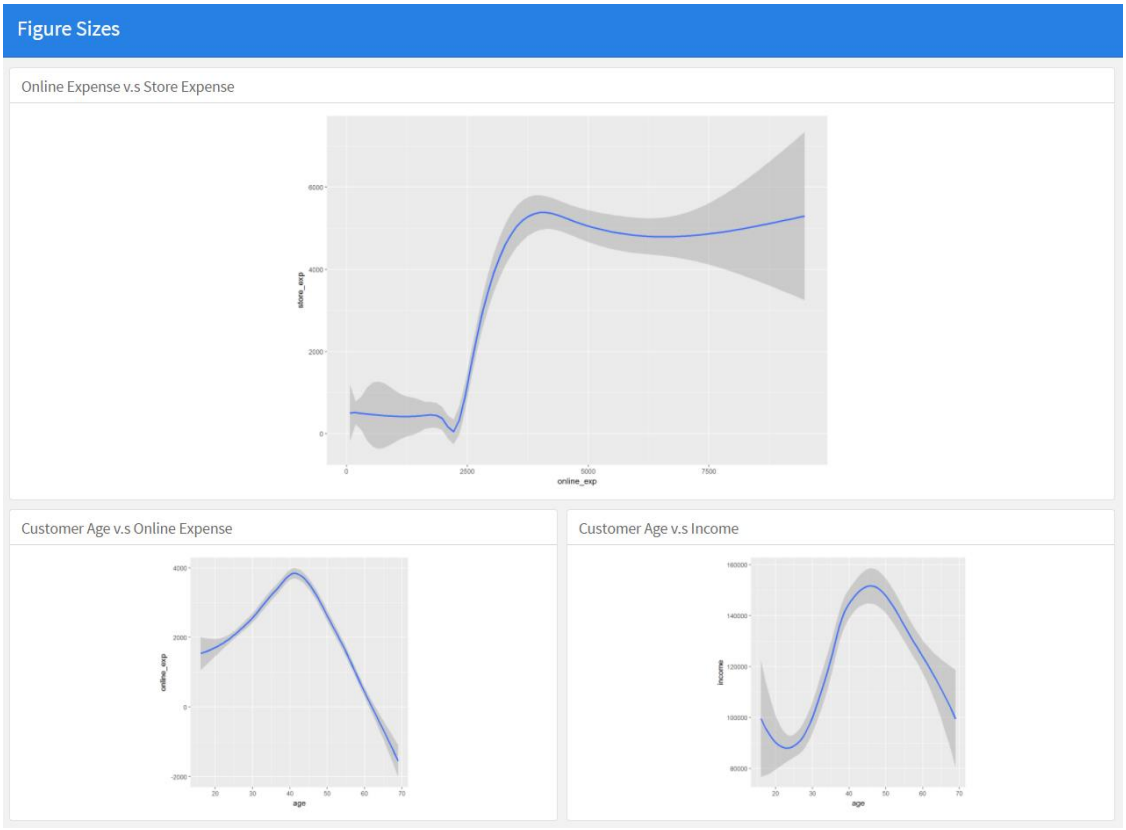
### Customer Age v.s Online Expense

```{r, fig.width=5, fig.height=5}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,online_exp))+geom_smooth()
```

### Customer Age v.s Income

```{r, fig.width=5, fig.height=5}
SegData%>%
  filter(age<100 & online_exp > 0)%>%
  ggplot(aes(age,income))+geom_smooth()
```
````

You can download the source code and see the complete output. Here is a screenshot of the output:



6.5.2.3.Tabular Data

Some of the previous examples included a DataTable component. It is interactive table that you can sort, filter and paginate. You can also display simple table. Here is an example of both:

```
---
title: "DataTable and Simple Table"
output:
  flexdashboard::flex_dashboard:
    orientation: rows
    vertical_layout: fill
---

```{r setup, include=FALSE}
library(flexdashboard)
```

Row 1

DataTable

```{r}
library(DataScienceR)
library(dplyr)
library(ggplot2)
library(DT)
data("SegData")
dat <- SegData%>%
  group_by(segment)%>%
  summarise( 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))
datatable(dat)
```

Row 2

Simple Table

```{r}
dat
```
```

You can download the source code and see the complete output.

6.5.2.4.Value Boxes

If you want to call out people's attention on one or more simple statistics in a dashboard, you can use the `valueBox` function. It allows you to display single values along with a title and optional icon. For example:

```
Row 1
-----------------------------------------------------------------------

### Conspicuous

```{r}
library(flexdashboard)
library(DataScienceR)
library(dplyr)
library(DT)
data("SegData")
value<-table(SegData$segment)
valueBox(value[1], icon = "fa-pencil", color = "success")
```

Price Sensitive

```{r}
valueBox(value[2], icon = "fa-pencil", color = "warning")
```

Quality

```{r}
valueBox(value[3], icon = "fa-pencil", color = "primary")
```

Style

```{r}
valueBox(value[4], icon = "fa-pencil",  color = "info")
```
```

You can download the source code and see the complete output. Here is a screenshot of part of the output:

The `valueBox` function will emit a value with a specified icon (`icon =`) and color (`color =`).

Specify Icon

There are three different icon sets you can refer to. You should specify it's full name including the prefix to `icon` parameter (e.g `"icon = "fa-pencil","icon = ion-social-twitter"`, etc.) :

- Font Awesome Icons
- Ionicons

- BooBootstrap Glyphicons

Specify Color

You can specify color using `color` parameter (e.g. `color = "success"`). Available colors include "primary", "info", "success", "warning", and "danger" (the default is "primary"). You can also specify and valid CSS color (e.g. "#ffffff", "rgb(100,100,100)", etc.)

6.5.2.5.Gauges

If your value is within a specified range such as percentage, it is more intuitive to use gauges.

```
Row 1
-----------------------------------------------------------------------

### Conspicuous

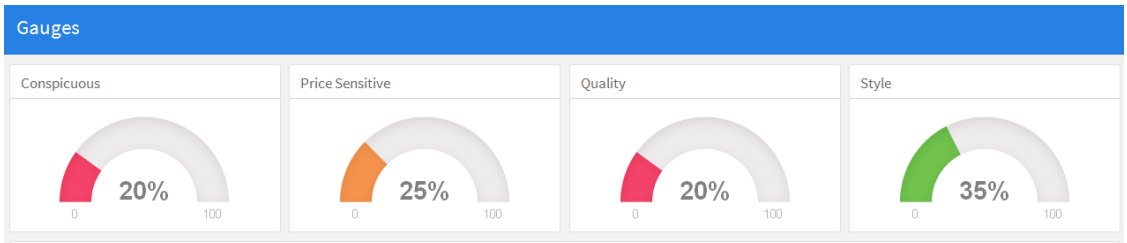
```{r}
library(flexdashboard)
library(DataScienceR)
library(dplyr)
library(DT)
data("SegData")
value <- table(SegData$segment)
value <- as.vector(round(value/sum(value),2)*100)

gauge(value[1], min = 0, max = 100,  symbol = '%', gaugeSectors(
  success = c(30, 100),warning = c(21, 29), danger = c(0, 20)
))
```

Price Sensitive

```{r}
gauge(value[2], min = 0, max = 100, symbol = '%', gaugeSectors(
  success = c(30, 100),warning = c(21, 29), danger = c(0, 20)
))
```
```

Output:



You can download the source code and see the complete output.

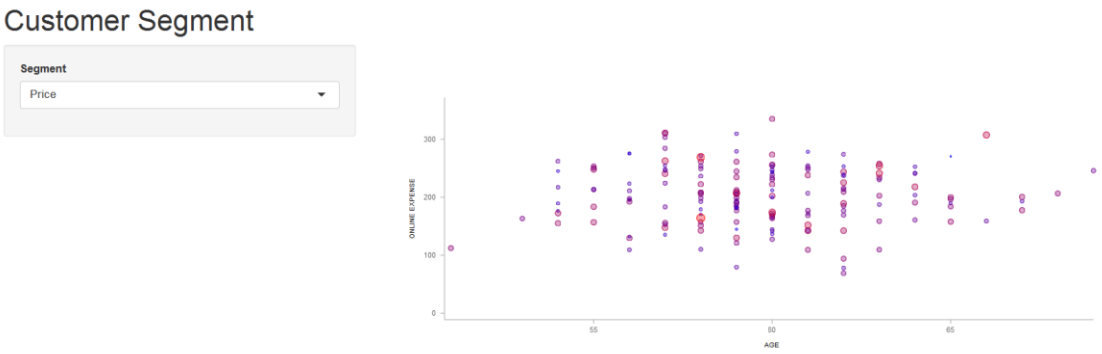
Those are the main components in a dashboard. More information about flesdashboard for R, refer to "flexdashboard: Easy interactive dashboards for R".

6.6.Shiny Dashboard

6.6.1.Brief Introduction to Shiny

Shiny is a web application framework for R that can help turn your analyses into interactive web applications. It is easy to learn and use. It doesn't require HTML, CSS, or JavaScript knowledge. This section will demonstrate two examples to help you understand the basic structure of a Shiny App. With some basic understanding, the next section will show how to include shiny in a dashboard.

Example 1: Customer Segment Plot



The Customer Segment example is a simple application that plots the clothes customer data by segments using htmlwidget `metricsgraphics`. Type the following code to run the example:

```
library(shiny)
source("https://raw.githubusercontent.com/happyrabbit/linhui.org/gh-
pages/CE_JSM2017/Examples/shiny1.R")
shinyApp(ui = ui, server = server)
```

A Shiny app contains two parts:

- `ui`: It defines user interface and controls the outlook of the web page.
- `server`: It includes the backend manipulation of the input.

The source code for both of the components is:

```
library(shiny)
library(dplyr)
library(metricsgraphics)

sim.dat<-
readr::read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR
/master/Data/SegData.csv")%>%
  filter(!is.na(income) & age<100)
```

```
# Define UI for application that draws a metricsgraphics interactive plot

ui <- pageWithSidebar(

  # inpute the panel header
  headerPanel('Customer Segment'),

  # sidebar with input for customer segment
  sidebarPanel(
    selectInput('seg', 'Segment', unique(sim.dat$segment))
  ),

  # show a metricsgraphics plot
  mainPanel(
    metricsgraphicsOutput('plot1')
  )
)

# Define server logic required to draw a metricsgraphics plot
server <-  function(input, output) {

  # Expression that generates a metricsgraphics The expression is
  # wrapped in a call to renderMetricsgraphics to indicate that:
  #
  #  1) It is "reactive" and therefore should be automatically
  #     re-executed when inputs change
  #  2) Its output type is a renderMetricsgraphics

  # select the part of data needed
  selectedData <- reactive({
    dplyr::filter(sim.dat, segment == input$seg)
  })

  # render plot
  output$plot1 <- renderMetricsgraphics({
    mjs_plot(selectedData(), x= age, y=online_exp) %>%
      mjs_point(color_accessor=income, size_accessor=income) %>%
      mjs_labs(x="Age", y="Online Expense")
  })

}

# Run the application
 shinyApp(ui = ui, server = server)
```

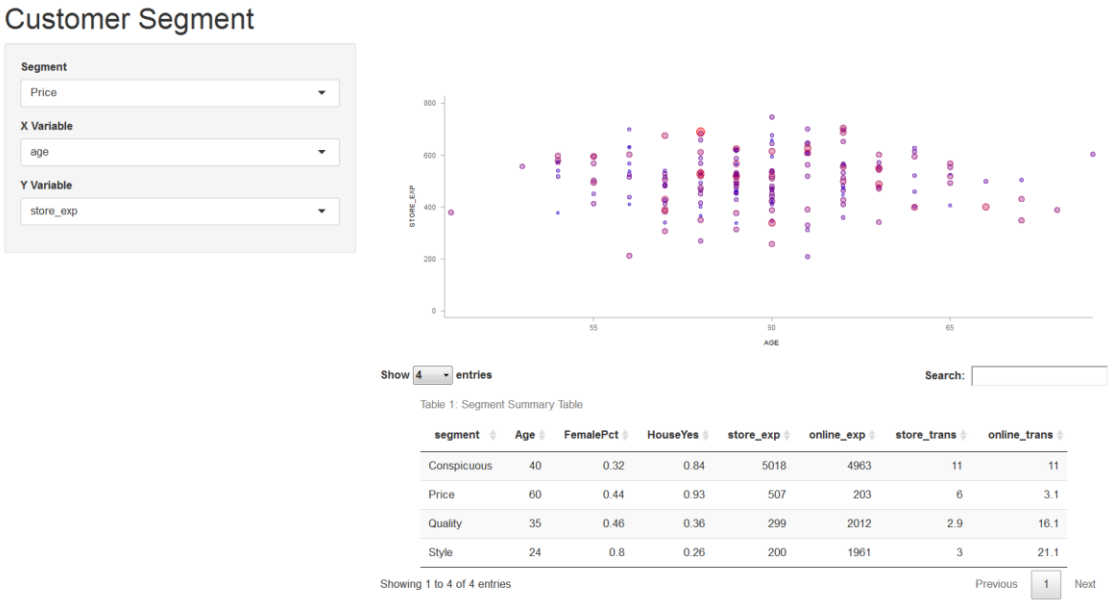
The example here has a single character input specified using a slider and a single `metricsgraphics` plot output. The server-side of the application generates a metricsgraphics

plot. Notice that the code generating the plot is wrapped in a call to `renderMetricsgraphics`. There are different render calls in Shiny:

- `renderDataTable`
- `renderImage`
- `renderPlot`
- `renderPrint`
- `renderTable`
- `renderText`
- `renderUI`

You can choose the appropriate one as needed. The next example is a little more complicated with more input controls. You may be confused by the `reactive` expression in example 1. Don't worry. We will explain the use in the next example.

Example 2: Customer Segment Plot and Summary Table



Example 2 demonstrates how to include multiple inputs and render both table and graphic using htmlwidgets. Type the following code to run the application:

```
library(shiny)
source("https://raw.githubusercontent.com/happyrabbit/linhui.org/gh-
pages/CE_JSM2017/Examples/shiny2.R")
shinyApp(ui = ui, server = server)
```

This example has a little more going on:

- three inputs: (1) customer segment; (2) x-axis variable of the plot; (3) y-axis variable of the plot
- two outputs: (1) a table on HTML pages with filtering, pagination, sorting features in the table; (2) a `metricsgraphics` plot

```
library(shiny)
library(dplyr)
library(DT)
library(metricsgraphics)

sim.dat<-
readr::read_csv("https://raw.githubusercontent.com/happyrabbit/DataScientistR
/master/Data/SegData.csv")%>%
  filter(!is.na(income) & age<100)

# Define UI for application that draws a histogram
ui <- pageWithSidebar(
  headerPanel('Customer Segment'),

  sidebarPanel(
    selectInput('seg', 'Segment', unique(sim.dat$segment)),
    selectInput('xcol', 'X Variable', c("age")),
    selectInput('ycol', 'Y Variable',
c("store_exp","online_exp","store_trans","online_trans"))
  ),

  mainPanel(
    metricsgraphicsOutput('plot1'),

    dataTableOutput("summary")
  )
)

# Define server logic required to draw a histogram
server <-  function(input, output) {

  # Combine the selected variables into a new data frame
  selectedData <- reactive({
    dplyr::filter(sim.dat, segment == input$seg)
  })

  output$plot1 <- renderMetricsgraphics({
    mjs_plot(selectedData(), x= input$xcol, y=input$ycol) %>%
      mjs_point(color_accessor=income, size_accessor=income) %>%
      mjs_labs(x=input$xcol, y=input$ycol)
  })

  # Generate a summary of the dataset
```

```
  output$summary <- renderDataTable({
    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))%>%
      data.frame()%>%
      datatable( rownames = FALSE,
                 caption = 'Table 1: Segment Summary Table',
                 options = list(
                   pageLength = 4,
                   autoWidth = TRUE)
      )

  })

}

# Run the application
shinyApp(ui = ui, server = server)
```

There are three `selectInput` calls in the user interface definition. Inside the `mainPanel()`, there are two calls `metricsgraphicsOutput` and `dataTableOutput`.

The server side also has some new elements. There are:

- A `reactive` expression to return the subset of data according to user's choice
- Rendering expression `renderMetricsgraphics` return the `output$plot1`
- Rendering expression `renderDataTable` return the `output$summary`

It is important to understand the concept of reactivity. The fundamental feature of Shiny is interactivity which means the output will change with input. The process is:

1. user provide input
2. backend R code will run using the input
3. report an output back to user

The changing step is through reactive programming. For more details about reactive programming, see the Reactivity Overview. RStudio provides an excellent Shiny tutorial from beginning to deep level: http://shiny.rstudio.com/tutorial/.

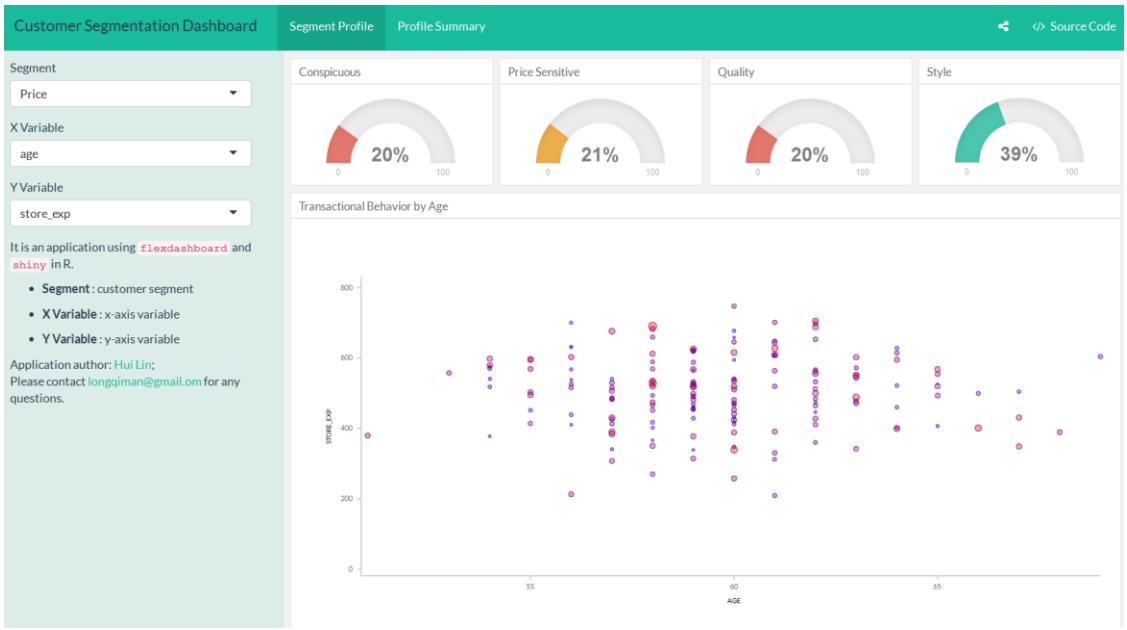
6.6.2.Using `shiny` with `flexdashboard`

You can also create a dashboard that enables viewers to change underlying parameters and see the corresponding results. You can add `shiny` to `flexdashboard` by specifying `runtime: shiny` in the front-matter of your document.

Then add one or more input controls and reactive expressions as in shiny. The difference is that when you add shiny function to flexdashboard, there is no need to use wrap the code to two components, `ui` and `server`. In that sense, using `shiny` in `flexdashboard` is easier than building Shiny App itself.

An alternative way to dashboards with Shiny is to use shinydashboard package. Example: Customer Segmentation Dashboard

Here is an example dashboard using the clothes customer data.



The control part of the source code is (analogy to `ui` in `shiny`):

```
Sidebar {.sidebar data-width=350}
=====================================================================

```{r}
selectInput('seg', 'Segment', unique(sim.dat$segment))
selectInput('xcol', 'X Variable', c("age"))
selectInput('ycol', 'Y Variable',
c("store_exp","online_exp","store_trans","online_trans"))
```
```

The reactive expressions are (analogy to `server` in `shiny`):

````
Row {data-width=650}
-----------------------------------------------------------------------

### Transactional Behavior by Age

```{r}

selectedData <- reactive({
    dplyr::filter(sim.dat, segment == input$seg)
  })

renderMetricsgraphics({
    mjs_plot(selectedData(), x= input$xcol, y=input$ycol) %>%
      mjs_point(color_accessor=income, size_accessor=income) %>%
      mjs_labs(x=input$xcol, y=input$ycol)
  })

```
````

Click here to download the complete source code. Here is the app.

7. SOFT SKILLS FOR DATA SCIENTISTS

7.1.Comparison between Statistician and Data Scientist

Statistics as a scientific area can be traced back to 1749 and statistician as a career has been around for hundreds of years with well-established theory and application. Data Scientist becomes an attractive career for only a few years along with the fact that data size and variety beyond the traditional statistician's toolbox and the fast growing of computation power. Statistician and data scientist have a lot of common backgrounds, but there are also some significant differences.

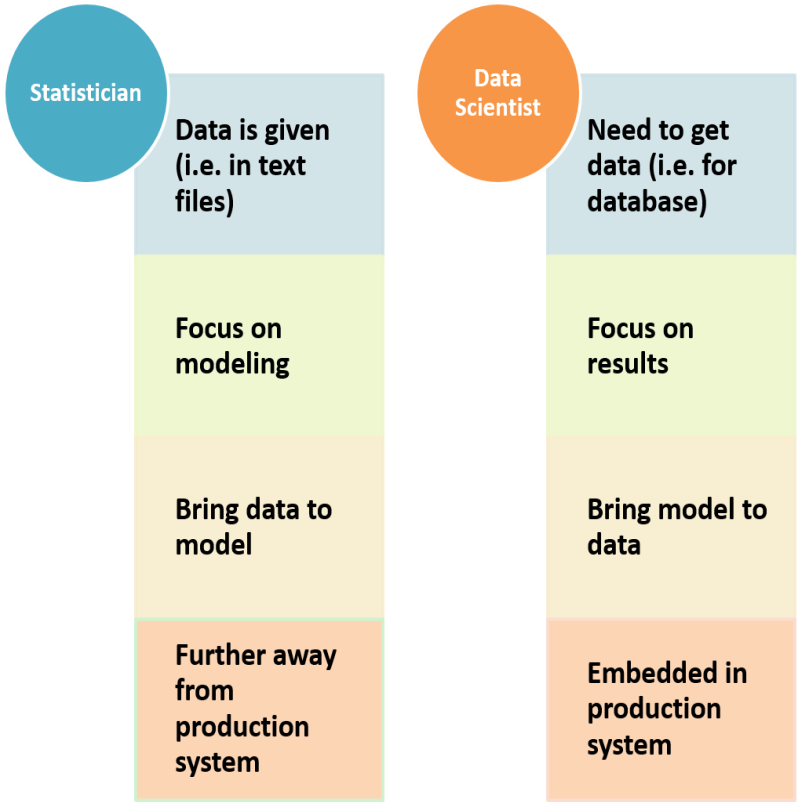


Figure 71 Comparison of Statistician and Data Scientist

Both statistician and data scientist work closely with data. For the traditional statistician, the data is usually well-formatted text files with numbers and labels. The size of the data usually can be fitted in a PC's memory. Comparing to statisticians, data scientists need to deal with more varieties of data: well-formatted data stored in a database system with size much larger than a PC's memory or hard-disk; huge amount of verbatim text, voice, image, and video; real-time streaming data and other types of records. One particular power of statistics is that statistician can fit model and make an inference based on limited data. It is quite common that once the data is given and cleaned, the majority of the work is developed

different models around the data. Today, data is relatively abundant, and modeling is just part of the overall effort. The focus is to deliver actionable results. Different from statisticians, data scientists, sometimes need to fit model on the cloud instead of reading data in since the data size is too large. From the entire problem-solving cycle point of view, statisticians are usually not well integrated with the production system where data is obtained in real time; while data scientists are more embedded in the production system and closer to the data generation procedures.

7.2.Where Data Science Team Fits?

During the past decade, a huge amount of data has become available and readily accessible for analysis in many companies across different business sectors. The size, complexity, and velocity of increment of data suddenly beyond the traditional scope of statistical analysis or BI reporting as mentioned above. To leverage the big data, many companies have established new data science organizations. Companies have gone through different paths to create their own data science and machine learning organizations. There are three major formats of data science teams:

(1) independent of any current organizations and the team report directly to senior leadership;

(2) within each business unit and the team report to business unit leaders;

(3) within in the traditional IT organizations and the team report to IT leaders.

Companies are different in many aspects, but in general, the most efficient option to mine big data is a team of data scientist independent of business units and IT organizations. The independence enables the data science team to collaborate across business units and IT organizations more efficiently and the independence also provides flexibility and potential to solve corporate level strategic big data problems. For each business units, there are many business unit specific data science related problems and embedding data scientist within each business units is also an efficient way to solve business unit specific data science problems. The full cycle of data science projects from data to decision (i.e. Data -> Information -> Knowledge -> Insight -> Decision) is relatively difficult to achieve if the data science team is part of traditional IT organizations.

7.3.Beyond Data and Analytics

Data scientists usually have a good sense of data and analytics, but data scientist project is definitely more than just data and analytics. A data science project may involve people with many different roles:

- a business owner or leader to identify opportunities in business value; program managers to ensure each data science project fit into the overall technical program development;
- data owners and computation resource and infrastructure owners from IT department;

- dedicated team to make sure the data and model are under model governance and privacy guidelines;
- a team to implement, maintain and refresh the model;
- project managers to coordinate all parties to set periodical tasks so that the project meets the preset milestones and delivery results;
- multiple rounds of discussion of resource allocation (i.e. who will pay for the data science project).

Effective communication and in-depth domain knowledge about the business problem are essential requirements for a successful data scientist. A data scientist will interact with people at various levels ranging from senior leaders who are setting the corporate strategies to front line employees who are doing the daily work. A data scientist needs to have the capability to view the problem from 10,000 feet above ground, as well as down to the detail to the very bottom. To convert a business question into a data problem, a data scientist needs to communicate using the language the other people can understand and obtain the required information.

7.4.Data Scientist as a Leader

During the entire process of data science project defining, planning, executing and implementation, the data scientist lead needs to be involved in every step to ensure the business problem is defined correctly and the business value and success metric are evaluated reasonable. Corporates are investing heavily in data science and machine learning with a very high expectation of big return. There are too many opportunities to introduce unrealistic goal and business impact for a particular data science project. The leading data scientist need to be the leader in these discussions to define the goal backed by data and analytics. Many data science projects over promise in deliverables and too optimistic on the timeline and these projects eventually fail by not delivering the preset business impact within the timeline. As the data scientist in the team, we need to identify these issues early in the stage and communicate to the entire team to make sure the project has a realistic deliverable and timeline. The data scientist team also need to work closely with data owners to identify relevant internal and external data source and evaluate the quality of the data; as well as working closely with the computation infrastructure team to understand the computation resources (i.e. hardware and software) available for the data science project.

7.5.Three Pillars of Knowledge

The following picture summarizes the needed three pillars of knowledge to be a successful data scientist.

(1) A successful data scientist needs to have a strong technical background in data mining, statistics and machine learning. The in-depth understanding of modeling with the insight about data enable a data scientist to convert a business problem to a data science problem.

(2) A successful data scientist needs some domain knowledge to understand business problem. For any data science project, the data scientist need to collaborate with other team members and effective communication and leadership skills are critical, especially when you are the only data person in the room and you need to make a decision with uncertainty.

(3) The last pillar is about computation environment and model implementation in big data platform. This used to be the most difficult one for a data scientist with statistics background (i.e. lack computer science or programming skills). The good news is that with the rise of cloud computation big data platform, this barrier is getting easier for a statistician to overcome and we will discuss in more detail in next chapter.

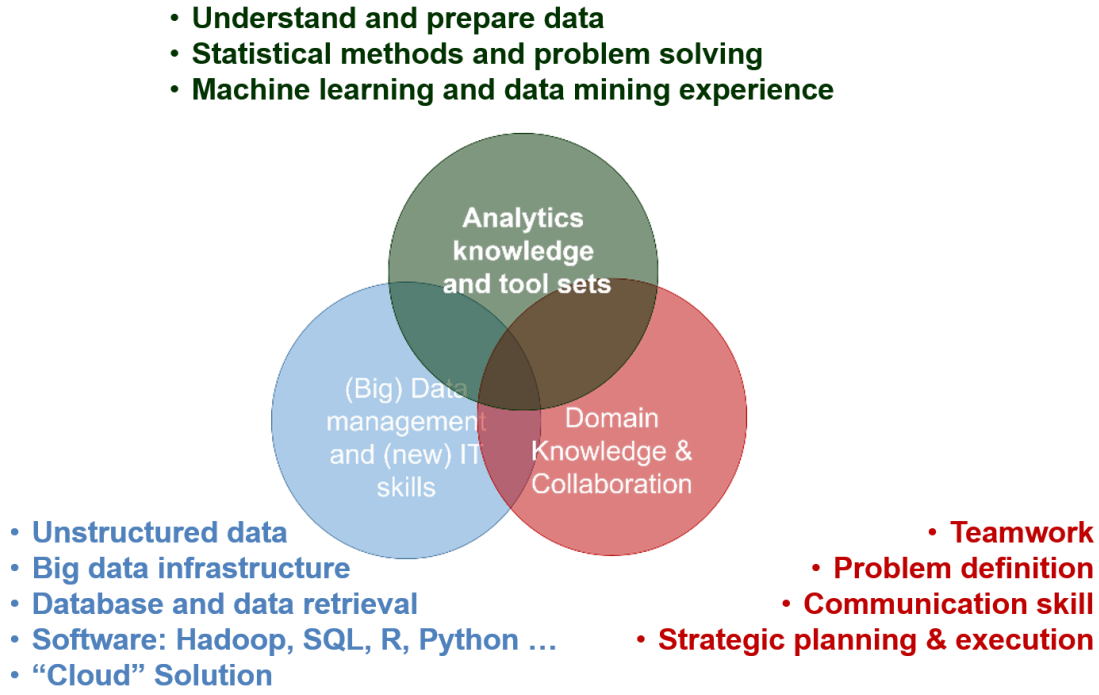


Figure 72 Three Pillars for Data Scientist

7.6.Common Pitfalls of Data Science Projects

Data science projects are usually complicated in nature, and many of these data science projects eventually fail due to various reasons. We will briefly discuss a few common pitfalls in data science projects and how to avoid them.

- **Solve the wrong problem:** data science project usually starts with a very vague description and a few rounds of detailed discussion with all stakeholders involved are needed to define the busses problem. There will be lots of opportunities to introduce misalignment when mapping the business problem into specific data science methods. Especially when the quality and availability of the data are not as good as what is expected at the first place. If not well-communicated during the project, the final data science solution may not be the right one to solve the business problem. As the data

scientist (sometimes the only data scientist) in the room, we must understand the business problem thoroughly and communicate regularly to business partners especially there is a change of status to make sure everyone is aligned with the progress and final deliverables.

- **Over promise on business value:** business leaders usually have high expectation on data science projects and the goal of business value and deliverables sometimes are set unrealistic and eventually beyond the scope of available data and computation resource. As the data scientist (sometimes the only data scientist) in the room, we must have our voice heard based on fact (i.e. data, analytics, and resources) instead of wishful thinking. Backed with fact-based evidence, it is easier to communicate what is a realistic goal for the entire team.

- **Too optimistic about the timeline:** there are lots of uncertainties in data science projects such as the data source availability and data quality, computation hardware and software, resource availability in the business team, implementation team and IT department, as well as project direction change which may delay the final delivery date. To have a better-estimated timeline, get as much detail as possible for all the needed tasks and estimated each task individually and reach out to each team member to confirm their availability. Most importantly, communicate with the entire team if there are blocking factors for the project in a prompt way such that everyone aware of the situation and potential impact on the timeline.

- **Too optimistic about data availability and quality:** the most important asset in data science project is data. Even though we are at the big data age, often times there is not enough relevant data for the data science projects. The data quality is also a general problem for data science projects. A thorough data availability and quality check are needed at the beginning of the data science project to estimate the needed effort to obtain data as well as data cleaning.

- **Model cannot be scaled:** be careful if you use a subset of data to fit the model and then scale it to the entire dataset. When developing the model using a smaller dataset, we must keep in mind how much computation resources needed for the whole dataset. With limited computation resource, it is important to maximize the computation time in production to a reasonable level based on the business application when fits the model with a sample dataset.

- **Take too long to fail:** data science projects usually are trying to push the boundary of current applications to new territory, people do not expect all data science projects to be successful. Fail fast is generally good practice such that we can quickly find a better way to solve the problem. A data scientist needs to have an open mindset to not stuck with one idea or one approach for a long time to avoid taking too long to fail.

8. REFERENCES

1. T, H.: The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence. 13, 340–354 (1998)

2. Y, A., D, G.: Shape quantization and recognition with randomized trees. Neural Computation. 9, 1545–1588 (1997)

3. L, V.: A theory of the learnable. Communications of the ACM. 27, 1134–1142 (1984)

4. M, K., L, V.: Cryptographic limitations on learning boolean formulae and finite automata, (1989)

5. Ton de Waal, S.S., Jeroen Pannekoek: Handbook of statistical data editing and imputation. John Wiley & Sons (2011)

6. Saar-Tsechansky M, P.F.: Handling missing values when applying classification models. Journal of Machine Learning Research. 8, 1625–1657 (2007b)

7. L, B.: Bagging predictors. Machine Learning. 24, 123–140 (1966a)

8. Jolliffe, I.: Principla component analysis. New York: Springer (2002)

9. Geladi P, K.B.: Partial least-squares regression: A tutorial. Analytica Chimica Acta. 1–17 (1986)

10. Mulaik, S.: Foundations of factor analysis. Boca Raton: Chapman&Hall/CRC (2009)

11. Box G, C.D.: An analysis of transformations. Journal of the Royal Statistical Society. Series B (Methodological), 211–252 (1964)

12. Iglewicz, B., Hoaglin, D.: How to detect and handle outliers. The ASQC Basic References in Quality Control: Statistical Techniques. 16, (1993)

13. Serneels S, E.P., Nolf ED: Spatial sign pre-processing: A simple way to impart moderate robustness to multivariate estimators. Journal of Chemical Information and Modeling. 46, 1402–1409 (2006)

14. Max Kuhn, K.J.: Applied predictive modeling. Springer (2013)

9. APPENDIX: SIMULATE SPARK SYSTEM IN R-STUDIO

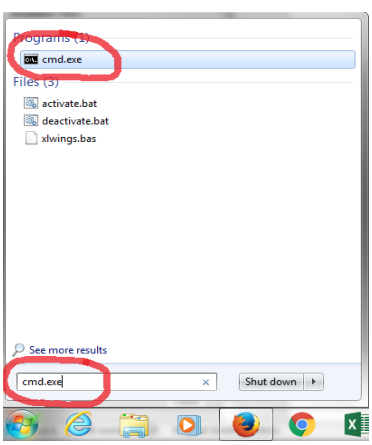
All the demo steps below are based on Windows 7 operating system!

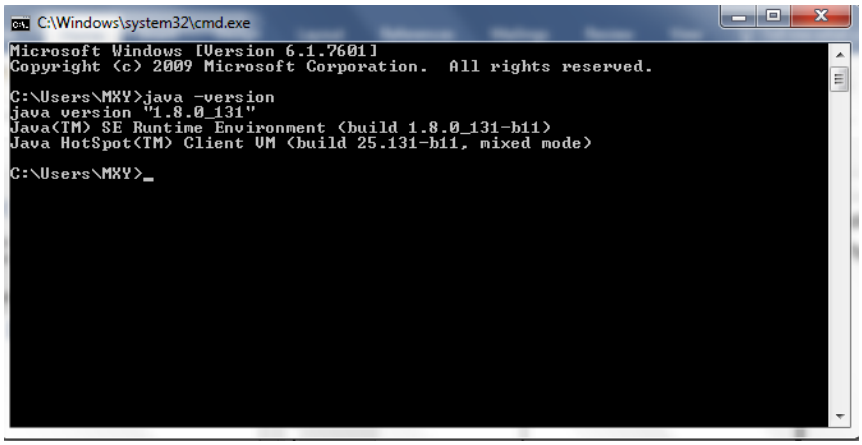
9.1.Install Required Software

We will need to install the following software in your laptop or desktop before we can run the codes included in this section:

- **R** which can be download at https://cran.rstudio.com/bin/windows/base/ and install R first
- **R Studio** can be download at https://www.rstudio.com/products/rstudio/download/ and once R is installed successfully, then install R Studio
- **Java 7.0 and above is needed,** which can be download at https://www.java.com/en/download/help/windows_offline_download.xml

After Java installation, please restart your computer. To test whether Java is installed successfully or not, we can open the command line window by typing in *cmd.exe* in the search box in the start menu as shown in the below graph (left). Once the command line window is open, type in *java -version* and we should see the version information in the below graph (right).





9.2.Leverage Hadoop and Spark Parallel using R Studio

R is a powerful tool for data analysis given the data can be fit into memory. Because of the memory bounded dataset limit, R itself cannot be used directly for big data analysis where the data is likely stored in Hadoop and Spark system. By leverage ***sparklyr*** package created by R Studio, we can connect R to Spark to analyze data stored in Spark system where data are across different nodes and computation are parallel in nature to use the collection of memory units across all nodes. And the process is relative simple. In this section, we will illustrate how to use R Studio for big data analysis on top of Spark environment through ***sparklyr*** package. For illustration purpose, we will install a local Spark version such that

even though there are no Spark cluster, but we can illustrate these big data analytics concepts.

9.2.1.Library Installation

First, we need to install ***sparklyr*** package which enables the connection between master or local node to Spark cluster environments. As it will install more than 10 dependencies, it may take more than 5 minutes to finish. Be patient while it is installing! Once the installation finishes, load the ***sparklyr*** package as illustrated by the following code:

```
# Installing sparklyr takes a few minutes,
# because it installs +10 dependencies.
if (!require("sparklyr")) {
  install.packages("sparklyr")
}
# Load sparklyr package.
library(sparklyr)
```

9.2.2.Create Connection

For illustration purpose, we use spark_install() function to download a local version of spark and let R to manage Spark related settings. Please be patient and it will take a few minutes to download and install. In real applications, we need to connect R Studio to a Spark Cluster.

```
## download and create a local Spark environment
spark_install(version = "1.6.2")
```

Once the local Spark environment is installed successfully, we need create a Spark Connection to link R Studio to Spark environment. As we are using a local Spark system, the *master* parameter is set to "local". In enterprise environment, please consult your administrator for details. The created Spark Connection (i.e. **sc**) will be the pipe that connect R Studio to the Spark Cluster. We can think of R Studio is running on a particular computation node which has its local memory and CPU and Spark is a collection of computation nodes with their own memory and CPU and Spark data frame are stored across Spark nodes. The Spark Connection can be established with:

```
# create a sparklyr connection
sc <- spark_connect(master = "local")
```

9.2.3.Sample Dataset

To simplify the learning process, let us use a very familiar dataset: the ***iris*** dataset. It is part of the *dplyr* library and let's load that library to use the ***iris*** data frame. Here the ***iris*** dataset is still in the local node where the R studio is running on. And we can see that the first a few lines of the iris dataset below the code after running:

```
library(dplyr)
```

```
head(iris)
```

9.2.4.IMPORTANT - Copy Data to Spark Environment

In real applications, your data is usually very big and cannot fit into one hard disk and it is very likely your data is already in Hadoop/Spark ecosystem. You can use SparkDataFrame to analyze your data in Spark system directly. Here, we illustrate how to copy a local dataset to Spark environment and then work on that dataset in the Spark system. As we have already created the Spark Connection ***sc***, it is fairly simple to copy data to spark system by ***sdf_copy_to***() function as below:

```
iris_tbl <- sdf_copy_to(sc = sc, x = iris, overwrite = T)
```

The above one line code copies ***iris*** dataset from local node to Spark cluster environment where ***sc*** is the Spark Connection we just created; ***x*** is the data frame that we want to copy; and ***overwrite*** is the option whether we want to overwrite the target object if the same name SparkDataFrame exists in the Spark environment. Finally ***sdf_copy_to***() function will return an R object wrapping the copied SparkDataFrame. So ***irir_tbl*** can be used to refer to the ***iris*** SparkDataFrame.

To check whether the iris dataset was copied to Spark environment successfully or not, we can use ***src_tbls***( ) function to the Spark Connection (***sc***):

```
src_tbls(sc) ## code to return all the dataframes associated with sc
```

9.2.5.Analyzing the Data

Now we have successfully copied the ***iris*** dataset to the Spark environment as a SparkDataFrame. And ***iris_tbl*** is an R object wrapping the ***iris*** SparkDataFrame and we can use ***iris_tbl*** to refer the ***iris*** dataset in the Spark system (i.e. the iris SparkDataFrame). With the ***sparklyr*** packages, we can use many functions in ***dplyr*** to SparkDataFrame directly through ***iris_tbl***, same as we are applying ***dplyr*** functions to a local R data frame in our laptop. For example, we can use %>% operator to pass ***iris_tbl*** to ***count***( ) function:

```
iris_tbl %>% count
```

or using the head( ) function to return to return the first a few rows in ***iris_tbl***:

```
head(iris_tbl)
```

or more advanced data manipulation directly to ***iris_tbl***:

```
iris_tbl %>%
  mutate(Sepal_Width = ROUND(Sepal_Width * 2) / 2) %>% # Bucketizing Sepal_Width
  group_by(Species, Sepal_Width) %>%
  summarize(count = n(), Sepal_Length = mean(Sepal_Length), stdev = sd(Sepal_Length))
```

9.2.6.Collect Results Back to Master Node

Even though we can run many of the ***dplyr*** functions on SparkDataFrame, we cannot apply functions from other packages to SparkDataFrame direction (such as ***ggplot***()). For functions that can only work on local R data frames, we must copy the SparkDataFrame back

to the local node. To copy SparkDataFrame back to the local node, we use the ***collect***() function where the argument to it is the name of the SparkDataFrame. The following code ***collect***() the results of a few operations and assign the collected data to ***iris_summary*** variable:

```
iris_summary <-
  iris_tbl %>%
  mutate(Sepal_Width = ROUND(Sepal_Width * 2) / 2) %>%
  group_by(Species, Sepal_Width) %>%
  summarize(count = n(), Sepal_Length = mean(Sepal_Length), stdev = sd(Sepal_Length)) %>%
  collect
```

Now ***iris_summary*** is a local variable to the R Studio and we can use all R packages and functions to it. In the following code, we will apply ***ggplot***() to it, exactly the same as a stand along R console:

```
library(ggplot2)
ggplot(iris_summary, aes(Sepal_Width, Sepal_Length, color = Species)) +
  geom_line(size = 1.2) +
  geom_errorbar(aes(ymin = Sepal_Length - stdev, ymax = Sepal_Length + stdev), width = 0.05) +
  geom_text(aes(label = count), vjust = -0.2, hjust = 1.2, color = "black") +
  theme(legend.position="top")
```

9.2.7.Fit Regression to SparkDataFrame

One of the largest advantage is that, within Spark system, there are already many statistical and machine learning algorithms developed to run parallel across many CPUs with data across many memory units. So, we can easily fit a linear regression for big dataset far beyond the memory limit of one single computer. Below is an illustration of how to fit a linear regression to SparkDataFrame using R:

```
fit1 <-  ml_linear_regression(x = iris_tbl, response = "Sepal_Length",
                              features = c("Sepal_Width", "Petal_Length", "Petal_Width"))
summary(fit1)
```

In the above code, ***x*** is the R object wrapping the SparkDataFrame; ***response*** is the y-variable, ***features*** is the collection of explanatory variables.

9.2.8.Fit a K-means Cluster

Through sparkly package, we can use R Studio to access many Spark Machine Learning Library (MLlib) algorithms such as linear regression, logistic regression, Survival Regression, Generalized Linear Regression, Decision Trees, Random Forests, Gradient-Boosted Trees, Principal Components Analysis, Naive-Bayes, K-Means Clustering and a few other methods. Below codes fit a k-means cluster algorithm:

```
## Now fit a k-means clustering using iris_tbl data
```

```
## with only two out of four features in iris_tbl
fit2 <- ml_kmeans(x = iris_tbl, centers = 3,
                  features = c("Petal_Length", "Petal_Width"))
# print our model fit
print(fit2)
```

After the k-means model is fit, we can apply the model to predict other datasets through ***sdf_predict***() function. Below code apply the model to ***iris_tbl*** again to predict and then the results are collected back to local variable ***prediction*** through ***collect***() function:

```
prediction = collect(sdf_predict(fit2, iris_tbl))
```

As ***prediction*** is a local variable, we can apply any R functions from any libraries to it. For example:

```
prediction  %>%
  ggplot(aes(Petal_Length, Petal_Width)) +
  geom_point(aes(Petal_Width, Petal_Length, col = factor(prediction + 1)),
             size = 2, alpha = 0.5) +
  geom_point(data = fit2$centers, aes(Petal_Width, Petal_Length),
             col = scales::muted(c("red", "green", "blue")),
             pch = 'x', size = 12) +
  scale_color_discrete(name = "Predicted Cluster",
                       labels = paste("Cluster", 1:3)) +
  labs(
    x = "Petal Length",
    y = "Petal Width",
    title = "K-Means Clustering",
    subtitle = "Use Spark.ML to predict cluster membership with the iris dataset."
  )
```

9.3.Summary

In the above a few sub-sections, we illustrated (1) the relationship between master / local node and Spark Clusters; (2) how to copy a local data frame to a SparkDataFrame (please note if your data is already in Spark environment, there is no need to copy. This is likely to be the case for enterprise environment); (3) how to manipulate SparkDataFrame through ***dplyr*** functions with the installation of ***sparklyr*** package; (4) how to fit statistical and machine learning models to SparkDataFrame; and (5) how to collect information from SparkDataFrame back to a local data frame for future analysis. These procedures are pretty much covered the basis of big data analysis that a data scientist need to know.

A YouTube Video for this demo is available at: https://youtu.be/RbDMvrSRXHM

JSM 2017 CE: Preparing Statistician/Statistics Graduates to Be Data Scientist

After taking the CE course, participants will:

- Understand data science is the process of defining the problem with business knowledge, gathering needed data from various sources, developing models, extracting insight and recommend actions.
- Get familiar with the cloud-based big data platform (Hadoop/ Hive/ Spark/ GPU etc.) that are widely used in the development and production setting for industry and know how to quickly transit from academia environment to enterprise environment.
- Get familiar with data extraction, transformation and load from various database systems such that participates can learn how to become self-sufficient to get needed data in enterprise environment.
- Understand how to leverage interactive dashboard to present insight and results and communicate efficient with business partners and customers.
- Understand what data scientists "in the wild" are doing to better prepared to be successfully data scientist in the future.
- Learn how to encode a real problem to a data science problem, search for the right data, preprocess data and deploy of analytical results through a case study.
- Get familiar with how to achieve high-performance computing for standard statistical procedures with big data infrastructure.