Data Analytics in Accounting
An R Programming Approach

Sharif Islam, DBA, CPA, CMA
Assistant Professor
School of Accountancy
College of Business & Analytics
Southern Illinois University Carbondale

January 31, 2021

Contents

List of Tables

List of Figures

Preface

Now a days, data science or data analytics has become a buzzword.The emergence of social media and some other platforms like social media have given birth to vast amount of data. The Economist [1] reports that "the world's most valuable resource is no longer oil, but data." Data is the raw materials for gaining additional insights and the data analytics is the means through which the insights are extracted from the data.

As we know accounting is called the "language of business" and accountants play significant role in generating vast amount of data about an organization. Additionally, accountants can leverage the data science tools to help business unlock valuable insights and thus make improved decision making.

How to Read this Book

This is how to read the book. The book is intended to a stand alone data analytics course in accounting. Alternatively, it can be used as a supplement to other accounting courses. In such cases, exercises specific to a particular accounting courses can be used. Moreover, the book is intended for both undergraduate and graduate accounting students. Further, doctoral students in Accountancy can also use the book as a preparation for research method classes in their PhD programs.

Structure of the Book

This is structure of the book.

[1] https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

Acknowledgements

I acknowledge the contributions of many whose materials online I use to prepare myself for this book.

About the Author

Sharif Islam is an Assistant Professor in School of Accountancy in Southern Illinois University Carbondale. He completed his doctorate in Accounting and Computer Information Systems from Louisiana Tech University and obtained MBA from Eastern Illinois University. He is a licensed CPA in Illinois and a CMA in Bangladesh. He has more than five years experiences in teaching and research. His research has been awarded in couple of conferences of American Accounting Association (AAA). He has a passion for Data Science, with its application in Accounting and Auditing.

Chapter 1

Introduction

Given the availability of vast amount of data, companies in numerous industries exploit such data for competitive advantage, aiming to either increase revenues or decrease costs. Data Driven Decisions (DDD) are making significant differences in productivity, on Return on Assets (ROA), Return on Equity (ROE), asset utilization, and on market value (Provost and Fawcett, 2013). Firms using data analytics in their operations can outperform their competitors by 5% in productivity and 6% in profitability (Barton and Court, 2012). In 2017, 53% companies have adopted big data, as compared to only 17% in 2015 (Columbus, 2017). Additionally, regulators are increasingly calling for organizations to use analytics (Protiviti, 2017). This emphasizes the significance of data analytics in organizations.

This is the first real chapter. Very good source for my book https://grantmcdermott.com/teaching/

1.1 Data Analytics

The meaning of (big) data analytics varies across different disciplines and there is substantive confusion between the slightly differing characterizations of "big data," "business intelligence," and "data analytics" (Vasarhelyi et al., 2015). Though many people consider big data in terms of quantities, it is also related to large-scale analysis of large amounts of data to generate insights and knowledge (Verver, 2015). Big data is characterized by four Vs: Volume; Velocity; Variety; and Veracity. Volume refers to the size of the dataset, velocity to the speed of data generation, variety to the multiplicity of data sources, and veracity to the elimination of noise and obtaining truthful information from big data. Sometimes big data are characterized by six Vs: Volume, Velocity, Variety, Veracity, Variability, and Value; or, even seven Vs: Volume, Velocity, Variety, Veracity, Variability, Value, and Visualization (Sivarajah et al., 2017).

Data analytics is defined by the American Institute of Certified Public Accountants (AICPA) (2015, p. 105) as "the art and science of discovering and analyzing patterns, identifying anomalies, and extracting other useful information in data underlying or related to the subject matter of an audit through analysis, modeling, and visualization for the purpose of planning or performing the audit." Cao et al. (2015) define big data analytics as the process of inspecting, cleaning, transforming, and modeling big data to discover and communicate useful information and patterns, suggest conclusions, and to provide support for decision-making.

Data analytics promises significant potential in auditing. Therefore, in accounting, sometimes data analytics becomes synonymous with audit analytics. Audit analytics involves the application of data analytics in the audit. Specifically, American Institute of Certified Public Accountants (AICPA) (2017) defines audit data analytics as "the science and art of discovering and analyzing patterns, identifying anomalies and extracting other useful information in data underlying or related to the subject matter of an audit through analysis, modeling and visualization for the purpose of planning or performing the audit." In other words, audit data analytics are techniques that can be used to perform a number of audit procedures such as risk assessment, tests of details, and substantive analytical procedure to gather audit evidence. The benefits of using audit data analytics include improved understanding of an entity's operations and associated risk including the risk of fraud, increased potential for detecting material misstatements, and improved communications with those charged with governance of audited entities.

1.2 Data Analytics & Accountancy

Data analytics is important for accounting profession because data gathering and analytics technologies have the potential to fundamentally change accounting and auditing task processes (Schneider et al., 2015). Scholars note that the emergence of data analytics will significantly change the infer/predict/assure (e.g., insight/foresight/oversight) tasks performed by accountants and auditors. Big data and analytics have increasingly important implications for accounting and will provide the means to improve managerial accounting, financial accounting, and financial reporting practices (Warren Jr et al., 2015). It is further suggested that big data offers an unprecedented potential for diverse, voluminous datasets and sophisticated analyses. Research indicates that big data has great potential to produce better forecast estimates, going concern calculations, fraud, and other variables that are of concern to both internal and external auditors (Alles, 2015). Moreover, auditors might reduce audit costs and enhance profitability and effectiveness by means of big data or data analytics. Sixty-six percent of internal audit departments currently utilize some form of data analytics as part of the audit process (Protiviti, 2017).

1.2.1 Data Analytics in Financial Accounting

Warren et al.(2015) note that "in financial accounting, big data will improve the quality and relevance of accounting information, thereby enhancing transparency and stakeholder decision-making. In reporting, big data can assist with the creation and refinement of accounting standards, helping to ensure that the accounting profession will continue to provide useful information as the dynamic, real-time, global economy evolves." In particular, they suggest that big data could significantly impact the future of financial accounting and Generally Accepted Accounting Principles (GAAP). Big data can also help to supplement financial statement disclosures by accumulating, processing, and analyzing information about a given intangible of interest. Furthermore, big data or data analytics can help in narrowing the differences between accounting standards such US GAAP and International Financial Reporting Standards (IFRS) and facilitate different measurement processes such as Fair Value Accounting (FVA) by analyzing different kinds of unstructured data (Warren Jr et al., 2015).

Crawley and Wahlen (2014) noted that data analytics allows researchers to explore a large amount of qualitative information disclosed by organizations, and examines the consequences of such disclosures. Moreover, data analytics now provides the opportunity to judge the informational content of qualitative financial information. For example, Davis, Piger, and Sedor (2012) found that the extent of optimism expressed in firms' earnings announcements is positively associated with Return on Assets (ROA) and stock reactions. By the same token, Li (2010) suggested that the tone of forward-looking statements is positively associated with future earnings performance. In addition, Feldman, Govindaraj, Livnat, and Segal (2010) found that changes in disclosure tone is indicative of future changes in earnings. Interestingly, research shows that even information on social media such as Twitter can predict stock market responses (Bollen et al., 2011).

Data analytics helps to relate textual data to earnings quality. For example, firms having more complicated and less transparent financial statement disclosures are more likely to have poor quality earnings, less persistent positive earnings and more persistent negative earnings (Li, 2008) . Li, Lundholm, and Minnis (2013) confirmed that firms discussing their competition frequently have ROAs that mean returns more severely than the firms discussing the competition infrequently.

With the help of textual data analytics, researchers recently documented the role that qualitative disclosures have in forming the information environment of organizations; such information environments include factors such as the number of analyst following a firm, characteristics of its investors, its trading activities, and the litigation it is involved with. Less readable 10-Ks are associated with greater number of analysts following the firm and a greater amount of effort needed to generate report about it (Lehavy et al., 2011). They also find that less readable 10-Ks are associated with greater dispersion, lower accuracy, and

greater uncertainty in analyst's earnings forecasts about a given firm.

1.2.2 Data Analytics in Management Accounting

Warren et al. (2015, 397) noted that "in managerial accounting, big data will contribute to the development and evolution of effective management control systems and budgeting processes." In particular, they elaborate on how big data or data analytics can play a role in management control systems by discovering behaviors that have correlation with specific goal outcomes. Essentially, big data analytics can locate new kinds of behaviors that might impact goal outcomes by simplifying the identification of important motivational measurement tools linked to organizational goals. Moreover, by analyzing non-structured data, big data analytics can help discern employee morale, productivity, and customer satisfaction. Data analytics can also be used to improve "beyond budgeting practices" since traditional budgeting sometimes creates barriers to creativity and flexibility (Warren, Moffitt, and Byrnes 2015).

Richins, Stapleton, Stratopoulos, and Wong (2017) suggest that big data analytics could improve customer service quality. They suggest that most of the time organizations use structured data that are in their records to evaluate customer service quality; however, this approach does not take into account the customer perspective. Big data analytics allow organizations to evaluate this customer perspective by using unstructured data from social media or e-commerce sites, thus permitting organizations to have a holistic view of customer service quality.

Managers recognize that financial measures, alone, are insufficient to forecast future financial success or to use for performance management. Big data analytics provides opportunities to incorporate non-financial measures by incorporating unstructured data (Richins et al. 2017). Using big data analytics (particularly the analysis of unstructured data) accountants can identify the causes of underlying problems, understand ramifications, and develop plans to mitigate adverse impacts (Richins et al. 2017). Data analytics can also provide accountants with additional tools to monitor operations and product quality, discover opportunities to reduce costs, and contribute to decision-making (Dai and Vasarhelyi 2016).

1.2.3 Data Analytics in Auditing

Data analytics has the potential to improve the effectiveness of auditing by providing new forms of audit evidence. Data analytics can be used in both auditing planning and in audit procedures, helping auditors to identify and assess risk by analyzing large volumes of data. Even organizations that have very immature capabilities indicate that a strong level of value is derived from including analytics in the audit process (Protiviti 2017).

Big data is being seen by practitioners as an essential part of assurance services (Alles and Gray 2016), but its application in auditing is not as straightforward

as it is in marketing and medical research. Appelbaum (2016) and Cao et al. (2015) identified several areas that are likely to benefit from the use of big data analytics. Some of the areas are:

a) At the engagement phase – supplementing auditors' industry and client knowledge
b) At the planning phase – supplementing auditors' risk assessment process
c) At the substantive test phase – verifying the management assertions
d) At the review phase – advanced data analytical tools as analytical procedures
e) At the continuous auditing phase – enhancing knowledge about the clients

Yoon, Hoogduin, and Zhang (2015) suggest that big data create great opportunities through providing audit evidence. They focused on the "sufficiency" and "appropriate" criteria and noted that though there are some issues about the propriety of big data due to different kinds of "noise," big data can be used as complementary audit evidence. Additionally, they discussed how big data can be integrated with traditional audit evidence in order to add value in the process. Big data or data analytics can also help auditors to test the existence of assertions (e.g. fixed assets) using non-conventional data such as video recording (Warren, Moffitt, and Byrnes 2015). In the world of big data, potential types and sources of audit evidence have changed (Appelbaum 2016). For this reason, Krahel and Titera (2015) suggest that big data might change the focus of auditors, shifting emphasis from management to the verification of data.

Data quality and reliability or verifiability have become important issues in auditors' evaluations of audit evidence. In this way, big data can be used as part of analytical procedures, which are required at the planning and review phase, but which are optional at the substantive procedure phase. However, many issues remain unresolved about how to use big data since analytical procedures and auditing standards are not very specific about the selection of analytical audit procedures; the choice depends on the professional judgment of auditors (Appelbaum, Kogan, and Vasarhelyi 2017). For this reason, auditors need to exercise increased professional skepticism in the big data era because in many cases sources of big data lack provenance and, subsequently, veracity, and sometimes auditors (particularly internal auditors) have little or no involvement in data quality evaluation of such sources (Appelbaum 2016). Considering the prediction that analytics will spell the demise of auditing, Richins et al. (2017) suggest that auditors in the big data era are still essential because they know "the language of business." Particularly, they suggest that big data analytics cannot replace the professional judgment used by auditors, suggesting that analytics will instead complement auditors' professional judgment.

Alles and Gray (2016) identify four potential advantages of incorporating big data into audit practices: strong predictive power to set expectations for financial statement audits, great opportunities to identify potential fraudulent activities, increased probabilities of discovering red flags, and the possibility of developing more predictive models for going concern assumptions. To that end,

internal audit groups with dedicated analytics functions and organizations that have attained a managed or optimized to the state of analytics maturity are far more likely to conduct continuous auditing (Protiviti 2017). Though big data creates many opportunities for improving auditing, it also suffers from different shortcomings that hinder its application in Continuous Auditing (CA). For example, Zhang, Yang, and Appelbaum (2015) suggest big data characteristics such as volume, velocity, variety, and veracity creates problems in its application in CA through different gaps such as data consistency, data integrity, data identification, data aggregation, and data confidentiality.

Rose, Rose, Sanderson, and Thibodeau (2017) found that the timing of the introduction of data analytics tools into the audit process affects the evaluation of evidence and professional judgment. Barr-Pulliam, Brown-Liburd, and Sanderson (2017) found that jurors consider auditors more negligent when they use traditional auditing technique rather than audit data analytics techniques. Additionally, they confirmed that audit data analytics tools increase the perceptions of audit quality. Schneider et al. (2015) suggest that data analytics can be used by auditors to evaluate the internal control effectiveness and policy compliance. They further suggest that by analyzing unusual data flows, unexpected large volumes of data, high frequency transactions, or duplicate vendor payments, auditors can better detect fraud.

Chapter 2

R Programming - A Great Tool for Data Science

There are many tools for data science. Of these tools, R Programming is one of the most powerful tools. It is powerful in that it has more than 11,000 packages on the CRAN (Comprehensive R Archive Network) and thousands of other packages on Github and other platforms. Moreover, many companies around the word use R. For example -

Companies	Companies	Companies
Facebook	Google	Twitter
Mircrosoft	Uber	Airbnb
IBM	Boeing	Ford
New York Times	Wells Fargo	American Express

Very good website for the introductory part - https://rc2e.com/gettingstarted#recipe-id001 and https://rc2e.com/navigatingthesoftware#intro-NavigatingTheSoftware

2.1 Starting with R Program

2.1.1 Installing R

2.1.2 Installing RStudio

2.2 Data Analytics Ecosystem in R

We will adopt the following ecosystem in data analytics in R.

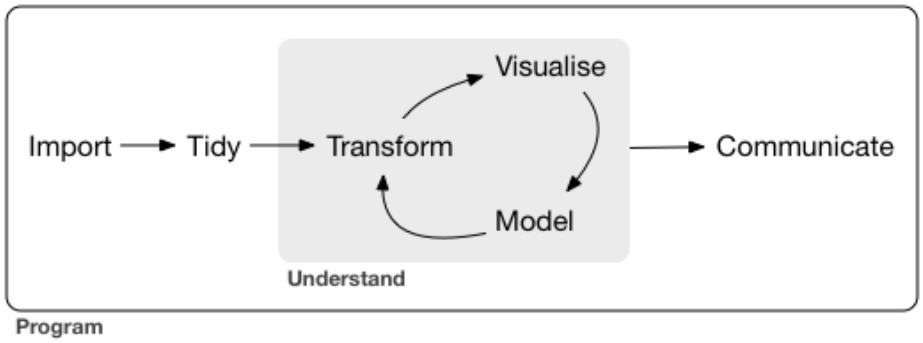

Figure 2.1: Data Analytics Ecosystem in R

Chapter 3

Getting Data into R

Data lie in different places and in different formats. Therefore, getting them into R is not always the same. Some data are stored in local directory in our computer, while other data are available online. Moreover, sometimes we need extract data from databases. To know how to extract different forms of data from different sources is important because the explosion of social media and similar platforms has given birth to tremendous amounts of data in different formats and in different places. Though accountants and auditors are good at dealing with structured data, they should also know how to deal with unstructured and non-conventional data (Richins et al., 2017).

3.1 Packages That Could Be Used to Import Data into R.

There are many packages that can be used to import data into R from many different sources. Of those packages, `readr`, `readxl`, and `purrr` will be discussed here. Additionally, data can be imported into R using `RStudio's Import Wizard`. To import data using `RStudio's Import Wizard`, one needs to go to the Environment tab and select Import Dataset; then, select appropriate type of data one wants to import and finally browse the data one wants to import. Please see Figure 3.1 to learn about how to use `RStudio's Import Wizard`.

3.2 Knowing & Setting Your Current (Working) Directory

When one wants to import the data from Personal Computer (PC), then knowing where the data reside (which is also called "Path") is necessary as this will help import the data easily. The function `getwd()` will help to know the current

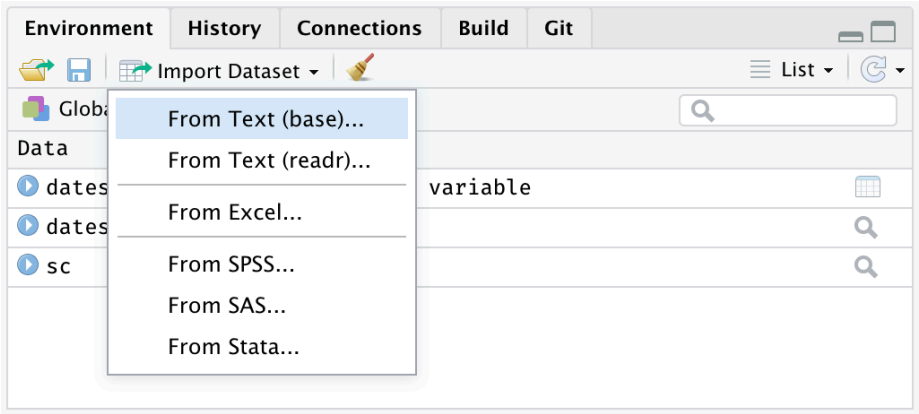

Figure 3.1: Import Data Using RStudio's Import Wizard

working directory. Unless a specific directory is set as a current (working) directory, `R` will always look for a file in the current working directory. The function `setwd()` helps to set any directory (folder) as current working directory. If you are working in an `R` project (The discussion about `R` project is in Appendix .2), then the project folder (directory) is the current directory. Figure 3.2 is an example of the address of a directory (folder).

If you click on the address bar of the directory (highlighted in Red), it will look like - `C:\Documents\Project Docs`. If you want to set it as your current working directory, then you have to write the code `setwd("C:/Documents/Project Docs")`. Note that though the address has back slash (\), in `setwd()` function, we use forward slash (/) as `R` cannot deal with forward slash. Once you set the working directory, running `getwd()` will show you your current working directory, which is the default directory for importing and exporting data into `R` (unless you specifically mention a different path).

3.3 Importing Data into R from Excel

To import data from excel, `readxl` package can be excellent. You can install `readxl` package by `install.packages("readxl")` or it will be installed when `tidyverse` package is installed. `readxl` package can deal with both `xls` and `xlsx` files. There are some built-in datasets with `readxl` package. The function `readxl_example` generates the names of the built in datasets in `readxl` package.

```
library(readxl)
readxl_example() # These are the example files from readxl package.
```

```
##  [1] "clippy.xls"    "clippy.xlsx"   "datasets.xls"
##  [4] "datasets.xlsx" "deaths.xls"    "deaths.xlsx"
##  [7] "geometry.xls"  "geometry.xlsx" "type-me.xls"
```

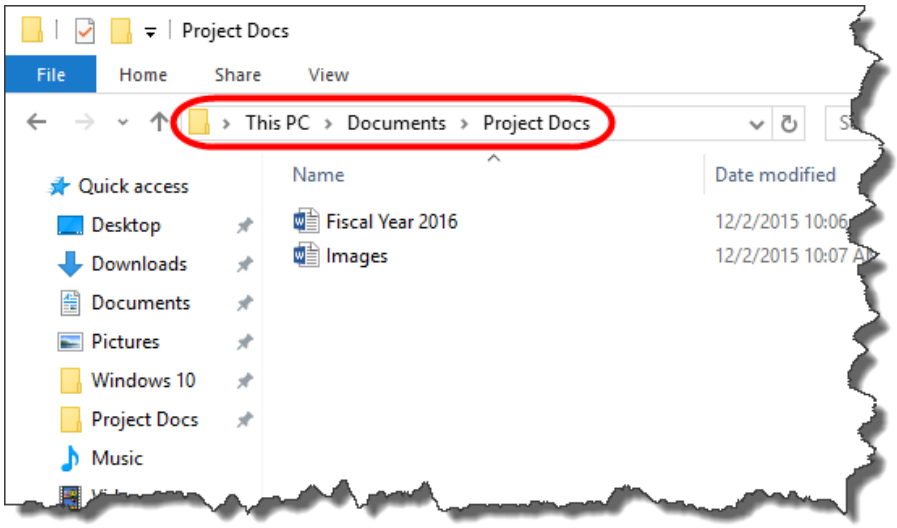

Figure 3.2: Directory

```
## [10] "type-me.xlsx"
```

The `read_excel` function will read the data from excel. For example - if we want to import an excel file such as climate change data from World Bank (https://datacatalog.worldbank.org/dataset/climate-change-data), we can use `read_excel` function.

```
climate_change <- read_excel("data/climate_change_download_0.xls")
# To know the names of the variables of the dataset
names(climate_change)
```

```
##  [1] "Country code" "Country name" "Series code"
##  [4] "Series name"  "SCALE"        "Decimals"
##  [7] "1990"         "1991"         "1992"
## [10] "1993"         "1994"         "1995"
## [13] "1996"         "1997"         "1998"
## [16] "1999"         "2000"         "2001"
## [19] "2002"         "2003"         "2004"
## [22] "2005"         "2006"         "2007"
## [25] "2008"         "2009"         "2010"
## [28] "2011"
```

```
# OR
colnames(climate_change)
```

```
##  [1] "Country code" "Country name" "Series code"
##  [4] "Series name"  "SCALE"        "Decimals"
```

```
##  [7] "1990"         "1991"         "1992"
## [10] "1993"         "1994"         "1995"
## [13] "1996"         "1997"         "1998"
## [16] "1999"         "2000"         "2001"
## [19] "2002"         "2003"         "2004"
## [22] "2005"         "2006"         "2007"
## [25] "2008"         "2009"         "2010"
## [28] "2011"
```

If the excel file `climate_change_download_0` is opened in excel, we can see that there are 3 worksheets in the files, but `read_excel` function in `climate_change` dataset only imports the first worksheet called `Data`. We can use `sheet` argument in `read_excel` function to specify which worksheet one wants to import. In `sheet` argument the position of the worksheet can be specified as well rather than the name of the worksheet. Also, `excel_sheets` functions can be used to know the names of all worksheets in a given dataset. The function `excel_format` can be used to know the format of the excel files (`xls` or `xlsx`).

```
excel_sheets("data/climate_change_download_0.xls")
```

```
## [1] "Data"    "Country" "Series"
```

```
excel_format("data/climate_change_download_0.xls")
```

```
## [1] "xls"
```

```
climate_change_country <- read_excel("data/climate_change_download_0.xls", sheet = "Cou
# OR
climate_change_country <- read_excel("data/climate_change_download_0.xls", sheet = 2)
```

Other arguments in `read_excel` function such as `range` can be used to import a subset of the excel file. The argument `na` are used for missing values (NA). In our dataset `climate_change` there are missing values represented by `...` This can be replaced by `na`argument.

```
climate_change2 <- read_excel("data/climate_change_download_0.xls",
                              na = "..")
```

Also, we can use `writexl` package to export (save) a dataset from R in excel format. The function `write_xlsx` is usually used to save the data in excel format in desired directory.

```
library(writexl)
write_xlsx(climate_change_country, "data/climatechange_country.xlsx")
```

Alternatively, one can use `file.choose()` function within `read_excel` function to **manually** import an excel file into R.

```
data <- read_excel(file.choose())
```

Also, using `read.table` function allows one to copy and paste an excel file in R.

```
df <- read.table(file = "clipboard",
                        sep = "\t", header=TRUE)
```

3.4 Importing Data into R from Local Directory of Computer

Importing data from local directory of computer can be done in couple of ways. For example, we can specify the path in which the data is stored. Alternatively, we can set the working directory first; then import the data by specifying the name of the data files.

3.5 Importing Data into R from Internet

3.6 Importing Data into R from WRDS Database

3.7 Some Other Sources

Very good website for the chapter - https://rc2e.com/inputandoutput#recipe-id245

Chapter 4

Data Wrangling

Data wrangling is the process of cleaning data, so that data become ready for further manipulation such as visualization and modeling. Sometimes, data wrangling is also called `Data Munging`. More specifically, data wrangling involves transforming and mapping data from one from to another form - particularly from *raw* form to *tidy* form.

There is an old saying that 90% of data science involves data wrangling. In many cases, data wrangling is difficult as it involves dealing with missing entries, ambiguous values, and different types of mixed data. In data analytics ecosystem in `R`, data wrangling involves three jobs - importing data into `R`, cleaning (tidying) the data, and transforming the data. Please see Figure 4.1 to learn about the data wrangling process.

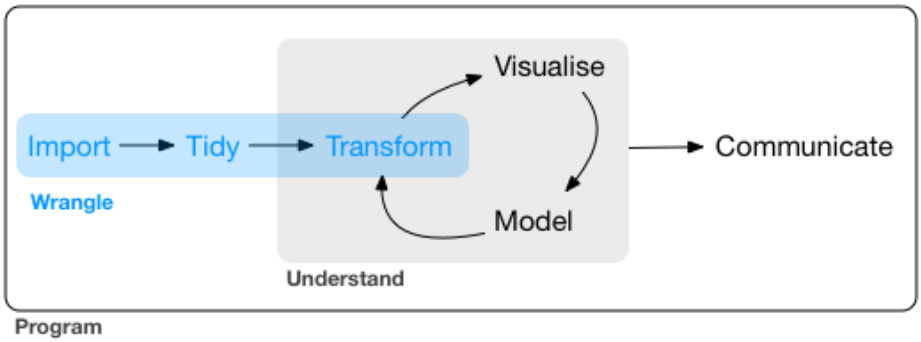

Figure 4.1: Data Wrangling in R

The first job - importing data into `R` - is discussed in chapter 03. In this chapter, cleaning (tidying) the data in `R` will be discussed. The last job will be discussed in next chapter - Exploratory Data Analysis (EDA).

4.1 `tidy` data

Data wrangling or data munging results in `tidy` data - which is storing data that makes further manipulation on data such as transformation, visualization, and modeling easier. The following rules make a dataset `tidy` -

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell

4.2 Same Data, but Different Formats (Presentations)

4.3 Tidying Messy Data

According to Hadley Wickham, "Tidy datasets are all alike, but every messy dataset is messy in its own way" (Wickham and Grolemund, 2017). Messy data can be in many forms; for example - Wickham (2014) mentions the five *most common problems* with messy datasets. These problems include -

- Column headers are values, not variable names
- Multiple variables are stored in one column
- Variables are stored in both rows and columns
- Multiple types of observational units are stored in the same table
- A single observational unit is stored in multiple tables

4.3.1 Column Hearders are Values, not Variable Names

The following dataset (`total_assets`) is an example of this case, in which the name of the variables (columns) are numbers. Though in some cases, this data format might be useful, in many cases usually it is not useful.

```
total_assets
```

```
## # A tibble: 5 x 7
##   TICKER COMPANYNAME `2000` `2001` `2002` `2003` `2004`
##   <chr>  <chr>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 AAPL   APPLE INC     6803   6021   6298   6815  8.05e3
## 2 WMT    WALMART INC 70349  78130  83451  94685  1.05e5
## 3 AABA   ALTABA INC   2270.  2379.  2790.  5932. 9.18e3
## 4 AMZN   AMAZON.COM~  2135.  1638.  1990.  2162. 3.25e3
## 5 GOOGL  ALPHABET I~    NA     NA    287.   871. 3.31e3
```

4.3.2 Multiple Variables are Stored in One Column

The `netincome_asset` dataset is a good example of this kind of dataset. For example, the variable `VALUES` includes `net income` and `total assets` sepa-

rated by /. This variable is very hard to manipulate. For example, if we want to calculate Return on Assets (ROA), which equals to net income divided by total assets, then it is not possible to use VALUES column to calculate ROA.

```
netincome_asset
```

```
## # A tibble: 23 x 4
##     year TICKER COMPANYNAME VALUES
##    <dbl> <chr>  <chr>       <chr>
##  1  2000 AAPL   APPLE INC   786/6803
##  2  2001 AAPL   APPLE INC   -25/6021
##  3  2002 AAPL   APPLE INC   65/6298
##  4  2003 AAPL   APPLE INC   69/6815
##  5  2004 AAPL   APPLE INC   276/8050
##  6  2000 WMT    WALMART INC 5377/70349
##  7  2001 WMT    WALMART INC 6295/78130
##  8  2002 WMT    WALMART INC 6671/83451
##  9  2003 WMT    WALMART INC 8039/94685
## 10  2004 WMT    WALMART INC 9054/104912
## # ... with 13 more rows
```

4.3.3 Variables are Stored in Both Rows and Columns

The sales_profit dataset is an example of this type. Here in the ITEM column, actually variables are included (such as SALES and NETINCOME). Also, the columns such as 2000 through 2004 should be a variable (e.g., year).

```
sales_profit
```

```
## # A tibble: 10 x 8
##    TICKER COMPANYNAME ITEM   `2000`  `2001`  `2002`
##    <chr>  <chr>       <chr>   <dbl>   <dbl>   <dbl>
##  1 AAPL   APPLE INC   SALES  7.98e3  5.36e3  5.74e3
##  2 AAPL   APPLE INC   NETI~  7.86e2 -2.50e1  6.50e1
##  3 WMT    WALMART INC SALES  1.66e5  1.92e5  2.19e5
##  4 WMT    WALMART INC NETI~  5.38e3  6.30e3  6.67e3
##  5 AABA   ALTABA INC  SALES  1.11e3  7.17e2  9.53e2
##  6 AABA   ALTABA INC  NETI~  7.08e1 -9.28e1  4.28e1
##  7 AMZN   AMAZON.COM~ SALES  2.76e3  3.12e3  3.93e3
##  8 AMZN   AMAZON.COM~ NETI~ -1.41e3 -5.67e2 -1.49e2
##  9 GOOGL  ALPHABET I~ SALES NA      NA       4.40e2
## 10 GOOGL  ALPHABET I~ NETI~ NA      NA       9.97e1
## # ... with 2 more variables: `2003` <dbl>,
## #   `2004` <dbl>
```

4.3.4 Multiple Types of Observational Units are Stored in the Same Table

4.3.5

4.4 `tidyr` Package for `tidy` Data

The `tidyr` package is widley used to `tidy` data in `R`. Specifically, `pivot_longer`, `pivot_wider`, `separate`, and `extract` functions are widely used to make a messy data into `tidy`.

4.4.1 `pivot_longer ()` function

If the `total_assets` dataset above (Column headers are values, not variable names) is made `tidy` using `pivot_longer` function, then it will be look like this (Assuming the name of the dataset is variable_number) -

```
# using pivot_longer () function
total_assets %>%
  pivot_longer(cols = -c("TICKER", "COMPANYNAME"),
               names_to = "year",
               values_to = "TOTALASSETS"
               ) %>%
  head()
```

```
## # A tibble: 6 x 4
##   TICKER COMPANYNAME year  TOTALASSETS
##   <chr>  <chr>       <chr>       <dbl>
## 1 AAPL   APPLE INC   2000         6803
## 2 AAPL   APPLE INC   2001         6021
## 3 AAPL   APPLE INC   2002         6298
## 4 AAPL   APPLE INC   2003         6815
## 5 AAPL   APPLE INC   2004         8050
## 6 WMT    WALMART INC 2000        70349
```

There are several arguments of `pivot_longer` function. Here three arguments are used. The `cols` argument specifies which columns should (not) be used in `pivot_longer` function. In this case, we select the variables that should not be used while using `pivot_longer` function. The second argument `names_to` specifies the name of the variable in which existing column values will be stored and finally `values_to` specifies the name of the column in which the cell values will be stored. Now, the dataset is a `tidy` dataset.

4.4.2 `pivot_wider ()` function

The `sales_profit` dataset can be put into `tidy` format using both `pivot_longer` and `pivot_wider` function.

```
sales_profit %>%
  pivot_longer(
    cols = !c ("TICKER", "COMPANYNAME", "ITEM"),
    names_to = "year",
    values_to = "AMOUNT"
  ) %>%
  pivot_wider(
    names_from = ITEM,
    values_from = AMOUNT
  )
```

```
## # A tibble: 25 x 5
##    TICKER COMPANYNAME year   SALES NETINCOME
##    <chr>  <chr>       <chr>  <dbl>     <dbl>
##  1 AAPL   APPLE INC   2000    7983       786
##  2 AAPL   APPLE INC   2001    5363       -25
##  3 AAPL   APPLE INC   2002    5742        65
##  4 AAPL   APPLE INC   2003    6207        69
##  5 AAPL   APPLE INC   2004    8279       276
##  6 WMT    WALMART INC 2000  165639      5377
##  7 WMT    WALMART INC 2001  192003      6295
##  8 WMT    WALMART INC 2002  218529      6671
##  9 WMT    WALMART INC 2003  245308      8039
## 10 WMT    WALMART INC 2004  257157      9054
## # ... with 15 more rows
```

4.4.3 `separate ()` function

The `separate` function can be used when multiple variables are stored in one column. The function separates one column into multiple columns. For example, in `netincome_asset` data, the column `VALUES` should be converted into two columns called `NETINCOME` and `TOTALASSETS`. In `separate` function, `col` argument is ued to select the column that needs to be broken; `into` argument is used to create new columns; and `sep` argument is ued to identify the character in which the column will be separated. In this case, it is `/`. Finally `remove` argument is used to decide whether the column (here `VALUES`) that is separated should be in new dataset.

```
netincome_asset %>%
  separate(
    col = VALUES, into = c ("NETINCMOE", "TOTALASSETS"), sep ="/",
    remove =TRUE
  )
```

```
## # A tibble: 23 x 5
##     year TICKER COMPANYNAME NETINCMOE TOTALASSETS
```

```
##    <dbl> <chr>  <chr>       <chr>     <chr>
##  1  2000 AAPL   APPLE INC   786       6803
##  2  2001 AAPL   APPLE INC   -25       6021
##  3  2002 AAPL   APPLE INC   65        6298
##  4  2003 AAPL   APPLE INC   69        6815
##  5  2004 AAPL   APPLE INC   276       8050
##  6  2000 WMT    WALMART INC 5377      70349
##  7  2001 WMT    WALMART INC 6295      78130
##  8  2002 WMT    WALMART INC 6671      83451
##  9  2003 WMT    WALMART INC 8039      94685
## 10  2004 WMT    WALMART INC 9054      104912
## # ... with 13 more rows
```

It is clear from the above dataset that the type of columns `NETINCOME` and `TOTALASSETS` are `chr`, but they should be number (e.g., `dbl`). Actually, `separate` functions maintain the type of the columns that are separated. The type of `VALUES` column was `chr` and it is maintained in new dataset. In order to get the true type, we can use `convert` argument, which is done below.

```
netincome_asset  %>%
  separate(
    col = VALUES, into = c ("NETINCMOE", "TOTALASSETS"), sep ="/",
    remove =TRUE,convert = TRUE
  )
```

```
## # A tibble: 23 x 5
##     year TICKER COMPANYNAME NETINCMOE TOTALASSETS
##    <dbl> <chr>  <chr>           <dbl>       <dbl>
##  1  2000 AAPL   APPLE INC         786        6803
##  2  2001 AAPL   APPLE INC         -25        6021
##  3  2002 AAPL   APPLE INC          65        6298
##  4  2003 AAPL   APPLE INC          69        6815
##  5  2004 AAPL   APPLE INC         276        8050
##  6  2000 WMT    WALMART INC      5377       70349
##  7  2001 WMT    WALMART INC      6295       78130
##  8  2002 WMT    WALMART INC      6671       83451
##  9  2003 WMT    WALMART INC      8039       94685
## 10  2004 WMT    WALMART INC      9054      104912
## # ... with 13 more rows
```

4.4.4 `extract ()` function

This is a very good source to discuss about data wrangling - https://dsapps-2020.github.io/Class_Slides/

This is the best data wrangling website - https://dcl-wrangle.stanford.edu/

The above link is also best about why `EXCEL` is not be best for Accounting &

Audit Analytics.

Chapter 5

Exploratory Data Analysis (EDA)

5.1 `tidyverse` package in R

The `tidyverse` package in R is a very useful package for manipulating data. The `tidyverse` is a collection of a set of packages. Of these packages, we will particularly focus on two pacakages - `dplyr` and `ggplot2`. The `dplyr` is for **data manipulation** and the `ggplot2` is for **data visualization**. In R, to install a package, you need to write `install.packages ()` code and to load the package, you need to write `library ()` code. Therefore, to install the package, you should write `install.packages("tidyverse")` and to load the follwing code -

```
library(tidyverse)
```

5.2 `dplyr` Package - Data Manipulation Tool

As stated above, the `dplyr` package is for **data manipulation**. There are many functions in `dplyr`; however, of these functions, six (06) functons are very much essential for data manipulation. In this project, we will learn those six necessary functions. These functions include - `select`, `filter`, `arrange`, `mutate`, `summarize` and , `group_by`. In additon to these functions, we will also use some other functions such as `glimpse`, `count`, `dim`, `str` and so on.

5.3 Data Set for Classroom Practice

There are many sources from which you can collect the data and manipulate in R. Some of the data sets are already included in some packages. In our class

room, we will use a package called `nycflights13` and install it by writing the code `install.packages("nycflights13")`. In the package, there are several data set, but we will use the `flights` data set. In order to get the data set in R Environment, you need to first install the package and load the data set and the following codes should be executed -

```
library(nycflights13)
flights <- flights
```

5.4 Assignment <- & Pipe %>% operator

Frequently, we will use the assignment `<-` and pipe `%>%` operator. The keyboard shortcurt for `<-` is **alt+-** and `%>%` is **ctrl+shift+M**

5.5 Meta Data - Data About Data

Once you load a data set in R, your next job should be to learn about some characteristcs about the data. To do so, you first need to load the `tidyverse` package by running the code `library(tidyverse)`. Then you should write the following code. See the Table 5.1.

```
library(tidyverse)
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 1...
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL...
## $ flight         <int> 1545, 1714, 1141, 725, 461,...
## $ tailnum        <chr> "N14228", "N24211", "N619AA...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK",...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN",...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

Table 5.1: A Table of First Five Variables.

year	month	day	dep_time	sched_dep_time
2013	1	1	517	515
2013	1	1	533	529
2013	1	1	542	540
2013	1	1	544	545
2013	1	1	554	600
2013	1	1	554	558

```
knitr::kable(head(flights[,1:5]), caption = "A Table of First Five Variables.")
```

It can be seen that there are 336,776 observations and 19 variables. Additionally, the label of the variables can be identified. For examplem, the variable "year" is **integer**, the variable "carrier" is **character** variable, and the variable "time_hour" is **date** variable. This metadata is important for further manipulation of the data. You can also use `dim ()` to see the number of rows and columns. Furhter, `str()` can be used. The `names ()` function gives you the names of the variables of the data set.

```
dim(flights)
```

```
## [1] 336776     19
```

```
str(flights)
```

```
## tibble [336,776 x 19] (S3: tbl_df/tbl/data.frame)
##  $ year          : int [1:336776] 2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int [1:336776] 517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int [1:336776] 515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num [1:336776] 2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int [1:336776] 830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int [1:336776] 819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num [1:336776] 11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr [1:336776] "UA" "UA" "AA" "B6" ...
##  $ flight        : int [1:336776] 1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr [1:336776] "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num [1:336776] 227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num [1:336776] 1400 1416 1089 1576 762 ...
##  $ hour          : num [1:336776] 5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num [1:336776] 15 29 40 45 0 58 0 0 0 0 ...
```

```
##  $ time_hour     : POSIXct[1:336776], format: "2013-01-01 05:00:00" ...
```

```
names(flights)
```

```
##  [1] "year"           "month"          "day"
##  [4] "dep_time"       "sched_dep_time" "dep_delay"
##  [7] "arr_time"       "sched_arr_time" "arr_delay"
## [10] "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"
## [16] "distance"       "hour"           "minute"
## [19] "time_hour"
```

You can also know more about the flights data set (built-in data set in packages) by the following code - `help ("flights")` or `?flights`

If you want to see the data set in excel-like spread sheet, you have to write `View (flights)`. This code will open the data in an excel-like spreadsheet. Note that the **V** in view is **capital** letter.

5.6 Changing the type of the variable

Sometimes we might need to change the type of the variable; e.g., converting an integer variable to a character variable. In such case, we need to write code. If we want to convert "flight" variable from **int** type to **chr**, you need to write the following code -

```
flights$flight <- as.character(flights$flight)
```

Other codes for the conversion should be like this - `as.character()`, `as.factor()`

Also by writing code, you can check the type of the variable. For example -

```
is.character(flights$hour)
```

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

```
is.numeric(flights$hour)
```

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

Alternatively, you can use `class()` function to know the type of the variable. For example -

```
class(flights$year)
```

```
## [1] "integer"
```

5.7 count() function

To know the frequency of different variables (particularly categorical variables), we can use the `count()` function. For example - we want to know whether the dataset includes information about American Airlines (AA); we should write -

```
flights %>%
  count(carrier)
```

```
## # A tibble: 16 x 2
##    carrier     n
##    <chr>   <int>
##  1 9E      18460
##  2 AA      32729
##  3 AS        714
##  4 B6      54635
##  5 DL      48110
##  6 EV      54173
##  7 F9        685
##  8 FL       3260
##  9 HA        342
## 10 MQ      26397
## 11 OO         32
## 12 UA      58665
## 13 US      20536
## 14 VX       5162
## 15 WN      12275
## 16 YV        601
```

If we want to know the name and the numbers of airports the flights left, we need to use the "origin" variable -

```
flights %>%
  count(origin)
```

```
## # A tibble: 3 x 2
##   origin      n
##   <chr>   <int>
## 1 EWR    120835
## 2 JFK    111279
## 3 LGA    104662
```

Simialrly, we can see where these flights go by the following code -

```
flights %>%
  count(dest)
```

```
## # A tibble: 105 x 2
##    dest      n
```

```
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     265
##  3 ALB     439
##  4 ANC       8
##  5 ATL   17215
##  6 AUS    2439
##  7 AVL     275
##  8 BDL     443
##  9 BGR     375
## 10 BHM     297
## # ... with 95 more rows
```

If we want to order the rows when we use `count()` function, then we have to use additional argument such as `sort = TRUE` in the function

```
flights %>%
  count(dest, sort = TRUE)
```

```
## # A tibble: 105 x 2
##    dest      n
##    <chr> <int>
##  1 ORD   17283
##  2 ATL   17215
##  3 LAX   16174
##  4 BOS   15508
##  5 MCO   14082
##  6 CLT   14064
##  7 SFO   13331
##  8 FLL   12055
##  9 MIA   11728
## 10 DCA    9705
## # ... with 95 more rows
```

We can see from the table that most of the flights' destination was Chicago Airport (ORD), followed by Atlanta Airport (ATL)

5.8 1st (First) verb - `select ()`

The `select ()` function is used to select some **columns** from your data set. For example, if you want to select the variables **year, month, day, dep_time** from your data set. Then you should write the following code (We created a new data set called **flights2**)

```
flights2 <- flights %>%
  select(year, month, day, dep_time)
glimpse(flights2)
```

```
## Rows: 336,776
## Columns: 4
## $ year     <int> 2013, 2013, 2013, 2013, 2013, 201...
## $ month    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time <int> 517, 533, 542, 544, 554, 554, 555...
```

Alternatively, you can write the following code to get the same results -

```
flights2 <- flights %>%
  select(year:dep_time)
glimpse(flights2)
```

```
## Rows: 336,776
## Columns: 4
## $ year     <int> 2013, 2013, 2013, 2013, 2013, 201...
## $ month    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time <int> 517, 533, 542, 544, 554, 554, 555...
```

There is another function called `starts_with()`, which we use to select those variables that start with some pre-selected phrases. For example - if we want to selet all variables that start with **arr**, then we should write the following code (the new data set is called - arr)

```
arr <- flights %>%
  select(starts_with("arr"))
glimpse(arr)
```

```
## Rows: 336,776
## Columns: 2
## $ arr_time  <int> 830, 850, 923, 1004, 812, 740, 9...
## $ arr_delay <dbl> 11, 20, 33, -18, -25, 12, 19, -1...
```

A similar function like `starts_with()` is `ends_with()`. `contains()` function can be used as well to select those variables that contain specific pharases/words. `matches()` function also serves the similar objective. Some of the applications of these arguments are given below -

```
flights %>%
  select(starts_with("dep")) %>%
  glimpse()
```

```
## Rows: 336,776
## Columns: 2
## $ dep_time  <int> 517, 533, 542, 544, 554, 554, 55...
## $ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3,...
```

```
flights %>%
  select(ends_with("time")) %>%
```

```
  glimpse()
```

```
## Rows: 336,776
## Columns: 5
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
```

```
flights %>%
  select(contains("time")) %>%
  glimpse()
```

```
## Rows: 336,776
## Columns: 6
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

```
flights %>%
  select(matches("time")) %>%
  glimpse()
```

```
## Rows: 336,776
## Columns: 6
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

If you want to rearrange the column (Variables) of your data set, then you can use `everything()` function. For example - you want to first put the **carrier** and **flight** variable and all variables after these two variables. In such case, you have to write the following code -

```
flights <- flights %>%
  select(carrier, flight, everything())
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 19
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL...
```

```
## $ flight         <chr> "1545", "1714", "1141", "72...
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 1...
## $ tailnum        <chr> "N14228", "N24211", "N619AA...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK",...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN",...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

5.9 2nd (Second) verb - `filter ()`

If we want to subset our dataset by **rows**, then `filter ()` is used. For example - we want the flights whose destination was Chicgo Airport (ORD).

```
chicago <- flights %>%
  filter(dest == "ORD")
glimpse(chicago)
```

```
## Rows: 17,283
## Columns: 19
## $ carrier        <chr> "UA", "AA", "MQ", "AA", "AA...
## $ flight         <chr> "1696", "301", "3768", "303...
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 554, 558, 608, 629, 656, 70...
## $ sched_dep_time <int> 558, 600, 600, 630, 700, 70...
## $ dep_delay      <dbl> -4, -2, 8, -1, -4, 9, 2, -6...
## $ arr_time       <int> 740, 753, 807, 824, 854, 85...
## $ sched_arr_time <int> 728, 745, 735, 810, 850, 83...
## $ arr_delay      <dbl> 12, 8, 32, 14, 4, 20, 21, -...
## $ tailnum        <chr> "N39463", "N3ALAA", "N9EAMQ...
## $ origin         <chr> "EWR", "LGA", "EWR", "LGA",...
## $ dest           <chr> "ORD", "ORD", "ORD", "ORD",...
## $ air_time       <dbl> 150, 138, 139, 140, 143, 13...
## $ distance       <dbl> 719, 733, 719, 733, 733, 73...
```

```
## $ hour           <dbl> 5, 6, 6, 6, 7, 7, 7, 7, 7, ...
## $ minute         <dbl> 58, 0, 0, 30, 0, 0, 13, 45,...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

Similarly, we want to subset the data for the month of January.

```
january <- flights %>%
  filter(month == "1")
glimpse(january)
```

```
## Rows: 27,004
## Columns: 19
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL...
## $ flight         <chr> "1545", "1714", "1141", "72...
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 1...
## $ tailnum        <chr> "N14228", "N24211", "N619AA...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK",...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN",...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

If we to subset the data for the month of January and February, the following code should be run

```
jan_feb <- flights %>%
  filter(month %in% c(1, 2))
glimpse(jan_feb)
```

```
## Rows: 51,955
## Columns: 19
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL...
## $ flight         <chr> "1545", "1714", "1141", "72...
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
```

```
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 1...
## $ tailnum        <chr> "N14228", "N24211", "N619AA...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK",...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN",...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

If we want to subset the data for all airlines other than American Airlines (AA), for the month of January and February and for the distance greater than 100, then the following code should be executed.

```
naa <- flights %>%
  filter(carrier != "AA", month %in% c(1, 2), distance > 1000
  )
glimpse(naa)
```

```
## Rows: 18,456
## Columns: 19
## $ carrier        <chr> "UA", "UA", "B6", "B6", "B6...
## $ flight         <chr> "1545", "1714", "725", "507...
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 544, 555, 558, 55...
## $ sched_dep_time <int> 515, 529, 545, 600, 600, 60...
## $ dep_delay      <dbl> 2, 4, -1, -5, -2, -2, -2, -...
## $ arr_time       <int> 830, 850, 1004, 913, 849, 8...
## $ sched_arr_time <int> 819, 830, 1022, 854, 851, 8...
## $ arr_delay      <dbl> 11, 20, -18, 19, -2, -3, 7,...
## $ tailnum        <chr> "N14228", "N24211", "N804JB...
## $ origin         <chr> "EWR", "LGA", "JFK", "EWR",...
## $ dest           <chr> "IAH", "IAH", "BQN", "FLL",...
## $ air_time       <dbl> 227, 227, 183, 158, 149, 15...
## $ distance       <dbl> 1400, 1416, 1576, 1065, 102...
## $ hour           <dbl> 5, 5, 5, 6, 6, 6, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 45, 0, 0, 0, 0, 0, ...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

Like the `select_*`, the verb `filter` has scoped versions such as `filter_if`, `filter_at`, and `filter_all`. If we want to learn more about these functions, we can use help functions such as `?filter_if` or `help(filter_if)` to know

more about these functions.

5.10 3rd (Third) verb - `arrange ()`

The `arrange ()`function allows you to reorder your data set by one or more variables. For example, if you want to reorder the `flights` dataset by distance, you need to execute the following code -

```
flights %>%
  arrange(distance)
```

```
## # A tibble: 336,776 x 19
##    carrier flight  year month   day dep_time
##    <chr>   <chr>  <int> <int> <int>    <int>
##  1 US      1632    2013     7    27       NA
##  2 EV      3833    2013     1     3     2127
##  3 EV      4193    2013     1     4     1240
##  4 EV      4502    2013     1     4     1829
##  5 EV      4645    2013     1     4     2128
##  6 EV      4193    2013     1     5     1155
##  7 EV      4619    2013     1     6     2125
##  8 EV      4619    2013     1     7     2124
##  9 EV      4619    2013     1     8     2127
## 10 EV      4619    2013     1     9     2126
## # ... with 336,766 more rows, and 13 more variables:
## #   sched_dep_time <int>, dep_delay <dbl>,
## #   arr_time <int>, sched_arr_time <int>,
## #   arr_delay <dbl>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

From the results, we can see the lowest distance was 17-mile flight between EWR and LGA. The next lowest distance was 80-mile flight between EWR and PHL. However, if you want to see the longest distance, then you have to use `desc ()` function becasue `arrange ()` function reorders the rows in ascending order (from lowest to highest).

```
flights %>%
  arrange(desc(distance))
```

```
## # A tibble: 336,776 x 19
##    carrier flight  year month   day dep_time
##    <chr>   <chr>  <int> <int> <int>    <int>
##  1 HA      51      2013     1     1      857
##  2 HA      51      2013     1     2      909
##  3 HA      51      2013     1     3      914
##  4 HA      51      2013     1     4      900
```

```
##  5 HA      51      2013     1     5      858
##  6 HA      51      2013     1     6     1019
##  7 HA      51      2013     1     7     1042
##  8 HA      51      2013     1     8      901
##  9 HA      51      2013     1     9      641
## 10 HA      51      2013     1    10      859
## # ... with 336,766 more rows, and 13 more variables:
## #   sched_dep_time <int>, dep_delay <dbl>,
## #   arr_time <int>, sched_arr_time <int>,
## #   arr_delay <dbl>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

It is evident that the highest distance was 4983 miles between JFK and HNL.

5.11 4th (Fourth) verb - `mutate ()`

The function `mutate ()` is used to **create new variables (columns)**. For example, we want to know the `total delay`, which is the sum of the `dep_delay` and `arr_delay`; then, we should write the following code -

```
flights <- flights %>%
  mutate(total_delay = dep_delay + arr_delay)
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 20
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL...
## $ flight         <chr> "1545", "1714", "1141", "72...
## $ year           <int> 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 542, 544, 554, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 55...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 7...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 1...
## $ tailnum        <chr> "N14228", "N24211", "N619AA...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK",...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN",...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 15...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-...
```

```
## $ total_delay    <dbl> 13, 24, 35, -19, -31, 8, 14...
```

We can use `mutate` function with other functions such as `ifelse` or `case_when` to create new variables. Examples are given below -

```
flights <- flights %>%
  mutate(total_delay_Dummy = ifelse(total_delay > mean(total_delay, na.rm = TRUE), "De
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 21
## $ carrier           <chr> "UA", "UA", "AA", "B6", ...
## $ flight            <chr> "1545", "1714", "1141", ...
## $ year              <int> 2013, 2013, 2013, 2013, ...
## $ month             <int> 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day               <int> 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time          <int> 517, 533, 542, 544, 554,...
## $ sched_dep_time    <int> 515, 529, 540, 545, 600,...
## $ dep_delay         <dbl> 2, 4, 2, -1, -6, -4, -5,...
## $ arr_time          <int> 830, 850, 923, 1004, 812...
## $ sched_arr_time    <int> 819, 830, 850, 1022, 837...
## $ arr_delay         <dbl> 11, 20, 33, -18, -25, 12...
## $ tailnum           <chr> "N14228", "N24211", "N61...
## $ origin            <chr> "EWR", "LGA", "JFK", "JF...
## $ dest              <chr> "IAH", "IAH", "MIA", "BQ...
## $ air_time          <dbl> 227, 227, 160, 183, 116,...
## $ distance          <dbl> 1400, 1416, 1089, 1576, ...
## $ hour              <dbl> 5, 5, 5, 5, 6, 5, 6, 6, ...
## $ minute            <dbl> 15, 29, 40, 45, 0, 58, 0...
## $ time_hour         <dttm> 2013-01-01 05:00:00, 20...
## $ total_delay       <dbl> 13, 24, 35, -19, -31, 8,...
## $ total_delay_Dummy <chr> "Not Delayed", "Delayed"...
```

Note that we can now use the `count` function to know the number of flights delayed -

```
flights %>%
  count(total_delay_Dummy)
```

```
## # A tibble: 3 x 2
##   total_delay_Dummy      n
##   <chr>              <int>
## 1 Delayed            86255
## 2 Not Delayed       241091
## 3 <NA>                9430
```

5.12 5th (Fifth) verb - summarize ()

The summarize () function is used to calculate different statistics such as mean, median, standard deviation, maximum, and minimum value. For example, we want to calucate the average distance and average delay of all flights in the month of January -

```
flights %>%
  filter(month == "1") %>%
  summarise(avg_distance = mean (distance),
            avg_delay = mean (total_delay, na.rm = TRUE),
            max_distance = max (distance),
            min_distance = min (distance, na.rm = TRUE),
            std_distance = sd (distance),
            med_distance = median (distance)
            )
```

```
## # A tibble: 1 x 6
##   avg_distance avg_delay max_distance min_distance
##          <dbl>     <dbl>        <dbl>        <dbl>
## 1        1007.      16.1         4983           80
## # ... with 2 more variables: std_distance <dbl>,
## #   med_distance <dbl>
```

5.13 6th (Sixth) verb - group_by ()

The group_by () function is very useful when it is used with summarize () function. For example, we want to know the average delay of each airport in New York in descending order; then, we should write the following code -

```
flights %>%
  group_by(origin) %>%
  summarize(avg_delay = mean (total_delay, na.rm = TRUE)) %>%
  arrange(desc(avg_delay))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   origin avg_delay
##   <chr>      <dbl>
## 1 EWR         24.1
## 2 JFK         17.6
## 3 LGA         16.1
```

If you want to know the average delay of each carrier, then you need to write the following code -

```
flights %>%
  group_by(carrier) %>%
  summarize(avg_delay = mean (total_delay, na.rm = TRUE)) %>%
  arrange(desc(avg_delay))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 16 x 2
##    carrier avg_delay
##    <chr>       <dbl>
##  1 F9          42.1
##  2 FL          38.7
##  3 EV          35.6
##  4 YV          34.5
##  5 WN          27.3
##  6 OO          24.5
##  7 9E          23.8
##  8 B6          22.4
##  9 MQ          21.2
## 10 UA          15.6
## 11 VX          14.5
## 12 DL          10.9
## 13 AA           8.93
## 14 US           5.87
## 15 HA          -2.01
## 16 AS          -4.10
```

If you want to know the average delay of each month, then you need to write the following code -

```
flights %>%
  group_by(month) %>%
  summarize(avg_delay = mean (total_delay, na.rm = TRUE)) %>%
  arrange(desc(avg_delay))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 12 x 2
##    month avg_delay
##    <int>     <dbl>
##  1     7     38.2
##  2     6     37.2
##  3    12     31.4
##  4     4     25.0
##  5     3     19.0
##  6     8     18.6
##  7     5     16.4
```

```
##  8     2     16.4
##  9     1     16.1
## 10    10      6.07
## 11    11      5.88
## 12     9      2.61
```

Chapter 6

Data Visualization

6.1 ggplot2 () Package - Data Visualization Tool

The package `ggplot2 ()` is a very powerful tool for data visualization.

Please use the following website to develop this chapter - https://r-graphics.org/index.html

6.2 Scatter Plot

In scatter Plot, we use to continuous variables in x axis and y axis. The following code is run to create a scatter plot of two variables called `time_hour` and `total_delay`. See Figure 6.1.

```
library(nycflights13)
library(tidyverse)
flights <- flights %>%
  mutate(total_delay = dep_delay + arr_delay)
```

```
flights %>%
  ggplot(mapping = aes( x = time_hour, y = total_delay))+
  geom_point()
```

```
## Warning: Removed 9430 rows containing missing values
## (geom_point).
```

However, we can also use a third variable in scatter plot. See Figure 6.2. Also see Figure 6.3.

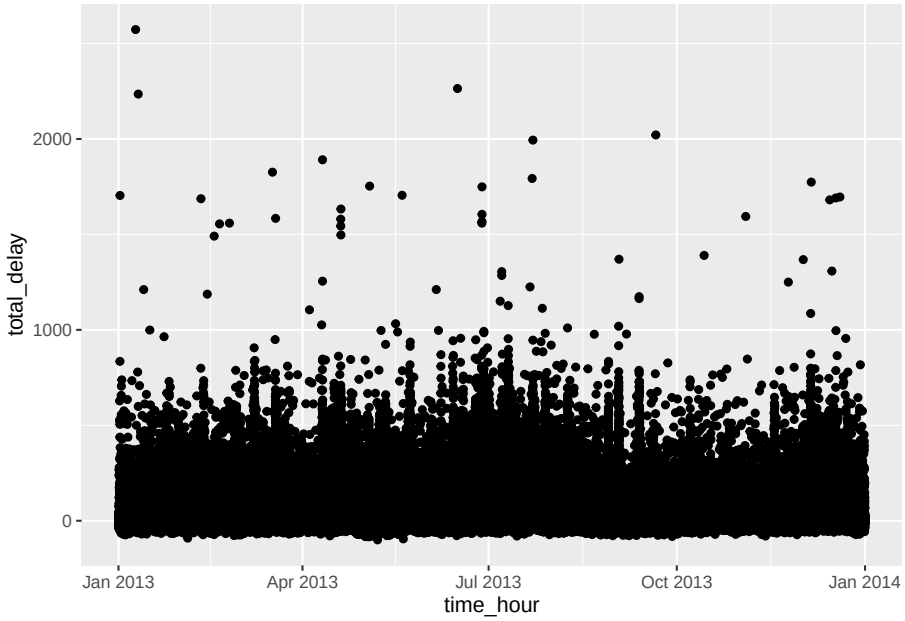

Figure 6.1: A Scatterplot of Total Delay and Month

```
flights %>%
  ggplot(mapping = aes( x = time_hour, y = total_delay))+
  geom_point()+
  facet_wrap(~origin)
```

```
## Warning: Removed 9430 rows containing missing values
## (geom_point).
```

```
flights %>%
  ggplot(mapping = aes( time_hour, total_delay, color = origin))+
  geom_point()
```

```
## Warning: Removed 9430 rows containing missing values
## (geom_point).
```

6.3 Line Plot

In line plot, we draw line for the points of two continuous variables. See Figure 6.4.

```
flights %>%
  group_by(month) %>%
```

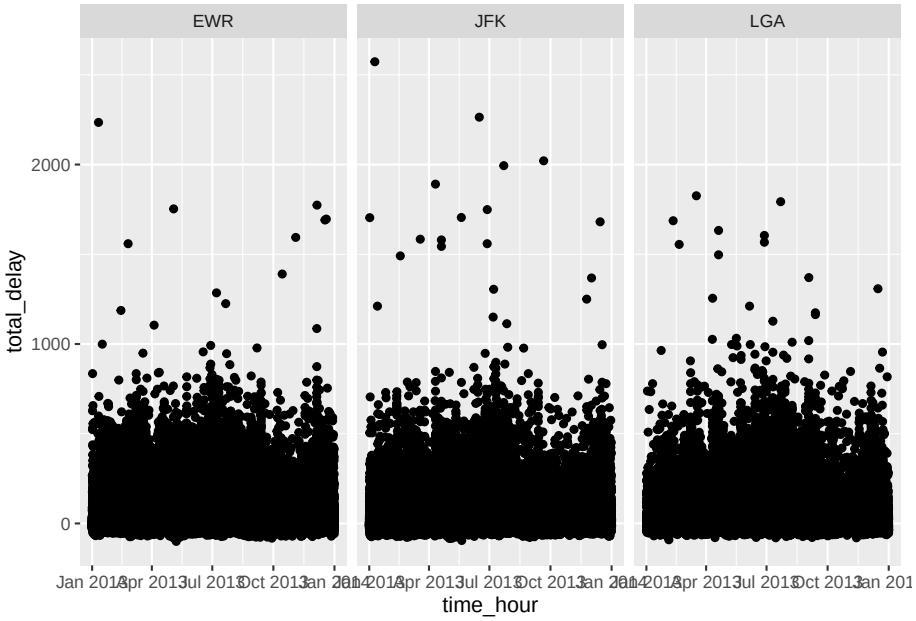

Figure 6.2: A Scatterplot of Total Delay and Month in Departing Airports in New York

```
  summarize(avg_delay = mean (total_delay, na.rm = TRUE)) %>%
  ggplot(mapping = aes(x = month, y = avg_delay)) +
  geom_smooth()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Like scatter plot, a third variable can also be added to the line plot. See Figure 6.5. Also see Figure 6.6.

```
flights %>%
  ggplot(mapping = aes(x = month, y = total_delay, color = origin)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
## Warning: Removed 9430 rows containing non-finite values
## (stat_smooth).
```

```
flights %>%
  ggplot(mapping = aes(x = day, y = total_delay, color = origin)) +
  geom_smooth()
```

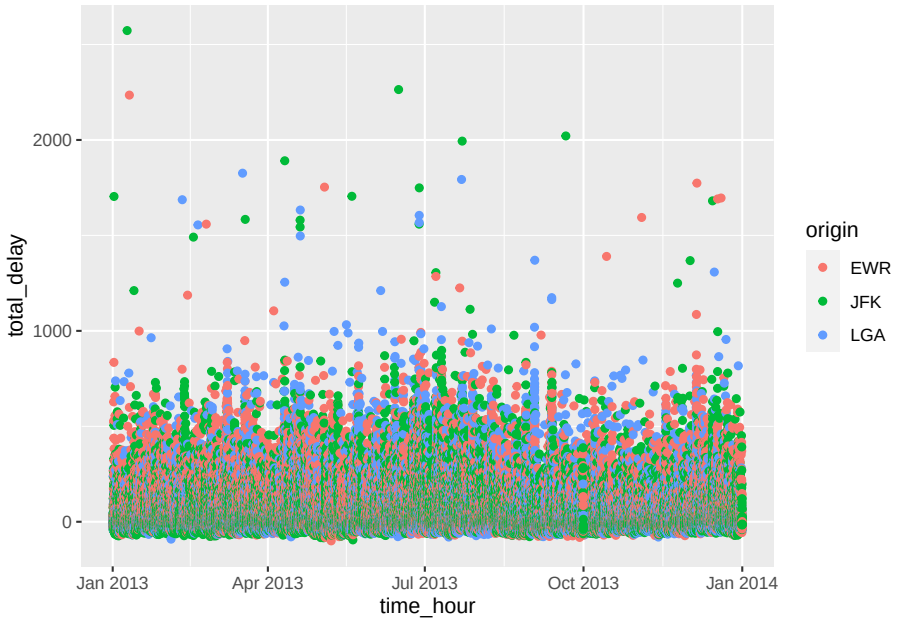

Figure 6.3: A Scatterplot of Total Delay and Month Using Origin as Third Variable

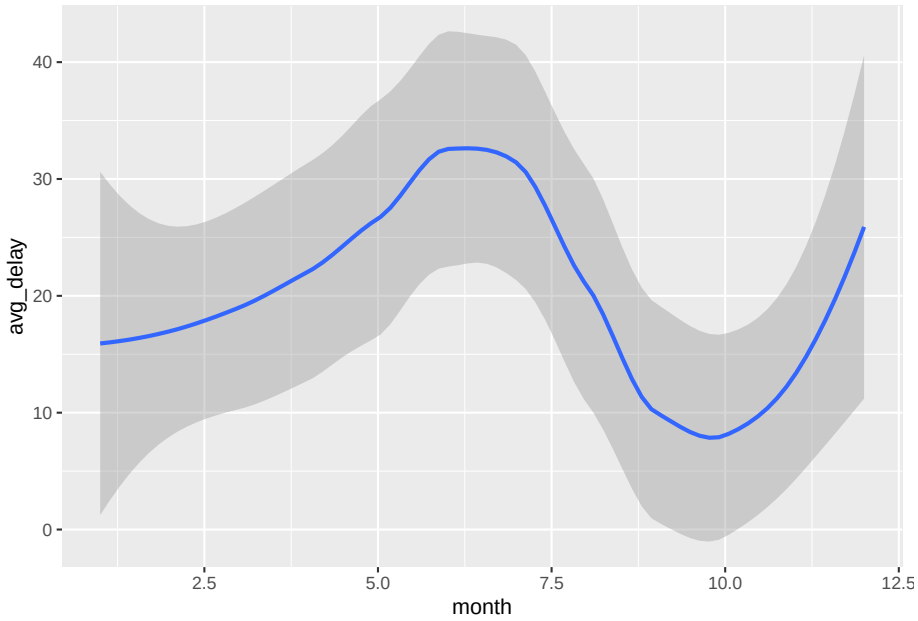

Figure 6.4: A lineplot of Average Delay and Month

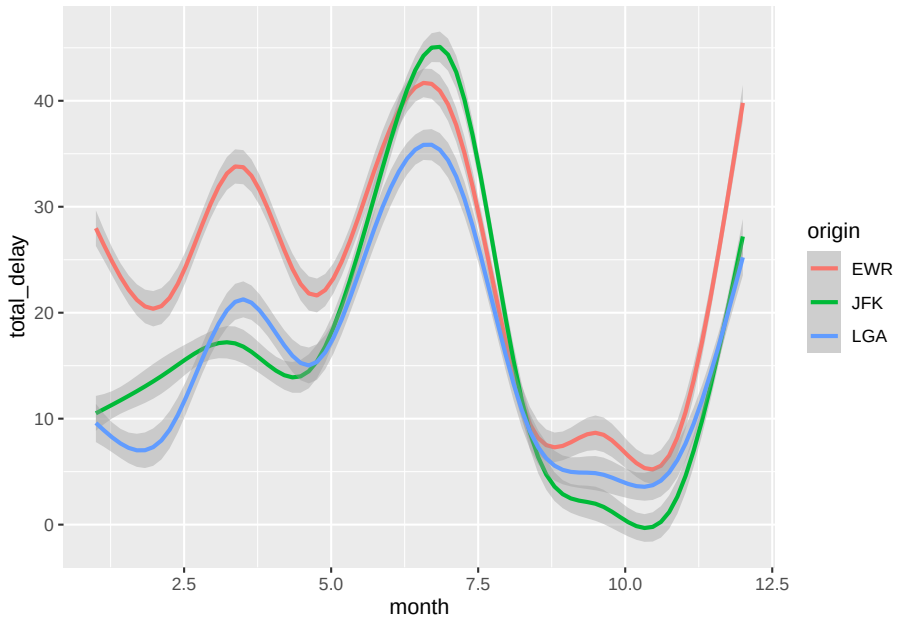

Figure 6.5: A lineplot of Average Delay and Month Using Origin as Third Variable

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
## Warning: Removed 9430 rows containing non-finite values
## (stat_smooth).
```

6.4 Bar Plot

We can also create bar diagram. See Figure 6.7.

```
ggplot(flights, aes(origin))+
  geom_bar()
```

We can also include a second variable in bar diagram. Please see Figure 6.8. Also see Figure 6.9.

```
ggplot(flights, aes(origin))+
  geom_bar(aes(fill = as.character(month)))
```

```
ggplot(flights, aes(origin))+
  geom_bar(aes(fill = carrier))
```

From the graph, we can figure out which carriers use which airport most.

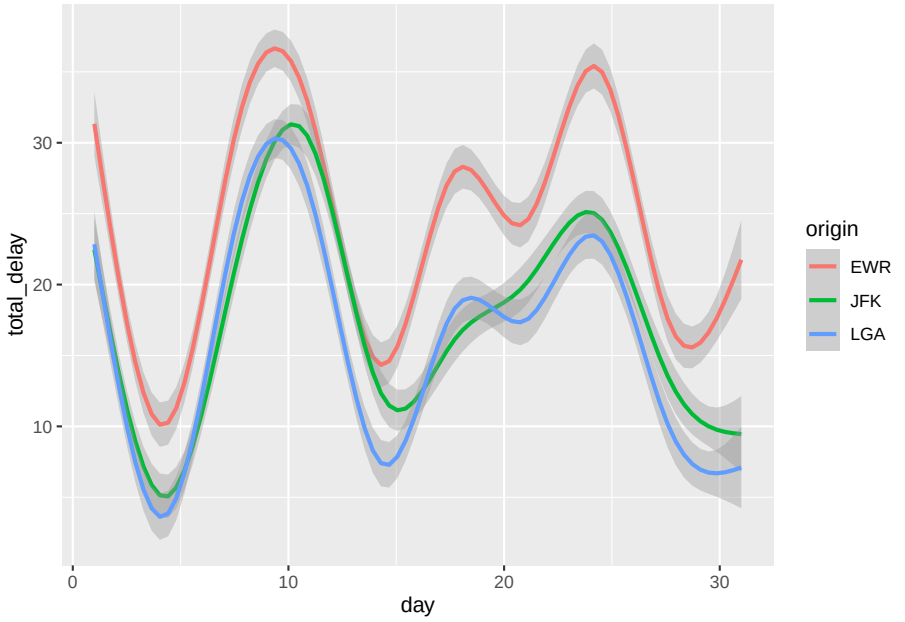

Figure 6.6: A lineplot of Total Delay and Day of the Week Using Origin as Third Variable

6.5 Box Plot

We can also create boxplot using `ggplot2`. Please see Figure 6.10.

```
ggplot(flights, aes ( origin,distance))+
  geom_boxplot()
```

It is evident that long distance flights use JFK because it has the highest distance. A thrid variable also can be included in box plot. Please see Figure 6.11.

```
ggplot(flights, aes ( origin,distance))+
  geom_boxplot()+
  facet_wrap(~ month)
```

Here boxplot is created for the `distance` variables by `origin` and `month` variables

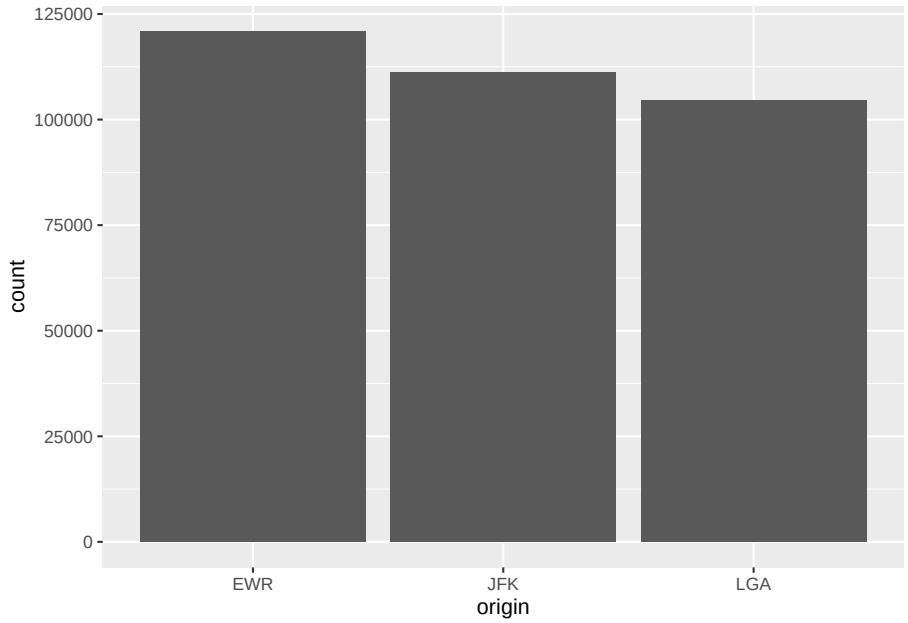

Figure 6.7: A barplot of Number of Flights from Departing Airports in New York

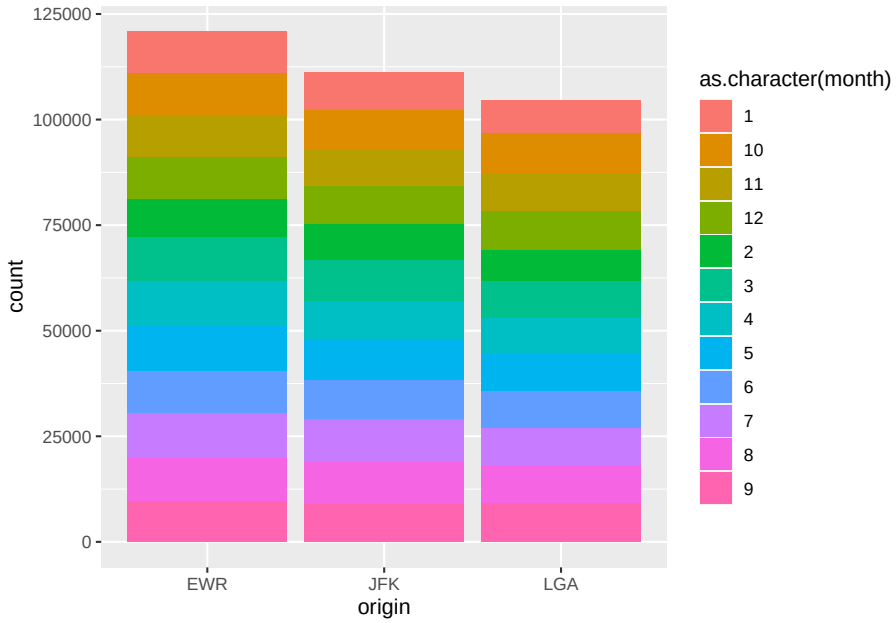

Figure 6.8: A barplot of Number of Flights from Departing Airports in New York in Different Months

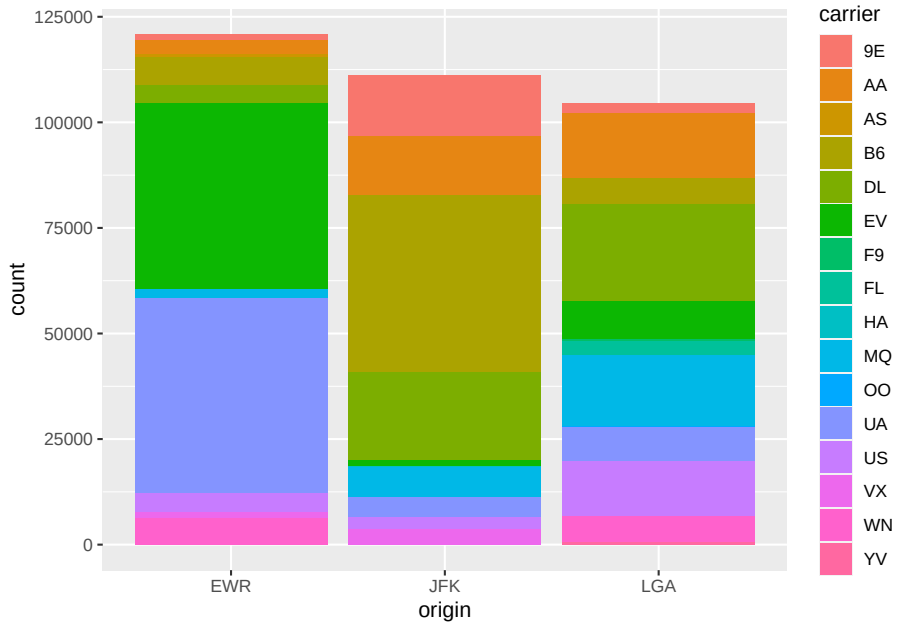

Figure 6.9: A barplot of Number of Flights from Departing Airports in New York by Different Carriers

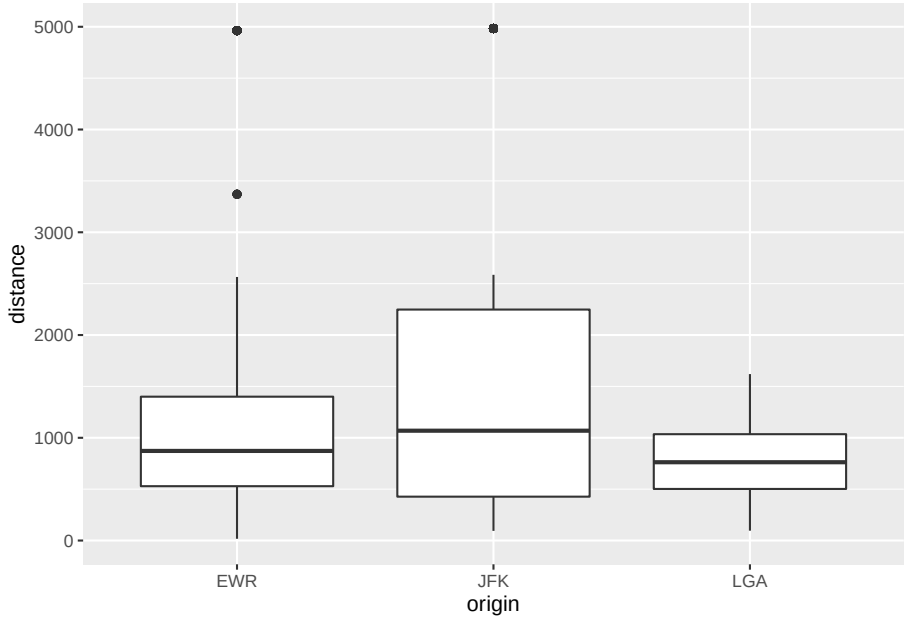

Figure 6.10: A boxplot of Distance from Departing Airports in New York

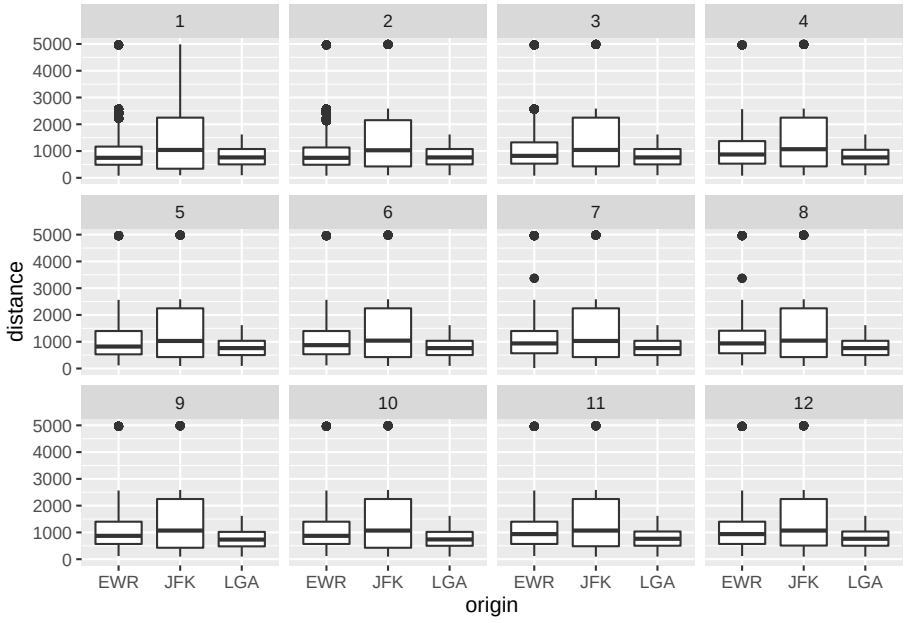

Figure 6.11: A boxplot of Distance from Departing Airports in New York in Different Months

Chapter 7

Data Modeling

This is the ____ real chapter. The best link for modelling in R - https://ademos.people.uic.edu/

Chapter 8

Communication of Analytics: An `Rmarkdown` Approach

Learning Objectives

By the end of this chapter, students should have learned the importance of communication in data science and particulary in auditing and accounting. The students should specifically learn the following:

1. The importance of communication in data science.
2. The importance of communication in Auditing
3. What `Shiny` is
4. How `Shiny` could improve the communication of data/audit analytics.

8.1 What is Rmarkdown

RMarkdown enables analysts to engage with code interactively, embrace literate programming, and rapidly produce a wide variety of high-quality data products such as documents, emails, dashboards, and websites. Emily Riederer[1], an analytics manager at Capital One, describes how reproducible programming using RMarkdown can help R users develop better software programming practices.

Use the following website to develop this chapter - https://m-clark.github.io/Introduction-to-Rmarkdown/

Another good Website -

[1] https://rstudio.com/resources/rstudioconf-2020/rmarkdown-driven-development/

https://rpubs.com/thealk/academic-writing This one is one of the best - https://rc2e.com/rmarkdown

For other forms of communication of analytics - https://rmd4sci.njtierney.com/different-outputs-and-extensions.html

Chapter 9

Communication of Analytics: A `Shiny` Approach

Learning Objectives

9.1 Introduction to `R Shiny`

This is the _____ real chapter. Good Website for `R Shiny` - https://uvastatlab.github.io/phdplus/shinyr.html.

Chapter 10

Conclusion

This is the _____ real chapter.

Appendix A

.1 Basic Data Structure in R

Though in the book, most of the time we talk about `data frames` (sometimes `tibble`), `vectors` are the buliding blocks of them. Traditional learners of `R` usually start learning it using `vectors`. However, for accountants (or would-be accountants) it is better to begin with `tibbles` and work down to the underlying concepts such as `vectors`.

Basically there are two types of vectors in `R`; one is called `Atomic` vectors (also called `homogeneous vectors`) and the other is called `lists`. `Atomic` vectors contain similar types of elements, while `lists` vectors do not; `lists` can hold differnet types elements in a vector. There are six types of `Atomic` vectors - `logical`, `integer`, `double`, `character`, `complex`, and `raw`. The `integer`and `double` are together called `numeric`. Of these types, the first four are widely used and most relevant for accounting analytics.

Each vector has 2 major characteristis - `type` and `length`. The function `typeof` can be used to know about the `types` of the vectors - namely `logical`, `integer`, `double`, `character`, `complex`, or `raw`. The `length` function is used to get or set the length of a vector. The function `nchar` can be used to get the length of a string. Some examples of vectors are given below -

```
# Character type
a <- c("1933","1934","2002")
length(a)
```

```
## [1] 3
```

```
typeof(a)
```

```
## [1] "character"
```

```
a
```

```
## [1] "1933" "1934" "2002"
```

```
b <- c("The Securities Act", "The Securities Exchange Act", "Sarbanes-Oxley Act")
length(b)
```

```
## [1] 3
```

```
typeof(b)
```

```
## [1] "character"
```

```
nchar(b)
```

```
## [1] 18 27 18
```

```
b
```

```
## [1] "The Securities Act"
## [2] "The Securities Exchange Act"
## [3] "Sarbanes-Oxley Act"
```

```
# Logical type
c <- c(TRUE, FALSE, TRUE, TRUE, TRUE, FALSE)
length(c)
```

```
## [1] 6
```

```
typeof(c)
```

```
## [1] "logical"
```

```
c
```

```
## [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE
```

```
# Double type
d <- c(1933, 1934, 2002)
length(d)
```

```
## [1] 3
```

```
typeof(d)
```

```
## [1] "double"
```

```
d
```

```
## [1] 1933 1934 2002
```

```
# Integer type
e <- c(1933L, 1934L, 2002L)
length(e)
```

```
## [1] 3
```

```
typeof(e)
```

```
## [1] "integer"
```

```
e
```

```
## [1] 1933 1934 2002
```

```
f <- list(a,b,c,d,e)
length(f)
```

```
## [1] 5
```

```
typeof(f)
```

```
## [1] "list"
```

```
f
```

```
## [[1]]
## [1] "1933" "1934" "2002"
##
## [[2]]
## [1] "The Securities Act"
## [2] "The Securities Exchange Act"
## [3] "Sarbanes-Oxley Act"
##
## [[3]]
## [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE
##
## [[4]]
## [1] 1933 1934 2002
##
## [[5]]
## [1] 1933 1934 2002
```

Very good website - https://rc2e.com/somebasics#recipe-id017

Appendix B

.2 Starting a Project in R

.3 Text Mining in R

This is the text mining in R.

.4 Social Media Analytics in R

This is the social media analytics in R.

.5 Web Scrapping Using R

This is web scrapping in R.

.6 Big Data in R with sparklyr

The best website to learn about how to use R with sparklyr for big data analytics is - https://therinspark.com/

Bibliography

Alles, M. G. (2015). Drivers of the Use and Facilitators and Obstacles of the Evolution of Big Data by the Audit Profession. *Accounting Horizons*, 29(2):439–449.

American Institute of Certified Public Accountants (AICPA) (2015). Audit Analytics and Continuous Audit: Looking toward the future.

American Institute of Certified Public Accountants (AICPA) (2017). *Description Criteria for Management's Description of the Entity's Cybersecurity Risk Management Program*.

Barton, D. and Court, D. (2012). Making advanced analytics work for you. *Harvard Business Review*, 90(10):78–83.

Bollen, J., Mao, H., and Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8. Publisher: Elsevier.

Cao, M., Chychyla, R., and Stewart, T. (2015). Big Data analytics in financial statement audits. *Accounting Horizons*, 29(2):423–429.

Columbus (2017). 53% Of Companies Are Adopting Big Data Analytics.

Crawley, M. and Wahlen, J. (2014). Analytics in empirical/archival financial accounting research. *Business Horizons*, 57(5):583–593. Publisher: Elsevier.

Davis, A. K., Piger, J. M., and Sedor, L. M. (2012). Beyond the numbers: Measuring the information content of earnings press release language. *Contemporary Accounting Research*, 29(3):845–868. Publisher: Wiley Online Library.

Feldman, R., Govindaraj, S., Livnat, J., and Segal, B. (2010). Management's tone change, post earnings announcement drift and accruals. *Review of Accounting Studies*, 15(4):915–953. Publisher: Springer.

Lehavy, R., Li, F., and Merkley, K. (2011). The effect of annual report readability on analyst following and the properties of their earnings forecasts. *The Accounting Review*, 86(3):1087–1115.

Li, F. (2008). Annual report readability, current earnings, and earnings persistence. *Journal of Accounting and economics*, 45(2-3):221–247. Publisher: Elsevier.

Li, F. (2010). The information content of forward-looking statements in corporate filings—A naïve Bayesian machine learning approach. *Journal of Accounting Research*, 48(5):1049–1102. Publisher: Wiley Online Library.

Li, F., Lundholm, R., and Minnis, M. (2013). A measure of competition based on 10-K filings. *Journal of Accounting Research*, 51(2):399–436. Publisher: Wiley Online Library.

Protiviti (2017). Embracing Analytics in Auditing.

Provost, F. and Fawcett, T. (2013). Data science and its relationship to big data and data-driven decision making. *Big data*, 1(1):51–59.

Richins, G., Stapleton, A., Stratopoulos, T. C., and Wong, C. (2017). Big Data analytics: Opportunity or threat for the accounting profession? *Journal of Information Systems*, 31(3):63–79.

Schneider, G. P., Dai, J., Janvrin, D. J., Ajayi, K., and Raschke, R. L. (2015). Infer, predict, and assure: Accounting opportunities in data analytics. *Accounting Horizons*, 29(3):719–742.

Sivarajah, U., Kamal, M. M., Irani, Z., and Weerakkody, V. (2017). Critical analysis of Big Data challenges and analytical methods. *Journal of Business Research*, 70:263–286.

Vasarhelyi, M. A., Kogan, A., and Tuttle, B. M. (2015). Big Data in accounting: An overview. *Accounting Horizons*, 29(2):381–396.

Verver, J. (2015). Six Audit Analytics Success Factors. *Internal Auditor*, 72(3):20–21.

Warren Jr, J. D., Moffitt, K. C., and Byrnes, P. (2015). How Big Data will change accounting. *Accounting Horizons*, 29(2):397–407.

Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, 59(10):1–23.

Wickham, H. and Grolemund, G. (2017). R for Data Science.