


Lesson 4 – Data Management and Visualization

Introduction to Data Analysis using R

Grant Hutchison

Welcome to this lesson on Data Management and Visualization in the Introduction to Data Analysis using R course.

Agenda

- Importing and Exporting Data
 - Filtering / Subsets
 - Sorting
 - Visualization
- 

In this lesson we will learn how to import external data from files into R. We will also learn how to export or write the data to files if needed.

Once we have our data loaded into memory we can perform data filtering or querying to focus on key aspects of our data set.

We will also learn how to reorder our data within data frames and finally we will explore data visualization techniques.

R provides extensive data visualization options, in this lesson we will examine how to plot data for trend analysis and distribution analysis.

In future lessons we will explore many other types of data visualization techniques.

Reading external data into R

Reading Data from text files

- Multiple functions to read in data from text files

Types of Data Formats

- Delimited
- Positional

3

© 2013 BigDataUniversity.com

Let's first examine how to read data from files into R.

Typically our data is provided as a semi-structured text file. The data is considered semi-structures as each line of data represents an observation and multiple variables or measurements associated with each observation. The measurements are either separated using a delimiter or by position.

R can read data from many other formats including spreadsheets and other statistical software products, but we will focus on the most commonly found formats for our analysis.

Reading external data into R

Delimited Text Files

Function	Header header	Separator sep
<code>read.table</code>	NO	white-space
<code>read.csv</code>	TRUE	" , "
<code>read.delim</code>	TRUE	"\t"

Useful Arguments for Reading Files

- `colClasses` vector of class names / col
- `col.names` vector of column names
- `dec` character for decimal point
- `skip` number (n) lines to skip
- `comment.char` ignore comment lines '#'

4

© 2013 BigDataUniversity.com

Let's first examine some of the options for importing data from text files with observations that use delimiters.

The `read.table` function will import data assuming the first row and every subsequent row contains observations. If your data includes header information in the first line then simply include the argument `header=TRUE`. If you specify the data has a header row then the values will be used as column names for the new data frame in R.

By default the `read.table` function will assume that the second and subsequent lines in the text file are measurements delimited with white-space characters. If your data is delimited using other means then use the `sep` argument.

R will attempt to automatically determine the data types for each measurement based on the type of data encountered. It is often a good idea to examine the data file and carefully consider the data types you plan to use for your data while it resides in R for analysis. To accomplish this simply use the `colClasses` argument to specify a character vector of class names such as "integer" or "Date" for example.

If you do not have a header in your data set, or you wish to define column names that

are different from the first row of data, use the `col.names` argument to specify a character vector of column names for your data frame.

Often the data file is delimited using special characters such as a comma `,`. These comma separated values or CSV files are often created by spreadsheet applications such as Microsoft Excel. By default, the `read.csv` function will import data where the elements for each row are separated with a comma character. Since the comma character is often used as a decimal point in non-English speaking countries the `dec` or `decimal` option.

The `read.delim` function can be used for tab delimited files.

Other useful options include the ability to skip over non-observational data at the beginning of the file and also to ignore any comment lines using a special character in the first position of a line of text.

The `skip` option can be used to bypass any initial information in the file and the `comment.char` can be used to ignore any data where the first character represents information that is not part of the data set under analysis. The `colClasses` option can be used to set the class names for the data being read. For example, if the first elements are date values the `colClasses` vector can specify the Date Class as the first datatype.

Reading external data into R

Fixed width text files

```
> read.fwf( file, width, ... )
```

- **file** - name of file to be imported
- **width** - vector of widths for each element

5

© 2013 BigDataUniversity.com

If the data you have been provided has its observations defined in fixed positions on each line the `read.fwf` (Read Fixed Width Format) function can be used.

You can specify the width of each measurement by providing a numeric vector containing the width of each value.

The column names and/or data types are defined in the same manner as previously discussed.

Now that we have the data in memory let's start exploring.

Exploring data – importing

Names example

– Contents of the CSV file

Year	Name	Frequency
1999	AADAM	5
2005	AADAM	6
2007	AADEN	10

File attributes

- approximately 66 thousand records in the file
- skip first line
- use second line as the header (column names)

```
> n <- read.csv("males.csv", skip=1, header=TRUE)
```

source: <http://www.ontario.ca/government/ontario-top-baby-names-male>

6

© 2013 BigDataUniversity.com

Let's explore a public data using R.

In this example we will work with birth registrations for males born in the province of Ontario in Canada from 1917 until 2010.

The data set is provided under an Open Data initiative by the Ontario government.

A snapshot of the Comma Separated Values file is shown here.

Upon examination of the file format we understand that we should skip the first row and use the second row as a header and therefore use the descriptions as column names for our data frame.

We will use the `read.csv` function and load our data into a data frame is called `n`.

Exploring data – post import

Data Verification - post import

>str(n) # structure of n

```
> str(n)
'data.frame': 66351 obs. of  3 variables:
 $ Year      : int  1999 2005 2007 2008 2009 2010
 $ Name      : Factor w/ 3736 levels "AADAM","AAD
 3 3 ...
 $ Frequency: int   5  6 10 34 38 12 10 6 10 5 ...
```

>dim(n) # rows and columns

```
> dim(n)
[1] 66351    3
```

as.is=TRUE
stringsAsFactors=FALSE

After a successful read operation we often examine the structure of the new data frame.

We have confirmed that n is a data.frame with 66,351 observations of 3 different variables.

The measurement or variable names are: Year, Name, and Frequency

R has automatically determined that the Year attribute and the Frequency attribute are both integer data types.

Notice how R has defined the Names to be Factors and not character vectors. This default behaviour can be changed using the option as.is=TRUE or stringsAsFactors=FALSE.

In this scenario we want to consider each name as a Factor and therefore we understand that there are 3,736 levels or unique names in our data.

R users will often examine a few rows of data using the head() function or in this case we validate the number of rows and columns using the dimension of dim() function.

Data Exploration - subsets

Examine most recent data - bracket [] notation

- Find the most recent year

```
>max(n$Year)
```

```
2010
```

Goal : Find the 5 most popular names in the most recent year.

- Create a new data frame with only 2010 data

```
>n.2010<-n[n$Year==2010,]
```

```
>nrow(n.2010)
```

```
[1] 1503
```

Filter by row
(all columns)

Now it is time to explore.

Our goal is to find out the most popular names in the most recent year.

Therefore we first need to determine the most recent year using the max function. We use the max function and the dollar sign (\$) to indicate that we would like to request the targets value for the Year measurement. We discover that the most recent year of data is from "2010".

Now, we will create a new data frame that contains only the male babies born and registered in 2010. This new data frame is called n.2010.

Notice how we use the bracket notation to specify the condition of our filter for the rows to be returned. If you are familiar with SQL this would be equivalent to the WHERE clause or selection operation. Since the indexing method for data frames involves rows and columns, separated by a comma (,), we must follow our conditional expression of n\$Year==2010 with the comma operator. There is no condition for the columns and therefore all of the defined column information will be returned.

We use the nrow() function to check the number of names registered in 2010. Now we know that there were 1,503 male names registered. As an aside, according to our

data source, if a name was registered less than five (5) times it was not included in the data set.

Our goal was to find the most popular baby names in the most recent year so now we will need to sort the data to get our answer.

Data Exploration - subsets


Examine most recent data – using `subset()`

- Create a new data frame with only 2010 data

```
>n.2010<-subset(n,n$Year==2010)
```

```
>nrow(n.2010)
```

```
[1] 1503
```



Filter by row
(all columns)

9

© 2013 BigDataUniversity.com

Before we sort the data and determine the 5 most popular name another option to filter data involves the use of the `subset()` function.

Here we pass the original data frame to the function and then we also provide a conditional expression. By default all of the column data will be returned, but this can be specified.

We validate the we achieve the same set of 1,503 observations.

Now let's sort the data.

Data Exploration – sorting vectors

Sorting data in vectors

```
> ages <- c(78, 45, 23, 12)
> sort(ages)
[1] 12 23 45 78
> sort(ages, decreasing=TRUE)
[1] 78 45 23 12
```

10

© 2013 BigDataUniversity.com

Sorting data in a vector is quite easy in R. You simply pass the vector to the sort function and new sorted vector is generated.

The default sort order is ascending, but this can be changed using the argument `decreasing=TRUE`.

However, in our scenario the data is not in a vector, it is in a data frame. Let's examine how we sort data stored contained in a data frame.

Data Exploration – sorting data frames

Sorting data in data frames

- Remove the Year column from the (*Projection*)

```
>n.2010<-n.2010[,c("Name", "Frequency")]  
>head(n.2010,2)
```

	Name	Frequency
6	AADEN	12
17	AAHIL	8

Filter by Columns
(all rows)

- Sort the data using `order()`

```
> n.2010.s <-n.2010[order(n.2010$Frequency, decreasing=TRUE),]  
> head(as.character(n.2010.s$Name),5)  
[1] "ETHAN" "LIAM" "JACOB" "LUCAS" "NOAH"
```

11

© 2013 BigDataUniversity.com

The data frame `n.2010` contains three columns: Year, Name and Frequency.

There is no need to keep the Year attribute so we decide to filter our data frame to remove the Year column.

Here is an example using the bracket notation of how we can filter out the Year column by specifying a character vector of column names within the brackets.

If you are familiar with SQL this task would be equivalent to using projection to specify the columns in the SELECT list for an SQL statement.

We use the `head()` function verify that the Year column has been removed. The function shows that the data is still in its original sort order by name.

To find the most popular names we must sort by Frequency in decreasing or descending order.

We can't use the function as we did with vectors, but with data frames we will use the `order` function within our bracket notation.

Here we want the data sorted by decreasing frequency values and all of the columns should be included in our output.

The final expression converts the Factor vector into a simple character vector to determine that the most popular name was: Ethan, followed by Lian, Jacob, Lucas,

and Noah.

Saving data

Saving Data frames

– Comma Separated Values (CSV)

```
> write.csv(n.2010.s, file="2010_male_popular.csv",  
            row.names=FALSE)
```

– Delimited

```
> write.table(n.2010.s,  
              file="2010_male_popular.del", sep="\t",  
              row.names=FALSE)
```

12

© 2013 BigDataUniversity.com

Whenever we have explored a data set and wish to save it for future reference we can use one of the write() family of functions.

Here we see some examples of exporting the sorted male names from the year 2010 to a comma separated values or csv file and also to a tab delimited file.

In many data frames the row names are not really used to identify data so we have used the row.names=FALSE option to avoid writing the row name information in our output file.

Data Visualization

Goal : Show trends from 1917 to 2010 for Names

1. Ask the user for a name to explore
2. Build a table of frequencies of registrations by year
3. Create a visualization (chart)

13

© 2013 BigDataUniversity.com

So far we have worked with data within data frames. Now let's visualize trends in our data using Data Visualizations or graphics.

In this task we would like to provide an interactive experience so that the user can provide a baby name and then view a single visualization of its popularity from 1979 to 2010.

Let's get started.

Data Visualization - Trends

```
> #Obtain the name from the user
> name.in <- readline(prompt="Male baby name? :")
> #create data frame with frequencies and years
> freq.year<-n[n$Name == toupper(name.in),
  c("Year", "Frequency")]

> #Ensure the data is sorted by year
> freq.year<-freq.year[order(freq.year$Year),]

> #Perform a simple scatterplot
> plot(freq.year)
```

14

© 2013 BigDataUniversity.com

So far we have imported data from external files, but we have not obtained input from users before.

We can use the `readline()` function to read data from text files, websites, or in this example we will use the function to obtain data from the standard input device or keyboard. We specify the prompt with the optional argument.

To visualize trends in baby names we would decide to use a scatter plot visualization based on frequencies over the years 1979 to 2010.

To do this we create a new data frame which is a subset of frequencies for the name provided by the user. The built-in `toupper()` function is used to convert the input character string to all uppercase so we can match the data stored in the data.frame. We also only require the two variables of Year and Frequency. If you are familiar with SQL you can imagine how this query would look using an SQL SELECT statement, but in R we do things a bit different.

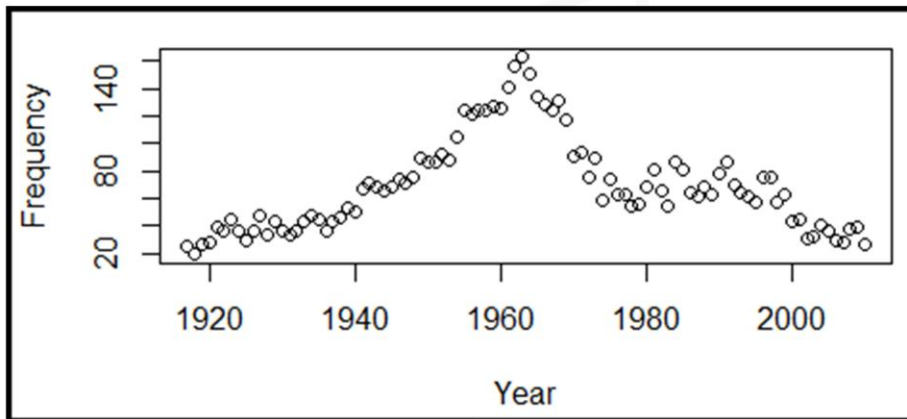
We then sort the data within the data frame and call the `plot()` function.

The plot function uses the values provided to generate a visualization.

Plot is an example of a generic function in R as it can accept many different data formats and it will attempt to create a visualization.

Let's take a look at the visualization.

Scatter Plot - default



15

© 2013 BigDataUniversity.com

When the data is provided to the plot function as a data frame of 2 variables, plot will create a scatterplot using circles for each observation.

Notice how default labels and scales were generated.

The x and y axis labels are determined based on the defined column names.

We will examine a few additional commonly used options for the `plot()` function next.

Plot function is extremely versatile

Options	Description
main	chart title (character string)
xlab	x-axis label (character string)
ylab	y-axis label (character string)
type	Type of plot 'p' - points (default) 'l' - lines 's' - stairs

Most visualizations should have a title. The main argument is used to provide a title for a visualization.

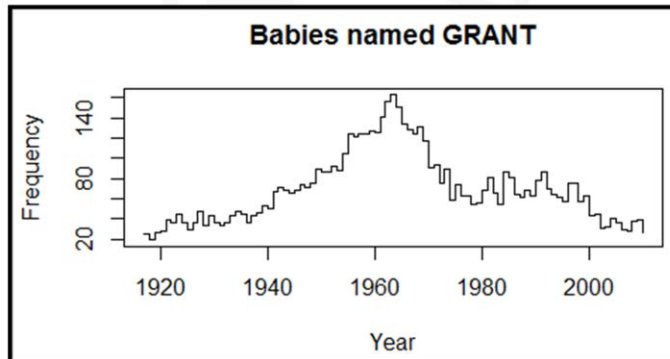
The label names can be specified for each axis and the method of plotting can be changed from the default to lines, stairs, or other representations.

Let's look at another example with our baby name data frame.

Data Visualization – plot() function

Plot - Example

```
> name.in <- readline(prompt="Male baby name? :")  
> plot.title <- paste("Babies named", toupper(name.in))  
> g <- plot(freq.year, main=plot.title, type='s')
```



17

© 2013 BigDataUniversity.com

This plot uses the same data, but it uses the paste function to construct a more precise title for our chart and the type of plot is changes from using circles to using stairs.

R has been able to help us spot trends in our data set across 2 variables. It is fairly obvious from this data that the first name or given name of "GRANT" was most popular in the early 1960s and from our earlier analysis we know that it was definitely not one of the top 5 names of newborns in 2010.

How hot was July in Toronto?

More advanced scatter plots

```
# Load data from weather file, skip the first 9 lines of metadata
>july.detail <-read.csv("july_toronto.csv", skip=9)

#examine the structure of the data loaded into the data frame
>str(july.detail)

'data.frame':  31 obs. of  5 variables:
 $ day      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ max.temp: num  23.8 22.9 26.5 28.3 25 29 27.8 29.1 25.4 30.5 ...
 $ min.temp: num  16.3 17 17.8 20.5 21.4 19.1 20.6 19.6 19.9 19.5 ...
 $ rain     : num  0 0 1.2 3.2 28.6 0.4 26.6 126 0.6 0.6 ...
 $ wind     : int  39 33 NA 33 NA NA 41 59 NA NA ...
```

18

© 2013 BigDataUniversity.com

Let's look at one more example of trend analysis and also learn a few more plotting options along the way.

In this dataset we plan to examine the maximum and minimum temperatures during the month of July 2013 in Toronto.

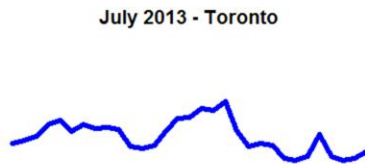
After the data is loaded we check the structure of the data frame and decide to move forward.

Note that there are some missing values of wind speed in our data, but we will only examine temperatures so we can ignore this variable at this time.

Plotting multiple data sets (daily minimum temperatures)

```
#set an appropriate range for y axis - lowest and highest temperatures
>yrange <-c (floor(min(july.detail$min.temp)),
ceiling(max(july.detail$max.temp)))

#plot the daily minimum temperatures
>plot(july.detail$day, july.detail$min.temp, type="l",lwd=6, ylim=yrange,
xlab="", ylab="", col="blue", main="July 2013 - Toronto", axes=FALSE)
```



19

© 2013 BigDataUniversity.com

We would like our graph to include both maximum and minimum values for each day of the month.

Therefore, we need to determine the range of values to set our axis properly.

Here we have created a variable called `yrange` that contains the lowest temperature and the highest temperature for the month.

As we plot the data we will specify a line graph type with a line width of 6 pixels and a colour of blue. The y-axis limit is specified using our `yrange` vector and we have also included a title in our initial plot.

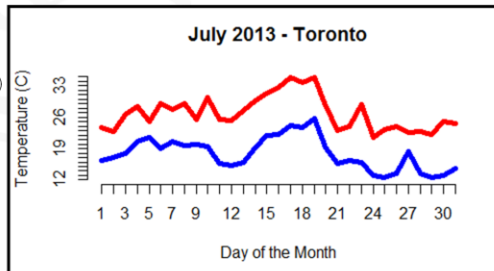
At this point we are not finished as we now wish to add the maximum temperatures and our axis labels to the same plot.

Plotting multiple data sets – maximums and labels/axis

```
#prevent a new plot from replacing the current plot
>par(new=TRUE)

#plot the maximum daily temperatures
>plot(july.detail$day, axes=FALSE, july.detail$max.temp, type="l",
      lwd=5, ylim=yrange, col="red", xlab="Day of the Month",
      ylab="Temperature (C)")

#add x and y axis
>axis(1,c(1:max(july.detail$day)))
>axis(2,yrange[1]:yrange[2])
```



20

© 2013 BigDataUniversity.com

The `par()` function can be used to control various features for our graph. Here we are telling R that our next `plot()` function call should reuse the existing plot.

Now we add the red line of maximum temperatures for the month. We also decide to add the x and y axis labels at this time.

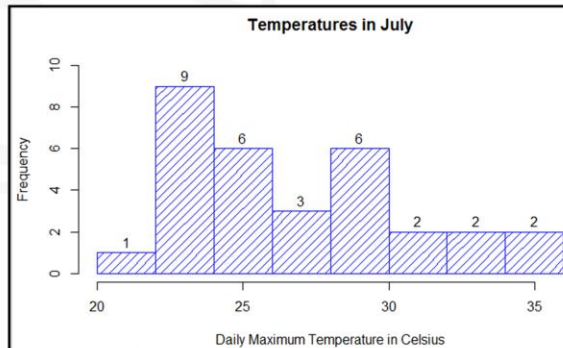
The final step is to include the axis range of values to our chart.

R has helped us explore how July was very warm in the middle of July followed by an extended cool period lasting until the end of the month.

Distribution Visualization - Histogram

Distribution of Temperatures

```
> hist(july.detail$max.temp,  
      main="Temperatures in July",  
      xlab="Daily Maximum Temperature in Celsius",  
      breaks=seq(20,36,2),  
      density=10,  
      col="blue",  
      labels=TRUE,  
      ylim = c(0,10))
```



21

© 2013 BigDataUniversity.com

We can use summary statistics to determine the mean or average temperature in the month of July or examine trends like we did in the previous example.

Now we would like to examine the distribution of temperatures during the month. A distribution of values are often visualized using a histogram where the data is grouped into categories and the categories are visualized.

Here is an example that shows the most frequent maximum temperature in July was between 22 and 24 degrees Celsius.

The breaks argument can be used to determine the size of the categories.

So now if someone asks what was the July 2013 like in Toronto you will have lots to discuss based on learning R.

Graphics

**You name it and
R can graph it !!**

- scatterplots
- barplot
- piecharts
- histograms
- densities
- pairs
- time-series
- mosaic
- associations
- 3D (contour, images)
- box plots
- many more...

22

© 2013 BigDataUniversity.com

R has many more data visualization capabilities.

One of the most popular R extensions for graphics is called ggplot2 by Hadley Wickham and if your goal is to use R to create data visualizations then exploring the ggplot2 package is a good idea.



Thank You

© 2013 BigDataUniversity.com

Thank you for completing lesson 4 on data management and visualization.

Have fun exploring trends and distributions using R.