CHAPTER

3 DATA SCIENCE TOOLS AND TECHNIQUES TO SUPPORT DATA ANALYTICS IN TRANSPORTATION APPLICATIONS

Linh B. Ngo
Clemson University, Clemson, SC, United States

3.1 INTRODUCTION

In recent years, technological advances have greatly simplified the generation and dissemination of information. This results in a deluge of traffic and infrastructure-related data coming from different sources, originating from traditional means such as roadway and roadside devices and transportation infrastructure survey to emerging ones such as embedded sensors from mobile devices and online media sources. The variety among sources contributes to the increased complexity of the preprocessing of data prior to the analytical steps. Researchers and engineers working with connected transportation system need to have access to a set of tools and methods that enable them to extract, ingest, and integrate meaningful data values from different sources and under various formats into a coherent data structure, to which specific data analytic techniques can be applied.

This introductory chapter will familiarize readers with such data science toolsets in anticipation of the analytical techniques in the remainder of this book. We cover the following topics within this chapter:

- Introduction to R, a de-facto statistical programming environments for complex data analytics.
- Introduction to Research Data Exchange (RDE), the largest repository for transportation data.
- Fundamental concepts about how to structure data in R.
- Techniques and libraries to ingest data files from external formats into R.
- Techniques and libraries to extract data from online sources into R.
- Introduction to Big Data processing.

Besides R, Python is another popular choice for data science. However, as R was developed by statisticians and has received significant contributions from the statistic community over several decades, it has a much richer collection of analytical libraries. On the other hand, Python surpasses R in terms of popularity as a multipurpose programming tool. As we progress through this chapter, we will also provide an alternative solution in Python to data ingestion and curation examples as necessary for reference purposes.

3.2 INTRODUCTION TO THE R PROGRAMMING ENVIRONMENT FOR DATA ANALYTICS

Courses that teach the fundamentals of programming skills have now become mandatory across the majority of science and engineering disciplines. Matlab, C, and Fortran are often selected to be taught in these courses, as these languages provide strong support for computational engineering and scientific simulation tasks. Each of these languages has the capability to support complex statistical and visual analytics. However, C and Fortran require users to follow the strict cycle of writing, compiling, running, and debugging codes, which make them not suited for the interactive nature of data analytics. Matlab can provide such an interactive environment, but it is proprietary and has a relatively complex GUI to support many other engineering tasks that Matlab was designed to perform (Fig. 3.1).

R was born out of the need to have a free open source environment to support statistic work. Designed and developed by two statisticians at Auckland University, New Zealand, R has quickly gained the acceptance and support of the statistic community as the standard open source statistical environment.

At the first glance, working with R is similar to Matlab and other interpreted languages. It is a functional programming with some object-oriented support. To support matrix arithmetic, R has built-in data structures such as vectors, matrices, data frames, and lists. Compared to other proprietary products, R has the advantage of having a significant level of community contributions in the form of custom libraries called *packages*. These contributions include a vast collection of various statistical and visual packages. Examples include linear and nonlinear techniques, statistical tests, time-series analysis, machine learning, and many others. R also has a rich collection of packages to help with nonanalytic tasks such as importing, cleaning, and manipulating data or optimizing performance (Fig. 3.2).

As an open source software, R can be downloaded from https://www.r-project.org/, and is available to all three platforms, Windows, Linux, and MacOS. By default, R comes with a standard GUI with limited capabilities. A recommended alternative is RStudio (https://www.rstudio.com/), an open source IDE (integrated development environment) for R. Installation guides for R and RStudio are straightforward and can be found at the above URLs (Fig. 3.3).

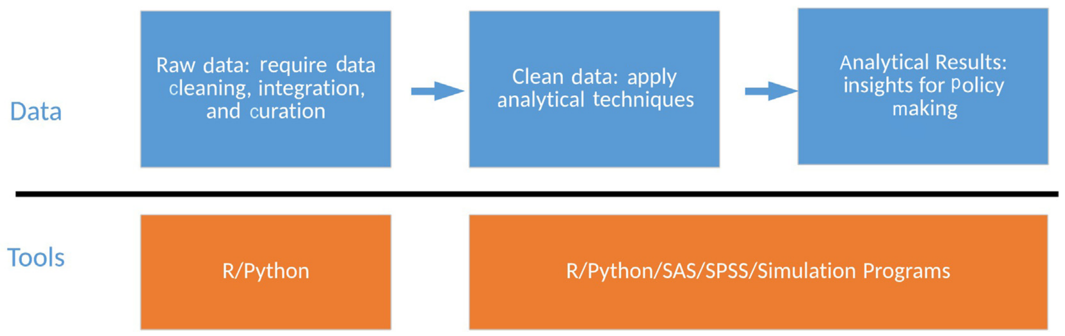
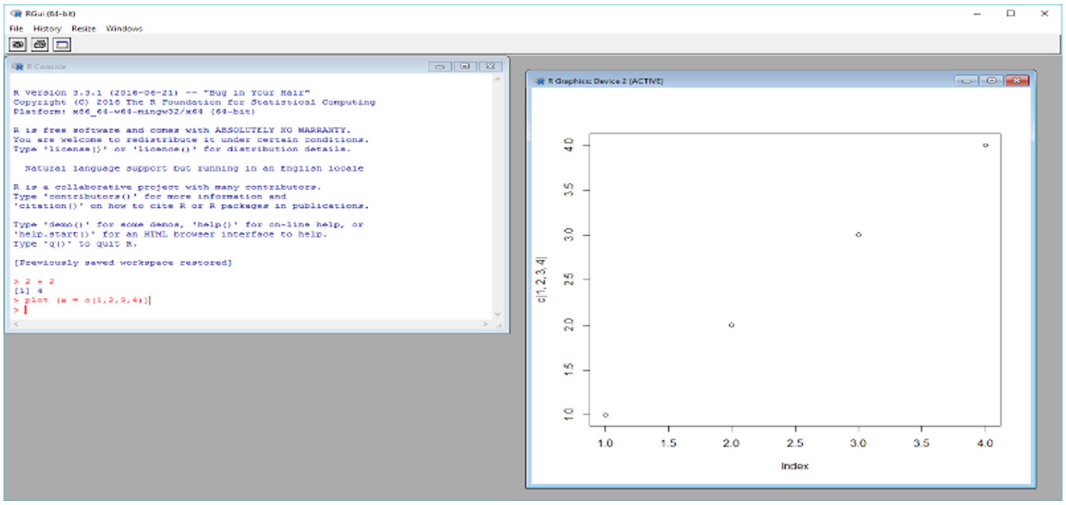

Data
Raw data: require data cleaning, integration, and curation
Clean data: apply analytical techniques
Analytical Results: insights for Policy making
Tools
R/Python
R/Python/SAS/SPSS/Simulation Programs

FIGURE 3.1

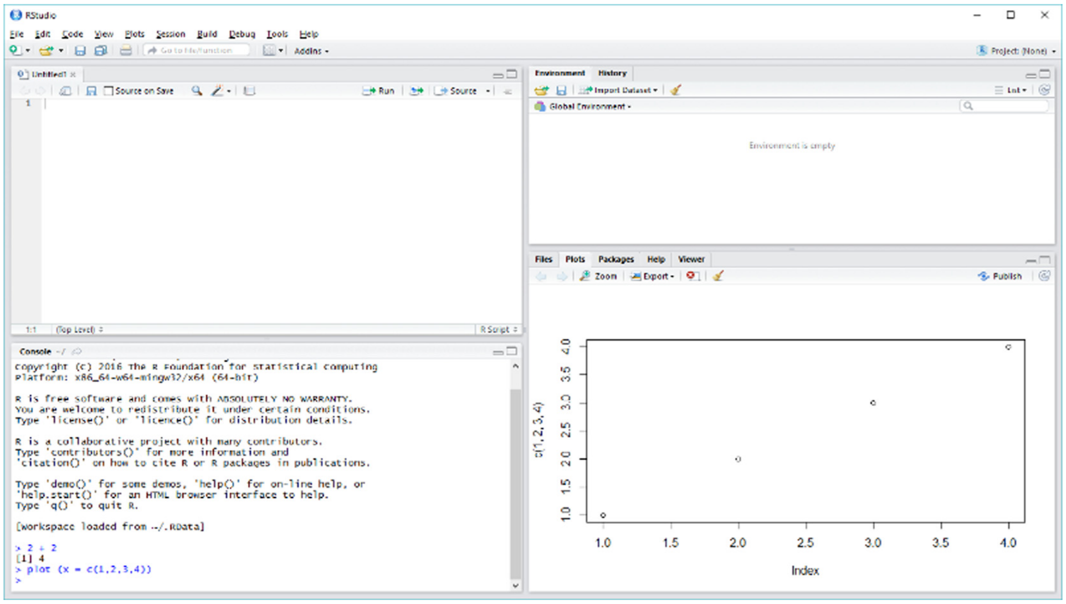
Applicability of R, Python, and other programming tools to the data processing pipeline.



```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 2 + 2
[1] 4
> plot (x = c(1,2,3,4))
>
```

FIGURE 3.2

The default Graphical User Interface (GUI) provided by R.



```
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> 2 + 2
[1] 4
> plot (x = c(1,2,3,4))
>
```

FIGURE 3.3

RStudio: RStudio, a popular customized Integrated Development Environment (IDE) for R.

3.3 RESEARCH DATA EXCHANGE

Data for connected vehicle systems come from a variety of sources. Typical transportation data includes all data generated by roadway/roadside devices with technologies such as inductive loop, magnetometer, infrared, acoustic detection, ultrasonic detection, charged coupled devices, Doppler radar detection, and pulsed radar [1]. The quality and accuracy of data provided by these devices vary under different traffic and environmental conditions, especially for real-time traffic incident detection. Other emerging sources of sensor-based traffic condition data include mobile and embedded on-board vehicle devices. Few academic institutions have the capability to set up laboratories to support CVS data collection, and even then, the collected data often serve specific transportation applications.

Transportation data collected from state and federal highways and streets are available to the public. However, the efforts required to curate and publish these data are nontrivial. As a result, few states have the resources to make data available online, others only provide aggregated data summary and make detailed information available upon requests. Most, if not all, of state-level transportation data are accessible or can be requested through individual state's Department of Transportation websites.

To facilitate research, analysis, and application development for the transportation community, the U.S. Department of Transportation (USDOT) has made available a public data repository called RDE. RDE is a critical component of the USDOT's Connected Data Systems Program to "improve safety, mobility, and environmental impacts of our surface transportation system" [2]. All RDE data are available at https://www.its-rde.net/, and depending on the size of individual data sets, we can either download them directly or request emails with download links to be sent to email addresses associated with registered accounts.

In this chapter, we will use RDE data sets from the Pasadena environment. This environment collects data from a variety of sources including highway network data, demand data, highway performance data, work zone, weather, and CCTV camera. More importantly, within this single environment, we have the three data formats that are representative across other environments: CSV, XML, and SQL File.

3.4 FUNDAMENTAL DATA TYPES AND STRUCTURES: DATA FRAMES AND LIST

The fundamental data type for R is not a single scalar but a vector, which is defined as an indexed set of values that have the same *type*. The type of these values defines the *class* of the vector. There are five primitive data types in R: numeric, integer, complex, logical, and character. Numeric, which is the most relevant type given the statistical nature of R, is a double precision type.

To support numerical arithmetic, R uses atomic vector, matrix, and array, for one, two, and *n*-dimensional data structures respectively. These data structures require homogeneous data.

While statistical analysis can be expressed mathematically using these structures, from an inferential perspective, it is not intuitive, especially when users have to include nonnumeric data for analytical purposes. To support heterogeneous and complex data, R uses two additional structures: data frames and list.

3.4.1 DATA FRAME

Data frame is a matrix-like data structure. Unlike the matrix structure, the columns of data frames can contain data of different types.

EXAMPLE 3.1

A matrix is created out of two numeric vectors

```
> A <- matrix (c(c(2,4,3),c(1,5,9)), nrow=3, ncol=2)
> A
     [,1] [,2]
[1,] 2    1
[2,] 4    5
[3,] 3    9
```

A matrix is created out of one numeric vector and one literal vector:

```
> A <- matrix (c(c(2,4,3),c('1','5','9')), nrow=3, ncol=2)
> A
     [,1] [,2]
[1,] "2"  "1"
[2,] "4"  "5"
[3,] "3"  "9"
```

In the latter case, the matrix structure cannot support two columns with different data types, and both columns are converted to type character. The reason for this is that in R, a matrix is simply a vector with additional dimensional attributes added. As show in the following example, value 4 can be reached through a two-dimensional coordinate [2, 1] or a one-dimensional coordinate index of 2.

```
> A[2,1]
[1] "4"
> A[2]
[1] "4"
>
```

While data frames have the same two-dimensional structure as matrices, their structure is considered a collection of vectors rather than a single vector with dimensional attributes. This can be observed from the syntax of matrix and data frame creation functions.

EXAMPLE 3.2

A matrix is created by adding dimensional information to a vector

```
    > A <- matrix (c(2,4,3,1,5,7), nrow = 3, ncol = 2)
    > A
         [,1] [,2]
    [1,]  2    1
    [2,]  4    5
    [3,]  3    7
```

EXAMPLE 3.3

A data frame is created by grouping multiple vectors together. Without specific column names, the data frame uses the vectors' contents as the default column names.

```
    > A <- data.frame (c(2,4,3),c('one','five','seven'))
    > A
      c.2..4..3.  c..1....5....7..
    1  2           "one"
    2  4           "five"
    3  3           "seven"
```

EXAMPLE 3.4

A one-dimensional coordinate index in a data frame points to an entire vector

```
    > A[2]
       c..1....5....7..
    1   "one"
    2   "five"
    3   "seven"
```

EXAMPLE 3.5

A data frame with named columns.

```
    > A = data.frame (x = c(2,4,3),y = c('one','five','seven'))
    > A
       x   y
    1  2   one
    2  4   five
    3  3   seven
```

A data frame's ability to group vectors with different data types into a single tabular data structure has made it the default structure to which external heterogeneous tabular data can be imported. Furthermore, data frames are considered the fundamental data structure input for the majority of statistical packages using R.

3.4.2 LIST

List is a vector structure. Unlike vectors, the elements of a list do not have to conform to a common type but can be any objects. To access an element of a list, a double bracket notation ([[X]]) is used.

EXAMPLE 3.6

A list with multiple data structures: vector, matrix, and data frame

```
> A <- c(1,2,3,4,5)
> B <- matrix (c(2,4,3,1,5,7), nrow=3, ncol=2)
> C <- data.frame(x=c(1,2,3), y=c("two","four","six"))
> D <- list(A, B, C)
> D
[[1]]
[1] 1 2 3 4 5
[[2]]
     [,1] [,2]
[1,]  2    1
[2,]  4    5
[3,]  3    7
[[3]]
  x  y
1 1  two
2 2  four
3 3  six
```

An element of a list can be another list, allowing for the construction of a tree-like data structure. As a result, the list can support nested data types such as XML and JSON.

3.5 IMPORTING DATA FROM EXTERNAL FILES

3.5.1 DELIMITED

A delimited file is a text-based file containing tabular data whose columns are separated by a predefined separator. This separator could be a tab, comma, semicolon, or any nonalphanumeric character. Within the Pasadena data sets, there are two types of delimited files: comma-delimited and tab-delimited with extended header lines.

A file whose ending extension is csv is often comma-delimited. CSV stands for comma separated values. A CSV file may or may not have the first line as a heading line, which contains the names of the data columns. Examples of CSV files in the Pasadena data sets include link and turn volume data, simulated link volumes, link capacity, link speed, turn capacity, turn delay, incident data, weather data, CMS data, and signal phase data.

A drawback of a CSV file is the comma itself, which is not appropriate for cases where data also contain commas (e.g., addresses or sentences). The work zone data from the Pasadena set is such a case, and its delimiter is the pipe character.

EXAMPLE 3.7

A portion of the comma-delimited (csv) file describing link and turn volume data of the Pasadena data environment is shown below:

```
2.01109E + 13,200011501,0,0,0,FALSE,FALSE,FALSE
2.01109E + 13,200011504,0,0,0,FALSE,FALSE,FALSE
2.01109E + 13,200011506,0,0,0,FALSE,FALSE,FALSE
2.01109E + 13,200028101,0,0,0,FALSE,FALSE,FALSE
2.01109E + 13,200028103,0,0,0,FALSE,FALSE,FALSE
```

EXAMPLE 3.8

A portion of the Pasadena's Work Zone data stored in a pipe-delimited file:

```
C5AA | 157 | 01/25/2011 18:38 | 02/06/2011 19:01 | 02/29/2012 06:01 | LA | LA | 5 | SB | 34.65
| EB/WB | Tuxford St | | 34.65 | SB | Golden State Frwy, Rte 5 | | On Ramp | Full | Bridge
Construction | | All | 2 | Y | 03/01/2011 22:46 | N | | N |
C5TA | 304 | 03/07/2011 11:21 | 03/14/2011 20:01 | 01/13/2012 06:01 | LA | LA | 5 | SB | 36.86
| | Brandford St | | 36.86 | SB | Golden State Frwy, Rte 5 | | On Ramp | Full | Slab
Replacement | | All | 1 | Y | 04/11/2011 20:01 | N | 08/10/2011 12:33 | N |
C5TA | 312 | 03/07/2011 11:24 | 03/14/2011 20:01 | 01/13/2012 06:01 | LA | LA | 5 | NB | 37.41
| EB | Osborne St | | 37.41 | NB | Golden State Frwy | | On Ramp | Full | Slab Replacement | |
All | 1 | Y | 03/08/2011 20:01 | N | 08/10/2011 12:34 | N |
C5TA | 308 | 03/07/2011 11:22 | 03/14/2011 21:01 | 01/13/2012 06:01 | LA | LA | 5 | NB | 37.41
| NB | Golden State Frwy, Rte 5 | | 37.41 | | Osborne St | | Off Ramp | Full | Slab Replacement
| | All | 1 | Y | 03/07/2011 20:01 | N | | N |
```

A second type of text-based files in the Pasadena set also contains tabular data. However, they have multiple heading lines containing information about the data itself before the tabular data is actually presented. Examples include network definition, census block groups, hourly origin–destination, and detector influence data.

EXAMPLE 3.9

A tab-delimited file with multiple heading lines to describe the Pasadena System detector reference is shown below:

```
   $VISION
   * Mygistics
   * 12/14/11
   *
   * Table: Version block
   *
   $VERSION:VERSNR  FILETYPE  LANGUAGE  UNIT
   8.10  Att  ENG  MI
   *
   * Table: Count locations
   *
   $COUNTLOCATION:NO
PEMSID    LINKNO FROMNODENO TONODENO  CODE     NAME XCOORD    YCOORD     SMS_ID SD_ID
200011501 0      709496419  800256726 49324755 SD   393064.58 3780735.74 115    1
200011504 0      37835893   49324757  49324755 SD   393085.94 3780745.23 115    4
200011506 0      24012907   49324747  49324755 SD   393081.62 3780728.17 115    6
200028101 0      24024003   49329468  49329496 SD   395223.74 3778307.46 281    1
```

R has a support function called *read.table* which can read in any delimited files. In order to view the full syntax of this function, users can type ?read.table from the console of RStudio, and a complete documentation of this function, including examples will be shown on the bottom right window of RStudio. Most of the parameters of read.table do not need to be altered, and in most cases, the read.table call can be completed with a handful of parameters.

In Example 3.10, a csv file is imported into R by the *read.table* function with only two parameters: *file* specifies the path to the csv file and *sep* specifies the delimiter, in this case a comma. As this csv file does not contain a header line, the column names are defaulted to be V *, with * representing column indices starting from 1.

EXAMPLE 3.10

Import Pasadena link and turn volume data from a csv file into R

```
> A <- read.table(file = "PasadenaDet_20110930_Sample.csv", sep = ",")
> A
  V1           V2         V3 V4 V5 V6     V7     V8
  2.01109e+13  200011501  0  0  0  FALSE  FALSE  FALSE
  2.01109e+13  200011504  0  0  0  FALSE  FALSE  FALSE
  2.01109e+13  200011506  0  0  0  FALSE  FALSE  FALSE
  2.01109e+13  200028101  0  0  0  FALSE  FALSE  FALSE
  2.01109e+13  200028103  0  0  0  FALSE  FALSE  FALSE
  2.01109e+13  200028303  0  0  0  FALSE  FALSE  FALSE
```

For Pasadena's tab-delimited files, in order to load all information including heading lines, two stages must be completed. First, the heading lines must be read using *file* and *readLines*. Next, *read.table* can be called, with the parameter skip specified as the number of heading lines to be omitted.

EXAMPLE 3.11

Read nontabular heading lines from the network definition data file by first opening a read-online connection to the file via the *file* function, then using *readLines* to read the first 12 nontabular heading lines from this connection.

```
> conn <- file("HighwayNetwork_Sample.att", "r")
> B <- readLines(conn, n = 12)
> close(conn)
> B
 [1] "$VISION"                                "* Mygistics"
 [3] "* 10/28/11"                             "*"
 [5] "* Table: Version block"                 "* "
 [7] "$VERSION:VERSNR\tFILETYPE\tLANGUAGE\tUNIT"  "8.10\tAtt\tENG\tMI"
 [9] ""                                       "* "
[11] "* Table: Links"                         "* "
```

EXAMPLE 3.12

Read the remaining tabular data from the network definition data file using *read.table*. In this example, the column headers are specified in the file (line 12) and the delimiter is a tab.

```
> C <- read.table(file = "HighwayNetwork_Sample.att", skip = 12, header =
TRUE, sep="\t")
> C
X.LINK.NO FROMNODENO TONODENO FROMNODE.XCOORD FROMNODE.YCOORD TONODE.XCOORD TONODE.YCOORD
3844829   49459044   49459045 407634.1        3776413         407634.6      3776461
23844829  49459045   49459044 407634.6        3776461         407634.1      3776413
23844838  49315506   49315507 397973.0        3776228         397967.7      3776254
23844838  49315507   49315506 397967.7        3776254         397973.0      3776228
23844844  49455491   49455527 401657.6        3776045         401900.5      3776082
```

The result of the read.table call is a data frame, whose columns represent the columns in the delimited file.

3.5.2 XML

XML files are written using Extensible Markup Language (XML), an open standard that allows documents to be encoded in such a way that it is both human-readable and machine-readable.

For this purpose, data in XML files are encapsulated in HTML-like tags that provide the necessary annotation. XML standard is formally defined by the World Wide Web Consortium (W3C) [3].

EXAMPLE 3.13

Example XML data file from the Pasadena data environment describing traffic incidents

```
<?xml version="1.0" encoding="UTF-8"?>
<informationResponse    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="RIITS_IAI_Schema.xsd">
  <messageHeader>
    <sender>
      <agencyName>CHP-LA</agencyName>
    </sender>
    <messageID>1317387669573</messageID>
    <responseTo>87654321</responseTo>
    <timeStamp>
      <date>20110930</date>
      <time>06010900</time>
    </timeStamp>
  </messageHeader>
  <responseGroups>
    <responseGroup>
      <head>
        <updateTime>
          <date>20110930</date>
          <time>05582400</time>
        </updateTime>
      </head>
      <events>
        <event> ... </event>
        <event> ... </event>
        ...
      </events>
    </responseGroup>
  </responseGroups>
</informationResponse>
```

Unlike *readLines* and *read.table*, the ability to read XML is not part of R’s core functionalities. However, there are many community packages that support XML ingestion. A well-known package among them is XML [4]. Setting up the XML package from R is straightforward and is done by calling *install.packages*. In order to enable the functionalities of the XML package, function *library* is used.

EXAMPLE 3.14

Installing and loading the XML package.

```
> install.packages("XML")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.1/XML_3.98-1.4.zip';
Content type ‘application/zip’ length 4293638 bytes (4.1 Mb)
opened URL
downloaded 4.1 Mb
package ‘XML’ successfully unpacked and MD5 sums checked
The downloaded binary packages are in ...
> library(XML)
>
```

For any well-formed XML file, data can be imported into R using XML’s function *xmlTreeParse* with parameter *userInternalNodes* set to TRUE (Example 3.15). Without this parameter, the XML file can still be imported, but the internal tag names beyond the first two levels of the XML tree will be lost.

EXAMPLE 3.15

```
Importing XML-based incident data into R and viewing all possible data elements.
X <- xmlTreeParse("Data/Pasadena/Pasadena/09 Incident Data/1event_from_chpla_
20110930060000.xml", useInternalNodes = TRUE)
> summary(X)
$nameCounts
  count  type  vehicleType  name  text
  54     45    36           27    27
  content  injury  injuryLevel  date  time
  18       18      18           11    11
  head  adminArea1  affectedLane  affectedLanes  ambulance
  10    9           9             9              9
  caltransMaintenance  caltransTMT  city  commentExternal  commentInternal
  9                    9            9     9                9
  coroner  countyFire  countySheriff  countySheriffTSB  crossStreets
  9        9           9              9                 9
  crossStreetsLink  description  direction  event  eventResponders
  9                 9            9          9      9
  eventStatus  fireDepartment  freewayServicePatrol  fromStreetInfo  hazmat
  9            9               9                     9               9
  highwayPatrol  id  injuries  issuingAgency  issuingUser
  9              9   9         9              9
  laneCnt  latitude  localEventInformation  location  longitude
  9        9         9                      9         9
  mait  onStreetInfo  other  otherText  postmile
  9     9             9      9          9
  severity  startGeoLocation  startTime  toStreetInfo  typeEvent
  9         9                 9          9             9
  types  agencyName  events  informationResponse  messageHeader
  9      1           1       1                    1
  messageID  responseGroup  responseGroups  responseTo  sender
  1          1              1               1           1
  timeStamp  updateTime
  1          1
$numNodes
[1] 691
```

Data from individual nodes of the XML object in R can be accessed via the function *xPathApply*. This function allows the values inside the specified tag to be extracted. In Example 3.16, to find out where the incidents happened, the tag *onStreetInfo* is specified, and all values of this tag are returned as a list.

EXAMPLE 3.16

Extracting specific data values from incident data based on tag names.

```
> xpathApply(X, "//onStreetInfo", xmlValue)
[[1]]
[1] "LOS ANGELES COUNTY"
[[2]]
[1] "I710"
[[3]]
[1] "SR2"
[[4]]
[1] "I605"
[[5]]
[1] "VAN NUYS BLVD"
[[6]]
[1] "GARFIELD AV ONR"
[[7]]
[1] "SR60"
[[8]]
[1] "LA & VENTURA COUNTY"
[[9]]
[1] "ELIZABETH LAKE RD"
```

The relationship between XML tags (nodes) is often not one-to-one. In order to traverse the XML structures and identify the hierarchy among the tags, the XML object first needs to be converted to a list object using *xmlToList*. Subsequent *summary* calls to various list elements at various levels will allow users to traverse the structure of this new object.

EXAMPLE 3.17

Traversing the XML data hierarchy after converting the XML object into a list.

```
Y <- xmlToList(X)
> summary(Y)
               Length  Class          Mode
messageHeader  4       -none-         list
responseGroups 1       -none-         list
.attrs         1       XMLAttributes  character
> summary(Y[[2]])
               Length  Class  Mode
responseGroup  2       -none- list
> summary(Y[[2]][[1]])
        Length  Class  Mode
head    1       -none- list
events  9       -none- list
> summary(Y[[2]][[1]][[2]])
      Length  Class   Mode
event  10     -none-  list
event  10     -none-  list
event  10     -none-  list
event  10     -none-  list
event  10     -none-  list
event  10     -none-  list
event  10     -none-  list
event  10     -none-  list
event  10     -none-  list
> summary(Y[[2]][[1]][[2]][[1]])
                      Length   Class   Mode
head                  2        -none-  list
location              1        -none-  list
typeEvent             1        -none-  character
severity              1        -none-  character
description           1        -none-  list
affectedLanes         1        -none-  list
types                 4        -none-  list
injuries              2        -none-  list
startTime             2        -none-  list
localEventInformation 5        -none-  list
```

3.5.3 SQL

SQL (Structured Query Language) is a special-purpose programming language. It was designed to interact with data stored in a relational database management system (RDBMS). Typically, there exist packages within R to enable the usage of SQL to access data within these databases directly. In the case of RDE, these databases are not opened to the public, but the data and the related structures are released through a mechanism called SQL dumps. SQL dumps are files with .sql extension, which contain not only all the data within a specific database but also all relevant SQL commands that allow the recreation of that database inside another RDBMS.

In order to read these files into R without a supporting database infrastructure, R needs to extract the data portion of the SQL file and convert them into data frames. The first step is to read the SQL file as text and identify all SQL commands:

```
sqlDumpConn <- file("Pasadena/08a WorkZone (SQL format)/pasadena_globals_wz_schedule_
Sample.sql", "r")
sqlLines <- readLines(sqlDumpConn)
sqlValid <- ""
for (i in 1:length(sqlLines)){
  tmpLine <- gsub("^\\s+|\\s+$", "", sqlLines[i])
  if (substr(tmpLine, 1, 2) != "--"){
    sqlValid <- paste(sqlValid, tmpLine, sep="")
    if (substr(sqlValid, nchar(sqlValid), nchar(sqlValid)) == ";" ){
      if (substr(sqlValid, 1, 2) != "/*" &
        substr(sqlValid, nchar(sqlValid)-2, nchar(sqlValid)-1) != "*/" ){
      print(sqlValid)
      }
    sqlValid <- ""
    }
  }
}
close(sqlDumpConn)
```

Next, we can identify the SQL command that is responsible for the insertion of data, extract the data themselves, and convert them into data frames in R. The format of this command typically uses the following syntax:

"INSERT INTO TABLE_NAME VALUES COMMA SEPARATED DATA TUPLES WITH EACH TUPLE INSIDE PARENTHESES;

To extract the data, this command must first be stripped of the initial INSERT … syntax as well as the final semicolon.

```
> sqlLists[5]
[1] "INSERT INTO `wz_schedule` VALUES (1,'C5AA',157,'2011-01-25 18:38:00','2011-02-06
19:01:00','2012-02-29  06:01:00','LA','LA',5,1,34.7,'EB/WB',34.7,'SB',1,1,1,NULL,'All',2,
NULL,'2011-03-01  22:46:00',NULL,NULL,NULL,NULL,0),(2,'C5TA',292,'2011-03-07  11:08:00',
‘2011-03-14  20:01:00’,‘2011-09-16  06:01:00’,‘LA’,‘LA’,5,2,35.9,NULL,35.9,‘NB’,1,1,2,
NULL,‘All’,1,NULL,‘2011-03-07 20:01:00’,NULL,NULL,NULL,NULL,0), ... ;
> parsedData <- substr(parsedData, 1, nchar(parsedData) - 1)
> parsedData
[1] “1,‘C5AA’,157,‘2011-01-25  18:38:00’,‘2011-02-06  19:01:00’,‘2012-02-29  06:01:00’,
‘LA’,‘LA’,5,1,34.7,‘EB/WB’,34.7,‘SB’,1,1,1,NULL,‘All’,2,NULL,‘2011-03-01  22:46:00’,NULL,
NULL,NULL,NULL,0),(2,‘C5TA’,292,‘2011-03-07  11:08:00’,‘2011-03-14  20:01:00’,‘2011-09-16
06:01:00’,‘LA’,‘LA’,5,2,35.9,NULL,35.9,‘NB’,1,1,2,NULL,‘All’,1,NULL,‘2011-03-07 20:01:00’,
NULL,NULL,NULL,NULL,0),(3,‘C5TA’,296,‘2011-03-07  11:15:00’,‘2011-03-14  20:01:00’,‘2011-
09-16 06:01:00’,‘LA’,‘LA’,5,2,35.9,‘NB’,35.9,‘NB’,2,1,2,NULL,‘All’,1,NULL,‘2011-03-07 20:
01:00’,NULL,NULL,NULL,NULL,0),...
```

R has a function called *strsplit* that would separate this data into individual tuples, with each tuple being an object of a list. The element that serves as the split token will be the string **),(** which represents the separation of each tuple within the initial data string.

```
datalist <- strsplit(parsedData, "),(", fixed=TRUE)
datalist[[1]][1]
```

```
[1] "1,‘C5AA’,157,‘2011-01-25 18:38:00’,‘2011-02-06 19:01:00’,‘2012-02-29 06:01:00’,‘LA’,
‘LA’,5,1,34.7,‘EB/WB’,34.7,‘SB’,1,1,1,NULL,‘All’,2,NULL,‘2011-03-01  22:46:00’,NULL,NULL,
NULL,NULL,0"
```

This process needs to be repeated one more time to separate the strings representing individual tuples into separate data elements, which then can be merged into a single data frame. A similar process can be used to extract the column names from the SQL command “CREATE TABLE . . .”

3.6 INGESTING ONLINE SOCIAL MEDIA DATA

Although Social Data generated by individuals using mobile devices is typically short, it occurs at a very high rate and assumes the form of various formats (e.g., text messages, images, link to online resources). We also include message boards and forums as a form of social media. Because these data are created by individuals, they also carry a degree of uncertainty about the accuracy of the information conveyed.

Popular platforms for large-scale social media data include Twitter and Facebook, which are being used by transportation agencies to disseminate news and real-time service alerts to the public [5]. An example of using social media in transportation application is the extraction and analysis of Twitter-based text for traffic incident detection [6]. However, according to Zang, such data has a limited range [7]. The reliable detection of traffic incidents in a given location required a higher number of tweets posted in that area. Crowdsourcing is another form of social media-based solution that aggregates data collected from user through multiple sources to develop useful services such as congestion and incident alerts. As new Social Media outlets are being created to compete against Twitter or Facebook in terms of social communication, the differences in target audiences, social situations, network ranges, and data availability will motivate new research to determine whether any emerging Social Media source can be usable.

Usually, social media providers allow to access their data through specific application programming interfaces (API). This is the only automated approach to acquiring large amounts of social media, aside from buying data in bulk directly from these providers. In the remainder of this section, we will examine the techniques to acquire social media data from Twitter in R. More specifically, we will look at two different ways to acquire data: static search and dynamic streaming.

The first step prior to enabling data acquisition from an online streaming source is to establish authentication and authorization. This is possible by registering a new application with Twitter and requesting the four necessary items: Consumer Key (API Key), Consumer Secret (APPI Secret), Access Token, and Access Token Secret [8].

3.6.1 STATIC SEARCH

The required package for static search from Twitter is twitter. Additional packages to support textual analysis such as tm and wordcloud are also recommended. Setting up *twitteR* is straightforward, with *install.packages* handling the installation of the package and all of the relevant dependencies.

```
library("twitter")
setup_twitter_oauth(CONSUMER_TOKEN,
                    CONSUMER_SECRET,
                    ACCESS_TOKEN,
                    ACCESS_SECRET)
```

Using the previously acquired tokens, authentication with Twitter is established through the use of setup_twitter_oauth(). Next, the searchTwitter function can be called to mine the Twitter text data. In Example 3.18, we use this function to identify possible mentions of traffic incidents in continental United States during the past two weeks since June 24, 2016. The plyr library is a supporting package that allows easy creation and manipulation of the data frame. In this case, the ldply function automatically iterates through each element (individual tweet) or the search result r_stats and combines them into a single data frame called tweets.df.

EXAMPLE 3.18

```
r_stats <- searchTwitter("traffic accident", lang="en", geoco-
de=‘39.5,98.35,5000mi’, n=1500)
library(plyr)
tweets.df = ldply(r_stats, function(t) t$toDataFrame() )
> tweets.df$text[1:10]
[1] "It’s taken me 38 minutes to drive 2km’s. No accident just traffic. Mxm."
[2] "TRAFFIC UPDATE: Delays continue 64 West past #Parham exit due to accident
    earlier this hour. https://t.co/qpzmr6RVx0"
[3] "TRAFFIC: Accident I-64 West just past Parham Road has a delay of one mile as
    of 7:17AM."
[4] "RT @uhfemergency: Traffic accident report - Braun St &amp; Sandgate Rd Deagon"
[5] "Traffic accident report - Braun St &amp; Sandgate Rd Deagon"
[6] "RT @uhfemergency: Traffic accident report - Sherwood Rd Toowong"
```

```
 [7] "Traffic accident report - Sherwood Rd Toowong"
 [8] "RT @Power987Traffic: Accident in Centurion, queueing traffic on the N1
     North for 15 minutes, from Jean Avenue to Solomon Mahlangu Drive. #P..."
 [9] "Accident in Centurion, queueing traffic on the N1 North for 15 minutes,
     from Jean Avenue to Solomon Mahlangu Drive. #PTATraffic"
[10] "Traffic diversions in place after fatal car accident near #Gloucester via
     @GloucesterAdv https://t.co/BXWZ1Yb09w https://t.co/V8tAvAMtAW"
```

3.6.2 DYNAMIC STREAMING

For dynamic live streaming of Twitter data, R requires a different set of community packages: *ROAuth* to support the authentication process, and *streamR* for the streaming activity itself. Once again, loading these packages into R is straightforward with *install.packages()*.

Authentication and authorization need to happen prior to the streaming process. However, the authentication token itself can be saved as an R object and reused for subsequent streams.

```
library(ROAuth)
requestURL <- "https://api.twitter.com/oauth/request_token"
accessURL <- "https://api.twitter.com/oauth/access_token"
authURL <- "https://api.twitter.com/oauth/authorize"
consumerKey <- "CONSUMER_KEY"
consumerSecret <- "CONSUMER_SECRET"
my_oauth <- OAuthFactory$new(consumerKey = consumerKey,
                             consumerSecret = consumerSecret,
                             requestURL = requestURL,
                             accessURL = accessURL,
                             authURL = authURL)
my_oauth$handshake(cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl"))
save(my_oauth, file = "my_oauth.Rdata")
```

With *my_oauth*, the streaming process comes down to specifying the correct parameters to filter the Twitter stream.

```
library(streamR)
load("my_oauth.Rdata")
file = "tweets.json"
file.remove(file)
filterStream(file.name = file,
             track = "traffic accident",
             locations = NULL,
             language = "en",
             timeout = 300,
             tweets = NULL,
             oauth = my_oauth,
             verbose = TRUE)
```

The Twitter stream, filtered by keyword “traffic accident,” will be saved into a file called *tweets.json*. After the stream stops, the file can be opened and converted into a data frame using the *parseTweets()* function.

```
streamingtweets.df <- parseTweets(file)
streamingtweets.df$text
 [1] "Accident cleared on Crescent Cty Connection NB at US-90 #traffic #NOLA https://t.
     co/9is45BqHqI"
 [2] "Accident in #ParagonMills on Nolensville Pike at Edmondson Pike #Nashville #traf-
     fic https://t.co/rU1w1uAgW1"
 [3] "RT @TotalTrafficBNA: Accident in #ParagonMills on Nolensville Pike at Edmondson
     Pike #Nashville #traffic https://t.co/rU1w1uAgW1"
 [4] "Houston traffic alert! Major accident at Hwy 59 North and 610. https://t.co/
     TVw9pibTAV"
 [5] "#NOLATraffic Accident cleared Lakebound past SK Center. Crash with injury backing
     up traffic on WB 90 through Bridge City"
 [6] "RT @RayRomeroTraf: #NOLATraffic Accident cleared Lakebound past SK Center. Crash
     with injury backing up traffic on WB 90 through Bridge Ci..."
 [7] "Accident. four right lanes blocked. in #NeSide on 59 Eastex Fwy Outbound before
     Crosstimbers, stop and go traffic back to The 610 N Lp"
 [8] "Accident in #Wayne on Van Born Rd at Beech Daly Rd #traffic https://t.co/
     93KUmrUJbm"
 [9] "RT @TotalTrafficBNA: Accident in #ParagonMills on Nolensville Pike at Edmondson
     Pike #Nashville #traffic https://t.co/rU1w1uAgW1"
[10] "#M3 eastbound between J3 and J2 - Accident - Full details at https://t.co/O3z6iRx7dE"
[11] "#M3 eastbound between J3 and J2 - Accident - Full details at https://t.co/LwI3prBa31"
[12] "RT @TotalTrafficDET: Accident in #Wayne on Van Born Rd at Beech Daly Rd #traffic
     https://t.co/93KUmrUJbm"
[13] "RT @TotalTrafficDET: Accident in #Wayne on Van Born Rd at Beech Daly Rd #traffic
     https://t.co/93KUmrUJbm"
[14] "RT @TotalTrafficDET: Accident in #Wayne on Van Born Rd at Beech Daly Rd #traffic
     https://t.co/93KUmrUJbm"
[15] ".@latimes Any word on the accident at 5ish this morning on the 10 WB at
     Arlington? I saw two coroner vans at the scene. Snarled traffic."
[16] "M5 Northbound blocked, stationary traffic due to serious multi-vehicle accident
     between J25 A358 (Taunton) and J24 A38 (Bridgwater South)."
```

3.7 BIG DATA PROCESSING: HADOOP MAPREDUCE

As technologies progress, the ability to capture and transmit information of entities within a connected transportation environment becomes trivial, resulting in massive amounts of information being collected for analytical purposes. To address this issue, new computing infrastructures are needed to assist with the management of Big Data together with existing statistical frameworks such as R. Apache Hadoop is one such infrastructure. Motivated by the two seminal papers describing the computing and data infrastructure of Google [9,10], Apache Hadoop provides both an underlying data management infrastructure (Hadoop Distributed File System) and a programming paradigm (MapReduce) that enables analytics on massive amounts of data orders of magnitude larger than the storage of a standard personal computer. Within the context of this book, we will focus on how to write MapReduce programs using R to manage large amounts of data. A MapReduce program maps multiple instances of the same task onto individual elements of a massive dataset and then aggregates (reduces) the outcomes of the map tasks to form the final result. By distributing massive data sets across a large collection of computers, the MapReduce framework allows multiple map and reduce tasks to process data at the same time, thus increasing performance.

EXAMPLE 3.19

In this example, we are looking at Pasadena's Turn Volume and Link Volume data from a roadside detector. More specifically, the 30-second turn volume and link volume data provide the counts of vehicles and estimated count of passengers from individual lanes of a road segment. In the mapping portion, the R code parses each line of data as it is being fed into the code via Hadoop Streaming. The parsing process will return a Key/Value pair, with the key representing a detector's unique ID, and the value representing the count of vehicles on a lane of that detector's road segment.

```
#!/software/R/3.0.2/bin/Rscript
stdin <- file('stdin', open='r')
open(stdin)
while(length(x <- readLines(stdin, n=1))>0) {
  tuple_elements <- strsplit(x,",", fixed=TRUE)[[1]]
  for (i in 1:10){
    if (!is.na(as.numeric(tuple_elements[i*3+1]))){
    cat(paste(tuple_elements[1],tuple_elements[i*3 + 1], sep='\t'), sep='\n')
    }
  }
}
```

The Hadoop infrastructure will shuffle and bring all Key/Value pairs that have the same key together (i.e.: all vehicle counts (value) of the same detector (key) will be grouped together). The reduce code will be applied to each of the keys, resulting in the final aggregated vehicle count across all lanes over time for each individual detector.

```
#!/software/R/3.0.2/bin/Rscript
trimWhiteSpace <- function(line) gsub("(^ +)|( +$)", "", line)
splitLine <- function(line) {
  val <- unlist(strsplit(line, "\t"))
  list(detectorID = val[1], count = as.integer(val[2]))
}
env <- new.env(hash = TRUE)
```

```
con <- file("stdin", open = "r")
while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
  line <- trimWhiteSpace(line)
  split <- splitLine(line)
  detectorID <- split$detectorID
  count <- split$count
  if (exists(detectorID, envir = env, inherits = FALSE)) {
    oldcount <- get(detectorID, envir = env)
    assign(detectorID, oldcount + count, envir = env)
  }
  else assign(word, count, envir = env)
}
close(con)
for (w in ls(env, all = TRUE))
  cat(w, "\t", get(w, envir = env), "\n", sep = "")
```

To execute the above procedure, the following command needs to be invoked. The location of the jar file for Hadoop streaming might differ depending on individual set up, but everything else should be the same.

```
yarn jar ~/software/hadoop-2.2.0.2.1.0.0-92/share/hadoop/tools/lib/hadoop-streaming-
2.2.0.2.1.0.0-92.jar -input /30sec_20110901235300.txt -output /outputrde -mapper
mapper.R -reducer reducer.R -file mapper.R -file reducer.R
```

A portion of the outcome of the procedure can be viewed below, with the first column of the data representing detector IDs, and the second column representing the total count of vehicle traveled over the road segment in 30 seconds.

```
[lngo@node0767 RDE]$ hdfs dfs -cat /outputrde/part-00000
715898  15
715900  0
715902  0
715905  2
715906  2
715912  1
715913  2
715915  27
715916  16
715917  0
715918  20
715919  0
715920  9
715921  13
715922  28
715923  1
...
```

3.8 SUMMARY

The deluge of data coming from various sources demands transportation planners and analysts to be able to ingest, curate, and integrate raw data prior to the application of analytical techniques. This chapter has presented approaches in dealing with common data formats, ranging from structured to semistructured and hierarchical format using the statistical software R. This chapter will set the stage for readers to apply the techniques from subsequent chapters to real world data from the RDE.

3.9 EXERCISES

Search for instructions to set up R and RStudio and perform the installations on your own personal devices.

REFERENCES

[1] R. Weil, J. Wootton, A. Garcia-Ortiz, Traffic incident detection: Sensors and algorithms,, Math. Comput. Model. 27.9 (1998) 257–291.
[2] US DOT, Research Data Exchange. <https://www.its-rde.net>, 2016.
[3] Extensible Markup Language (XML) 1.0 (Fifth Edition), <https://www.w3.org/TR/REC-xml/>.
[4] XML, <https://cran.r-project.org/web/packages/XML/index.html>.
[5] S. Bregman, Uses of social media in public transportation. Transit Cooperative Research Program (TCRP) Synthesis 99, Transportation Research Board, Washington, 2012.
[6] K. Fu, et al., Steds: Social media based transportation event detection with text summarization, Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on, IEEE, 2015.
[7] S. Zhang, Using Twitter to Enhance Traffic Incident Awareness, Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on, IEEE, 2015.
[8] Twitter Developer Documentation, <https://dev.twitter.com/overview/documentation>, 2016.
[9] S. Ghemawat, H. Gobioff, S.-T. Leung, The Google file systemvol. 37, no. 5, pp. 29–43 ACM SIGOPS operating systems review, ACM, 2003.
[10] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.