Chapter 5
Data analytics and visualization with R

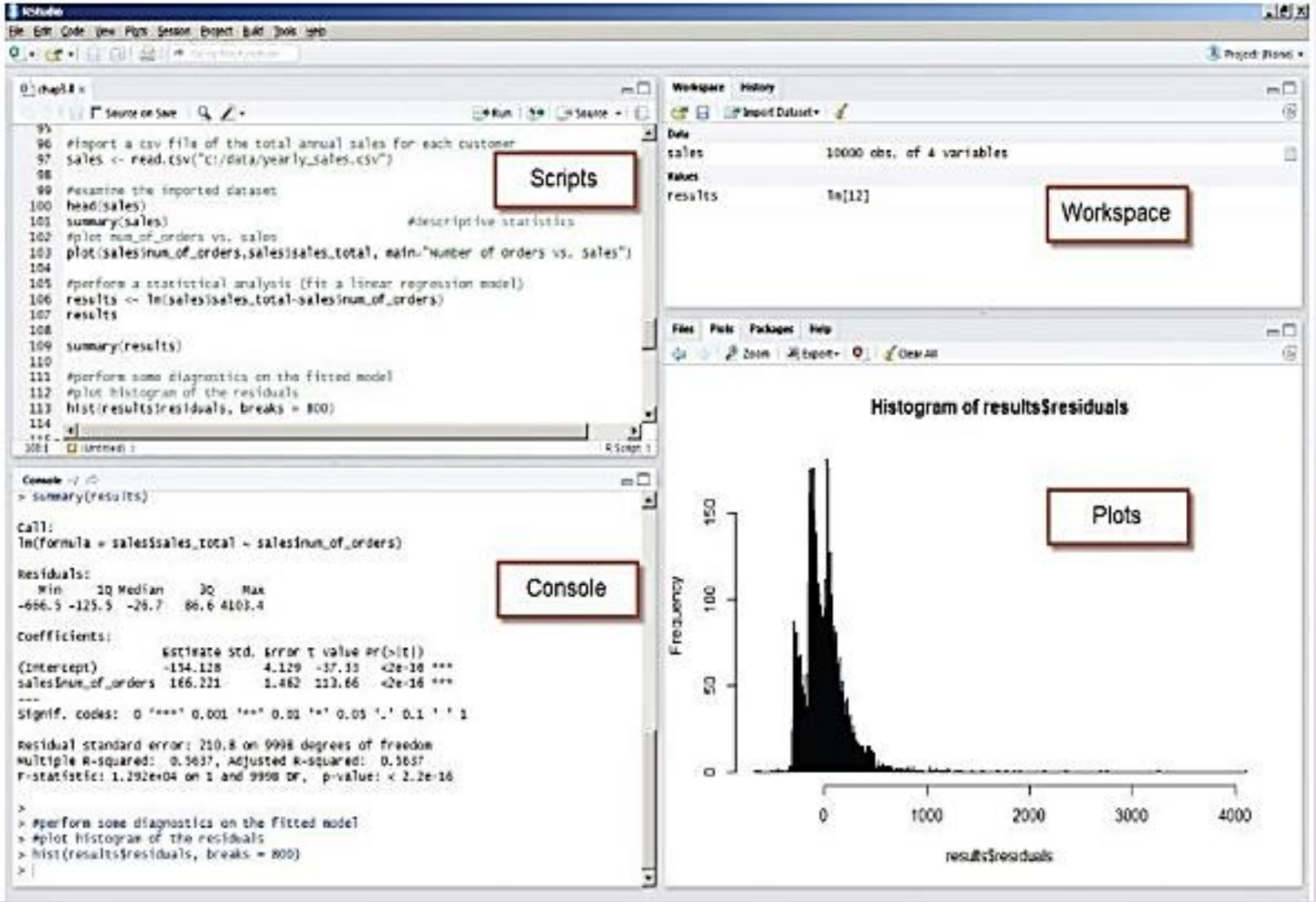
Topics

- Introduction to R:
  - Data Import and Export, Attribute and Data type,
  - Descriptive statistics.
- Exploratory Data Analysis:
  - Visualization before analysis,
  - Dirty Data,
  - Visualizing single variable,
  - Examining Multiple variable,
  - Data Exploration versus presentation.

R Graphical User Interfaces

- Software uses a command-line interface (CLI) that is similar to the BASH shell in Linux or the interactive versions of scripting languages such as Python. UNIX and Linux users can enter command R at the terminal prompt to use the CLI.
- For Windows installations, R comes with RGui.exe, which provides a basic graphical user interface (GUI). However, to improve the ease of writing, executing, and debugging R code, several additional GUIs have been written for R. Popular GUIs include the R commande, Rattle, and RStudio.

R Graphical User Interfaces



R Graphical User Interfaces

- The four highlighted window panes follow.
  - Scripts: Serves as an area to write and save R code
  - Workspace: Lists the datasets and variables in the R environment
  - Plots: Displays the plots generated by the R code and provides a straightforward
  - mechanism to export the plots
  - Console: Provides a history of the executed R code and the output.

Basic functions in R

- help(): obtain help information in R example help(lm).
- edit(): allow the user to edit the contents of an R variable.
- fix(): allow the user to update the contents of an R variable.
- save.image(): save the workspace environment, including variables and loaded libraries, into an .Rdata file.
- load.image(): An existing .Rdata file can be loaded using this.
- The dataset was imported into R using the read.csv() function as in the following code. sales <- read.csv("c:/data/yearly_sales.csv")
- R uses a forward slash (/) as the separator character in the directory and file paths.

Basic functions in R

- To simplify the import of multiple files with long path names, the **setwd()** function can be used to set the working directory for the subsequent import and export operations, as shown in the following R code.

```
setwd("c:/data/")

sales <- read.csv("yearly_sales.csv")
```

- read.table() and read.delim(), which are intended to import other common file types such as TXT.

```
sales_table <- read.table("yearly_sales.csv", header=TRUE, sep=",")

sales_delim <- read.delim("yearly_sales.csv", sep=",")
```

Basic functions in R

- The main difference between these import functions is the default values.
- For example, the read.delim() function expects the column separator to be a tab ("\t").
- In the event that the numerical data in a data file uses a comma for the decimal, R also provides two additional functions—read.csv2() and read.delim2()—to import such data.

Function	Headers	Separator	Decimal Point
read.table()	FALSE	" "	" " "."
read.csv()	TRUE	" " ","	" " "."
read.csv2()	TRUE	"." ","	" " ","
read.delim()	TRUE	"\t"	" " "."
read.delim2()	TRUE	"\t"	" " ","

Basic functions in R

- The analogous R functions such as write.table(), write.csv(), and write.csv2() enable exporting of R datasets to an external file.
- For example, the following R code adds an additional column to the sales dataset and exports the modified dataset to an external file.

```
# add a column for the average sales per order

sales$per_order <- sales$sales_total/sales$num_of_orders

# export data as tab delimited without the row names

write.table(sales,"sales_modified.txt", sep="\t", row.names=FALSE
```

Basic functions in R

- Sometimes it is necessary to read data from a database management system (DBMS).
- R packages such as **DBI** and **RODBC** are available for this purpose.
- These packages provide database interfaces for communication between R and DBMSs such as MySQL, Oracle, SQL Server, PostgreSQL, and Pivotal Greenplum.

Basic functions in R

- The following R code demonstrates how to install the RODBC package with the install.packages() function.
- The library() function loads the package into the R workspace.
- Finally, a connector (conn) is initialized for connecting to a Pivotal Greenplum database training2 via open database connectivity (ODBC) with user *user*.
- The training2 database must be defined either in the /etc/ODBC.ini configuration file or using the Administrative Tools under the Windows Control Panel.

```
install.packages("RODBC")

library(RODBC)

conn <- odbcConnect("training2", uid="user", pwd="password")
```

Basic functions in R

- The sqlQuery() function from the RODBC package.
- The following R code retrieves specific columns from the housing table in which household income (hinc) is greater than $1,000,000.

```
housing_data <- sqlQuery(conn, "select serialno, state, persons, rooms
                         from housing
                          where hinc > 1000000")
head(housing_data)
```

```
  serialno state persons rooms
1  3417867     6       2     7
2  3417867     6       2     7
3  4552088     6       5     9
4  4552088     6       5     9
5  8699293     6       5     5
6  8699293     6       5     5
```

Basic functions in R

- The jpeg() function, the following R code creates a new JPEG file, adds a histogram plot to the file, and then closes the file. Such techniques are useful when automating standard reports.
- Other functions, such as png(), bmp(), pdf(), and postscript(), are available in R to save plots in the desired format.

```
jpeg(file="c:/data/sales_hist.jpeg") # create a new jpeg file

hist(sales$num_of_orders) # export histogram to jpeg

dev.off() # shut off the graphic device
```

- http://cran.rproject.org/doc/manuals/r-release/R-data.html

Attributes in R

- Attributes can be categorized into four types: nominal, ordinal, interval, and ratio (NOIR).

	Categorical (Qualitative)		Numeric (Quantitative)	
	Nominal	**Ordinal**	**Interval**	**Ratio**
Definition	The values represent labels that distinguish one from another.	Attributes imply a sequence.	The difference between two values is meaningful.	Both the difference and the ratio of two values are meaningful.
Examples	ZIP codes, nationality, street names, gender, employee ID numbers, `TRUE` or `FALSE`	Quality of diamonds, academic grades, magnitude of earthquakes	Temperature in Celsius or Fahrenheit, calendar dates, latitudes	Age, temperature in Kelvin, counts, length, weight
Operations	$=, \neq$	$=, \neq,$ $<, \leq, >, \geq$	$=, \neq,$ $<, \leq, >, \geq,$ $+, -$	$=, \neq,$ $<, \leq, >, \geq,$ $+, -,$ $\times, \div$

Attributes in R

- Data of one attribute type may be converted to another.
- For example, the quality of diamonds {Fair, Good, Very Good, Premium, Ideal} is considered ordinal but can be converted to nominal {Good, Excellent} with a defined mapping.
- Similarly, a ratio attribute like Age can be converted into an ordinal attribute such as {Infant, Adolescent, Adult, Senior}.
- Understanding the attribute types in a given dataset is important to ensure that the appropriate descriptive statistics and analytic methods are applied and properly interpreted.
- For example, the mean and standard deviation of U.S. postal ZIP codes are not very meaningful or appropriate.

Numeric, Character, and Logical Data Types

```
i <- 1 # create a numeric variable

sport <- “football” # create a character variable

flag <- TRUE # create a logical variable
```

Numeric, Character, and Logical Data Types

- The class() function represents the abstract class of an object.
- The typeof() function determines the way an object is stored in memory.

class(i) # returns “numeric”

typeof(i) # returns “double”

class(sport) # returns “character”

typeof(sport) # returns “character”

class(flag) # returns “logical”

typeof(flag) # returns “logical”

Numeric, Character, and Logical Data Types

- The following R code illustrates how to test if i is an integer using the is.integer() function and to coerce i into a new integer variable, j, using the as.integer() function. Similar functions can be applied for double, character, and logical types.

```
is.integer(i) # returns FALSE

j <- as.integer(i) # coerces contents of i into an integer

is.integer(j) # returns TRUE
```

Numeric, Character, and Logical Data Types

- The application of the length() function reveals that the created variables each have a length of 1.
- One might have expected the returned length of sport to have been 8 for each of the characters in the string “football”. However, these three variables are actually one element, vectors.

length(i) # returns 1

length(flag) # returns 1

length(sport) # returns 1 (not 8 for “football”)

Vectors

- Vectors are a basic building block for data in R.
- A vector can only consist of values in the same class.
- The tests for vectors can be conducted using the is.vector() function.

```
is.vector(i) # returns TRUE

is.vector(flag) # returns TRUE

is.vector(sport) # returns TRUE
```

Vectors

- The following R code illustrates how a vector can be created using the combine function, c() or the colon operator, :, to build a vector from the sequence of integers from 1 to 5.

```
u <- c("red", "yellow", "blue") # create a vector "red" "yellow" "blue"
u # returns "red" "yellow" "blue"
u[1] # returns "red" (1st element in u)
v <- 1:5 # create a vector 1 2 3 4 5
v # returns 1 2 3 4 5
sum(v) # returns 15
w <- v * 2 # create a vector 2 4 6 8 10
w # returns 2 4 6 8 10
w[3] # returns 6 (the 3rd element of w)
```

- The code, related to the z vector, indicates how logical comparisons can be built to extract certain elements of a given vector.

```
z <- v + w # sums two vectors element by element

z # returns 3 6 9 12 15

z > 8 # returns FALSE FALSE TRUE TRUE TRUE

z[z > 8] # returns 9 12 15

z[z > 8 | z < 5] # returns 3 9 12 15 ("|" denotes "or")
```

Vectors

- The vector() function, by default, creates a logical vector.
- A vector of a different type can be specified by using the mode parameter.
- The vector c, an integer vector of length 0, may be useful when the number of elements is not initially known and the new elements will later be added to the end of the vector as the values become available.

a <- vector(length=3) # create a logical vector of length 3

a # returns FALSE FALSE FALSE

b <- vector(mode="numeric", 3) # create a numeric vector of length 3

typeof(b) # returns "double"b[2] <- 3.1 # assign 3.1 to the 2nd element

b # returns 0.0 3.1 0.0

c <- vector(mode="integer", 0) # create an integer vector of length 0

c # returns integer(0)

length(c) # returns 0

Vectors

- Although vectors may appear to be analogous to arrays of one dimension, they are technically dimensionless, as seen in the following R code.

```
length(b) # returns 3

dim(b) # returns NULL (an undefined value)
```

Array

- The array() function can be used to restructure a vector as an array.
- For example, the following R code builds a three-dimensional array to hold the quarterly sales for three regions over a two-year period and then assign the sales amount of $158,000 to the second region for the first quarter of the first year.

```
# the dimensions are 3 regions, 4 quarters, and 2 years

quarterly_sales <- array(0, dim=c(3,4,2))

quarterly_sales[2,1,1] <- 158000

quarterly_sales
```

Matrix

- A two-dimensional array is known as a matrix. The following code initializes a matrix to hold the quarterly sales for the three regions. The parameters nrow and ncol define the number of rows and columns, respectively, for the sales_matrix.

```
sales_matrix <- matrix(0, nrow = 3, ncol = 4)

sales_matrix
```

Matrix

- R provides the standard matrix operations such as addition, subtraction, and multiplication, as well as the transpose function t() and the inverse matrix function matrix.inverse() included in the matrixcalc package. The following R code builds a 3 × 3 matrix, M, and multiplies it by its inverse to obtain the identity matrix.

```
library(matrixcalc)

M <- matrix(c(1,3,3,5,0,4,3,3,3),nrow = 3,ncol = 3) # build a 3x3 matrix

M %*% matrix.inverse(M) # multiply M by inverse(M)
```

Data Frame

- Data frames provide a structure for storing and accessing several variables of possibly different data types. In fact, as the
- a data frame is created by the read.csv() function and is.data.frame() function indicates where the said variable is data frame or not.

```
#import a CSV file of the total annual sales for each customer

sales <- read.csv("c:/data/yearly_sales.csv")

is.data.frame(sales) # returns TRUE
```

Data Frame

- The variables stored in the data frame can be easily accessed using the $ notation.
- The following R code illustrates that in this example, each variable is a vector with the exception of gender, which was, by a read.csv() default, imported as a factor.
- A factor denotes a categorical variable, typically with a few finite levels such as “F” and “M” in the case of gender.

```
length(sales$num_of_orders) # returns 10000
(number of customers)
is.vector(sales$cust_id) # returns TRUE
is.vector(sales$sales_total) # returns TRUE
is.vector(sales$num_of_orders) # returns TRUE
is.vector(sales$gender) # returns FALSE
is.factor(sales$gender) # returns TRUE
```

Data Frame

- The following use of the str() function provides the structure of the sales data frame.
- This function identifies the integer and numeric (double) data types, the factor variables and levels, as well as the first few values for each variable.

```
str(sales) # display structure of the data frame object
```

Data Frame

- A subset of the data frame can be retrieved through subsetting operators. R's subsetting operators are powerful in that they allow one to express complex operations in a succinct fashion and easily retrieve a subset of the dataset.

```
# extract the fourth column of the sales data frame
sales[,4]
# extract the gender column of the sales data frame
sales$gender
# retrieve the first two rows of the data frame
sales[1:2,]
# retrieve the first, third, and fourth columns
sales[,c(1,3,4)]
# retrieve both the cust_id and the sales_total columns
sales[,c("cust_id", "sales_total")]
# retrieve all the records whose gender is female
sales[sales$gender=="F",]
```

Data Frames

- The class of the sales variable is a data frame. However, the type of the sales variable is a list. A list is a collection of objects that can be of various types, including other lists.

class(sales)

“data.frame”

typeof(sales)

“list”

Data Frames

- The class of the sales variable is a data frame. However, the type of the sales variable is a list. A list is a collection of objects that can be of various types, including other lists.

class(sales)

“data.frame”

typeof(sales)

“list”

Lists

- Lists can contain any type of objects, including other lists.
- Using the vector v and the matrix M created in earlier examples, the following R code creates assortment, a list of different object types.

```
# build an assorted list of a string, a numeric, a list, a vector,

# and a matrix

housing <- list("own", "rent")

assortment <- list("football", 7.5, housing, v, M)

assortment
```

Lists

- In displaying the contents of assortment, the use of the double brackets, [[]], is of particular importance.
- As the following R code illustrates, the use of the single set of brackets only accesses an item in the list, not its content.

```
# examine the fifth object, M, in the list

class(assortment[5]) # returns "list"

length(assortment[5]) # returns 1

class(assortment[[5]]) # returns "matrix"

length(assortment[[5]]) # returns 9 (for the 3x3 matrix)
```

- As seen earlier in the data frame discussion, the str() function offers details about the structure of a list.

```
str(assortment)
```

Factor

- Factors were briefly introduced during the discussion of the **gender** variable in the data frame sales.
- In this case, gender could assume one of two levels: F or M.
- Factors can be ordered or not ordered. In the case of gender, the levels are not ordered.

class(sales$gender) # returns “factor”

is.ordered(sales$gender) # returns FALSE

head(sales$gender) # display first six values and the levels.

Factor

- Included with the ggplot2 package, the diamonds data frame contains three ordered factors.
- Examining the cut factor, there are five levels in order of improving cut: Fair, Good, Very Good, Premium, and Ideal.
- Thus sales$gender contains nominal data, and diamonds$cut contains ordinal data.

```
library(ggplot2)

data(diamonds) # load the data frame into the R workspace

str(diamonds) #details about the structure of data.

head(diamonds$cut) # display first six values and the levels
```

Factor

- To categorize sales$sales_totals into three groups—small, medium, and big—according to the amount of the sales with the following code. These groupings are the basis for the new ordinal factor, spender, with levels {small, medium, big}.

```
# build an empty character vector of the same length as sales

sales_group <- vector(mode="character",

length=length(sales$sales_total))

# group the customers according to the sales amount

sales_group[sales$sales_total<100] <- "small"

sales_group[sales$sales_total>=100 & sales$sales_total<500] <- "medium"

sales_group[sales$sales_total>=500] <- "big"
```

Factor

```
# create and add the ordered factor to the sales data frame
spender <- factor(sales_group,levels=c("small", "medium", "big"),
                  ordered = TRUE)
sales <- cbind(sales,spender)
str(sales$spender)
head(sales$spender)
```

Factor

```
# create and add the ordered factor to the sales data frame
spender <- factor(sales_group,levels=c("small", "medium", "big"),
                  ordered = TRUE)
sales <- cbind(sales,spender)
str(sales$spender)
head(sales$spender)
```

- The cbind() function is used to combine variables column-wise.
- The rbind() function is used to combine datasets row-wise.
- The use of factors is important in several R statistical modeling functions, such as analysis of variance, aov(),

Contingency Tables

- In R, table refers to a class of objects used to store the observed counts across the factors for a given dataset.
- Such a table is commonly referred to as a contingency table and is the basis for performing a statistical test on the independence of the factors used to build the table.
- The following R code builds a contingency table based on the sales$gender and sales$spender factors.

```
# build a contingency table based on the gender and spender factors

sales_table <- table(sales$gender,sales$spender)

sales_table

class(sales_table) # returns "table"

typeof(sales_table) # returns "integer"

dim(sales_table) # returns 2 3

# performs a chi-squared test

summary(sales_table)
```

Descriptive Statistics

- The summary() function provides several descriptive statistics, such as the mean and median, about a variable such as the sales data frame.
- The results now include the counts for the three levels of the spender variable based on the earlier examples involving factors.

```
summary(sales)
```

Descriptive Statistics

- The following code provides some common R functions that include descriptive statistics.In parentheses, the comments describe the functions.

```
# to simplify the function calls, assign

x <- sales$sales_total

y <- sales$num_of_orders

cor(x,y) # returns 0.7508015 (correlation)

cov(x,y) # returns 345.2111 (covariance)

IQR(x) # returns 215.21 (interquartile range). The IQR() function provides the difference between the third and the first quartiles.

mean(x) # returns 249.4557 (mean)

median(x) # returns 151.65 (median)

range(x) # returns 30.02 7606.09 (min max)

sd(x) # returns 319.0508 (std. dev.)

var(x) # returns 101793.4 (variance)
```

Descriptive Statistics

- The function apply() is useful when the same function is to be applied to several variables in a data frame.
- For example, the following R code calculates the standard deviation for the first three variables in sales. In the code, setting MARGIN=2 specifies that the sd() function is applied over the columns.
- Other functions, such as lapply() and sapply(), apply a function to a list or vector.

```
apply(sales[,c(1:3)], MARGIN=2, FUN=sd)
```

Descriptive Statistics

- my_range(), to compute the difference between the maximum and minimum values returned by the range() function. In general, user-defined functions are useful for any task or operation that needs to be frequently repeated. More information on user-defined functions is available by entering help("function") in the console.

```
# build a function to provide the difference between

# the maximum and the minimum values

my_range <- function(v) {range(v)[2] - range(v)[1]}

my_range(x)

7576.07
```