Application Development Laboratory (CS 33002)

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School of Computer Engineering

2 Credit

Analytics Application Development using R

Lab Contents

Sr #	Major and Detailed Coverage Area	Lab#
1	Using Built-in Datasets in R	4
2	Package	
3	Data Visualization	
3.1	Basic Visualization	
3.2	Advance Visualization	
3.3	Specialized Visualization	

Using Built-in Datasets in R

R has a number of base datasets that come with the install, and there are many packages that also include additional datasets. These are very useful and easy to work with.

?datasets # to get help info on the datasets package

library(help="datasets") # provides detailed information on the datasets package, including listing and description of the datasets in the package

data() # lists all the datasets currently available by package

data(package = .packages(all.available = TRUE)) # lists all the available datasets by package even if not installed

data(EuStockMarkets) # loads the dataset EuStockMarkets into the workspace

summary(EuStockMarkets) # provides a summary of the dataset EuStockMarkets

View(EuStockMarkets) # shows the data of the dataset "EuStockMarkets" in spreadsheet format

str(EuStockMarkets) # shows the structure of the dataset "EuStockMarkets"

head(EuStockMarkets, 6) # Print the first 6 rows

nrow(EuStockMarkets) # Number of rows (observations)

ncol(EuStockMarkets) # Number of columns (variables)

Most used R built-in data sets

- mtcars: Motor Trend Car Road Tests

  The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models)

- iris

  iris data set gives the measurements in centimeters of the variables sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

- ToothGrowth

  ToothGrowth data set contains the result from an experiment studying the effect of vitamin C on tooth growth in 60 Guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, (orange juice or ascorbic acid)

- PlantGrowth

  Results obtained from an experiment to compare yields (as measured by dried weight of plants) obtained under a control and two different treatment condition.

- USArrests

  This data set contains statistics about violent crime rates by us state.

Package

An R package is an extension of R containing data sets and specific functions to solve specific questions. R comes with standard (or base) packages, which contain the basic functions and data sets as well as standard statistical and graphical functions that allow R to work. There are also thousands other R packages available for download and installation from **CRAN**, Bioconductor and **GitHub** repositories. After installation, one must first load the package for using the functions in the package.

Installing R packages

Packages can be installed either from CRAN (for general packages), from Bioconductor (for biology-related packages) or from Github (developing versions of packages).

Install a package from CRAN: The function install.packages() is used to install a package from CRAN. The syntax is install.packages("package_name"). For example, to install the package named readr, type: install.packages("readr"). It's also possible to install multiple packages at the same time, as : install.packages(c("readr", "ggplot2"))

Install a package from Bioconductor: Bioconductor contains packages for analyzing biological related data. In the following R code, we want to install the R/Bioconductor package limma, which is dedicated to analyse genomic data. To install a package from Bioconductor, use this:

source("https://bioconductor.org/biocLite.R")

biocLite("limma")

Package cont'd

Install a package from Github: GitHub is a repository useful for all software development and data analysis, including R packages. To install a package from GitHub, the R package devtools (by Hadley Wickham) can be used. You should first install devtools if you don't have it installed on your computer. For example, the following R code installs the latest version of survminer R package developed by A. Kassambara (https://github.com/kassambara/survminer). Example:

```
install.packages("devtools")
devtools::install_github("kassambara/survminer")
```

View the list of installed packages

To view the list of the already installed packages on your computer, type : installed.packages()

Folder containing installed packages

R packages are installed in a directory called library. The R function **.libPaths()** can be used to get the path to the library.

Package cont'd

Load and use an R package

To use a specific function available in an R package, you have to load the R package using the function library(). In the following R code, we want to import a file ("http://www.sthda.com/upload/decathlon.txt") into R using the R package readr, which has been installed. The function read_tsv() [in readr] can be used to import a tab separated .txt file:

```
library("readr") # Import my data
my_data <- read_tsv("http://www.sthda.com/upload/decathlon.txt")
# View the first 6 rows and the first 6 columns and syntax: my_data[row, column]
my_data[1:6, 1:6]
```

View loaded R packages

To view the list of loaded (or attached) packages during an R session, search() is used.

Remove installed packages

To remove an installed R package, use the function remove.packages() as follow:

```
remove.packages("package_name")
```

Data Visualization

Data visualization is the presentation of data with graphics. It's a way to summarize the findings and display it in a form that facilitates interpretation and can help in identifying patterns or trends. R is an amazing platform for data analysis, capable of creating almost any type of graph.

When it comes time to develop a model, it may be difficult to understand where to begin simply by looking at the data in its raw form. But by properly visualizing the data, the patterns may start to become clear, allowing to more effectively decide on the best model to use.

R provides built-in and external libraries that can be used to make impressive data visualizations. In addition, it provides several tools for building models off of the insights gained during data visualization.

R offers three main graphics packages: **traditional (or base)**, **lattice** and **ggplot2**. Traditional graphics are built into R, create nice looking graphs, and are very flexible. However, they require a lot of work when repeating a graph for different groups in the data. Lattice graphics excel at repeating graphs for various groups. The ggplot2 package also deals with groups well and is quite a bit more flexible than lattice graphics.

Ways to visualize data

Basic Visualization

1. Line Charts
2. Bar Plot
3. Histogram
4. Pie Chart
5. Dot Plots
6. Box Plot
7. Scatter Plot
8. Kernel Density Plot

Advance Visualization

1. 3D Graphs
2. Heatmap
3. Correlogram
4. Mosaic Map
5. Map Visualization

Specialized Visualization

1. Word Clouds
2. Radar Charts
3. Waffle Charts

Line Charts

A line chart is a series of data points called 'markers' connected by straight line segments. It is the most often used to visualize data that changes over time. A standard example would be how the stock value for a certain company develops over time on the stock market. However, it does not necessarily need to be time along the X-axis. Line charts emphasize time flow and rate of change rather than the amount of change.

Syntax:

plot(v, type, main, col, xlab, ylab) and the description of the parameters:

- v is a vector containing the numeric values.
- type takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- xlab is the label for x axis.
- ylab is the label for y axis.
- main is the Title of the chart.
- col is used to give colors to both the points and lines.

Line Charts Example


```
cars <- c(1, 3, 6, 4, 9) # Define the cars vector with 5 values
plot(cars) # Graph the cars vector with all defaults
plot(cars, type = "o")
plot(cars, type="o", col="blue")
plot(cars,type = "o", col = "red", xlab = "Month", ylab = "Unit Produced")
trucks <- c(1, 5, 7, 5, 5)
lines(trucks, type = "o", col = "blue")
```

Lab work

Draw the line chart for the following

cars	trucks	suvs
1	2	4
3	5	4
6	4	6
4	5	6
9	12	16

Bar Plot

Bar charts are representations of grouped data, and are extremely versatile in their application. For example, they're good at plotting the counts of groups, like the number of employees in each division of a company. They're also good at plotting the averages of groups, like the average height for both men and women. R uses the function **barplot()** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax:

barplot(H, xlab, ylab, main, names.arg, col). Following is the description of the parameters used –

- H is a vector or matrix containing numeric values used in bar chart.
- xlab is the label for x axis.
- ylab is the label for y axis.
- main is the title of the bar chart.
- names.arg is a vector of names appearing under each bar.
- col is used to give colors to the bars in the graph.

Bar Plot Example


```
H <- c(7,12,28,3,41) # Create the data for the chart
barplot(H) # Plot the bar chart
M <- c("Mar","Apr","May","Jun","Jul") # Plot for categorical data
# Plot the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",
    main="Revenue Chart", horiz = FALSE ,border="red")
```

Group Bar Plot and Stacked Bar Plot

```
# Create the input vectors.
colors = c("green","orange","brown")
months <- c("Mar","Apr","May","Jun","Jul")
regions <- c("East","West","North")
# Create the matrix of the values.
values <- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11), nrow = 3, ncol = 5, byrow = TRUE)
# Create the bar chart
barplot(values, main = “Total Revenue", names.arg = months, xlab = "month", ylab =
"revenue", col = colors)
```

Histogram

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range. R creates histogram using hist() function. This function takes a vector as an input and uses some more parameters to plot histograms.

Syntax:

hist(v,main,xlab,xlim,ylim,breaks,col,border) wherein

- v is a vector containing numeric values used in histogram.
- main indicates title of the chart.
- col is used to set color of the bars.
- border is used to set border color of each bar.
- xlab is used to give description of x-axis.
- xlim is used to specify the range of values on the x-axis.
- ylim is used to specify the range of values on the y-axis.
- breaks is used to mention the width of each bar.

Histogram Example

Example 1

```
# Create data for the graph.
v <-  c(9,13,21,8,36,22,12,41,31,33,19)

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")

 # Create the histogram with Range of X and Y values
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),
    breaks = 5)
```

Example 2

```
temperature <- airquality$Temp
hist(temperature)
```

Pie Chart

A type of graph in which a circle is divided into sectors that each represent a proportion of the whole. It is created using the pie() function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax:

pie(x, labels, radius, main, col, clockwise) wherein

- x is a vector containing the numeric values used in the pie chart.
- labels is used to give description to the slices.
- radius indicates the radius of the circle of the pie chart.(value between –1 and +1).
- main indicates the title of the chart.
- col indicates the color palette.
- clockwise is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Pie Chart Example


```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Plot the chart.
pie(x,labels)

# Plot the chart with title and rainbow color pallet.
pie(x, labels, main = "City pie chart", col = rainbow(length(x)))

# Plot the chart with Slice Percentages and Chart Legend
piepercent<- round(100*x/sum(x), 1)
pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,
  fill = rainbow(length(x)))
```

3D Pie Chart Example

A pie chart with 3 dimensions can be drawn using additional packages. The package plotrix has a function called pie3D() that is used for this.

```
# install plotrix library
install.packages("plotrix")

# Get the library.
library(plotrix)

# Create data for the graph.
x <-  c(21, 62, 10,53)
lbl <-  c("London","New York","Singapore","Mumbai")

# Plot the chart.
pie3D(x,labels = lbl, explode = 0.1, main = "Pie Chart of Countries")
```

Dot Chart

It is an alternative to bar charts, where the bars are replaced by dots i.e. it is a graphical display of data using dots. It is suitable for small to moderate sized data sets and are useful for highlighting clusters and gaps, as well as outliers.

Syntax:

dotchart (NumericVector, cex = 1, col = “black”, labels = NULL, main = NULL, pch = 1, sub = NULL, xlab = NULL) wherein

- NumericVector is the Numeric vector to be plotted
- cex is the plot scaling factor(size) . More the value of cex, more the plot size will be
- col is the colour of the dot
- labels is the A vector containing the label names for each plotted value.
- main is the Title of the dot chart
- pch is the numeric value which decides the type of plot … if pch=1 then dot, pch=2 then triangle, pch=3 then ‘+’
- sub is the subtitle of the dot chart
- xlab is the x axis label

Dot Chart Example

Example 1

```
dotchart(PlantGrowth$weight,col="red",pch=1,labels=PlantGrowth$group,
main="group vs weight", xlab="weight")

# different dot plots for different group of the same data set
pg <- PlantGrowth
pg$color[pg$group=="ctrl"] <- "red"
pg$color[pg$group=="trt1"] <- "Violet"
pg$color[pg$group=="trt2"] <- "blue“
# plot the dot chart
dotchart(PlantGrowth$weight, labels=PlantGrowth$group,cex=0.8,groups= PlantGrowth$group,
main="group vs weight",xlab="weight",gcolor="black",color=pg$color)
```

Example 2

```
data2 = USArrests[order(USArrests$Murder),]
dotchart(data2$Murder, labels = row.names(data2),
   cex = .5, main = "Murder arrests by state, 1973",
   xlab = "Murder arrests per 100,000 population")
```

Box Plots

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them. Boxplots are created in R by using the boxplot() function.

Syntax:

boxplot(x, data, notch, varwidth, names, main) wherein

- x is a vector or a formula.
- data is the data frame.
- notch is a logical value. Set as TRUE to draw a notch.
- varwidth is a logical value. Set as true to draw width of the box proportionate to the sample size.
- names are the group labels which will be printed under each boxplot.
- main is used to give a title to the graph.

First Quartile and Third Quartile

Definition:

- The **lower half** of a data set is the set of all values that are to the left of the median value when the data has been put into increasing order.
- The **upper half** of a data set is the set of all values that are to the right of the median value when the data has been put into increasing order.
- The **first quartile**, denoted by Q_1, is the median of the lower half of the data set. This means that about 25% of the numbers in the data set lie below Q_1 and about 75% lie above Q_1 .
- The **third quartile**, denoted by Q_3 , is the median of the upper half of the data set. This means that about 75% of the numbers in the data set lie below Q_3 and about 25% lie above Q_3 .

Example:

Let the dataset have values {3, 7, 8, 5, 12, 14, 21, 13, 18}.

Arrange the data in increasing order : 3, 5, 7, 8, 12, 13, 14, 18, 21.

First Quartile and Third Quartile cont’d

$$\underbrace{3,\ 5,\ 7,\ 8,}_{\text{“lower half”}}\ \underset{\text{median}}{12}\ \underbrace{13,\ 14,\ 18,\ 21}_{\text{“upper half”}}$$

The median is 12. and the lower half of the data is: {3, 5, 7, 8}.
The first quartile, Q_1, is the median of {3, 5, 7, 8}.
Since there is an even number of values, we need the mean of the middle two values to find the first quartile: $Q_1 = \frac{5+7}{2} = 6$
Similarly, the upper half of the data is: {13, 14, 18, 21}, so $Q_3 = \frac{14+18}{2} = 16$

Lab Exercise

- Find the first and third quartiles of the set {3, 7, 8, 5, 12, 14, 21, 15, 18, 14}.
- The following dollar amounts were the hourly collections from a Salvation Army kettle at a local store one day in December: \$19, \$26, \$25, \$37, \$32, \$28, \$22, \$23, \$29, \$34, \$39, and \$31. Determine the first quartile and third quartile for the amount collected.

Box Plots cont'd


```
# data set "mtcars" with the columns "mpg" and "cyl"
input <- mtcars[,c('mpg','cyl')]
print(head(input))

# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
      ylab = "Miles Per Gallon", main = "Mileage Data")

# draw boxplot with notch to find out how the medians of different data groups match  with each other.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
  ylab = "Miles Per Gallon", main = "Mileage Data", notch = TRUE,
  varwidth = TRUE, col = c("green","yellow","purple"),
  names = c("High","Medium","Low")
```

Lab Exercise

- Plot Boxplot for the set {3, 7, 8, 5, 12, 14, 21, 15, 18, 14}.
- Plot Boxplot for the set {3, 7, 8, 5, 12, 14, 21, 13, 18}.

Scatter Plot

A scatter plot is a two-dimensional data visualization that uses dots to represent the values obtained for two different variables - one plotted along the x-axis and the other plotted along the y-axis. It is used when the need is to show the relationship between two variables. It is also called correlation plots because it show how two variables are correlated.

Syntax:

plot(x, y, main, xlab, ylab, xlim, ylim, axes) and the description of the parameters:

- x is the data set whose values are the horizontal coordinates.
- y is the data set whose values are the vertical coordinates.
- main is the tile of the graph.
- xlab is the label in the horizontal axis.
- ylab is the label in the vertical axis.
- xlim is the limits of the values of x used for plotting.
- ylim is the limits of the values of y used for plotting.
- axes indicates whether both axes should be drawn on the plot.

Scatter Plot Example


```
# Get the input values.
input <- mtcars[,c('wt','mpg')]

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
  xlab = "Weight",
  ylab = "Milage",
  xlim = c(2.5,5),
  ylim = c(15,30),
  main = "Weight vs Milage"
)
```

Kernel Density Plot

Kernel density plot is usually a much more effective way to view the distribution of a variable.

Example:

```
# Kernel Density Plot
den = density(mtcars$mpg)
plot(den,main=" Kernel Density of Miles Per Gallon")
polygon(d, col="red", border="blue")
```

How to select a data Visualization?

- To trend your data over time:
  - Line charts
  - Bar charts
- To compare values of different categories:
  - Bar chart
  - Pie chart
- To show the composition of a total:
  - Pie chart
  - Stacked bar chart
- To understand relationships between factors:
  - Scatter plot
- To understand the distribution of data:
  - Scatter plot
  - Box plot

Saving the Plot

All the graphs (bar plot, pie chart, histogram, etc.) we plot in R are displayed on the screen by default. We can save these plots as a file on disk with the help of built-in functions. Plots can be saved as bitmap image (raster) which are fixed size or as vector image which are easily resizable.

Save plot as a bitmap image

Most of the image we come across like jpeg or png are bitmap image. They have a fixed resolution and are pixelated when zoomed enough. Functions that help us save plots in this format are jpeg(), png(), bmp() and tiff().

```
# get the temperature column of built-in dataset airquality
temp <- airquality$Temp # remainder of this section used this variable
print(temp)
```

- **Save as Jpeg image:**

  ```
  jpeg(file="saving_plot1.jpeg")
  hist(temp, col="darkgreen")
  dev.off() # call after all the plotting, to save the file and return control to the screen.
  ```

Save plot as a bitmap image

- **Save as Jpeg image cont'd:**

  This will save a jpeg image in the current directory. The resolution of the image by default will be 480x480 pixel.

- **Save as png image:**

  We can specify the resolution we want with arguments width and height. We can also specify the full path of the file we want to save if we don't want to save it in the current directory.

  The following code saves a png file with resolution 600x350.

```
png(file="C:/Datamentor/R-tutorial/saving_plot2.png",
width=600, height=350)
hist(temp, col="gold")
dev.off()
```

Save plot as a bitmap image cont'd

- **Save as bmp image:**

  Similarly, we can specify the size of our image in inch, cm or mm with the argument units and specify ppi with res.

  The following code saves a bmp file of size 6x4 inch and 100 ppi.

  ```
  bmp(file="saving_plot3.bmp", width=6, height=4, units="in", res=100)
  hist(temp, col="steelblue")
  dev.off()
  ```

- **Save as tiff image:**

  ```
  tiff(file="saving_plot3.tiff", width=6, height=4, units="in", res=100)
  hist(Temperature, col="steelblue")
  dev.off()
  ```

Save plot as a vector image

We can save our plots as vector image in pdf or postscript formats. The beauty of vector image is that it is easily resizable. Zooming on the image will not compromise its quality.

- **Save as pdf file:**

```
pdf(file="saving_plot4.pdf")
hist(temp, col="violet")
dev.off()
```

- **Save as postscript file:**

```
postscript(file="saving_plot4.ps")
hist(Temperature, col="violet")
dev.off()
```

3D Plot

There are many functions in R programming for creating 3D plots. In this section, weWe will discuss only the persp() function which can be used to create 3D surfaces in perspective view. This function mainly takes in three variables, x, y and z where x and y are vectors defining the location along x- and y-axis. The height of the surface (z-axis) will be in the matrix z.

```
cone <- function(x, y){ # function
sqrt(x^2+y^2)
}

#variable preparation
x <- y <- seq(-1, 1, length= 20)
z <- outer(x, y, cone)
persp(x, y, z) # simple 3D plot
persp(x, y, z, main="Perspective Plot of a Cone", zlab = "Height", theta = 30, phi = 15, col = "springgreen", shade = 0.5)
```

Heatmap

A heat map (or heatmap) is a graphical representation of data where the individual values contained in a matrix are represented as colors. It is a bit like looking a data table from above. It is really useful to display a general view of numerical data, not to extract specific data point.

```
# The mtcars dataset
data=as.matrix(mtcars)
head(data)

# Default Heatmap (left)
heatmap(data)

# Use 'scale' to normalize (right)
heatmap(data, scale="column")
```

Thank You
End of Lab 4

Experiment

- Explore any 5 built-in dataset and perform the following for each dataset
  - Display the summary
  - Display the structure
  - Display the first 5 rows
  - Display the number of rows
  - Display the number of columns
- Explore any built-in dataset and plot the following
  - Line Charts
  - Bar Plot
  - Histogram
  - Pie Chart
  - Dot Plots
  - Box Plot
  - Scatter Plot
  - Kernel Density Plot
- List and count the number of packages installed in the workspace
- Draw a Bubble Chart using ggplot2 package